Abstract

We utilize and extend the method of *shallow semantic embeddings* (SSEs) in classical higher-order logic (HOL) to construct a custom theorem proving environment for *abstract objects theory* (AOT) on the basis of Isabelle/HOL.

SSEs are a means for universal logical reasoning by translating a target logic to HOL using a representation of its semantics. AOT is a foundational metaphysical theory, developed by Edward Zalta, that explains the objects presupposed by the sciences as *abstract objects* that reify property patterns. In particular, AOT aspires to provide a philosphically grounded basis for the construction and analysis of the objects of mathematics.

We can support this claim by verifying Uri Nodelman's and Edward Zalta's reconstruction of Frege's theorem: we can confirm that the Dedekind-Peano postulates for natural numbers are consistently derivable in AOT using Frege's method. Furthermore, we can suggest and discuss generalizations and variants of the construction and can thereby provide theoretical insights into, and contribute to the philosophical justification of, the construction.

In the process, we can demonstrate that our method allows for a nearly transparent exchange of results between traditional pen-and-paper-based reasoning and the computerized implementation, which in turn can retain the automation mechanisms available for Isabelle/HOL.

During our work, we could significantly contribute to the evolution of our target theory itself, while simultaneously solving the technical challenge of using an SSE to implement a theory that is based on logical foundations that significantly differ from the meta-logic HOL.

In general, our results demonstrate the fruitfulness of the practice of Computational Metaphysics, i.e. the application of computational methods to metaphysical questions and theories.

A full description of this formalization including references can be found at http://dx.doi.org/10.17169/refubium-35141.

The version of Principia Logico-Metaphysica (PLM) implemented in this formalization can be found at http://mally.stanford.edu/principia-2021-10-13.pdf, while the latest version of PLM is available at http://mally.stanford.edu/principia.pdf.

Contents

1 References

A full description of this formalization including references can be found at http://dx.doi.org/10.17169/refubium-35141.

The version of Principia Logico-Metaphysica (PLM) implemented in this formalization can be found at http://mally.stanford.edu/principia-2021-10-13.pdf, while the latest version of PLM is available at http://mally.stanford.edu/principia.pdf.

2 Model for the Logic of AOT

We introduce a primitive type for hyperintensional propositions.

typedecl o

To be able to model modal operators following Kripke semantics, we introduce a primitive type for possible worlds and assert, by axiom, that there is a surjective function mapping propositions to the boolean-valued functions acting on possible worlds. We call the result of applying this function to a proposition the Montague intension of the proposition.

typedecl *w* — The primtive type of possible worlds.
axiomatization *AOT-model-d*o :: ‹o⇒(*w*⇒*bool*)› **where**
  *d*o*-surj*: ‹*surj AOT-model-d*o›

The axioms of PLM require the existence of a non-actual world.

consts w_0 :: *w* — The designated actual world.
axiomatization where *AOT-model-nonactual-world*: ‹∃ *w* . *w* ≠ w_0›

Validity of a proposition in a given world can now be modelled as the result of applying that world to the Montague intension of the proposition.

definition *AOT-model-valid-in* :: ‹*w*⇒o⇒*bool*› **where**
  ‹*AOT-model-valid-in w φ* ≡ *AOT-model-d*o *φ w*›

By construction, we can choose a proposition for any given Montague intension, s.t. the proposition is valid in a possible world iff the Montague intension evaluates to true at that world.

definition *AOT-model-proposition-choice* :: ‹(*w*⇒*bool*) ⇒ o› (**binder** ‹ε_o › *8*)
  where ‹ε_o *w*. *φ w* ≡ (*inv AOT-model-d*o) *φ*›
lemma *AOT-model-proposition-choice-simp*: ‹*AOT-model-valid-in w* (ε_o *w*. *φ w*) = *φ w*›
  by (*simp add*: *surj-f-inv-f*[*OF d*o*-surj*] *AOT-model-valid-in-def*
        AOT-model-proposition-choice-def)

Nitpick can trivially show that there are models for the axioms above.

lemma ‹*True*› **nitpick**[*satisfy*, *user-axioms*, *expect* = *genuine*] **..**

typedecl *ω* — The primtive type of ordinary objects/urelements.

Validating extended relation comprehension requires a large set of special urelements. For simple models that do not validate extended relation comprehension (and consequently the predecessor axiom in the theory of natural numbers), it suffices to use a primitive type as *σ*, i.e. **typedecl** *σ*.

typedecl *σ′*
typedef *σ* = ‹*UNIV*::((*ω* ⇒ *w* ⇒ *bool*) *set* × (*ω* ⇒ *w* ⇒ *bool*) *set* × *σ′*) *set*› **..**

typedecl *null* — Null-urelements representing non-denoting terms.

datatype *υ* = *ωυ ω* | *συ σ* | *is-nullυ*: *nullυ null* — Type of urelements

Urrelations are proposition-valued functions on urelements. Urrelations are required to evaluate to necessarily false propositions for null-urelements (note that there may be several distinct necessarily false propositions).

typedef *urrel* = ‹{ *φ* . ∀ *x w* . ¬*AOT-model-valid-in w* (*φ* (*nullυ x*)) }›
  by (*rule exI*[**where** *x*=‹λ *x* . (ε_o *w* . ¬*is-nullυ x*)›])

(*auto simp*: *AOT-model-proposition-choice-simp*)

Abstract objects will be modelled as sets of urrelations and will have to be mapped surjectively into the set of special urelements. We show that any mapping from abstract objects to special urelements has to involve at least one large set of collapsed abstract objects. We will use this fact to extend arbitrary mappings from abstract objects to special urelements to surjective mappings.

lemma *ασ-pigeonhole*:
— For any arbitrary mapping $\alpha\sigma$ from sets of urrelations to special urelements, there exists an abstract object x, s.t. the cardinal of the set of special urelements is strictly smaller than the cardinal of the set of abstract objects that are mapped to the same urelement as x under $\alpha\sigma$.
‹$\exists x$. $|UNIV{::}\sigma\ set| <o\ |\{y\ .\ \alpha\sigma\ x = \alpha\sigma\ y\}|$›
for $\alpha\sigma$ :: ‹*urrel set* ⇒ σ›
proof(*rule ccontr*)
have *card-σ-set-set-bound*: ‹$|UNIV{::}\sigma\ set\ set| \leq o\ |UNIV{::}urrel\ set|$›
proof −
let *?pick* = ‹$\lambda u\ s\ .\ \varepsilon_o\ w$. *case u of* $(\sigma\upsilon\ s') \Rightarrow s' \in s$ | - ⇒ *False*›
have ‹$\exists f :: \sigma\ set \Rightarrow urrel$. *inj f*›
proof
show ‹*inj* (λ*s* . *Abs-urrel* (λ*u* . *?pick u s*))›
proof(*rule injI*)
fix *x y*
assume ‹*Abs-urrel* (λ*u*. *?pick u x*) = *Abs-urrel* (λ*u*. *?pick u y*)›
hence ‹(λ*u*. *?pick u x*) = (λ*u*. *?pick u y*)›
by (*auto intro*!: *Abs-urrel-inject*[*THEN iffD1*]
simp: *AOT-model-proposition-choice-simp*)
hence ‹*AOT-model-valid-in* w_0 (*?pick* (συ *s*) *x*) =
AOT-model-valid-in w_0 (*?pick* (συ *s*) *y*)›
for *s* **by** *metis*
hence ‹(*s* ∈ *x*) = (*s* ∈ *y*)› **for** *s*
by (*auto simp*: *AOT-model-proposition-choice-simp*)
thus ‹*x* = *y*›
by *blast*
qed
qed
thus *?thesis*
by (*metis card-of-image inj-imp-surj-inv*)
qed

Assume, for a proof by contradiction, that there is no large collapsed set.

assume ‹$\nexists x$. $|UNIV{::}\sigma\ set| <o\ |\{y\ .\ \alpha\sigma\ x = \alpha\sigma\ y\}|$›
hence *A*: ‹$\forall x$. $|\{y\ .\ \alpha\sigma\ x = \alpha\sigma\ y\}| \leq o\ |UNIV{::}\sigma\ set|$›
by *auto*
have *union-univ*: ‹$(\bigcup x \in range(inv\ \alpha\sigma)\ .\ \{y\ .\ \alpha\sigma\ x = \alpha\sigma\ y\}) = UNIV$›
by *auto* (*meson f-inv-into-f range-eqI*)

We refute by case distinction: there is either finitely many or infinitely many special urelements and in both cases we can derive a contradiction from the assumption above.

{

Finite case.

assume *finite-σ-set*: ‹*finite* (*UNIV*::σ *set*)›
hence *finite-collapsed*: ‹*finite* {*y* . ασ *x* = ασ *y*}› **for** *x*
using *A card-of-ordLeq-infinite* **by** *blast*
hence *0*: ‹$\forall x$. *card* {*y* . ασ *x* = ασ *y*} ≤ *card* (*UNIV*::σ *set*)›
by (*metis A finite-σ-set card-of-ordLeq inj-on-iff-card-le*)
have *1*: ‹*card* (*range* (*inv* ασ)) ≤ *card* (*UNIV*::σ *set*)›
using *finite-σ-set card-image-le* **by** *blast*
hence *2*: ‹*finite* (*range* (*inv* ασ))›
using *finite-σ-set* **by** *blast*

define *n* **where** ‹*n* = *card* (*UNIV*::*urrel set set*)›
define *m* **where** ‹*m* = *card* (*UNIV*::σ *set*)›

```
  have ‹n = card (⋃x ∈ range(inv ασ) . {y . ασ x = ασ y})›
    unfolding n-def using union-univ by argo
  also have ‹... ≤ (∑ i∈range (inv ασ). card {y. ασ i = ασ y})›
    using card-UN-le 2 by blast
  also have ‹... ≤ (∑ i∈range (inv ασ). card (UNIV::σ set))›
    by (metis (no-types, lifting) 0 sum-mono)
  also have ‹... ≤ card (range (inv ασ)) ∗ card (UNIV::σ set)›
    using sum-bounded-above by auto
  also have ‹... ≤ card (UNIV::σ set) ∗ card (UNIV::σ set)›
    using 1 by force
  also have ‹... = m∗m›
    unfolding m-def by blast
  finally have n-upper: ‹n ≤ m∗m›.

  have ‹finite (⋃x ∈ range(inv ασ) . {y . ασ x = ασ y})›
    using 2 finite-collapsed by blast
  hence finite-αset: ‹finite (UNIV::urrel set set)›
    using union-univ by argo

  have ‹2^2^m = (2::nat)^(card (UNIV::σ set set))›
    by (metis Pow-UNIV card-Pow finite-σ-set m-def)
  moreover have ‹card (UNIV::σ set set) ≤ (card (UNIV::urrel set))›
    using card-σ-set-set-bound
    by (meson Finite-Set.finite-set card-of-ordLeq finite-αset
          finite-σ-set inj-on-iff-card-le)
  ultimately have ‹2^2^m ≤ (2::nat)^(card (UNIV:: urrel set))›
    by simp
  also have ‹... = n›
    unfolding n-def
    by (metis Finite-Set.finite-set Pow-UNIV card-Pow finite-αset)
  finally have ‹2^2^m ≤ n› by blast
  hence ‹2^2^m ≤ m∗m› using n-upper by linarith
  moreover {
    have ‹(2::nat)^(2^m) ≥ (2^(m + 1))›
      by (metis Suc-eq-plus1 Suc-leI less-exp one-le-numeral power-increasing)
    also have ‹(2^(m + 1)) = (2::nat) ∗ 2^m›
      by auto
    have ‹m < 2^m›
      by (simp add: less-exp)
    hence ‹m∗m < (2^m)∗(2^m)›
      by (simp add: mult-strict-mono)
    moreover have ‹... = 2^(m+m)›
      by (simp add: power-add)
    ultimately have ‹m∗m < 2 ^ (m + m)› by presburger
    moreover have ‹m+m ≤ 2^m›
    proof (induct m)
      case 0
      thus ?case by auto
    next
      case (Suc m)
      thus ?case
        by (metis Suc-leI less-exp mult-2 mult-le-mono2 power-Suc)
    qed
    ultimately have ‹m∗m < 2^2^m›
      by (meson less-le-trans one-le-numeral power-increasing)
  }
  ultimately have False by auto
 }
 moreover {
```

Infinite case.

```
  assume ‹infinite (UNIV::σ set)›
```

hence *Cinfσ*: ‹*Cinfinite* |*UNIV*::*σ set*|›
by (*simp add*: *cinfinite-def*)
have *1*: ‹|*range* (*inv ασ*)| ≤*o* |*UNIV*::*σ set*|›
by *auto*
have *2*: ‹∀ *i*∈*range* (*inv ασ*). |{*y* . *ασ i* = *ασ y*}| ≤*o* |*UNIV*::*σ set*|›
proof
fix *i* :: ‹*urrel set*›
assume ‹*i* ∈ *range* (*inv ασ*)›
show ‹|{*y* . *ασ i* = *ασ y*}| ≤*o* |*UNIV*::*σ set*|›
using *A* **by** *blast*
qed
have ‹|⋃ ((λ*i*. {*y*. *ασ i* = *ασ y*}) ‘ (*range* (*inv ασ*)))| ≤*o*
|*Sigma* (*range* (*inv ασ*)) (λ*i*. {*y*. *ασ i* = *ασ y*})|›
using *card-of-UNION-Sigma* **by** *blast*
hence ‹|*UNIV*::*urrel set set*| ≤*o*
|*Sigma* (*range* (*inv ασ*)) (λ*i*. {*y*. *ασ i* = *ασ y*})|›
using *union-univ* **by** *argo*
moreover have ‹|*Sigma* (*range* (*inv ασ*)) (λ*i*. {*y*. *ασ i* = *ασ y*})| ≤*o* |*UNIV*::*σ set*|›
using *card-of-Sigma-ordLeq-Cinfinite*[*OF Cinfσ*, *OF 1*, *OF 2*] **by** *blast*
ultimately have ‹|*UNIV*::*urrel set set*| ≤*o* |*UNIV*::*σ set*|›
using *ordLeq-transitive* **by** *blast*
moreover {
have ‹|*UNIV*::*σ set*| <*o* |*UNIV*::*σ set set*|›
by *auto*
moreover have ‹|*UNIV*::*σ set set*| ≤*o* |*UNIV*::*urrel set*|›
using *card-σ-set-set-bound* **by** *blast*
moreover have ‹|*UNIV*::*urrel set*| <*o* |*UNIV*::*urrel set set*|›
by *auto*
ultimately have ‹|*UNIV*::*σ set*| <*o* |*UNIV*::*urrel set set*|›
by (*metis ordLess-imp-ordLeq ordLess-ordLeq-trans*)
}
ultimately have *False*
using *not-ordLeq-ordLess* **by** *blast*
}
ultimately show *False* **by** *blast*
qed

We introduce a mapping from abstract objects (i.e. sets of urrelations) to special urelements $\alpha\sigma$ that is surjective and distinguishes all abstract objects that are distinguished by a (not necessarily surjective) mapping $\alpha\sigma'$. $\alpha\sigma'$ will be used to model extended relation comprehension.

consts *ασ'* :: ‹*urrel set* ⇒ *σ*›
consts *ασ* :: ‹*urrel set* ⇒ *σ*›

specification(*ασ*)
ασ-surj: ‹*surj ασ*›
ασ-ασ': ‹*ασ x* = *ασ y* ⟹ *ασ' x* = *ασ' y*›
proof –
obtain *x* **where** *x-prop*: ‹|*UNIV*::*σ set*| <*o* |{*y*. *ασ' x* = *ασ' y*}|›
using *ασ-pigeonhole* **by** *blast*
have ‹∃*f* :: *urrel set* ⇒ *σ* . *f* ‘ {*y*. *ασ' x* = *ασ' y*} = *UNIV* ∧ *f x* = *ασ' x*›
proof –
have ‹∃*f* :: *urrel set* ⇒ *σ* . *f* ‘ {*y*. *ασ' x* = *ασ' y*} = *UNIV*›
by (*simp add*: *x-prop card-of-ordLeq2 ordLess-imp-ordLeq*)
then obtain *f* :: ‹*urrel set* ⇒ *σ*› **where** ‹*f* ‘ {*y*. *ασ' x* = *ασ' y*} = *UNIV*›
by *presburger*
moreover obtain *a* **where** ‹*f a* = *ασ' x*› **and** ‹*ασ' a* = *ασ' x*›
by (*smt* (*verit*, *best*) *calculation UNIV-I image-iff mem-Collect-eq*)
ultimately have ‹(*f* (*a* := *f x*, *x* := *f a*)) ‘ {*y*. *ασ' x* = *ασ' y*} = *UNIV* ∧
(*f* (*a* := *f x*, *x* := *f a*)) *x* = *ασ' x*›
by (*auto simp*: *image-def*)
thus *?thesis* **by** *blast*
qed
then obtain *f* **where** *fimage*: ‹*f* ‘ {*y*. *ασ' x* = *ασ' y*} = *UNIV*›

```
            and fx: ‹f x = ασ' x›
    by blast

  define ασ :: ‹urrel set ⇒ σ› where
    ‹ασ ≡ λ urrels . if ασ' urrels = ασ' x ∧ f urrels ∉ range ασ'
                     then f urrels
                     else ασ' urrels›
  have ‹surj ασ›
  proof -
    {
    fix s :: σ
    {
      assume ‹s ∈ range ασ'›
      hence 0: ‹ασ' (inv ασ' s) = s›
        by (meson f-inv-into-f)
      {
        assume ‹s = ασ' x›
        hence ‹ασ x = s›
          using ασ-def fx by presburger
        hence ‹∃ f . ασ (f s) = s›
          by auto
      }
      moreover {
        assume ‹s ≠ ασ' x›
        hence ‹ασ (inv ασ' s) = s›
          unfolding ασ-def 0 by presburger
        hence ‹∃ f . ασ (f s) = s›
          by blast
      }
      ultimately have ‹∃ f . ασ (f s) = s›
        by blast
    }
    moreover {
      assume ‹s ∉ range ασ'›
      moreover obtain urrels where ‹f urrels = s› and ‹ασ' x = ασ' urrels›
        by (smt (verit, best) UNIV-I fimage image-iff mem-Collect-eq)
      ultimately have ‹ασ urrels = s›
        using ασ-def by presburger
      hence ‹∃ f . ασ (f s) = s›
        by (meson f-inv-into-f range-eqI)
    }
    ultimately have ‹∃ f . ασ (f s) = s›
      by blast
    }
    thus ?thesis
      by (metis surj-def)
  qed
  moreover have ‹∀ x y. ασ x = ασ y ⟶ ασ' x = ασ' y›
    by (metis ασ-def rangeI)
  ultimately show ?thesis
    by blast
qed
```

For extended models that validate extended relation comprehension (and consequently the predecessor axiom), we specify which abstract objects are distinguished by $\alpha\sigma'$.

```
definition urrel-to-ωrel :: ‹urrel ⇒ (ω ⇒ w ⇒ bool)› where
  ‹urrel-to-ωrel ≡ λ r u w . AOT-model-valid-in w (Rep-urrel r (ωυ u))›
definition ωrel-to-urrel :: ‹(ω ⇒ w ⇒ bool) ⇒ urrel› where
  ‹ωrel-to-urrel ≡ λ φ . Abs-urrel
    (λ u . ε∘ w . case u of ωυ x ⇒ φ x w | - ⇒ False)›

definition AOT-urrel-ωequiv :: ‹urrel ⇒ urrel ⇒ bool› where
  ‹AOT-urrel-ωequiv ≡ λ r s . ∀ u v . AOT-model-valid-in v (Rep-urrel r (ωυ u)) =
```

```
                              AOT-model-valid-in v (Rep-urrel s (ων u))›

lemma urrel-ωrel-quot: ‹Quotient3 AOT-urrel-ωequiv urrel-to-ωrel ωrel-to-urrel›
proof(rule Quotient3I)
  show ‹urrel-to-ωrel (ωrel-to-urrel a) = a› for a
    unfolding ωrel-to-urrel-def urrel-to-ωrel-def
    apply (rule ext)
    apply (subst Abs-urrel-inverse)
    by (auto simp: AOT-model-proposition-choice-simp)
next
  show ‹AOT-urrel-ωequiv (ωrel-to-urrel a) (ωrel-to-urrel a)› for a
    unfolding ωrel-to-urrel-def AOT-urrel-ωequiv-def
    apply (subst (1 2) Abs-urrel-inverse)
    by (auto simp: AOT-model-proposition-choice-simp)
next
  show ‹AOT-urrel-ωequiv r s = (AOT-urrel-ωequiv r r ∧ AOT-urrel-ωequiv s s ∧
                              urrel-to-ωrel r = urrel-to-ωrel s)› for r s
  proof
    assume ‹AOT-urrel-ωequiv r s›
    hence ‹AOT-model-valid-in v (Rep-urrel r (ων u)) =
          AOT-model-valid-in v (Rep-urrel s (ων u))› for u v
      using AOT-urrel-ωequiv-def by metis
    hence ‹urrel-to-ωrel r = urrel-to-ωrel s›
      unfolding urrel-to-ωrel-def
      by simp
    thus ‹AOT-urrel-ωequiv r r ∧ AOT-urrel-ωequiv s s ∧
         urrel-to-ωrel r = urrel-to-ωrel s›
      unfolding AOT-urrel-ωequiv-def
      by auto
  next
    assume ‹AOT-urrel-ωequiv r r ∧ AOT-urrel-ωequiv s s ∧
           urrel-to-ωrel r = urrel-to-ωrel s›
    hence ‹AOT-model-valid-in v (Rep-urrel r (ων u)) =
          AOT-model-valid-in v (Rep-urrel s (ων u))› for u v
      by (metis urrel-to-ωrel-def)
    thus ‹AOT-urrel-ωequiv r s›
      using AOT-urrel-ωequiv-def by presburger
  qed
qed

specification (ασ′)
  ασ-eq-ord-exts-all:
    ‹ασ′ a = ασ′ b ⟹ (⋀s . urrel-to-ωrel s = urrel-to-ωrel r ⟹ s ∈ a)
      ⟹ (⋀ s . urrel-to-ωrel s = urrel-to-ωrel r ⟹ s ∈ b)›
  ασ-eq-ord-exts-ex:
    ‹ασ′ a = ασ′ b ⟹ (∃ s . s ∈ a ∧ urrel-to-ωrel s = urrel-to-ωrel r)
      ⟹ (∃ s . s ∈ b ∧ urrel-to-ωrel s = urrel-to-ωrel r)›
proof −
  define ασ-wit-intersection where
      ‹ασ-wit-intersection ≡ λ urrels .
        {ordext . ∀ urrel . urrel-to-ωrel urrel = ordext ⟶ urrel ∈ urrels}›
  define ασ-wit-union where
      ‹ασ-wit-union ≡ λ urrels .
        {ordext . ∃ urrel∈urrels . urrel-to-ωrel urrel = ordext}›

  let ?ασ-wit = ‹λ urrels .
    let ordexts = ασ-wit-intersection urrels in
    let ordexts′ = ασ-wit-union urrels in
    (ordexts, ordexts′, undefined)›
  define ασ-wit :: ‹urrel set ⇒ σ› where
    ‹ασ-wit ≡ λ urrels . Abs-σ (?ασ-wit urrels)›
  {
    fix a b :: ‹urrel set› and r s
```

```
    assume ‹ασ-wit a = ασ-wit b›
    hence 0: ‹{ordext. ∀ urrel. urrel-to-ωrel urrel = ordext ⟶ urrel ∈ a} =
              {ordext. ∀ urrel. urrel-to-ωrel urrel = ordext ⟶ urrel ∈ b}›
      unfolding ασ-wit-def Let-def
      apply (subst (asm) Abs-σ-inject)
      by (auto simp: ασ-wit-intersection-def ασ-wit-union-def)
    assume ‹urrel-to-ωrel s = urrel-to-ωrel r ⟹ s ∈ a› for s
    hence ‹urrel-to-ωrel r ∈
           {ordext. ∀ urrel. urrel-to-ωrel urrel = ordext ⟶ urrel ∈ a}›
      by auto
    hence ‹urrel-to-ωrel r ∈
           {ordext. ∀ urrel. urrel-to-ωrel urrel = ordext ⟶ urrel ∈ b}›
      using 0 by blast
    moreover assume ‹urrel-to-ωrel s = urrel-to-ωrel r›
    ultimately have ‹s ∈ b›
      by blast
  }
  moreover {
    fix a b :: ‹urrel set› and s r
    assume ‹ασ-wit a = ασ-wit b›
    hence 0: ‹{ordext. ∃ urrel ∈ a. urrel-to-ωrel urrel = ordext} =
              {ordext. ∃ urrel ∈ b. urrel-to-ωrel urrel = ordext}›
      unfolding ασ-wit-def
      using Abs-σ-inject ασ-wit-union-def by auto
    assume ‹s ∈ a›
    hence ‹urrel-to-ωrel s ∈ {ordext. ∃ urrel ∈ a. urrel-to-ωrel urrel = ordext}›
      by blast
    moreover assume ‹urrel-to-ωrel s = urrel-to-ωrel r›
    ultimately have ‹urrel-to-ωrel r ∈
                     {ordext. ∃ urrel ∈ b. urrel-to-ωrel urrel = ordext}›
      using 0 by argo
    hence ‹∃ s. s ∈ b ∧ urrel-to-ωrel s = urrel-to-ωrel r›
      by blast
  }
  ultimately show ?thesis
    by (safe intro!: exI[where x=ασ-wit]; metis)
qed
```

We enable the extended model version.

```
abbreviation (input) AOT-ExtendedModel where ‹AOT-ExtendedModel ≡ True›
```

Individual terms are either ordinary objects, represented by ordinary urelements, abstract objects, modelled as sets of urrelations, or null objects, used to represent non-denoting definite descriptions.

```
datatype κ = ωκ ω | ακ ‹urrel set› | is-nullκ: nullκ null
```

The mapping from abstract objects to urelements can be naturally lifted to a surjective mapping from individual terms to urelements.

```
primrec κυ :: ‹κ⇒υ› where
  ‹κυ (ωκ x) = ωυ x›
| ‹κυ (ακ x) = συ (ασ x)›
| ‹κυ (nullκ x) = nullυ x›

lemma κυ-surj: ‹surj κυ›
  using ασ-surj by (metis κυ.simps(1) κυ.simps(2) κυ.simps(3) υ.exhaust surj-def)
```

By construction if the urelement of an individual term is exemplified by an urrelation, it cannot be a null-object.

```
lemma urrel-null-false:
  assumes ‹AOT-model-valid-in w (Rep-urrel f (κυ x))›
  shows ‹¬is-nullκ x›
  by (metis (mono-tags, lifting) assms Rep-urrel κ.collapse(3) κυ.simps(3)
        mem-Collect-eq)
```

AOT requires any ordinary object to be *possibly concrete* and that there is an object that is not actually, but possibly concrete.

```
consts AOT-model-concreteω :: ‹ω ⇒ w ⇒  bool›
specification (AOT-model-concreteω)
  AOT-model-ω-concrete-in-some-world:
  ‹∃ w . AOT-model-concreteω x w›
  AOT-model-contingent-object:
  ‹∃ x w . AOT-model-concreteω x w ∧ ¬AOT-model-concreteω x w_0›
  by (rule exI[where x=‹λ- w. w ≠ w_0›]) (auto simp: AOT-model-nonactual-world)
```

We define a type class for AOT's terms specifying the conditions under which objects of that type denote and require the set of denoting terms to be non-empty.

```
class AOT-Term =
  fixes AOT-model-denotes :: ‹'a ⇒ bool›
  assumes AOT-model-denoting-ex: ‹∃ x . AOT-model-denotes x›
```

All types except the type of propositions involve non-denoting terms. We define a refined type class for those.

```
class AOT-IncompleteTerm = AOT-Term +
  assumes AOT-model-nondenoting-ex: ‹∃ x . ¬AOT-model-denotes x›
```

Generic non-denoting term.

```
definition AOT-model-nondenoting :: ‹'a::AOT-IncompleteTerm› where
  ‹AOT-model-nondenoting ≡ SOME τ . ¬AOT-model-denotes τ›
lemma AOT-model-nondenoing: ‹¬AOT-model-denotes (AOT-model-nondenoting)›
  using someI-ex[OF AOT-model-nondenoting-ex]
  unfolding AOT-model-nondenoting-def by blast
```

AOT-model-denotes can trivially be extended to products of types.

```
instantiation prod :: (AOT-Term, AOT-Term) AOT-Term
begin
definition AOT-model-denotes-prod :: ‹'a×'b ⇒ bool› where
  ‹AOT-model-denotes-prod ≡ λ(x,y) . AOT-model-denotes x ∧ AOT-model-denotes y›
instance proof
  show ‹∃x::'a×'b. AOT-model-denotes x›
    by (simp add: AOT-model-denotes-prod-def AOT-model-denoting-ex)
qed
end
```

We specify a transformation of proposition-valued functions on terms, s.t. the result is fully determined by *regular* terms. This will be required for modelling n-ary relations as functions on tuples while preserving AOT's definition of n-ary relation identity.

```
locale AOT-model-irregular-spec =
  fixes AOT-model-irregular :: ‹('a ⇒ o) ⇒ 'a ⇒ o›
    and AOT-model-regular :: ‹'a ⇒ bool›
    and AOT-model-term-equiv :: ‹'a ⇒ 'a ⇒ bool›
  assumes AOT-model-irregular-false:
    ‹¬AOT-model-valid-in w (AOT-model-irregular φ x)›
  assumes AOT-model-irregular-equiv:
    ‹AOT-model-term-equiv x y ⟹
     AOT-model-irregular φ x = AOT-model-irregular φ y›
  assumes AOT-model-irregular-eqI:
    ‹(⋀ x . AOT-model-regular x ⟹ φ x = ψ x) ⟹
     AOT-model-irregular φ x = AOT-model-irregular ψ x›
```

We introduce a type class for individual terms that specifies being regular, being equivalent (i.e. conceptually *sharing urelements*) and the transformation on proposition-valued functions as specified above.

```
class AOT-IndividualTerm = AOT-IncompleteTerm +
  fixes AOT-model-regular :: ‹'a ⇒ bool›
  fixes AOT-model-term-equiv :: ‹'a ⇒ 'a ⇒ bool›
  fixes AOT-model-irregular :: ‹('a ⇒ o) ⇒ 'a ⇒ o›
```

```
  assumes AOT-model-irregular-nondenoting:
    ‹¬AOT-model-regular x ⟹ ¬AOT-model-denotes x›
  assumes AOT-model-term-equiv-part-equivp:
    ‹equivp AOT-model-term-equiv›
  assumes AOT-model-term-equiv-denotes:
    ‹AOT-model-term-equiv x y ⟹ (AOT-model-denotes x = AOT-model-denotes y)›
  assumes AOT-model-term-equiv-regular:
    ‹AOT-model-term-equiv x y ⟹ (AOT-model-regular x = AOT-model-regular y)›
  assumes AOT-model-irregular:
    ‹AOT-model-irregular-spec AOT-model-irregular AOT-model-regular
                              AOT-model-term-equiv›

interpretation AOT-model-irregular-spec AOT-model-irregular AOT-model-regular
                                      AOT-model-term-equiv
  using AOT-model-irregular .
```

Our concrete type for individual terms satisfies the type class of individual terms. Note that all unary individuals are regular. In general, an individual term may be a tuple and is regular, if at most one tuple element does not denote.

```
instantiation κ :: AOT-IndividualTerm
begin
definition AOT-model-term-equiv-κ :: ‹κ ⇒ κ ⇒ bool› where
  ‹AOT-model-term-equiv-κ ≡ λ x y . κυ x = κυ y›
definition AOT-model-denotes-κ :: ‹κ ⇒ bool› where
  ‹AOT-model-denotes-κ ≡ λ x . ¬is-nullκ x›
definition AOT-model-regular-κ :: ‹κ ⇒ bool› where
  ‹AOT-model-regular-κ ≡ λ x . True›
definition AOT-model-irregular-κ :: ‹(κ ⇒ o) ⇒ κ ⇒ o› where
  ‹AOT-model-irregular-κ ≡ SOME φ . AOT-model-irregular-spec φ
                                    AOT-model-regular AOT-model-term-equiv›
instance proof
  show ‹∃ x :: κ. AOT-model-denotes x›
    by (rule exI[where x=‹ωκ undefined›])
      (simp add: AOT-model-denotes-κ-def)
next
  show ‹∃ x :: κ. ¬AOT-model-denotes x›
    by (rule exI[where x=‹nullκ undefined›])
      (simp add: AOT-model-denotes-κ-def AOT-model-regular-κ-def)
next
  show ¬AOT-model-regular x ⟹ ¬ AOT-model-denotes x for x :: κ
    by (simp add: AOT-model-regular-κ-def)
next
  show ‹equivp (AOT-model-term-equiv :: κ ⇒ κ ⇒ bool)›
    by (rule equivpI; rule reflpI exI sympI transpI)
      (simp-all add: AOT-model-term-equiv-κ-def)
next
  fix x y :: κ
  show ‹AOT-model-term-equiv x y ⟹ AOT-model-denotes x = AOT-model-denotes y›
    by (metis AOT-model-denotes-κ-def AOT-model-term-equiv-κ-def κ.exhaust-disc
          κυ.simps υ.disc(1,3,5,6) is-ακ-def is-ωκ-def is-nullκ-def)
next
  fix x y :: κ
  show ‹AOT-model-term-equiv x y ⟹ AOT-model-regular x = AOT-model-regular y›
    by (simp add: AOT-model-regular-κ-def)
next
  have AOT-model-irregular-spec (λ φ (x::κ) . εₒ w . False)
        AOT-model-regular AOT-model-term-equiv
    by standard (auto simp: AOT-model-proposition-choice-simp)
  thus ‹AOT-model-irregular-spec (AOT-model-irregular::(κ⇒o) ⇒ κ ⇒ o)
        AOT-model-regular AOT-model-term-equiv›
    unfolding AOT-model-irregular-κ-def by (metis (no-types, lifting) someI-ex)
qed
end
```

We define relations among individuals as proposition valued functions. *Denoting* unary relations (among κ) will match the urrelations introduced above.

```
typedef 'a rel (‹<->›) = ‹UNIV::('a::AOT-IndividualTerm ⇒ o) set› ..
setup-lifting type-definition-rel
```

We will use the transformation specified above to "fix" the behaviour of functions on irregular terms when defining λ-expressions.

```
definition fix-irregular :: ‹('a::AOT-IndividualTerm ⇒ o) ⇒ ('a ⇒ o)› where
  ‹fix-irregular ≡ λ φ x . if AOT-model-regular x
                            then φ x else AOT-model-irregular φ x›
lemma fix-irregular-denoting:
  ‹AOT-model-denotes x ⟹ fix-irregular φ x = φ x›
  by (meson AOT-model-irregular-nondenoting fix-irregular-def)
lemma fix-irregular-regular:
  ‹AOT-model-regular x ⟹ fix-irregular φ x = φ x›
  by (meson AOT-model-irregular-nondenoting fix-irregular-def)
lemma fix-irregular-irregular:
  ‹¬AOT-model-regular x ⟹ fix-irregular φ x = AOT-model-irregular φ x›
  by (simp add: fix-irregular-def)
```

Relations among individual terms are (potentially non-denoting) terms. A relation denotes, if it agrees on all equivalent terms (i.e. terms sharing urelements), is necessarily false on all non-denoting terms and is well-behaved on irregular terms.

```
instantiation rel :: (AOT-IndividualTerm) AOT-IncompleteTerm
begin

lift-definition AOT-model-denotes-rel :: ‹<'a> ⇒ bool› is
  ‹λ φ . (∀ x y . AOT-model-term-equiv x y ⟶ φ x = φ y) ∧
        (∀ w x . AOT-model-valid-in w (φ x) ⟶ AOT-model-denotes x) ∧
        (∀ x . ¬AOT-model-regular x ⟶ φ x = AOT-model-irregular φ x)› .
instance proof
  have ‹AOT-model-irregular (fix-irregular φ) x = AOT-model-irregular φ x›
    for φ and x :: 'a
    by (rule AOT-model-irregular-eqI) (simp add: fix-irregular-def)
  thus ‹∃ x :: <'a> . AOT-model-denotes x›
    by (safe intro!: exI[where x=‹Abs-rel (fix-irregular (λx. ε_o w . False))›])
      (transfer; auto simp: AOT-model-proposition-choice-simp fix-irregular-def
                          AOT-model-irregular-equiv AOT-model-term-equiv-regular
                          AOT-model-irregular-false)
next
  show ‹∃f :: <'a> . ¬AOT-model-denotes f›
    by (rule exI[where x=‹Abs-rel (λx. ε_o w . True)›];
        auto simp: AOT-model-denotes-rel.abs-eq AOT-model-nondenoting-ex
                   AOT-model-proposition-choice-simp)
qed
end
```

Auxiliary lemmata.

```
lemma AOT-model-term-equiv-eps:
  shows ‹AOT-model-term-equiv (Eps (AOT-model-term-equiv κ)) κ›
    and ‹AOT-model-term-equiv κ (Eps (AOT-model-term-equiv κ))›
    and ‹AOT-model-term-equiv κ κ' ⟹
         (Eps (AOT-model-term-equiv κ)) = (Eps (AOT-model-term-equiv κ'))›
  apply (metis AOT-model-term-equiv-part-equivp equivp-def someI-ex)
  apply (metis AOT-model-term-equiv-part-equivp equivp-def someI-ex)
  by (metis AOT-model-term-equiv-part-equivp equivp-def)

lemma AOT-model-denotes-Abs-rel-fix-irregularI:
  assumes ‹⋀ x y . AOT-model-term-equiv x y ⟹ φ x = φ y›
      and ‹⋀ w x . AOT-model-valid-in w (φ x) ⟹ AOT-model-denotes x›
    shows ‹AOT-model-denotes (Abs-rel (fix-irregular φ))›
proof -
```

```
  have ‹AOT-model-irregular φ x = AOT-model-irregular
          (λx. if AOT-model-regular x then φ x else AOT-model-irregular φ x) x›
    if ‹¬ AOT-model-regular x›
    for x
    by (rule AOT-model-irregular-eqI) auto
  thus ?thesis
  unfolding AOT-model-denotes-rel.rep-eq
  using assms by (auto simp: AOT-model-irregular-false Abs-rel-inverse
                             AOT-model-irregular-equiv fix-irregular-def
                             AOT-model-term-equiv-regular)
qed

lemma AOT-model-term-equiv-rel-equiv:
  assumes ‹AOT-model-denotes x›
      and ‹AOT-model-denotes y›
    shows ‹AOT-model-term-equiv x y = (∀ Π w . AOT-model-denotes Π ⟶
            AOT-model-valid-in w (Rep-rel Π x) = AOT-model-valid-in w (Rep-rel Π y))›
proof
  assume ‹AOT-model-term-equiv x y›
  thus ‹∀ Π w . AOT-model-denotes Π ⟶ AOT-model-valid-in w (Rep-rel Π x) =
                                        AOT-model-valid-in w (Rep-rel Π y)›
    by (simp add: AOT-model-denotes-rel.rep-eq)
next
  have 0: ‹(AOT-model-denotes x' ∧ AOT-model-term-equiv x' y) =
          (AOT-model-denotes y' ∧ AOT-model-term-equiv y' y)›
    if ‹AOT-model-term-equiv x' y'› for x' y'
    by (metis that AOT-model-term-equiv-denotes AOT-model-term-equiv-part-equivp
              equivp-def)
  assume ‹∀ Π w . AOT-model-denotes Π ⟶ AOT-model-valid-in w (Rep-rel Π x) =
                                          AOT-model-valid-in w (Rep-rel Π y)›
  moreover have ‹AOT-model-denotes (Abs-rel (fix-irregular
    (λ x . ε∘ w . AOT-model-denotes x ∧ AOT-model-term-equiv x y)))›
    (is AOT-model-denotes ?r)
    by (rule AOT-model-denotes-Abs-rel-fix-irregularI)
       (auto simp: 0 AOT-model-denotes-rel.rep-eq Abs-rel-inverse fix-irregular-def
                   AOT-model-proposition-choice-simp AOT-model-irregular-false)
  ultimately have ‹AOT-model-valid-in w (Rep-rel ?r x) =
                   AOT-model-valid-in w (Rep-rel ?r y)› for w
    by blast
  thus ‹AOT-model-term-equiv x y›
    by (simp add: Abs-rel-inverse AOT-model-proposition-choice-simp
                  fix-irregular-denoting[OF assms(1)] AOT-model-term-equiv-part-equivp
                  fix-irregular-denoting[OF assms(2)] assms equivp-reflp)
qed
```

Denoting relations among terms of type κ correspond to urrelations.

```
definition rel-to-urrel :: ‹<κ> ⇒ urrel› where
  ‹rel-to-urrel ≡ λ Π . Abs-urrel (λ u . Rep-rel Π (SOME x . κυ x = u))›
definition urrel-to-rel :: ‹urrel ⇒ <κ>› where
  ‹urrel-to-rel ≡ λ φ . Abs-rel (λ x . Rep-urrel φ (κυ x))›
definition AOT-rel-equiv :: ‹<'a::AOT-IndividualTerm> ⇒ <'a> ⇒ bool› where
  ‹AOT-rel-equiv ≡ λ f g . AOT-model-denotes f ∧ AOT-model-denotes g ∧ f = g›

lemma urrel-quotient3: ‹Quotient3 AOT-rel-equiv rel-to-urrel urrel-to-rel›
proof (rule Quotient3I)
  have ‹(λu. Rep-urrel a (κυ (SOME x. κυ x = u))) = (λu. Rep-urrel a u)› for a
    by (rule ext) (metis (mono-tags, lifting) κυ-surj surj-f-inv-f verit-sko-ex')
  thus ‹rel-to-urrel (urrel-to-rel a) = a› for a
    by (simp add: Abs-rel-inverse rel-to-urrel-def urrel-to-rel-def
                  Rep-urrel-inverse)
next
  show ‹AOT-rel-equiv (urrel-to-rel a) (urrel-to-rel a)› for a
    unfolding AOT-rel-equiv-def urrel-to-rel-def
```

```
    by transfer (simp add: AOT-model-regular-κ-def AOT-model-denotes-κ-def
                           AOT-model-term-equiv-κ-def urrel-null-false)
next
  {
    fix a
    assume ‹∀ w x. AOT-model-valid-in w (a x) ⟶ ¬ is-nullκ x›
    hence ‹(λu. a (SOME x. κυ x = u)) ∈
        {φ. ∀ x w. ¬ AOT-model-valid-in w (φ (nullυ x))}›
      by (simp; metis (mono-tags, lifting) κ.exhaust-disc κυ.simps υ.disc(1,3,5)
                                           υ.disc(6) is-ακ-def is-ωκ-def someI-ex)
  } note 1 = this
  {
    fix r s :: ‹κ ⇒ o›
    assume A: ‹∀ x y. AOT-model-term-equiv x y ⟶ r x = r y›
    assume ‹∀ w x. AOT-model-valid-in w (r x) ⟶ AOT-model-denotes x›
    hence 2: ‹(λu. r (SOME x. κυ x = u)) ∈
            {φ. ∀ x w. ¬ AOT-model-valid-in w (φ (nullυ x))}›
      using 1 AOT-model-denotes-κ-def by meson
    assume B: ‹∀ x y. AOT-model-term-equiv x y ⟶ s x = s y›
    assume ‹∀ w x. AOT-model-valid-in w (s x) ⟶ AOT-model-denotes x›
    hence 3: ‹(λu. s (SOME x. κυ x = u)) ∈
            {φ. ∀ x w. ¬ AOT-model-valid-in w (φ (nullυ x))}›
      using 1 AOT-model-denotes-κ-def by meson
    assume ‹Abs-urrel (λu. r (SOME x. κυ x = u)) =
          Abs-urrel (λu. s (SOME x. κυ x = u))›
    hence 4: ‹r (SOME x. κυ x = u) = s (SOME x::κ. κυ x = u)› for u
      unfolding Abs-urrel-inject[OF 2 3] by metis
    have ‹r x = s x› for x
      using 4[of ‹κυ x›]
      by (metis (mono-tags, lifting) A B AOT-model-term-equiv-κ-def someI-ex)
    hence ‹r = s› by auto
  }
  thus ‹AOT-rel-equiv r s = (AOT-rel-equiv r r ∧ AOT-rel-equiv s s ∧
                            rel-to-urrel r = rel-to-urrel s)› for r s
    unfolding AOT-rel-equiv-def rel-to-urrel-def
    by transfer auto
qed

lemma urrel-quotient:
  ‹Quotient AOT-rel-equiv rel-to-urrel urrel-to-rel
          (λx y. AOT-rel-equiv x x ∧ rel-to-urrel x = y)›
  using Quotient3-to-Quotient[OF urrel-quotient3] by auto
```

Unary individual terms are always regular and equipped with encoding and concreteness. The specification of the type class anticipates the required properties for deriving the axiom system.

```
class AOT-UnaryIndividualTerm =
  fixes AOT-model-enc :: ‹'a ⇒ <'a::AOT-IndividualTerm> ⇒ bool›
    and AOT-model-concrete :: ‹w ⇒ 'a ⇒ bool›
  assumes AOT-model-unary-regular:
      ‹AOT-model-regular x› — All unary individual terms are regular.
      and AOT-model-enc-relid:
        ‹AOT-model-denotes F ⟹
         AOT-model-denotes G ⟹
         (⋀ x . AOT-model-enc x F ⟷ AOT-model-enc x G)
         ⟹ F = G›
      and AOT-model-A-objects:
        ‹∃ x . AOT-model-denotes x ∧
              (∀ w. ¬ AOT-model-concrete w x) ∧
              (∀ F. AOT-model-denotes F ⟶ AOT-model-enc x F = φ F)›
      and AOT-model-contingent:
        ‹∃ x w. AOT-model-concrete w x ∧ ¬ AOT-model-concrete w0 x›
      and AOT-model-nocoder:
        ‹AOT-model-concrete w x ⟹ ¬AOT-model-enc x F›
```

and *AOT-model-concrete-equiv*:
‹*AOT-model-term-equiv x y* ⟹
AOT-model-concrete w x = *AOT-model-concrete w y*›
and *AOT-model-concrete-denotes*:
‹*AOT-model-concrete w x* ⟹ *AOT-model-denotes x*›
— The following are properties that will only hold in the extended models.
and *AOT-model-enc-indistinguishable-all*:
‹*AOT-ExtendedModel* ⟹
AOT-model-denotes a ⟹ ¬(∃ *w* . *AOT-model-concrete w a*) ⟹
AOT-model-denotes b ⟹ ¬(∃ *w* . *AOT-model-concrete w b*) ⟹
AOT-model-denotes Π ⟹
(⋀ Π′ . *AOT-model-denotes* Π′ ⟹
(⋀ *v* . *AOT-model-valid-in v* (*Rep-rel* Π′ *a*) =
AOT-model-valid-in v (*Rep-rel* Π′ *b*))) ⟹
(⋀ Π′ . *AOT-model-denotes* Π′ ⟹
(⋀ *v x* . ∃ *w* . *AOT-model-concrete w x* ⟹
AOT-model-valid-in v (*Rep-rel* Π′ *x*) =
AOT-model-valid-in v (*Rep-rel* Π *x*)) ⟹
AOT-model-enc a Π′) ⟹
(⋀ Π′ . *AOT-model-denotes* Π′ ⟹
(⋀ *v x* . ∃ *w* . *AOT-model-concrete w x* ⟹
AOT-model-valid-in v (*Rep-rel* Π′ *x*) =
AOT-model-valid-in v (*Rep-rel* Π *x*)) ⟹
AOT-model-enc b Π′)›
and *AOT-model-enc-indistinguishable-ex*:
‹*AOT-ExtendedModel* ⟹
AOT-model-denotes a ⟹ ¬(∃ *w* . *AOT-model-concrete w a*) ⟹
AOT-model-denotes b ⟹ ¬(∃ *w* . *AOT-model-concrete w b*) ⟹
AOT-model-denotes Π ⟹
(⋀ Π′ . *AOT-model-denotes* Π′ ⟹
(⋀ *v* . *AOT-model-valid-in v* (*Rep-rel* Π′ *a*) =
AOT-model-valid-in v (*Rep-rel* Π′ *b*))) ⟹
(∃ Π′ . *AOT-model-denotes* Π′ ∧ *AOT-model-enc a* Π′ ∧
(∀ *v x* . (∃ *w* . *AOT-model-concrete w x*) ⟶
AOT-model-valid-in v (*Rep-rel* Π′ *x*) =
AOT-model-valid-in v (*Rep-rel* Π *x*))) ⟹
(∃ Π′ . *AOT-model-denotes* Π′ ∧ *AOT-model-enc b* Π′ ∧
(∀ *v x* . (∃ *w* . *AOT-model-concrete w x*) ⟶
AOT-model-valid-in v (*Rep-rel* Π′ *x*) =
AOT-model-valid-in v (*Rep-rel* Π *x*)))›

Instantiate the class of unary individual terms for our concrete type of individual terms κ.

instantiation κ :: *AOT-UnaryIndividualTerm*
begin

definition *AOT-model-enc-κ* :: ‹κ ⇒ <κ> ⇒ *bool*› **where**
‹*AOT-model-enc-κ* ≡ λ *x F* .
case x of ακ a ⇒ *AOT-model-denotes F* ∧ *rel-to-urrel F* ∈ *a*
| - ⇒ *False*›
primrec *AOT-model-concrete-κ* :: ‹*w* ⇒ κ ⇒ *bool*› **where**
‹*AOT-model-concrete-κ w* (*ωκ x*) = *AOT-model-concreteω x w*›
| ‹*AOT-model-concrete-κ w* (*ακ x*) = *False*›
| ‹*AOT-model-concrete-κ w* (*nullκ x*) = *False*›

lemma *AOT-meta-A-objects-κ*:
‹∃ *x* :: κ. *AOT-model-denotes x* ∧
(∀ *w*. ¬ *AOT-model-concrete w x*) ∧
(∀ *F*. *AOT-model-denotes F* ⟶ *AOT-model-enc x F* = φ *F*)› **for** φ
apply (*rule exI*[**where** *x*=‹*ακ* {*f* . φ (*urrel-to-rel f*)}›])
apply (*simp add*: *AOT-model-enc-κ-def AOT-model-denotes-κ-def*)
by (*metis* (*no-types*, *lifting*) *AOT-rel-equiv-def urrel-quotient*
Quotient-rep-abs-fold-unmap)

```
instance proof
  show ‹AOT-model-regular x› for x :: κ
    by (simp add: AOT-model-regular-κ-def)
next
  fix F G :: ‹<κ>›
  assume ‹AOT-model-denotes F›
  moreover assume ‹AOT-model-denotes G›
  moreover assume ‹⋀x. AOT-model-enc x F = AOT-model-enc x G›
  moreover obtain x where ‹∀ G. AOT-model-denotes G ⟶ AOT-model-enc x G = (F = G)›
    using AOT-meta-A-objects-κ by blast
  ultimately show ‹F = G› by blast
next
  show ‹∃ x :: κ. AOT-model-denotes x ∧
                (∀ w. ¬ AOT-model-concrete w x) ∧
                (∀ F. AOT-model-denotes F ⟶ AOT-model-enc x F = φ F)› for φ
    using AOT-meta-A-objects-κ .
next
  show ‹∃ (x::κ) w. AOT-model-concrete w x ∧ ¬ AOT-model-concrete w_0 x›
    using AOT-model-concrete-κ.simps(1) AOT-model-contingent-object by blast
next
  show ‹AOT-model-concrete w x ⟹ ¬ AOT-model-enc x F› for w and x :: κ and F
    by (metis AOT-model-concrete-κ.simps(2) AOT-model-enc-κ-def κ.case-eq-if
          κ.collapse(2))
next
  show ‹AOT-model-concrete w x = AOT-model-concrete w y›
    if ‹AOT-model-term-equiv x y›
    for x y :: κ and w
    using that by (induct x; induct y; auto simp: AOT-model-term-equiv-κ-def)
next
  show ‹AOT-model-concrete w x ⟹ AOT-model-denotes x› for w and x :: κ
    by (metis AOT-model-concrete-κ.simps(3) AOT-model-denotes-κ-def κ.collapse(3))

next
  fix κ κ' :: κ and Π Π' :: ‹<κ>› and w :: w
  assume ext: ‹AOT-ExtendedModel›
  assume ‹AOT-model-denotes κ›
  moreover assume ‹∄ w. AOT-model-concrete w κ›
  ultimately obtain a where a-def: ‹ακ a = κ›
    by (metis AOT-model-ω-concrete-in-some-world AOT-model-concrete-κ.simps(1)
          AOT-model-denotes-κ-def κ.discI(3) κ.exhaust-sel)
  assume ‹AOT-model-denotes κ'›
  moreover assume ‹∄ w. AOT-model-concrete w κ'›
  ultimately obtain b where b-def: ‹ακ b = κ'›
    by (metis AOT-model-ω-concrete-in-some-world AOT-model-concrete-κ.simps(1)
          AOT-model-denotes-κ-def κ.discI(3) κ.exhaust-sel)
  assume ‹AOT-model-denotes Π' ⟹ AOT-model-valid-in w (Rep-rel Π' κ) =
                              AOT-model-valid-in w (Rep-rel Π' κ')› for Π' w
  hence ‹AOT-model-valid-in w (Rep-urrel r (κυ κ)) =
        AOT-model-valid-in w (Rep-urrel r (κυ κ'))› for r
    by (metis AOT-rel-equiv-def Abs-rel-inverse Quotient3-rel-rep
          iso-tuple-UNIV-I urrel-quotient3 urrel-to-rel-def)
  hence ‹let r = (Abs-urrel (λ u . ε∘ w . u = κυ κ)) in
        AOT-model-valid-in w (Rep-urrel r (κυ κ)) =
        AOT-model-valid-in w (Rep-urrel r (κυ κ'))›
    by presburger
  hence ασ-eq: ‹ασ a = ασ b›
    unfolding Let-def
    apply (subst (asm) (1 2) Abs-urrel-inverse)
    using AOT-model-proposition-choice-simp a-def b-def by force+
  assume Π-den: ‹AOT-model-denotes Π›
  have ‹¬AOT-model-valid-in w (Rep-rel Π (SOME xa. κυ xa = nullυ x))› for x w
    by (metis (mono-tags, lifting) AOT-model-denotes-κ-def
          AOT-model-denotes-rel.rep-eq κ.exhaust-disc κυ.simps(1,2,3))
```

‹*AOT-model-denotes* Π› *υ.disc*(*8*,*9*) *υ.distinct*(*3*)
is-ακ-def is-ωκ-def verit-sko-ex′)
moreover have ‹*Rep-rel* Π (*ωκ x*) = *Rep-rel* Π (*SOME y. κυ y* = *ωυ x*)› **for** *x*
by (*metis* (*mono-tags*, *lifting*) *AOT-model-denotes-rel.rep-eq*
AOT-model-term-equiv-κ-def κυ.simps(*1*) Π*-den verit-sko-ex′*)
ultimately have ‹*Rep-rel* Π (*ωκ x*) = *Rep-urrel* (*rel-to-urrel* Π) (*ωυ x*)› **for** *x*
unfolding *rel-to-urrel-def*
by (*subst Abs-urrel-inverse*) *auto*
hence ‹∃ *r* . ∀ *x* . *Rep-rel* Π (*ωκ x*) = *Rep-urrel r* (*ωυ x*)›
by (*auto intro*!: *exI*[**where** *x*=‹*rel-to-urrel* Π›])
then obtain *r* **where** *r-prop*: ‹*Rep-rel* Π (*ωκ x*) = *Rep-urrel r* (*ωυ x*)› **for** *x*
by *blast*
assume ‹*AOT-model-denotes* Π′ ⟹
(⋀*v x*. ∃ *w*. *AOT-model-concrete w x* ⟹
AOT-model-valid-in v (*Rep-rel* Π′ *x*) =
AOT-model-valid-in v (*Rep-rel* Π *x*)) ⟹ *AOT-model-enc κ* Π′› **for** Π′
hence ‹*AOT-model-denotes* Π′ ⟹
(⋀*v x*. *AOT-model-valid-in v* (*Rep-rel* Π′ (*ωκ x*)) =
AOT-model-valid-in v (*Rep-rel* Π (*ωκ x*))) ⟹ *AOT-model-enc κ* Π′› **for** Π′
by (*metis AOT-model-concrete-κ.simps*(*2*) *AOT-model-concrete-κ.simps*(*3*)
κ.exhaust-disc is-ακ-def is-ωκ-def is-nullκ-def)
hence ‹(⋀*v x*. *AOT-model-valid-in v* (*Rep-urrel r* (*ωυ x*)) =
AOT-model-valid-in v (*Rep-rel* Π (*ωκ x*))) ⟹ *r* ∈ *a*› **for** *r*
unfolding *a-def*[*symmetric*] *AOT-model-enc-κ-def* **apply** *simp*
by (*smt* (*verit*, *best*) *AOT-rel-equiv-def Abs-rel-inverse Quotient3-def*
κυ.simps(*1*) *iso-tuple-UNIV-I urrel-quotient3 urrel-to-rel-def*)
hence ‹(⋀*v x*. *AOT-model-valid-in v* (*Rep-urrel r′* (*ωυ x*)) =
AOT-model-valid-in v (*Rep-urrel r* (*ωυ x*))) ⟹ *r′* ∈ *a*› **for** *r′*
unfolding *r-prop***.**
hence ‹⋀*s*. *urrel-to-ωrel s* = *urrel-to-ωrel r* ⟹ *s* ∈ *a*›
by (*metis urrel-to-ωrel-def*)
hence *0*: ‹⋀*s*. *urrel-to-ωrel s* = *urrel-to-ωrel r* ⟹ *s* ∈ *b*›
using *ασ-eq-ord-exts-all ασ-eq ext ασ-ασ′* **by** *blast*

assume Π′*-den*: ‹*AOT-model-denotes* Π′›
assume ‹∃ *w*. *AOT-model-concrete w x* ⟹ *AOT-model-valid-in v* (*Rep-rel* Π′ *x*) =
AOT-model-valid-in v (*Rep-rel* Π *x*)› **for** *v x*
hence ‹*AOT-model-valid-in v* (*Rep-rel* Π′ (*ωκ x*)) =
AOT-model-valid-in v (*Rep-rel* Π (*ωκ x*))› **for** *v x*
using *AOT-model-ω-concrete-in-some-world AOT-model-concrete-κ.simps*(*1*)
by *presburger*
hence ‹*AOT-model-valid-in v* (*Rep-urrel* (*rel-to-urrel* Π′) (*ωυ x*)) =
AOT-model-valid-in v (*Rep-urrel r* (*ωυ x*))› **for** *v x*
by (*smt* (*verit*, *best*) *AOT-rel-equiv-def Abs-rel-inverse Quotient3-def*
κυ.simps(*1*) *iso-tuple-UNIV-I r-prop urrel-quotient3 urrel-to-rel-def* Π′*-den*)
hence ‹*urrel-to-ωrel* (*rel-to-urrel* Π′) = *urrel-to-ωrel r*›
by (*metis* (*full-types*) *AOT-urrel-ωequiv-def Quotient3-def urrel-ωrel-quot*)
hence ‹*rel-to-urrel* Π′ ∈ *b*› **using** *0* **by** *blast*
thus ‹*AOT-model-enc κ′* Π′›
unfolding *b-def*[*symmetric*] *AOT-model-enc-κ-def* **by** (*auto simp*: Π′*-den*)
next
fix *κ κ′* :: *κ* **and** Π Π′ :: ‹<κ>› **and** *w* :: *w*
assume *ext*: ‹*AOT-ExtendedModel*›
assume ‹*AOT-model-denotes κ*›
moreover assume ‹∄ *w*. *AOT-model-concrete w κ*›
ultimately obtain *a* **where** *a-def*: ‹*ακ a* = *κ*›
by (*metis AOT-model-ω-concrete-in-some-world AOT-model-concrete-κ.simps*(*1*)
AOT-model-denotes-κ-def κ.discI(*3*) *κ.exhaust-sel*)
assume ‹*AOT-model-denotes κ′*›
moreover assume ‹∄ *w*. *AOT-model-concrete w κ′*›
ultimately obtain *b* **where** *b-def*: ‹*ακ b* = *κ′*›
by (*metis AOT-model-ω-concrete-in-some-world AOT-model-concrete-κ.simps*(*1*)
AOT-model-denotes-κ-def κ.discI(*3*) *κ.exhaust-sel*)

```
assume ‹AOT-model-denotes Π′ ⟹ AOT-model-valid-in w (Rep-rel Π′ κ) =
                            AOT-model-valid-in w (Rep-rel Π′ κ′)› for Π′ w
hence ‹AOT-model-valid-in w (Rep-urrel r (κυ κ)) =
      AOT-model-valid-in w (Rep-urrel r (κυ κ′))› for r
  by (metis AOT-rel-equiv-def Abs-rel-inverse Quotient3-rel-rep
          iso-tuple-UNIV-I urrel-quotient3 urrel-to-rel-def)
hence ‹let r = (Abs-urrel (λ u . εₒ w . u = κυ κ)) in
      AOT-model-valid-in w (Rep-urrel r (κυ κ)) =
      AOT-model-valid-in w (Rep-urrel r (κυ κ′))›
  by presburger
hence ασ-eq: ‹ασ a = ασ b›
  unfolding Let-def
  apply (subst (asm) (1 2) Abs-urrel-inverse)
  using AOT-model-proposition-choice-simp a-def b-def by force+
assume Π-den: ‹AOT-model-denotes Π›
have ‹¬AOT-model-valid-in w (Rep-rel Π (SOME xa. κυ xa = nullυ x))› for x w
  by (metis (mono-tags, lifting) AOT-model-denotes-κ-def
       AOT-model-denotes-rel.rep-eq κ.exhaust-disc κυ.simps(1,2,3)
       ‹AOT-model-denotes Π› υ.disc(8) υ.disc(9) υ.distinct(3)
       is-ακ-def is-ωκ-def verit-sko-ex′)
moreover have ‹Rep-rel Π (ωκ x) = Rep-rel Π (SOME xa. κυ xa = ωυ x)› for x
  by (metis (mono-tags, lifting) AOT-model-denotes-rel.rep-eq
       AOT-model-term-equiv-κ-def κυ.simps(1) Π-den verit-sko-ex′)
ultimately have ‹Rep-rel Π (ωκ x) = Rep-urrel (rel-to-urrel Π) (ωυ x)› for x
  unfolding rel-to-urrel-def
  by (subst Abs-urrel-inverse) auto
hence ‹∃ r . ∀ x . Rep-rel Π (ωκ x) = Rep-urrel r (ωυ x)›
  by (auto intro!: exI[where x=‹rel-to-urrel Π›])
then obtain r where r-prop: ‹Rep-rel Π (ωκ x) = Rep-urrel r (ωυ x)› for x
  by blast

assume ‹∃ Π′. AOT-model-denotes Π′ ∧
  AOT-model-enc κ Π′ ∧
  (∀ v x. (∃ w. AOT-model-concrete w x) ⟶ AOT-model-valid-in v (Rep-rel Π′ x) =
                                     AOT-model-valid-in v (Rep-rel Π x))›
then obtain Π′ where
    Π′-den: ‹AOT-model-denotes Π′› and
    κ-enc-Π′: ‹AOT-model-enc κ Π′› and
    Π′-prop: ‹∃ w. AOT-model-concrete w x ⟹
              AOT-model-valid-in v (Rep-rel Π′ x) =
              AOT-model-valid-in v (Rep-rel Π x)› for v x
  by blast
have ‹AOT-model-valid-in v (Rep-rel Π′ (ωκ x)) =
     AOT-model-valid-in v (Rep-rel Π (ωκ x))› for x v
  by (simp add: AOT-model-ω-concrete-in-some-world Π′-prop)
hence 0: ‹AOT-urrel-ωequiv (rel-to-urrel Π′) (rel-to-urrel Π)›
  unfolding AOT-urrel-ωequiv-def
  by (smt (verit) AOT-rel-equiv-def Abs-rel-inverse Quotient3-def
              κυ.simps(1) iso-tuple-UNIV-I urrel-quotient3 urrel-to-rel-def
              Π-den Π′-den)
have ‹rel-to-urrel Π′ ∈ a›
 and ‹urrel-to-ωrel (rel-to-urrel Π′) = urrel-to-ωrel (rel-to-urrel Π)›
  apply (metis AOT-model-enc-κ-def κ.simps(11) κ-enc-Π′ a-def)
  by (metis Quotient3-rel 0 urrel-ωrel-quot)
hence ‹∃ s. s ∈ b ∧ urrel-to-ωrel s = urrel-to-ωrel (rel-to-urrel Π)›
  using ασ-eq-ord-exts-ex ασ-eq ext ασ-ασ′ by blast
then obtain s where
  s-prop: ‹s ∈ b ∧ urrel-to-ωrel s = urrel-to-ωrel (rel-to-urrel Π)›
  by blast
then obtain Π″ where
  Π″-prop: ‹rel-to-urrel Π″ = s› and Π″-den: ‹AOT-model-denotes Π″›
  by (metis AOT-rel-equiv-def Quotient3-def urrel-quotient3)
moreover have ‹AOT-model-enc κ′ Π″›
```

```
    by (metis AOT-model-enc-κ-def Π''-den Π''-prop κ.simps(11) b-def s-prop)
  moreover have ‹AOT-model-valid-in v (Rep-rel Π'' x) =
                 AOT-model-valid-in v (Rep-rel Π x)›
             if ‹∃ w. AOT-model-concrete w x› for v x
  proof(insert that)
    assume ‹∃ w. AOT-model-concrete w x›
    then obtain u where x-def: ‹x = ωκ u›
      by (metis AOT-model-concrete-κ.simps(2,3) κ.exhaust)
    show ‹AOT-model-valid-in v (Rep-rel Π'' x) =
          AOT-model-valid-in v (Rep-rel Π x)›
      unfolding x-def
      by (smt (verit, best) AOT-rel-equiv-def Abs-rel-inverse Quotient3-def
            Π''-den Π''-prop Π-den κv.simps(1) iso-tuple-UNIV-I s-prop
            urrel-quotient3 urrel-to-ωrel-def urrel-to-rel-def)
  qed
  ultimately show ‹∃ Π'. AOT-model-denotes Π' ∧ AOT-model-enc κ' Π' ∧
    (∀ v x. (∃ w. AOT-model-concrete w x) ⟶ AOT-model-valid-in v (Rep-rel Π' x) =
                                        AOT-model-valid-in v (Rep-rel Π x))›
    apply (safe intro!: exI[where x=Π''])
    by auto
qed
end
```

Products of unary individual terms and individual terms are individual terms. A tuple is regular, if at most one element does not denote. I.e. a pair is regular, if the first (unary) element denotes and the second is regular (i.e. at most one of its recursive tuple elements does not denote), or the first does not denote, but the second denotes (i.e. all its recursive tuple elements denote).

```
instantiation prod :: (AOT-UnaryIndividualTerm, AOT-IndividualTerm) AOT-IndividualTerm
begin
definition AOT-model-regular-prod :: ‹'a×'b ⇒ bool› where
  ‹AOT-model-regular-prod ≡ λ (x,y) . AOT-model-denotes x ∧ AOT-model-regular y ∨
                                  ¬AOT-model-denotes x ∧ AOT-model-denotes y›
definition AOT-model-term-equiv-prod :: ‹'a×'b ⇒ 'a×'b ⇒ bool› where
  ‹AOT-model-term-equiv-prod ≡  λ (x₁,y₁) (x₂,y₂) .
    AOT-model-term-equiv x₁ x₂ ∧ AOT-model-term-equiv y₁ y₂›
function AOT-model-irregular-prod :: ‹('a×'b ⇒ o) ⇒ 'a×'b ⇒ o› where
  AOT-model-irregular-proj2: ‹AOT-model-denotes x ⟹
    AOT-model-irregular φ (x,y) =
    AOT-model-irregular (λy. φ (SOME x' . AOT-model-term-equiv x x', y)) y›
| AOT-model-irregular-proj1: ‹¬AOT-model-denotes x ∧ AOT-model-denotes y ⟹
    AOT-model-irregular φ (x,y) =
    AOT-model-irregular (λx. φ (x, SOME y' . AOT-model-term-equiv y y')) x›
| AOT-model-irregular-prod-generic: ‹¬AOT-model-denotes x ∧ ¬AOT-model-denotes y ⟹
    AOT-model-irregular φ (x,y) =
    (SOME Φ . AOT-model-irregular-spec Φ AOT-model-regular AOT-model-term-equiv)
      φ (x,y)›
  by auto blast
termination using termination by blast

instance proof
  obtain x :: 'a and y :: 'b where
    ‹¬AOT-model-denotes x› and ‹¬AOT-model-denotes y›
    by (meson AOT-model-nondenoting-ex AOT-model-denoting-ex)
  thus ‹∃ x::'a×'b. ¬AOT-model-denotes x›
    by (auto simp: AOT-model-denotes-prod-def AOT-model-regular-prod-def)
next
  show ‹equivp (AOT-model-term-equiv :: 'a×'b ⇒ 'a×'b ⇒ bool)›
    by (rule equivpI; rule reflpI sympI transpI;
        simp add: AOT-model-term-equiv-prod-def AOT-model-term-equiv-part-equivp
                  equivp-reflp prod.case-eq-if case-prod-unfold equivp-symp)
       (metis equivp-transp[OF AOT-model-term-equiv-part-equivp])
next
  show ‹¬AOT-model-regular x ⟹ ¬ AOT-model-denotes x› for x :: ‹'a×'b›
```

```
    by (metis (mono-tags, lifting) AOT-model-denotes-prod-def case-prod-unfold
          AOT-model-irregular-nondenoting AOT-model-regular-prod-def)
next
  fix x y :: ‹'a×'b›
  show ‹AOT-model-term-equiv x y ⟹ AOT-model-denotes x = AOT-model-denotes y›
    by (metis (mono-tags, lifting) AOT-model-denotes-prod-def case-prod-beta
        AOT-model-term-equiv-denotes AOT-model-term-equiv-prod-def )
next
  fix x y :: ‹'a×'b›
  show ‹AOT-model-term-equiv x y ⟹ AOT-model-regular x = AOT-model-regular y›
    by (induct x; induct y;
        simp add: AOT-model-term-equiv-prod-def AOT-model-regular-prod-def)
       (meson AOT-model-term-equiv-denotes AOT-model-term-equiv-regular)
next
  interpret sp: AOT-model-irregular-spec ‹λφ (x::'a×'b) . εo w . False›
                  AOT-model-regular AOT-model-term-equiv
    by (simp add: AOT-model-irregular-spec-def AOT-model-proposition-choice-simp)
  have ex-spec: ‹∃ φ :: ('a×'b ⇒ o) ⇒ 'a×'b ⇒ o .
    AOT-model-irregular-spec φ AOT-model-regular AOT-model-term-equiv›
    using sp.AOT-model-irregular-spec-axioms by blast
  have some-spec: ‹AOT-model-irregular-spec
    (SOME φ :: ('a×'b ⇒ o) ⇒ 'a×'b ⇒ o .
        AOT-model-irregular-spec φ AOT-model-regular AOT-model-term-equiv)
    AOT-model-regular AOT-model-term-equiv›
    using someI-ex[OF ex-spec] by argo
  interpret sp-some: AOT-model-irregular-spec
    ‹SOME φ :: ('a×'b ⇒ o) ⇒ 'a×'b ⇒ o .
        AOT-model-irregular-spec φ AOT-model-regular AOT-model-term-equiv›
    AOT-model-regular AOT-model-term-equiv
    using some-spec by blast
  show ‹AOT-model-irregular-spec (AOT-model-irregular :: ('a×'b ⇒ o) ⇒ 'a×'b ⇒ o)
          AOT-model-regular AOT-model-term-equiv›
  proof
    have ‹¬AOT-model-valid-in w (AOT-model-irregular φ (a, b))›
      for w φ and a :: 'a and b :: 'b
      by (induct arbitrary: φ rule: AOT-model-irregular-prod.induct)
         (auto simp: AOT-model-irregular-false sp-some.AOT-model-irregular-false)
    thus ¬AOT-model-valid-in w (AOT-model-irregular φ x) for w φ and x :: ‹'a×'b›
      by (induct x)
  next
    {
      fix x1 y1 :: 'a and x2 y2 :: 'b and φ :: ‹'a×'b⇒o›
      assume x1y1-equiv: ‹AOT-model-term-equiv x1 y1›
      moreover assume x2y2-equiv: ‹AOT-model-term-equiv x2 y2›
      ultimately have xy-equiv: ‹AOT-model-term-equiv (x1,x2) (y1,y2)›
        by (simp add: AOT-model-term-equiv-prod-def)
      {
        assume ‹AOT-model-denotes x1›
        moreover hence ‹AOT-model-denotes y1›
          using AOT-model-term-equiv-denotes AOT-model-term-equiv-regular
                x1y1-equiv x2y2-equiv by blast
        ultimately have ‹AOT-model-irregular φ (x1,x2) =
                         AOT-model-irregular φ (y1,y2)›
          using AOT-model-irregular-equiv AOT-model-term-equiv-eps(3)
                x1y1-equiv x2y2-equiv by fastforce
      }
      moreover {
        assume ‹~AOT-model-denotes x1 ∧ AOT-model-denotes x2›
        moreover hence ‹~AOT-model-denotes y1 ∧ AOT-model-denotes y2›
          by (meson AOT-model-term-equiv-denotes x1y1-equiv x2y2-equiv)
        ultimately have ‹AOT-model-irregular φ (x1,x2) =
                         AOT-model-irregular φ (y1,y2)›
          using AOT-model-irregular-equiv AOT-model-term-equiv-eps(3)
```

```
        x1y1-equiv x2y2-equiv by fastforce
    }
    moreover {
      assume denotes-x: ‹(¬AOT-model-denotes x1 ∧ ¬AOT-model-denotes x2)›
      hence denotes-y: ‹(¬AOT-model-denotes y1 ∧ ¬AOT-model-denotes y2)›
        by (meson AOT-model-term-equiv-denotes AOT-model-term-equiv-regular
              x1y1-equiv x2y2-equiv)
      have eps-eq: ‹Eps (AOT-model-term-equiv x1) = Eps (AOT-model-term-equiv y1)›
        by (simp add: AOT-model-term-equiv-eps(3) x1y1-equiv)
      have ‹AOT-model-irregular φ (x1,x2) = AOT-model-irregular φ (y1,y2)›
        using denotes-x denotes-y
        using sp-some.AOT-model-irregular-equiv xy-equiv by auto
    }
    moreover {
      assume denotes-x: ‹¬AOT-model-denotes x1 ∧ AOT-model-denotes x2›
      hence denotes-y: ‹¬AOT-model-denotes y1 ∧ AOT-model-denotes y2›
        by (meson AOT-model-term-equiv-denotes x1y1-equiv x2y2-equiv)
      have eps-eq: ‹Eps (AOT-model-term-equiv x2) = Eps (AOT-model-term-equiv y2)›
        by (simp add: AOT-model-term-equiv-eps(3) x2y2-equiv)
      have ‹AOT-model-irregular φ (x1,x2) = AOT-model-irregular φ (y1,y2)›
        using denotes-x denotes-y
        using AOT-model-irregular-nondenoting calculation(2) by blast
    }
    ultimately have ‹AOT-model-irregular φ (x1,x2) = AOT-model-irregular φ (y1,y2)›
      using AOT-model-term-equiv-denotes AOT-model-term-equiv-regular
          sp-some.AOT-model-irregular-equiv x1y1-equiv x2y2-equiv xy-equiv
      by blast
  } note 0 = this
  show ‹AOT-model-term-equiv x y ⟹
      AOT-model-irregular φ x = AOT-model-irregular φ y›
    for x y :: ‹'a×'b› and φ
    by (induct x; induct y; simp add: AOT-model-term-equiv-prod-def 0)
next
  fix φ ψ :: ‹'a×'b ⇒ o›
  assume ‹AOT-model-regular x ⟹ φ x = ψ x› for x
  hence ‹φ (x, y) = ψ (x, y)›
    if ‹AOT-model-denotes x ∧ AOT-model-regular y ∨
        ¬AOT-model-denotes x ∧ AOT-model-denotes y› for x y
    using that unfolding AOT-model-regular-prod-def by simp
  hence ‹AOT-model-irregular φ (x,y) = AOT-model-irregular ψ (x,y)›
    for x :: 'a and y :: 'b
  proof (induct arbitrary: ψ φ rule: AOT-model-irregular-prod.induct)
    case (1 x y φ)
    thus ?case
      apply simp
      by (meson AOT-model-irregular-eqI AOT-model-irregular-nondenoting
              AOT-model-term-equiv-denotes AOT-model-term-equiv-eps(1))
  next
    case (2 x y φ)
    thus ?case
      apply simp
      by (meson AOT-model-irregular-nondenoting AOT-model-term-equiv-denotes
              AOT-model-term-equiv-eps(1))
  next
    case (3 x y φ)
    thus ?case
      apply simp
      by (metis (mono-tags, lifting) AOT-model-regular-prod-def case-prod-conv
                              sp-some.AOT-model-irregular-eqI surj-pair)
  qed
  thus ‹AOT-model-irregular φ x = AOT-model-irregular ψ x› for  x :: ‹'a×'b›
    by (metis surjective-pairing)
qed
```

qed
end

Introduction rules for term equivalence on tuple terms.

lemma *AOT-meta-prod-equivI*:
  shows ⋀ (*a*::′*a*::*AOT-UnaryIndividualTerm*) *x* (*y* :: ′*b*::*AOT-IndividualTerm*) .
      AOT-model-term-equiv x y ⟹ *AOT-model-term-equiv* (*a*,*x*) (*a*,*y*)
    and ⋀ (*x*::′*a*::*AOT-UnaryIndividualTerm*) *y* (*b* :: ′*b*::*AOT-IndividualTerm*) .
      AOT-model-term-equiv x y ⟹ *AOT-model-term-equiv* (*x*,*b*) (*y*,*b*)
    unfolding *AOT-model-term-equiv-prod-def*
    by (*simp add*: *AOT-model-term-equiv-part-equivp equivp-reflp*)+

The type of propositions are trivial instances of terms.

instantiation o :: *AOT-Term*
begin
definition *AOT-model-denotes*-o :: ‹o ⇒ *bool*› **where**
  ‹*AOT-model-denotes*-o ≡ λ-. *True*›
instance proof
  show ‹∃ *x*::o. *AOT-model-denotes x*›
    by (*simp add*: *AOT-model-denotes*-o-*def*)
qed
end

AOT's variables are modelled by restricting the type of terms to those terms that denote.

typedef ′*a AOT-var* = ‹{ *x* :: ′*a*::*AOT-Term* . *AOT-model-denotes x* }›
  morphisms *AOT-term-of-var AOT-var-of-term*
  by (*simp add*: *AOT-model-denoting-ex*)

Simplify automatically generated theorems and rules.

declare *AOT-var-of-term-induct*[*induct del*]
    AOT-var-of-term-cases[*cases del*]
    AOT-term-of-var-induct[*induct del*]
    AOT-term-of-var-cases[*cases del*]
lemmas *AOT-var-of-term-inverse* = *AOT-var-of-term-inverse*[*simplified*]
  and *AOT-var-of-term-inject* = *AOT-var-of-term-inject*[*simplified*]
  and *AOT-var-of-term-induct* =
      AOT-var-of-term-induct[*simplified*, *induct type*: *AOT-var*]
  and *AOT-var-of-term-cases* =
      AOT-var-of-term-cases[*simplified*, *cases type*: *AOT-var*]
  and *AOT-term-of-var* = *AOT-term-of-var*[*simplified*]
  and *AOT-term-of-var-cases* =
      AOT-term-of-var-cases[*simplified*, *induct pred*: *AOT-term-of-var*]
  and *AOT-term-of-var-induct* =
      AOT-term-of-var-induct[*simplified*, *induct pred*: *AOT-term-of-var*]
  and *AOT-term-of-var-inverse* = *AOT-term-of-var-inverse*[*simplified*]
  and *AOT-term-of-var-inject* = *AOT-term-of-var-inject*[*simplified*]

Equivalence by definition is modelled as necessary equivalence.

consts *AOT-model-equiv-def* :: ‹o ⇒ o ⇒ *bool*›
specification(*AOT-model-equiv-def*)
  AOT-model-equiv-def: ‹*AOT-model-equiv-def* φ ψ = (∀ *v* . *AOT-model-valid-in v* φ =
                AOT-model-valid-in v ψ)›
  by (*rule exI*[**where** *x*=‹λ φ ψ . ∀ *v* . *AOT-model-valid-in v* φ =
                AOT-model-valid-in v ψ›]) *simp*

Identity by definition is modelled as identity for denoting terms plus co-denoting.

consts *AOT-model-id-def* :: ‹(′*b* ⇒ ′*a*::*AOT-Term*) ⇒ (′*b* ⇒ ′*a*) ⇒ *bool*›
specification(*AOT-model-id-def*)
  AOT-model-id-def: ‹(*AOT-model-id-def* τ σ) = (∀ α . *if AOT-model-denotes* (σ α)
                then τ α = σ α
                else ¬*AOT-model-denotes* (τ α))›
  by (*rule exI*[**where** *x*=λ τ σ . ∀ α . *if AOT-model-denotes* (σ α)

then $\tau\ \alpha = \sigma\ \alpha$
else $\neg$*AOT-model-denotes* $(\tau\ \alpha)$])
blast

To reduce definitions by identity without free variables to definitions by identity with free variables acting on the unit type, we give the unit type a trivial instantiation to *AOT-Term*.

instantiation *unit* :: *AOT-Term*
begin
definition *AOT-model-denotes-unit* :: ‹*unit* ⇒ *bool*› **where**
‹*AOT-model-denotes-unit* ≡ λ-. *True*›
instance proof qed(*simp add*: *AOT-model-denotes-unit-def*)
end

Modally-strict and modally-fragile axioms are as necessary, resp. actually valid propositions.

definition *AOT-model-axiom* **where**
‹*AOT-model-axiom* ≡ λ φ . ∀ v . *AOT-model-valid-in* v φ›
definition *AOT-model-act-axiom* **where**
‹*AOT-model-act-axiom* ≡ λ φ . *AOT-model-valid-in* w_0 φ›

lemma *AOT-model-axiomI*:
assumes ‹⋀v . *AOT-model-valid-in* v φ›
shows ‹*AOT-model-axiom* φ›
unfolding *AOT-model-axiom-def* **using** *assms* **..**

lemma *AOT-model-act-axiomI*:
assumes ‹*AOT-model-valid-in* w_0 φ›
shows ‹*AOT-model-act-axiom* φ›
unfolding *AOT-model-act-axiom-def* **using** *assms* **.**

3 Outer Syntax Commands

nonterminal *AOT-prop*
nonterminal φ
nonterminal φ′
nonterminal τ
nonterminal τ′
nonterminal *AOT-axiom*
nonterminal *AOT-act-axiom*
ML-file *AOT-keys.ML*
ML-file *AOT-commands.ML*
setup‹*AOT-Theorems.setup*›
setup‹*AOT-Definitions.setup*›
setup‹*AOT-no-atp.setup*›

4 Approximation of the Syntax of PLM

locale *AOT-meta-syntax*
begin
notation *AOT-model-valid-in* (‹[- ⊨ -]›)
notation *AOT-model-axiom* (‹□[-]›)
notation *AOT-model-act-axiom* (‹𝓐[-]›)
end
locale *AOT-no-meta-syntax*
begin
no-notation *AOT-model-valid-in* (‹[- ⊨ -]›)
no-notation *AOT-model-axiom* (‹□[-]›)
no-notation *AOT-model-act-axiom* (‹𝓐[-]›)
end

```
consts AOT-denotes :: ‹'a::AOT-Term ⇒ o›
    AOT-imp :: ‹[o, o] ⇒ o›
    AOT-not :: ‹o ⇒ o›
    AOT-box :: ‹o ⇒ o›
    AOT-act :: ‹o ⇒ o›
    AOT-forall :: ‹('a::AOT-Term ⇒ o) ⇒ o›
    AOT-eq :: ‹'a::AOT-Term ⇒ 'a::AOT-Term ⇒ o›
    AOT-desc :: ‹('a::AOT-UnaryIndividualTerm ⇒ o) ⇒ 'a›
    AOT-exe :: ‹<'a::AOT-IndividualTerm> ⇒ 'a ⇒ o›
    AOT-lambda :: ‹('a::AOT-IndividualTerm ⇒ o) ⇒ <'a>›
    AOT-lambda0 :: ‹o ⇒ o›
    AOT-concrete :: ‹<'a::AOT-UnaryIndividualTerm> AOT-var›

nonterminal κs and Π and Π0 and α and exe-arg and exe-args
    and lambda-args and desc and free-var and free-vars
    and AOT-props and AOT-premises and AOT-world-relative-prop

syntax -AOT-process-frees :: ‹φ ⇒ φ'› (‹-›)
    -AOT-verbatim :: ‹any ⇒ φ› (‹«-»›)
    -AOT-verbatim :: ‹any ⇒ τ› (‹«-»›)
    -AOT-quoted :: ‹φ' ⇒ any› (‹«-»›)
    -AOT-quoted :: ‹τ' ⇒ any› (‹«-»›)
    :: ‹φ ⇒ φ› (‹'(-')›)
    -AOT-process-frees :: ‹τ ⇒ τ'› (‹-›)
    :: ‹κs ⇒ τ› (‹-›)
    :: ‹Π ⇒ τ› (‹-›)
    :: ‹φ ⇒ τ› (‹'(-')›)
    -AOT-term-var :: ‹id-position ⇒ τ› (‹-›)
    -AOT-term-var :: ‹id-position ⇒ φ› (‹-›)
    -AOT-exe-vars :: ‹id-position ⇒ exe-arg› (‹-›)
    -AOT-lambda-vars :: ‹id-position ⇒ lambda-args› (‹-›)
    -AOT-var :: ‹id-position ⇒ α› (‹-›)
    -AOT-vars :: ‹id-position ⇒ any›
    -AOT-verbatim :: ‹any ⇒ α› (‹«-»›)
    -AOT-valid :: ‹w ⇒ φ' ⇒ bool› (‹[- ⊨ -]›)
    -AOT-denotes :: ‹τ ⇒ φ› (‹-↓›)
    -AOT-imp :: ‹[φ, φ] ⇒ φ› (infixl ‹→› 25)
    -AOT-not :: ‹φ ⇒ φ› (‹~-› [50] 50)
    -AOT-not :: ‹φ ⇒ φ› (‹¬-› [50] 50)
    -AOT-box :: ‹φ ⇒ φ› (‹□-› [49] 54)
    -AOT-act :: ‹φ ⇒ φ› (‹𝓐-› [49] 54)
    -AOT-all :: ‹α ⇒ φ ⇒ φ› (‹∀ - -› [1,40])
syntax (input)
    -AOT-all-ellipse
        :: ‹id-position ⇒ id-position ⇒ φ ⇒ φ› (‹∀ -...∀ - -› [1,40])
syntax (output)
    -AOT-all-ellipse
        :: ‹id-position ⇒ id-position ⇒ φ ⇒ φ› (‹∀ -...∀ -'(-')› [1,40])
syntax
    -AOT-eq :: ‹[τ, τ] ⇒ φ› (infixl ‹=› 50)
    -AOT-desc :: ‹α ⇒ φ ⇒ desc› (‹ι--› [1,1000])
    :: ‹desc ⇒ κs› (‹-›)
    -AOT-lambda :: ‹lambda-args ⇒ φ ⇒ Π› (‹[λ- -]›)
    -explicitRelation :: ‹τ ⇒ Π› (‹[-]›)
    :: ‹κs ⇒ exe-arg› (‹-›)
    :: ‹exe-arg ⇒ exe-args› (‹-›)
    -AOT-exe-args :: ‹exe-arg ⇒ exe-args ⇒ exe-args› (‹--›)
    -AOT-exe-arg-ellipse :: ‹id-position ⇒ id-position ⇒ exe-arg› (‹-...-›)
    -AOT-lambda-arg-ellipse
        :: ‹id-position ⇒ id-position ⇒ lambda-args› (‹-...-›)
    -AOT-term-ellipse :: ‹id-position ⇒ id-position ⇒ τ› (‹-...-›)
    -AOT-exe :: ‹Π ⇒ exe-args ⇒ φ› (‹--›)
    -AOT-enc :: ‹exe-args ⇒ Π ⇒ φ› (‹--›)
```

-*AOT-lambda0* :: ‹φ ⇒ Π*0*› (‹[λ -]›)
:: ‹Π*0* ⇒ φ› (‹-›)
:: ‹Π*0* ⇒ τ› (‹-›)
-*AOT-concrete* :: ‹Π› (‹*E*!›)
:: ‹*any* ⇒ *exe-arg*› (‹«-»›)
:: ‹*desc* ⇒ *free-var*› (‹-›)
:: ‹Π ⇒ *free-var*› (‹-›)
-*AOT-appl* :: ‹*id-position* ⇒ *free-vars* ⇒ φ› (‹-'{-'}›)
-*AOT-appl* :: ‹*id-position* ⇒ *free-vars* ⇒ τ› (‹-'{-'}›)
-*AOT-appl* :: ‹*id-position* ⇒ *free-vars* ⇒ *free-vars*› (‹-'{-'}›)
-*AOT-appl* :: ‹*id-position* ⇒ *free-vars* ⇒ *free-vars*› (‹-'{-'}›)
-*AOT-term-var* :: ‹*id-position* ⇒ *free-var*› (‹-›)
:: ‹*any* ⇒ *free-var*› (‹«-»›)
:: ‹*free-var* ⇒ *free-vars*› (‹-›)
-*AOT-args* :: ‹*free-var* ⇒ *free-vars* ⇒ *free-vars*› (‹-,-›)
-*AOT-free-var-ellipse* :: ‹*id-position* ⇒ *id-position* ⇒ *free-var*› (‹-...-›)
syntax -*AOT-premises*
:: ‹*AOT-world-relative-prop* ⇒ *AOT-premises* ⇒ *AOT-premises*› (**infixr** ‹,› *3*)
-*AOT-world-relative-prop* :: φ ⇒ *AOT-world-relative-prop* (‹-›)
:: *AOT-world-relative-prop* ⇒ *AOT-premises* (‹-›)
-*AOT-prop* :: ‹*AOT-world-relative-prop* ⇒ *AOT-prop*› (‹-›)
:: ‹*AOT-prop* ⇒ *AOT-props*› (‹-›)
-*AOT-derivable* :: *AOT-premises* ⇒ φ' ⇒ *AOT-prop* (**infixl** ‹⊢› *2*)
-*AOT-nec-derivable* :: *AOT-premises* ⇒ φ' ⇒ *AOT-prop* (**infixl** ‹$\vdash_\Box$› *2*)
-*AOT-theorem* :: φ' ⇒ *AOT-prop* (‹⊢ -›)
-*AOT-nec-theorem* :: φ' ⇒ *AOT-prop* (‹$\vdash_\Box$ -›)
-*AOT-equiv-def* :: ‹φ ⇒ φ ⇒ *AOT-prop*› (**infixl** ‹$\equiv_{df}$› *3*)
-*AOT-axiom* :: φ' ⇒ *AOT-axiom* (‹-›)
-*AOT-act-axiom* :: φ' ⇒ *AOT-act-axiom* (‹-›)
-*AOT-axiom* :: φ' ⇒ *AOT-prop* (‹- ∈ $\Lambda_\Box$›)
-*AOT-act-axiom* :: φ' ⇒ *AOT-prop* (‹- ∈ Λ›)
-*AOT-id-def* :: ‹τ ⇒ τ ⇒ *AOT-prop*› (**infixl** ‹$=_{df}$› *3*)
-*AOT-for-arbitrary*
:: ‹*id-position* ⇒ *AOT-prop* ⇒ *AOT-prop*› (‹*for arbitrary* -: -› [*1000*,*1*] *1*)
syntax (**output**) -*lambda-args* :: ‹*any* ⇒ *patterns* ⇒ *patterns*› (‹--›)

translations
[*w* ⊨ φ] => *CONST AOT-model-valid-in w* φ

AOT-syntax-print-translations
[*w* ⊨ φ] <= *CONST AOT-model-valid-in w* φ

ML-file *AOT-syntax.ML*

AOT-register-type-constraints
Individual: ‹-::*AOT-UnaryIndividualTerm*› ‹-::*AOT-IndividualTerm*› **and**
Proposition: o **and**
Relation: ‹<-::*AOT-IndividualTerm*>› **and**
Term: ‹-::*AOT-Term*›

AOT-register-variable-names
Individual: *x y z* ν μ *a b c d* **and**
Proposition: *p q r s* **and**
Relation: *F G H P Q R S* **and**
Term: α β γ δ

AOT-register-metavariable-names
Individual: κ **and**
Proposition: φ ψ χ ϑ ζ ξ Θ **and**
Relation: Π **and**
Term: τ σ

AOT-register-premise-set-names Γ Δ Λ

```
parse-ast-translation‹[
  (syntax-const ‹-AOT-var›, K AOT-check-var),
  (syntax-const ‹-AOT-exe-vars›, K AOT-split-exe-vars),
  (syntax-const ‹-AOT-lambda-vars›, K AOT-split-lambda-args)
]›

translations
  -AOT-denotes τ => CONST AOT-denotes τ
  -AOT-imp φ ψ => CONST AOT-imp φ ψ
  -AOT-not φ => CONST AOT-not φ
  -AOT-box φ => CONST AOT-box φ
  -AOT-act φ => CONST AOT-act φ
  -AOT-eq τ τ' => CONST AOT-eq τ τ'
  -AOT-lambda0 φ => CONST AOT-lambda0 φ
  -AOT-concrete => CONST AOT-term-of-var (CONST AOT-concrete)
  -AOT-lambda α φ => CONST AOT-lambda (-abs α φ)
  -explicitRelation Π => Π

AOT-syntax-print-translations
  -AOT-lambda (-lambda-args x y) φ <= CONST AOT-lambda (-abs (-pattern x y) φ)
  -AOT-lambda (-lambda-args x y) φ <= CONST AOT-lambda (-abs (-patterns x y) φ)
  -AOT-lambda x φ <= CONST AOT-lambda (-abs x φ)
  -lambda-args x (-lambda-args y z) <= -lambda-args x (-patterns y z)
  -lambda-args (x y z) <= -lambda-args (-tuple x (-tuple-arg (-tuple y z)))


AOT-syntax-print-translations
  -AOT-imp φ ψ <= CONST AOT-imp φ ψ
  -AOT-not φ <= CONST AOT-not φ
  -AOT-box φ <= CONST AOT-box φ
  -AOT-act φ <= CONST AOT-act φ
  -AOT-all α φ <= CONST AOT-forall (-abs α φ)
  -AOT-all α φ <= CONST AOT-forall (λα. φ)
  -AOT-eq τ τ' <= CONST AOT-eq τ τ'
  -AOT-desc x φ <= CONST AOT-desc (-abs x φ)
  -AOT-desc x φ <= CONST AOT-desc (λx. φ)
  -AOT-lambda0 φ <= CONST AOT-lambda0 φ
  -AOT-concrete <= CONST AOT-term-of-var (CONST AOT-concrete)

translations
  -AOT-appl φ (-AOT-args a b) => -AOT-appl (φ a) b
  -AOT-appl φ a => φ a


parse-translation‹
[
  (syntax-const ‹-AOT-var›, parseVar true),
  (syntax-const ‹-AOT-vars›, parseVar false),
  (syntax-const ‹-AOT-valid›, fn ctxt => fn [w,x] =>
    const ‹AOT-model-valid-in› $ w $ x),
  (syntax-const ‹-AOT-quoted›, fn ctxt => fn [x] => x),
  (syntax-const ‹-AOT-process-frees›, fn ctxt => fn [x] => processFrees ctxt x),
  (syntax-const ‹-AOT-world-relative-prop›, fn ctxt => fn [x] => let
    val (x, premises) = processFreesAndPremises ctxt x
    val (world::formulas) = Variable.variant-names (Variable.declare-names x ctxt)
        ((v, dummyT)::(map (fn - => (φ, dummyT)) premises))
    val term = HOLogic.mk-Trueprop
        (@{const AOT-model-valid-in} $ Free world $ processFrees ctxt x)
    val term = fold (fn (premise,form) => fn trm =>
         @{const Pure.imp} $
        HOLogic.mk-Trueprop
          (Const (const-name ‹Set.member›, dummyT) $ Free form $ premise) $
```

```
      (Term.absfree (Term.dest-Free (dropConstraints premise)) trm $ Free form)
   ) (ListPair.zipEq (premises,formulas)) term
   val term = fold (fn (form) => fn trm =>
        Const (const-name ‹Pure.all›, dummyT) $
       (Term.absfree form trm)
   ) formulas term
   val term = Term.absfree world term
   in term end),
  (syntax-const ‹-AOT-prop›, fn ctxt => fn [x] => let
   val world = case (AOT-ProofData.get ctxt) of SOME w => w
       | - => raise Fail Expected world to be stored in the proof state.
   in x $ world end),
  (syntax-const ‹-AOT-theorem›, fn ctxt => fn [x] =>
     HOLogic.mk-Trueprop (@{const AOT-model-valid-in} $ @{const w₀} $ x)),
  (syntax-const ‹-AOT-axiom›, fn ctxt => fn [x] =>
     HOLogic.mk-Trueprop (@{const AOT-model-axiom} $ x)),
  (syntax-const ‹-AOT-act-axiom›, fn ctxt => fn [x] =>
     HOLogic.mk-Trueprop (@{const AOT-model-act-axiom} $ x)),
  (syntax-const ‹-AOT-nec-theorem›, fn ctxt => fn [trm] => let
   val world = singleton (Variable.variant-names (Variable.declare-names trm ctxt)) (v, @{typ w})
   val trm = HOLogic.mk-Trueprop (@{const AOT-model-valid-in} $ Free world $ trm)
   val trm = Term.absfree world trm
   val trm = Const (const-name ‹Pure.all›, dummyT) $ trm
   in trm end),
  (syntax-const ‹-AOT-derivable›, fn ctxt => fn [x,y] => let
   val world = case (AOT-ProofData.get ctxt) of SOME w => w
     | - => raise Fail Expected world to be stored in the proof state.
   in foldPremises world x y end),
  (syntax-const ‹-AOT-nec-derivable›, fn ctxt => fn [x,y] => let
   in Const (const-name ‹Pure.all›, dummyT) $
      Abs (v, dummyT, foldPremises (Bound 0) x y) end),
  (syntax-const ‹-AOT-for-arbitrary›, fn ctxt => fn [- $ var $ pos,trm] => let
   val trm = Const (const-name ‹Pure.all›, dummyT) $
       (Const (-constrainAbs, dummyT) $ Term.absfree (Term.dest-Free var) trm $ pos)
   in trm end),
  (syntax-const ‹-AOT-equiv-def›, parseEquivDef),
  (syntax-const ‹-AOT-exe›, parseExe),
  (syntax-const ‹-AOT-enc›, parseEnc)
]
›

parse-ast-translation‹
[
  (syntax-const ‹-AOT-exe-arg-ellipse›, parseEllipseList -AOT-term-vars),
  (syntax-const ‹-AOT-lambda-arg-ellipse›, parseEllipseList -AOT-vars),
  (syntax-const ‹-AOT-free-var-ellipse›, parseEllipseList -AOT-term-vars),
  (syntax-const ‹-AOT-term-ellipse›, parseEllipseList -AOT-term-vars),
  (syntax-const ‹-AOT-all-ellipse›, fn ctx => fn [a,b,c] =>
     Ast.mk-appl (Ast.Constant const-name ‹AOT-forall›) [
       Ast.mk-appl (Ast.Constant -abs) [parseEllipseList -AOT-vars ctx [a,b],c]
     ])
]
›

syntax (output)
  -AOT-individual-term :: ‹'a ⇒ tuple-args› (‹-›)
  -AOT-individual-terms :: ‹tuple-args ⇒ tuple-args ⇒ tuple-args› (‹--›)
  -AOT-relation-term :: ‹'a ⇒ Π›
  -AOT-any-term :: ‹'a ⇒ τ›


print-ast-translation‹AOT-syntax-print-ast-translations[
 (syntax-const ‹-AOT-individual-term›, AOT-print-individual-term),
```

(*syntax-const* ‹-*AOT-relation-term*›, *AOT-print-relation-term*),
(*syntax-const* ‹-*AOT-any-term*›, *AOT-print-generic-term*)
]›

AOT-syntax-print-translations
-*AOT-individual-terms* (-*AOT-individual-term* *x*) (-*AOT-individual-terms* (-*tuple* *y* *z*))
<= -*AOT-individual-terms* (-*tuple* *x* (-*tuple-args* *y* *z*))
-*AOT-individual-terms* (-*AOT-individual-term* *x*) (-*AOT-individual-term* *y*)
<= -*AOT-individual-terms* (-*tuple* *x* (-*tuple-arg* *y*))
-*AOT-individual-terms* (-*tuple* *x* *y*) <= -*AOT-individual-term* (-*tuple* *x* *y*)
-*AOT-exe* (-*AOT-relation-term* Π) (-*AOT-individual-term* κ) <= *CONST* *AOT-exe* Π κ
-*AOT-denotes* (-*AOT-any-term* κ) <= *CONST* *AOT-denotes* κ

AOT-define *AOT-conj* :: ‹[φ, φ] ⇒ φ› (**infixl** ‹&› *35*) ‹φ & ψ $\equiv_{df}$ ¬(φ → ¬ψ)›
declare *AOT-conj*[*AOT del*, *AOT-defs del*]
AOT-define *AOT-disj* :: ‹[φ, φ] ⇒ φ› (**infixl** ‹∨› *35*) ‹φ ∨ ψ $\equiv_{df}$ ¬φ → ψ›
declare *AOT-disj*[*AOT del*, *AOT-defs del*]
AOT-define *AOT-equiv* :: ‹[φ, φ] ⇒ φ› (**infix** ‹≡› *20*) ‹φ ≡ ψ $\equiv_{df}$ (φ → ψ) & (ψ → φ)›
declare *AOT-equiv*[*AOT del*, *AOT-defs del*]
AOT-define *AOT-dia* :: ‹φ ⇒ φ› (‹◇-› [*49*] *54*) ‹◇φ $\equiv_{df}$ ¬□¬φ›
declare *AOT-dia*[*AOT del*, *AOT-defs del*]

context *AOT-meta-syntax*
begin
notation *AOT-dia* (‹◇-› [*49*] *54*)
notation *AOT-conj* (**infixl** ‹**&**› *35*)
notation *AOT-disj* (**infixl** ‹∨› *35*)
notation *AOT-equiv* (**infixl** ‹≡› *20*)
end
context *AOT-no-meta-syntax*
begin
no-notation *AOT-dia* (‹◇-› [*49*] *54*)
no-notation *AOT-conj* (**infixl** ‹**&**› *35*)
no-notation *AOT-disj* (**infixl** ‹∨› *35*)
no-notation *AOT-equiv* (**infixl** ‹≡› *20*)
end

print-translation ‹
AOT-syntax-print-translations
[
AOT-preserve-binder-abs-tr′
const-syntax ‹*AOT-forall*›
syntax-const ‹-*AOT-all*›
(*syntax-const* ‹-*AOT-all-ellipse*›, *true*)
const-name ‹*AOT-imp*›,
AOT-binder-trans @{*theory*} @{*binding AOT-forall-binder*} *syntax-const* ‹-*AOT-all*›,
(*const-syntax* ‹*AOT-desc*›, *fn ctxt* => *Syntax-Trans.preserve-binder-abs-tr′* *syntax-const* ‹-*AOT-desc*› *ctxt* *dummyT*),
AOT-binder-trans @{*theory*} @{*binding AOT-desc-binder*} *syntax-const* ‹-*AOT-desc*›,
AOT-preserve-binder-abs-tr′
const-syntax ‹*AOT-lambda*›
syntax-const ‹-*AOT-lambda*›
(*syntax-const* ‹-*AOT-lambda-arg-ellipse*›, *false*)
const-name ‹*undefined*›,
AOT-binder-trans
@{*theory*}
@{*binding AOT-lambda-binder*}
syntax-const ‹-*AOT-lambda*›
]
›

parse-translation‹

[(*syntax-const* ‹-AOT-id-def›, *parseIdDef*)]
›

parse-ast-translation‹[
(*syntax-const* ‹-AOT-all›,
AOT-restricted-binder *const-name* ‹*AOT-forall*› *const-name* ‹*AOT-imp*›),
(*syntax-const* ‹-AOT-desc›,
AOT-restricted-binder *const-name* ‹*AOT-desc*› *const-name* ‹*AOT-conj*›)
]›

AOT-define *AOT-exists* :: ‹α ⇒ φ ⇒ φ› ‹«*AOT-exists* φ» $\equiv_{df}$ ¬∀ α ¬φ{α}›
declare *AOT-exists*[*AOT del*, *AOT-defs del*]
syntax *-AOT-exists* :: ‹α ⇒ φ ⇒ φ› (‹∃ - -› [*1*,*40*])

AOT-syntax-print-translations
-AOT-exists α φ <= *CONST AOT-exists* (*-abs* α φ)
-AOT-exists α φ <= *CONST AOT-exists* (λα. φ)

parse-ast-translation‹
[(*syntax-const* ‹-AOT-exists›,
AOT-restricted-binder *const-name* ‹*AOT-exists*› *const-name* ‹*AOT-conj*›)]
›

context *AOT-meta-syntax*
begin
notation *AOT-exists* (**binder** ‹∃ › *8*)
end
context *AOT-no-meta-syntax*
begin
no-notation *AOT-exists* (**binder** ‹∃ › *8*)
end

syntax (*input*)
-AOT-exists-ellipse :: ‹*id-position* ⇒ *id-position* ⇒ φ ⇒ φ› (‹∃ -...∃ - -› [*1*,*40*])
syntax (**output**)
-AOT-exists-ellipse :: ‹*id-position* ⇒ *id-position* ⇒ φ ⇒ φ› (‹∃ -...∃ - ′(-′)› [*1*,*40*])
parse-ast-translation‹[(*syntax-const* ‹-AOT-exists-ellipse›, *fn ctx* => *fn* [*a*,*b*,*c*] =>
Ast.mk-appl (*Ast.Constant AOT-exists*)
[*Ast.mk-appl* (*Ast.Constant -abs*) [*parseEllipseList -AOT-vars ctx* [*a*,*b*],*c*]])]›
print-translation‹*AOT-syntax-print-translations* [
AOT-preserve-binder-abs-tr′
const-syntax ‹*AOT-exists*›
syntax-const ‹-AOT-exists›
(*syntax-const* ‹-AOT-exists-ellipse›,*true*) *const-name* ‹*AOT-conj*›,
AOT-binder-trans
@{*theory*}
@{*binding AOT-exists-binder*}
syntax-const ‹-AOT-exists›
]›

syntax *-AOT-DDDOT* :: φ (‹...›)
syntax *-AOT-DDDOT* :: φ (‹. . .›)
parse-translation‹[(*syntax-const* ‹-AOT-DDDOT›, *parseDDOT*)]›

print-translation‹*AOT-syntax-print-translations*
[(*const-syntax* ‹*Pure.all*›, *fn ctxt* => *fn* [*Abs* (-, -,
Const (*const-syntax* ‹*HOL.Trueprop*›, -) $
(*Const* (*const-syntax* ‹*AOT-model-valid-in*›, -) $ *Bound 0* $ *y*))] => *let*
val y = (*Const* (*syntax-const* ‹-AOT-process-frees›, *dummyT*) $ *y*)
in (*Const* (*syntax-const* ‹-AOT-nec-theorem›, *dummyT*) $ *y*) *end*

```
| [p as Abs (name, -,
  Const (const-syntax ‹HOL.Trueprop›, -) $
  (Const (const-syntax ‹AOT-model-valid-in›, -) $ w $ y))]
=> (Const (syntax-const ‹-AOT-for-arbitrary›, dummyT) $
    (Const (-bound, dummyT) $ Free (name, dummyT)) $
    (Term.betapply (p, (Const (-bound, dummyT) $ Free (name, dummyT)))))
),

 (const-syntax ‹AOT-model-valid-in›, fn ctxt =>
  fn [w as (Const (-free, -) $ Free (v, -)), y] => let
    val is-world = (case (AOT-ProofData.get ctxt)
        of SOME (Free (w, -)) => Name.clean w = Name.clean v | - => false)
    val y = (Const (syntax-const ‹-AOT-process-frees›, dummyT) $ y)
    in if is-world then y else Const (syntax-const ‹-AOT-valid›, dummyT) $ w $ y end
  | [Const (const-syntax ‹w0›, -), y] => let
    val y = (Const (syntax-const ‹-AOT-process-frees›, dummyT) $ y)
    in case (AOT-ProofData.get ctxt) of SOME (Const (const-name ‹w0›, -)) => y |
          - => Const (syntax-const ‹-AOT-theorem›, dummyT) $ y end
  | [Const (-var, -) $ -, y] => let
    val y = (Const (syntax-const ‹-AOT-process-frees›, dummyT) $ y)
    in Const (syntax-const ‹-AOT-nec-theorem›, dummyT) $ y end
  ),
 (const-syntax ‹AOT-model-axiom›, fn ctxt => fn [trm] =>
    Const (syntax-const ‹-AOT-axiom›, dummyT) $
    (Const (syntax-const ‹-AOT-process-frees›, dummyT) $ trm)),
 (const-syntax ‹AOT-model-act-axiom›, fn ctxt => fn [trm] =>
    Const (syntax-const ‹-AOT-axiom›, dummyT) $
    (Const (syntax-const ‹-AOT-process-frees›, dummyT) $ trm)),
(syntax-const ‹-AOT-process-frees›, fn - =>  fn [t] => let
  fun mapAppls (x as Const (-free, -) $
                    Free (-, Type (-ignore-type, [Type (fun, -)])))
        = (Const (-AOT-raw-appl, dummyT) $ x)
    | mapAppls (x as Const (-free, -) $ Free (-, Type (fun, -)))
        = (Const (-AOT-raw-appl, dummyT) $ x)
    | mapAppls (x as Const (-var, -) $
                    Var (-, Type (-ignore-type, [Type (fun, -)])))
        = (Const (-AOT-raw-appl, dummyT) $ x)
    | mapAppls (x as Const (-var, -) $ Var (-, Type (fun, -)))
        = (Const (-AOT-raw-appl, dummyT) $ x)
    | mapAppls (x $ y) = mapAppls x $ mapAppls y
    | mapAppls (Abs (x,y,z)) = Abs (x,y, mapAppls z)
    | mapAppls x = x
  in mapAppls t end
)
]
›

print-ast-translation‹AOT-syntax-print-ast-translations
let
fun handleTermOfVar x kind name = (
let
val - = case kind of -free => () | -var => () | -bound => () | - => raise Match
in
  case printVarKind name
    of (SingleVariable name) => Ast.Appl [Ast.Constant kind, Ast.Variable name]
    | (Ellipses (s, e)) =>  Ast.Appl [Ast.Constant -AOT-free-var-ellipse,
    Ast.Appl [Ast.Constant kind, Ast.Variable s],
    Ast.Appl [Ast.Constant kind, Ast.Variable e]
      ]
  | Verbatim name => Ast.mk-appl (Ast.Constant -AOT-quoted)
                       [Ast.mk-appl (Ast.Constant -AOT-term-of-var) [x]]
end
)
```

```
fun termOfVar ctxt (Ast.Appl [Ast.Constant -constrain,
      x as Ast.Appl [Ast.Constant kind, Ast.Variable name], -]) = termOfVar ctxt x
  | termOfVar ctxt (x as Ast.Appl [Ast.Constant kind, Ast.Variable name])
      = handleTermOfVar x kind name
  | termOfVar ctxt (x as Ast.Appl [Ast.Constant rep, y]) = (
let
val (restr,-) = Local-Theory.raw-theory-result (fn thy => (
let
val restrs = Symtab.dest (AOT-Restriction.get thy)
val restr = List.find (fn (n,(-,Const (c,t))) => (
  c = rep orelse c = Lexicon.unmark-const rep) | - => false) restrs
in
(restr,thy)
end
)) ctxt
in
  case restr of SOME r => Ast.Appl [Ast.Constant (const-syntax ‹AOT-term-of-var›), y]
  | - => raise Match
end)

in
[(const-syntax ‹AOT-term-of-var›, fn ctxt => fn [x] => termOfVar ctxt x),
(-AOT-raw-appl, fn ctxt => fn t::a::args => let
fun applyTermOfVar (t as Ast.Appl (Ast.Constant const-syntax ‹AOT-term-of-var›::[x]))
    = (case try (termOfVar ctxt) x of SOME y => y | - => t)
  | applyTermOfVar y = (case try (termOfVar ctxt) y of SOME x => x | - => y)
val ts = fold (fn a => fn b => Ast.mk-appl (Ast.Constant syntax-const ‹-AOT-args›)
             [b,applyTermOfVar a]) args (applyTermOfVar a)
in Ast.mk-appl (Ast.Constant syntax-const ‹-AOT-appl›) [t,ts] end)]
end
›
```

context *AOT-meta-syntax*
begin
notation *AOT-denotes* (‹-↓›)
notation *AOT-imp* (**infixl** ‹→› *25*)
notation *AOT-not* (‹¬-› [*50*] *50*)
notation *AOT-box* (‹□-› [*49*] *54*)
notation *AOT-act* (‹𝓐-› [*49*] *54*)
notation *AOT-forall* (**binder** ‹∀ › *8*)
notation *AOT-eq* (**infixl** ‹=› *50*)
notation *AOT-desc* (**binder** ‹ι› *100*)
notation *AOT-lambda* (**binder** ‹λ› *100*)
notation *AOT-lambda0* (‹[λ -]›)
notation *AOT-exe* (‹⦇-,-⦈›)
notation *AOT-model-equiv-def* (**infixl** ‹$\equiv_{df}$› *10*)
notation *AOT-model-id-def* (**infixl** ‹$=_{df}$› *10*)
notation *AOT-term-of-var* (‹⟨-⟩›)
notation *AOT-concrete* (‹**E!**›)
end
context *AOT-no-meta-syntax*
begin
no-notation *AOT-denotes* (‹-↓›)
no-notation *AOT-imp* (**infixl** ‹→› *25*)
no-notation *AOT-not* (‹¬-› [*50*] *50*)
no-notation *AOT-box* (‹□-› [*49*] *54*)
no-notation *AOT-act* (‹𝓐-› [*49*] *54*)
no-notation *AOT-forall* (**binder** ‹∀ › *8*)
no-notation *AOT-eq* (**infixl** ‹=› *50*)
no-notation *AOT-desc* (**binder** ‹ι› *100*)
no-notation *AOT-lambda* (**binder** ‹λ› *100*)
no-notation *AOT-lambda0* (‹[λ -]›)
no-notation *AOT-exe* (‹⦇-,-⦈›)

```
no-notation AOT-model-equiv-def (infixl ‹≡df› 10)
no-notation AOT-model-id-def (infixl ‹=df› 10)
no-notation AOT-term-of-var (‹⟨-⟩›)
no-notation AOT-concrete (‹E!›)
end

bundle AOT-syntax
begin
declare[[show-AOT-syntax=true, show-question-marks=false, eta-contract=false]]
end

bundle AOT-no-syntax
begin
declare[[show-AOT-syntax=false, show-question-marks=true]]
end

parse-translation‹
[(-AOT-restriction, fn ctxt => fn [Const (name,-)] =>
let
val (restr, ctxt) = ctxt |> Local-Theory.raw-theory-result
  (fn thy => (Option.map fst (Symtab.lookup (AOT-Restriction.get thy) name), thy))
val restr = case restr of SOME x => x
  | - => raise Fail (Unknown restricted type:  ^ name)
in restr end
)]
›

print-translation‹
AOT-syntax-print-translations
[
  (const-syntax‹AOT-model-equiv-def›, fn ctxt => fn [x,y] =>
    Const (syntax-const‹-AOT-equiv-def›, dummyT) $
    (Const (syntax-const‹-AOT-process-frees›, dummyT) $ x) $
    (Const (syntax-const‹-AOT-process-frees›, dummyT) $ y))
]
›

print-translation‹
AOT-syntax-print-translations [
(const-syntax‹AOT-model-id-def›, fn ctxt =>
  fn [lhs as Abs (lhsName, lhsTy, lhsTrm), rhs as Abs (rhsName, rhsTy, rhsTrm)] =>
    let
      val (name,-) = Name.variant lhsName
        (Syntax-Trans.declare-term-names ctxt rhsTrm
          (Name.build-context (Syntax-Trans.declare-term-names ctxt lhsTrm)));
      val lhs = Term.betapply (lhs, Const (-bound, dummyT) $ Free (name, lhsTy))
      val rhs = Term.betapply (rhs, Const (-bound, dummyT) $ Free (name, rhsTy))
    in
      Const (const-syntax‹AOT-model-id-def›, dummyT) $ lhs $ rhs
    end
  | [Const (const-syntax‹case-prod›, -) $ lhs,
     Const (const-syntax‹case-prod›, -) $ rhs] =>
    Const (const-syntax‹AOT-model-id-def›, dummyT) $ lhs $ rhs
  | [Const (const-syntax‹case-unit›, -) $ lhs,
      Const (const-syntax‹case-unit›, -) $ rhs] =>
    Const (const-syntax‹AOT-model-id-def›, dummyT) $ lhs $ rhs
  | [x, y] =>
      Const (syntax-const‹-AOT-id-def›, dummyT) $
        (Const (syntax-const‹-AOT-process-frees›, dummyT) $ x) $
        (Const (syntax-const‹-AOT-process-frees›, dummyT) $ y)
)]›
```

Special marker for printing propositions as theorems and for pretty-printing AOT terms.

definition *print-as-theorem* :: ‹o ⇒ *bool*› **where**
‹*print-as-theorem* ≡ λ φ . ∀ v . [v ⊨ φ]›
lemma *print-as-theoremI*:
assumes ‹⋀ v . [v ⊨ φ]›
shows ‹*print-as-theorem* φ›
using *assms* **by** (*simp add*: *print-as-theorem-def*)
attribute-setup *print-as-theorem* =
‹*Scan.succeed* (*Thm.rule-attribute* []
(*K* (*fn thm* => *thm RS* @{*thm print-as-theoremI*})))›
Print as theorem.
print-translation‹*AOT-syntax-print-translations* [
(***const-syntax***‹*print-as-theorem*›, *fn ctxt* => *fn* [x] =>
(*Const* (***syntax-const***‹*-AOT-process-frees*›, *dummyT*) $ x))
]›

definition *print-term* :: ‹'a ⇒ 'a› **where** ‹*print-term* ≡ λ x . x›
syntax *-AOT-print-term* :: ‹τ ⇒ 'a› (‹*AOT'-TERM*[-]›)
translations
-AOT-print-term φ => *CONST print-term* (*-AOT-process-frees* φ)
print-translation‹*AOT-syntax-print-translations* [
(***const-syntax***‹*print-term*›, *fn ctxt* => *fn* [x] =>
(*Const* (***syntax-const***‹*-AOT-process-frees*›, *dummyT*) $ x))
]›

interpretation *AOT-no-meta-syntax*.

unbundle *AOT-syntax*

5 Abstract Semantics for AOT

specification(*AOT-denotes*)
— Relate object level denoting to meta-denoting. AOT's definitions of denoting will become derivable at each type.
AOT-sem-denotes: ‹[w ⊨ τ↓] = *AOT-model-denotes* τ›
by (*rule exI*[**where** x=‹λ τ . ε_o w . *AOT-model-denotes* τ›])
(*simp add*: *AOT-model-proposition-choice-simp*)

lemma *AOT-sem-var-induct*[*induct type*: *AOT-var*]:
assumes *AOT-denoting-term-case*: ‹⋀ τ . [v ⊨ τ↓] ⟹ [v ⊨ φ{τ}]›
shows ‹[v ⊨ φ{α}]›
by (*simp add*: *AOT-denoting-term-case AOT-sem-denotes AOT-term-of-var*)

specification(*AOT-imp*)
AOT-sem-imp: ‹[w ⊨ φ → ψ] = ([w ⊨ φ] ⟶ [w ⊨ ψ])›
by (*rule exI*[**where** x=‹λ φ ψ . ε_o w . ([w ⊨ φ] ⟶ [w ⊨ ψ])›])
(*simp add*: *AOT-model-proposition-choice-simp*)

specification(*AOT-not*)
AOT-sem-not: ‹[w ⊨ ¬φ] = (¬[w ⊨ φ])›
by (*rule exI*[**where** x=‹λ φ . ε_o w . ¬[w ⊨ φ]›])
(*simp add*: *AOT-model-proposition-choice-simp*)

specification(*AOT-box*)
AOT-sem-box: ‹[w ⊨ □φ] = (∀ w . [w ⊨ φ])›
by (*rule exI*[**where** x=‹λ φ . ε_o w . ∀ w . [w ⊨ φ]›])

(*simp add*: *AOT-model-proposition-choice-simp*)

specification(*AOT-act*)
AOT-sem-act: ‹$[w \models \mathcal{A}\varphi] = [w_0 \models \varphi]$›
by (*rule exI*[**where** x=‹$\lambda\ \varphi\ .\ \varepsilon_o\ w\ .\ [w_0 \models \varphi]$›])
(*simp add*: *AOT-model-proposition-choice-simp*)

Derived semantics for basic defined connectives.

lemma *AOT-sem-conj*: ‹$[w \models \varphi\ \&\ \psi] = ([w \models \varphi] \wedge [w \models \psi])$›
using *AOT-conj AOT-model-equiv-def AOT-sem-imp AOT-sem-not* **by** *auto*
lemma *AOT-sem-equiv*: ‹$[w \models \varphi \equiv \psi] = ([w \models \varphi] = [w \models \psi])$›
using *AOT-equiv AOT-sem-conj AOT-model-equiv-def AOT-sem-imp* **by** *auto*
lemma *AOT-sem-disj*: ‹$[w \models \varphi \vee \psi] = ([w \models \varphi] \vee [w \models \psi])$›
using *AOT-disj AOT-model-equiv-def AOT-sem-imp AOT-sem-not* **by** *auto*
lemma *AOT-sem-dia*: ‹$[w \models \Diamond\varphi] = (\exists\ w\ .\ [w \models \varphi])$›
using *AOT-dia AOT-sem-box AOT-model-equiv-def AOT-sem-not* **by** *auto*

specification(*AOT-forall*)
AOT-sem-forall: ‹$[w \models \forall\alpha\ \varphi\{\alpha\}] = (\forall\ \tau\ .\ [w \models \tau\downarrow] \longrightarrow [w \models \varphi\{\tau\}])$›
by (*rule exI*[**where** x=‹$\lambda\ op\ .\ \varepsilon_o\ w\ .\ \forall\ \tau\ .\ [w \models \tau\downarrow] \longrightarrow [w \models$ «$op\ \tau$»$]$›])
(*simp add*: *AOT-model-proposition-choice-simp*)

lemma *AOT-sem-exists*: ‹$[w \models \exists\alpha\ \varphi\{\alpha\}] = (\exists\ \tau\ .\ [w \models \tau\downarrow] \wedge [w \models \varphi\{\tau\}])$›
unfolding *AOT-exists*[*unfolded AOT-model-equiv-def*, *THEN spec*]
by (*simp add*: *AOT-sem-forall AOT-sem-not*)

specification(*AOT-eq*)
— Relate identity to denoting identity in the meta-logic. AOT's definitions of identity will become derivable at each type.
AOT-sem-eq: ‹$[w \models \tau = \tau'] = ([w \models \tau\downarrow] \wedge [w \models \tau'\downarrow] \wedge \tau = \tau')$›
by (*rule exI*[**where** x=‹$\lambda\ \tau\ \tau'\ .\ \varepsilon_o\ w\ .\ [w \models \tau\downarrow] \wedge [w \models \tau'\downarrow] \wedge \tau = \tau'$›])
(*simp add*: *AOT-model-proposition-choice-simp*)

specification(*AOT-desc*)
— Descriptions denote, if there is a unique denoting object satisfying the matrix in the actual world.
AOT-sem-desc-denotes: ‹$[w \models \iota x(\varphi\{x\})\downarrow] = (\exists!\ \kappa\ .\ [w_0 \models \kappa\downarrow] \wedge [w_0 \models \varphi\{\kappa\}])$›
— Denoting descriptions satisfy their matrix in the actual world.
AOT-sem-desc-prop: ‹$[w \models \iota x(\varphi\{x\})\downarrow] \Longrightarrow [w_0 \models \varphi\{\iota x(\varphi\{x\})\}]$›
— Uniqueness of denoting descriptions.
AOT-sem-desc-unique: ‹$[w \models \iota x(\varphi\{x\})\downarrow] \Longrightarrow [w \models \kappa\downarrow] \Longrightarrow [w_0 \models \varphi\{\kappa\}] \Longrightarrow$
$[w \models \iota x(\varphi\{x\}) = \kappa]$›
proof –
have ‹$\exists\,x{::}'a\ .\ \neg$*AOT-model-denotes* x›
using *AOT-model-nondenoting-ex*
by *blast*

Note that we may choose a distinct non-denoting object for each matrix. We do this explicitly merely to convince ourselves that our specification can still be satisfied.

then obtain *nondenoting* :: ‹$('a \Rightarrow o) \Rightarrow 'a$› **where**
nondenoting: ‹$\forall\ \varphi\ .\ \neg$*AOT-model-denotes* (*nondenoting* φ)›
by *fast*
define *desc* **where**
‹*desc* = ($\lambda\ \varphi$. *if* $(\exists!\ \kappa\ .\ [w_0 \models \kappa\downarrow] \wedge [w_0 \models \varphi\{\kappa\}])$
then (*THE* $\kappa\ .\ [w_0 \models \kappa\downarrow] \wedge [w_0 \models \varphi\{\kappa\}]$)
else nondenoting φ)›
{
fix φ :: ‹$'a \Rightarrow o$›
assume *ex1*: ‹$\exists!\ \kappa\ .\ [w_0 \models \kappa\downarrow] \wedge [w_0 \models \varphi\{\kappa\}]$›
then obtain κ **where** *x-prop*: $[w_0 \models \kappa\downarrow] \wedge [w_0 \models \varphi\{\kappa\}]$
unfolding *AOT-sem-denotes* **by** *blast*
moreover have (*desc* φ) = κ
unfolding *desc-def* **using** *x-prop ex1* **by** *fastforce*
ultimately have $[w_0 \models$ «*desc* φ»$\downarrow] \wedge [w_0 \models$ «φ (*desc* φ)»$]$

by *blast*
} **note** *1* = *this*
moreover {
fix φ :: ‹'*a* ⇒ o›
assume *nex1*: ‹∄! κ . [w_0 ⊨ κ↓] ∧ [w_0 ⊨ φ{κ}]›
hence (*desc* φ) = *nondenoting* φ **by** (*simp add*: *desc-def AOT-sem-denotes*)
hence [*w* ⊨ ¬«*desc* φ»↓] **for** *w*
by (*simp add*: *AOT-sem-denotes nondenoting AOT-sem-not*)
}
ultimately have *desc-denotes-simp*:
‹[*w* ⊨ «*desc* φ»↓] = (∃! κ . [w_0 ⊨ κ↓] ∧ [w_0 ⊨ φ{κ}])› **for** φ *w*
by (*simp add*: *AOT-sem-denotes desc-def nondenoting*)
have ‹(∀ φ *w*. [*w* ⊨ «*desc* φ»↓] = (∃!κ. [w_0 ⊨ κ↓] ∧ [w_0 ⊨ φ{κ}])) ∧
(∀ φ *w*. [*w* ⊨ «*desc* φ»↓] ⟶ [w_0 ⊨ «φ (*desc* φ)»]) ∧
(∀ φ *w* κ. [*w* ⊨ «*desc* φ»↓] ⟶ [*w* ⊨ κ↓] ⟶ [w_0 ⊨ φ{κ}] ⟶
[*w* ⊨ «*desc* φ» = κ])›
by (*insert 1*; *auto simp*: *desc-denotes-simp AOT-sem-eq AOT-sem-denotes*
desc-def nondenoting)
thus *?thesis*
by (*safe intro*!: *exI*[**where** *x*=*desc*]; *presburger*)
qed

specification(*AOT-exe AOT-lambda*)
— Truth conditions of exemplification formulas.
AOT-sem-exe: ‹[*w* ⊨ [Π]$\kappa_1 ... \kappa_n$] = ([*w* ⊨ Π↓] ∧ [*w* ⊨ $\kappa_1 ... \kappa_n$↓] ∧
[*w* ⊨ «*Rep-rel* Π $\kappa_1\kappa_n$»])›
— η-conversion for denoting terms; equivalent to AOT's axiom
AOT-sem-lambda-eta: ‹[*w* ⊨ Π↓] ⟹ [*w* ⊨ [λ$\nu_1 ... \nu_n$ [Π]$\nu_1 ... \nu_n$] = Π]›
— β-conversion; equivalent to AOT's axiom
AOT-sem-lambda-beta: ‹[*w* ⊨ [λ$\nu_1 ... \nu_n$ φ{$\nu_1 ... \nu_n$}]↓] ⟹ [*w* ⊨ $\kappa_1 ... \kappa_n$↓] ⟹
[*w* ⊨ [λ$\nu_1 ... \nu_n$ φ{$\nu_1 ... \nu_n$}]$\kappa_1 ... \kappa_n$] = [*w* ⊨ φ{$\kappa_1 ... \kappa_n$}]›
— Necessary and sufficient conditions for relations to denote. Equivalent to a theorem of AOT and used to derive the base cases of denoting relations (cqt.2).
AOT-sem-lambda-denotes: ‹[*w* ⊨ [λ$\nu_1 ... \nu_n$ φ{$\nu_1 ... \nu_n$}]↓] =
(∀ *v* $\kappa_1\kappa_n$ $\kappa_1{}'\kappa_n{}'$. [*v* ⊨ $\kappa_1 ... \kappa_n$↓] ∧ [*v* ⊨ $\kappa_1{}' ... \kappa_n{}'$↓] ∧
(∀ Π *v* . [*v* ⊨ Π↓] ⟶ [*v* ⊨ [Π]$\kappa_1 ... \kappa_n$] = [*v* ⊨ [Π]$\kappa_1{}' ... \kappa_n{}'$]) ⟶
[*v* ⊨ φ{$\kappa_1 ... \kappa_n$}] = [*v* ⊨ φ{$\kappa_1{}' ... \kappa_n{}'$}])›
— Equivalent to AOT's coexistence axiom.
AOT-sem-lambda-coex: ‹[*w* ⊨ [λ$\nu_1 ... \nu_n$ φ{$\nu_1 ... \nu_n$}]↓] ⟹
(∀ *w* $\kappa_1\kappa_n$. [*w* ⊨ $\kappa_1 ... \kappa_n$↓] ⟶ [*w* ⊨ φ{$\kappa_1 ... \kappa_n$}] = [*w* ⊨ ψ{$\kappa_1 ... \kappa_n$}]) ⟹
[*w* ⊨ [λ$\nu_1 ... \nu_n$ ψ{$\nu_1 ... \nu_n$}]↓]›
— Only the unary case of the following should hold, but our specification has to range over all types. We might move *AOT-exe* and *AOT-lambda* to type classes in the future to solve this.
AOT-sem-lambda-eq-prop-eq: ‹«[λ$\nu_1 ... \nu_n$ φ]» = «[λ$\nu_1 ... \nu_n$ ψ]» ⟹ φ = ψ›
— The following is solely required for validating n-ary relation identity and has the danger of implying artifactual theorems. Possibly avoidable by moving *AOT-exe* and *AOT-lambda* to type classes.
AOT-sem-exe-denoting: ‹[*w* ⊨ Π↓] ⟹ *AOT-exe* Π κ*s* = *Rep-rel* Π κ*s*›
— The following is required for validating the base cases of denoting relations (cqt.2). A version of this meta-logical identity will become derivable in future versions of AOT, so this will ultimately not result in artifactual theorems.
AOT-sem-exe-equiv: ‹*AOT-model-term-equiv x y* ⟹ *AOT-exe* Π *x* = *AOT-exe* Π *y*›
proof –
have ‹∃ *x* :: <'*a*> . ¬*AOT-model-denotes x*›
by (*rule exI*[**where** *x*=‹*Abs-rel* (λ *x* . ε_o *w*. *True*)›])
(*meson AOT-model-denotes-rel.abs-eq AOT-model-nondenoting-ex*
AOT-model-proposition-choice-simp)
define *exe* :: ‹<'*a*> ⇒ '*a* ⇒ o› **where**
‹*exe* ≡ λ Π κ*s* . *if AOT-model-denotes* Π
then Rep-rel Π κ*s*
else (ε_o *w* . *False*)›
define *lambda* :: ‹('*a*⇒o) ⇒ <'*a*>› **where**
‹*lambda* ≡ λ φ . *if AOT-model-denotes* (*Abs-rel* φ)
then (*Abs-rel* φ)
else

if (∀ κ κ′ w . (*AOT-model-denotes* κ ∧ *AOT-model-term-equiv* κ κ′) ⟶
[w ⊨ «φ κ»] = [w ⊨ «φ κ′»])
then
Abs-rel (*fix-irregular* (λ x . *if AOT-model-denotes* x
then φ (*SOME* y . *AOT-model-term-equiv* x y)
else (ε∘ w . *False*)))
else
Abs-rel φ›
have *fix-irregular-denoting-simp*[*simp*]:
‹*fix-irregular* (λx. *if AOT-model-denotes* x *then* φ x *else* ψ x) κ = φ κ›
if ‹*AOT-model-denotes* κ›
for κ :: ′a **and** φ ψ
by (*simp add*: *that fix-irregular-denoting*)
have *denoting-eps-cong*[*cong*]:
‹[w ⊨ «φ (*Eps* (*AOT-model-term-equiv* κ))»] = [w ⊨ «φ κ»]›
if ‹*AOT-model-denotes* κ›
and ‹∀ κ κ′. *AOT-model-denotes* κ ∧ *AOT-model-term-equiv* κ κ′ ⟶
(∀ w. [w ⊨ «φ κ»] = [w ⊨ «φ κ′»])›
for w :: w **and** κ :: ′a **and** φ :: ‹′a⇒o›
using *that AOT-model-term-equiv-eps*(*2*) **by** *blast*
have *exe-rep-rel*: ‹[w ⊨ «*exe* Π κ₁κₙ»] = ([w ⊨ Π↓] ∧ [w ⊨ κ₁...κₙ↓] ∧
[w ⊨ «*Rep-rel* Π κ₁κₙ»])› **for** w Π κ₁κₙ
by (*metis AOT-model-denotes-rel.rep-eq exe-def AOT-sem-denotes*
AOT-model-proposition-choice-simp)
moreover have ‹[w ⊨ «Π»↓] ⟹ [w ⊨ «*lambda* (*exe* Π)» = «Π»]› **for** Π w
by (*auto simp*: *Rep-rel-inverse lambda-def AOT-sem-denotes AOT-sem-eq*
AOT-model-denotes-rel-def Abs-rel-inverse exe-def)
moreover have *lambda-denotes-beta*:
‹[w ⊨ «*exe* (*lambda* φ) κ »] = [w ⊨ «φ κ»]›
if ‹[w ⊨ «*lambda* φ»↓]› **and** ‹[w ⊨ «κ»↓]›
for φ κ w
using *that* **unfolding** *exe-def AOT-sem-denotes*
by (*auto simp*: *lambda-def Abs-rel-inverse split*: *if-split-asm*)
moreover have *lambda-denotes-simp*:
‹[w ⊨ «*lambda* φ»↓] = (∀ v κ₁κₙ κ₁′κₙ′ . [v ⊨ κ₁...κₙ↓] ∧ [v ⊨ κ₁′...κₙ′↓] ∧
(∀ Π v . [v ⊨ Π↓] ⟶ [v ⊨ «*exe* Π κ₁κₙ»] = [v ⊨ «*exe* Π κ₁′κₙ′»]) ⟶
[v ⊨ φ{κ₁...κₙ}] = [v ⊨ φ{κ₁′...κₙ′}])› **for** φ w
proof
assume ‹[w ⊨ «*lambda* φ»↓]›
hence ‹*AOT-model-denotes* (*lambda* φ)›
unfolding *AOT-sem-denotes* **by** *simp*
moreover have ‹[w ⊨ «φ κ»] ⟹ [w ⊨ «φ κ′»]›
and ‹[w ⊨ «φ κ′»] ⟹ [w ⊨ «φ κ»]›
if ‹*AOT-model-denotes* κ› **and** ‹*AOT-model-term-equiv* κ κ′›
for w κ κ′
by (*metis* (*no-types*, *lifting*) *AOT-model-denotes-rel.abs-eq lambda-def*
that calculation)+
ultimately show ‹∀ v κ₁κₙ κ₁′κₙ′ . [v ⊨ κ₁...κₙ↓] ∧ [v ⊨ κ₁′...κₙ′↓] ∧
(∀ Π v . [v ⊨ Π↓] ⟶ [v ⊨ «*exe* Π κ₁κₙ»] = [v ⊨ «*exe* Π κ₁′κₙ′»]) ⟶
[v ⊨ φ{κ₁...κₙ}] = [v ⊨ φ{κ₁′...κₙ′}]›
unfolding *AOT-sem-denotes*
by (*metis* (*no-types*) *AOT-sem-denotes lambda-denotes-beta*)
next
assume ‹∀ v κ₁κₙ κ₁′κₙ′ . [v ⊨ κ₁...κₙ↓] ∧ [v ⊨ κ₁′...κₙ′↓] ∧
(∀ Π v . [v ⊨ Π↓] ⟶ [v ⊨ «*exe* Π κ₁κₙ»] = [v ⊨ «*exe* Π κ₁′κₙ′»]) ⟶
[v ⊨ φ{κ₁...κₙ}] = [v ⊨ φ{κ₁′...κₙ′}]›
hence ‹[w ⊨ «φ κ»] = [w ⊨ «φ κ′»]›
if ‹*AOT-model-denotes* κ ∧ *AOT-model-denotes* κ′ ∧ *AOT-model-term-equiv* κ κ′›
for w κ κ′
using *that*
by (*auto simp*: *AOT-sem-denotes*)
(*meson AOT-model-term-equiv-rel-equiv AOT-sem-denotes exe-rep-rel*)+
hence ‹[w ⊨ «φ κ»] = [w ⊨ «φ κ′»]›

```
      if ‹AOT-model-denotes κ ∧ AOT-model-term-equiv κ κ'›
      for w κ κ'
      using that AOT-model-term-equiv-denotes by blast
    hence ‹AOT-model-denotes (lambda φ)›
      by (auto simp: lambda-def Abs-rel-inverse AOT-model-denotes-rel.abs-eq
                     AOT-model-irregular-equiv AOT-model-term-equiv-eps(3)
                     AOT-model-term-equiv-regular fix-irregular-def AOT-sem-denotes
                     AOT-model-term-equiv-denotes AOT-model-proposition-choice-simp
                     AOT-model-irregular-false
               split: if-split-asm
               intro: AOT-model-irregular-eqI)
    thus ‹[w ⊨ «lambda φ»↓]›
      by (simp add: AOT-sem-denotes)
  qed
  moreover have ‹[w ⊨ «lambda ψ»↓]›
    if ‹[w ⊨ «lambda φ»↓]›
    and ‹∀ w κ1κn . [w ⊨ κ1...κn↓] ⟶ [w ⊨ φ{κ1...κn}] = [w ⊨ ψ{κ1...κn}]›
    for φ ψ w using that unfolding lambda-denotes-simp by auto
  moreover have ‹[w ⊨ Π↓] ⟹ exe Π κs = Rep-rel Π κs› for Π κs w
    by (simp add: exe-def AOT-sem-denotes)
  moreover have ‹lambda (λx. p) = lambda (λx. q) ⟹ p = q› for p q
    unfolding lambda-def
    by (auto split: if-split-asm simp: Abs-rel-inject fix-irregular-def)
       (meson AOT-model-irregular-nondenoting AOT-model-denoting-ex)+
  moreover have ‹AOT-model-term-equiv x y ⟹ exe Π x = exe Π y› for x y Π
    unfolding exe-def
    by (meson AOT-model-denotes-rel.rep-eq)
  note calculation = calculation this
  show ?thesis
    apply (safe intro!: exI[where x=exe] exI[where x=lambda])
    using calculation apply simp-all
    using lambda-denotes-simp by blast+
qed

lemma AOT-model-lambda-denotes:
  ‹AOT-model-denotes (AOT-lambda φ) = (∀ v κ κ' .
    AOT-model-denotes κ ∧ AOT-model-denotes κ' ∧ AOT-model-term-equiv κ κ' ⟶
    [v ⊨ «φ κ»] = [v ⊨ «φ κ'»])›
proof(safe)
  fix v and κ κ' :: 'a
  assume ‹AOT-model-denotes (AOT-lambda φ)›
  hence 1: ‹AOT-model-denotes κ1κn ∧
        AOT-model-denotes κ1'κn' ∧
        (∀Π v. AOT-model-denotes Π ⟶ [v ⊨ [Π]κ1...κn] = [v ⊨ [Π]κ1'...κn']) ⟶
        [v ⊨ φ{κ1...κn}] = [v ⊨ φ{κ1'...κn'}]› for κ1κn κ1'κn' v
    using AOT-sem-lambda-denotes[simplified AOT-sem-denotes] by blast
  {
    fix v and κ1κn κ1'κn' :: 'a
    assume d: ‹AOT-model-denotes κ1κn ∧ AOT-model-denotes κ1'κn' ∧
               AOT-model-term-equiv κ1κn κ1'κn'›
    hence ‹∀Π w. AOT-model-denotes Π ⟶ [w ⊨ [Π]κ1...κn] = [w ⊨ [Π]κ1'...κn']›
      by (metis AOT-sem-exe-equiv)
    hence ‹[v ⊨ φ{κ1...κn}] = [v ⊨ φ{κ1'...κn'}]› using d 1 by auto
  }
  moreover assume ‹AOT-model-denotes κ›
  moreover assume ‹AOT-model-denotes κ'›
  moreover assume ‹AOT-model-term-equiv κ κ'›
  ultimately show ‹[v ⊨ «φ κ»] ⟹ [v ⊨ «φ κ'»]›
              and ‹[v ⊨ «φ κ'»] ⟹ [v ⊨ «φ κ»]›
    by auto
next
  assume 0: ‹∀ v κ κ' . AOT-model-denotes κ ∧ AOT-model-denotes κ' ∧
                       AOT-model-term-equiv κ κ' ⟶ [v ⊨ «φ κ»] = [v ⊨ «φ κ'»]›
```

{
fix $\kappa_1\kappa_n$ $\kappa_1'\kappa_n'$ :: $'a$
assume *den*: ‹*AOT-model-denotes* $\kappa_1\kappa_n$›
moreover assume *den′*: ‹*AOT-model-denotes* $\kappa_1'\kappa_n'$›
assume ‹$\forall\,\Pi\ v$. *AOT-model-denotes* $\Pi \longrightarrow$
$[v \models [\Pi]\kappa_1...\kappa_n] = [v \models [\Pi]\kappa_1'...\kappa_n']$›
hence ‹$\forall\,\Pi\ v$. *AOT-model-denotes* $\Pi \longrightarrow$
$[v \models$ «*Rep-rel* $\Pi\ \kappa_1\kappa_n$»$] = [v \models$ «*Rep-rel* $\Pi\ \kappa_1'\kappa_n'$»$]$›
by (*simp add*: *AOT-sem-denotes AOT-sem-exe den den′*)
hence *AOT-model-term-equiv* $\kappa_1\kappa_n$ $\kappa_1'\kappa_n'$
unfolding *AOT-model-term-equiv-rel-equiv*[*OF den*, *OF den′*]
by *argo*
hence ‹$[v \models \varphi\{\kappa_1...\kappa_n\}] = [v \models \varphi\{\kappa_1'...\kappa_n'\}]$› **for** *v*
using *den den′ 0* **by** *blast*
}
thus ‹*AOT-model-denotes* (*AOT-lambda* φ)›
unfolding *AOT-sem-lambda-denotes*[*simplified AOT-sem-denotes*]
by *blast*
qed

specification (*AOT-lambda0*)
AOT-sem-lambda0: *AOT-lambda0* $\varphi = \varphi$
by (*rule exI*[**where** *x*=‹$\lambda x.\ x$›]) *simp*

specification(*AOT-concrete*)
AOT-sem-concrete: ‹$[w \models [E!]\kappa] =$
AOT-model-concrete $w\ \kappa$›
by (*rule exI*[**where** *x*=‹*AOT-var-of-term* (*Abs-rel*
($\lambda\ x$. $\varepsilon_\circ\ w$. *AOT-model-concrete* $w\ x$))›];
subst AOT-var-of-term-inverse)
(*auto simp*: *AOT-model-unary-regular AOT-model-concrete-denotes*
AOT-model-concrete-equiv AOT-model-regular-κ-def
AOT-model-proposition-choice-simp AOT-sem-exe Abs-rel-inverse
AOT-model-denotes-rel-def AOT-sem-denotes)

lemma *AOT-sem-equiv-defI*:
assumes ‹$\bigwedge\ v$. $[v \models \varphi] \Longrightarrow [v \models \psi]$›
and ‹$\bigwedge\ v$. $[v \models \psi] \Longrightarrow [v \models \varphi]$›
shows ‹*AOT-model-equiv-def* $\varphi\ \psi$›
using *AOT-model-equiv-def assms* **by** *blast*

lemma *AOT-sem-id-defI*:
assumes ‹$\bigwedge\ \alpha\ v$. $[v \models$ «$\sigma\ \alpha$»$\downarrow] \Longrightarrow [v \models$ «$\tau\ \alpha$» $=$ «$\sigma\ \alpha$»$]$›
assumes ‹$\bigwedge\ \alpha\ v$. $\neg[v \models$ «$\sigma\ \alpha$»$\downarrow] \Longrightarrow [v \models \neg$«$\tau\ \alpha$»$\downarrow]$›
shows ‹*AOT-model-id-def* $\tau\ \sigma$›
using *assms*
unfolding *AOT-sem-denotes AOT-sem-eq AOT-sem-not*
using *AOT-model-id-def*[*THEN iffD2*]
by *metis*

lemma *AOT-sem-id-def2I*:
assumes ‹$\bigwedge\ \alpha\ \beta\ v$. $[v \models$ «$\sigma\ \alpha\ \beta$»$\downarrow] \Longrightarrow [v \models$ «$\tau\ \alpha\ \beta$» $=$ «$\sigma\ \alpha\ \beta$»$]$›
assumes ‹$\bigwedge\ \alpha\ \beta\ v$. $\neg[v \models$ «$\sigma\ \alpha\ \beta$»$\downarrow] \Longrightarrow [v \models \neg$«$\tau\ \alpha\ \beta$»$\downarrow]$›
shows ‹*AOT-model-id-def* (*case-prod* τ) (*case-prod* σ)›
apply (*rule AOT-sem-id-defI*)
using *assms* **by** (*auto simp*: *AOT-sem-conj AOT-sem-not AOT-sem-eq AOT-sem-denotes*
AOT-model-denotes-prod-def)

lemma *AOT-sem-id-defE1*:
assumes ‹*AOT-model-id-def* $\tau\ \sigma$›
and ‹$[v \models$ «$\sigma\ \alpha$»$\downarrow]$›
shows ‹$[v \models$ «$\tau\ \alpha$» $=$ «$\sigma\ \alpha$»$]$›
using *assms* **by** (*simp add*: *AOT-model-id-def AOT-sem-denotes AOT-sem-eq*)

lemma *AOT-sem-id-defE2*:
assumes ‹*AOT-model-id-def* τ σ›
and ‹¬[v ⊨ «σ α»↓]›
shows ‹¬[v ⊨ «τ α»↓]›
using *assms* **by** (*simp add*: *AOT-model-id-def AOT-sem-denotes AOT-sem-eq*)

lemma *AOT-sem-id-def0I*:
assumes ‹⋀ v . [v ⊨ σ↓] ⟹ [v ⊨ τ = σ]›
and ‹⋀ v . ¬[v ⊨ σ↓] ⟹ [v ⊨ ¬τ↓]›
shows ‹*AOT-model-id-def* (*case-unit* τ) (*case-unit* σ)›
apply (*rule AOT-sem-id-defI*)
using *assms*
by (*simp-all add*: *AOT-sem-conj AOT-sem-eq AOT-sem-not AOT-sem-denotes*
AOT-model-denotes-unit-def case-unit-Unity)

lemma *AOT-sem-id-def0E1*:
assumes ‹*AOT-model-id-def* (*case-unit* τ) (*case-unit* σ)›
and ‹[v ⊨ σ↓]›
shows ‹[v ⊨ τ = σ]›
by (*metis* (*full-types*) *AOT-sem-id-defE1 assms*(*1*) *assms*(*2*) *case-unit-Unity*)

lemma *AOT-sem-id-def0E2*:
assumes ‹*AOT-model-id-def* (*case-unit* τ) (*case-unit* σ)›
and ‹¬[v ⊨ σ↓]›
shows ‹¬[v ⊨ τ↓]›
by (*metis AOT-sem-id-defE2 assms*(*1*) *assms*(*2*) *case-unit-Unity*)

lemma *AOT-sem-id-def0E3*:
assumes ‹*AOT-model-id-def* (*case-unit* τ) (*case-unit* σ)›
and ‹[v ⊨ σ↓]›
shows ‹[v ⊨ τ↓]›
using *AOT-sem-id-def0E1*[*OF assms*]
by (*simp add*: *AOT-sem-eq AOT-sem-denotes*)

lemma *AOT-sem-ordinary-def-denotes*: ‹[w ⊨ [λx ◇[E!]x]↓]›
unfolding *AOT-sem-denotes AOT-model-lambda-denotes*
by (*auto simp*: *AOT-sem-dia AOT-model-concrete-equiv*
AOT-sem-concrete AOT-sem-denotes)
lemma *AOT-sem-abstract-def-denotes*: ‹[w ⊨ [λx ¬◇[E!]x]↓]›
unfolding *AOT-sem-denotes AOT-model-lambda-denotes*
by (*auto simp*: *AOT-sem-dia AOT-model-concrete-equiv*
AOT-sem-concrete AOT-sem-denotes AOT-sem-not)

Relation identity is constructed using an auxiliary abstract projection mechanism with suitable instantiations for κ and products.

class *AOT-RelationProjection* =
fixes *AOT-sem-proj-id* :: ‹*'a*::*AOT-IndividualTerm* ⇒ (*'a* ⇒ o) ⇒ (*'a* ⇒ o) ⇒ o›
assumes *AOT-sem-proj-id-prop*:
‹[v ⊨ Π = Π'] =
[v ⊨ Π↓ & Π'↓ & ∀α («*AOT-sem-proj-id* α (λ τ . «[Π]τ») (λ τ . «[Π']τ»)»)]›
and *AOT-sem-proj-id-refl*:
‹[v ⊨ τ↓] ⟹ [v ⊨ [λν₁...νₙ φ{ν₁...νₙ}] = [λν₁...νₙ φ{ν₁...νₙ}]] ⟹
[v ⊨ «*AOT-sem-proj-id* τ φ φ»]›

class *AOT-UnaryRelationProjection* = *AOT-RelationProjection* +
assumes *AOT-sem-unary-proj-id*:
‹*AOT-sem-proj-id* κ φ ψ = «[λν₁...νₙ φ{ν₁...νₙ}] = [λν₁...νₙ ψ{ν₁...νₙ}]»›

instantiation κ :: *AOT-UnaryRelationProjection*
begin
definition *AOT-sem-proj-id-κ* :: ‹κ ⇒ (κ ⇒ o) ⇒ (κ ⇒ o) ⇒ o› **where**
‹*AOT-sem-proj-id-κ* κ φ ψ = «[λz φ{z}] = [λz ψ{z}]»›

```
instance proof
  show ‹[v ⊨ Π = Π'] =
        [v ⊨ Π↓ & Π'↓ & ∀α («AOT-sem-proj-id α (λ τ . «[Π]τ») (λ τ . «[Π']τ»)»)]›
    for v and Π Π' :: ‹<κ>›
    unfolding AOT-sem-proj-id-κ-def
    by (simp add: AOT-sem-eq AOT-sem-conj AOT-sem-denotes AOT-sem-forall)
       (metis AOT-sem-denotes AOT-model-denoting-ex AOT-sem-eq AOT-sem-lambda-eta)
next
  show ‹AOT-sem-proj-id κ φ ψ = «[λν1...νn φ{ν1...νn}] = [λν1...νn ψ{ν1...νn}]»›
    for κ :: κ and φ ψ
    unfolding AOT-sem-proj-id-κ-def ..
next
  show ‹[v ⊨ «AOT-sem-proj-id τ φ φ»]›
    if ‹[v ⊨ τ↓]› and ‹[v ⊨ [λν1...νn φ{ν1...νn}] = [λν1...νn φ{ν1...νn}]]›
    for τ :: κ and v φ
    using that by (simp add: AOT-sem-proj-id-κ-def AOT-sem-eq)
qed
end

instantiation prod ::
  ({AOT-UnaryRelationProjection, AOT-UnaryIndividualTerm}, AOT-RelationProjection)
  AOT-RelationProjection
begin
definition AOT-sem-proj-id-prod :: ‹'a×'b ⇒ ('a×'b ⇒ o) ⇒ ('a×'b ⇒ o) ⇒ o› where
  ‹AOT-sem-proj-id-prod ≡ λ (x,y) φ ψ . «[λz «φ (z,y)»] = [λz «ψ (z,y)»] &
    «AOT-sem-proj-id y (λ a . φ (x,a)) (λ a . ψ (x,a))»»›
instance proof
```

This is the main proof that allows to derive the definition of n-ary relation identity. We need to show that our defined projection identity implies relation identity for relations on pairs of individual terms.

```
  fix v and Π Π' :: ‹<'a×'b>›
  have AOT-meta-proj-denotes1: ‹AOT-model-denotes (Abs-rel (λz. AOT-exe Π (z, β)))›
    if ‹AOT-model-denotes Π› for Π :: ‹<'a×'b>› and β
    using that unfolding AOT-model-denotes-rel.rep-eq
    apply (simp add: Abs-rel-inverse AOT-meta-prod-equivI(2) AOT-sem-denotes
                     that)
    by (metis (no-types, lifting) AOT-meta-prod-equivI(2) AOT-model-denotes-prod-def
            AOT-model-unary-regular AOT-sem-exe AOT-sem-exe-equiv case-prodD)
  {
    fix κ :: 'a and Π :: ‹<'a×'b>›
    assume Π-denotes: ‹AOT-model-denotes Π›
    assume α-denotes: ‹AOT-model-denotes κ›
    hence ‹AOT-exe Π (κ, x) = AOT-exe Π (κ, y)›
      if ‹AOT-model-term-equiv x y› for x y :: 'b
      by (simp add: AOT-meta-prod-equivI(1) AOT-sem-exe-equiv that)
    moreover have ‹AOT-model-denotes κ1'κn'›
             if ‹[w ⊨ [Π]κ κ1'...κn']› for w κ1'κn'
      by (metis that AOT-model-denotes-prod-def AOT-sem-exe
              AOT-sem-denotes case-prodD)
    moreover {
      fix x :: 'b
      assume x-irregular: ‹¬AOT-model-regular x›
      hence prod-irregular: ‹¬AOT-model-regular (κ, x)›
        by (metis (no-types, lifting) AOT-model-irregular-nondenoting
                                      AOT-model-regular-prod-def case-prodD)
      hence ‹(¬AOT-model-denotes κ ∨ ¬AOT-model-regular x) ∧
           (AOT-model-denotes κ ∨ ¬AOT-model-denotes x)›
        unfolding AOT-model-regular-prod-def by blast
      hence x-nonden: ‹¬AOT-model-regular x›
        by (simp add: α-denotes)
      have ‹Rep-rel Π (κ, x) = AOT-model-irregular (Rep-rel Π) (κ, x)›
        using AOT-model-denotes-rel.rep-eq Π-denotes prod-irregular by blast
      moreover have ‹AOT-model-irregular (Rep-rel Π) (κ, x) =
```

AOT-model-irregular (*λz. Rep-rel* Π (*κ*, *z*)) *x*›
using Π-*denotes x-irregular prod-irregular x-nonden*
using *AOT-model-irregular-prod-generic*
apply (*induct arbitrary*: Π *x rule*: *AOT-model-irregular-prod.induct*)
by (*auto simp*: *α-denotes AOT-model-irregular-nondenoting*
AOT-model-regular-prod-def AOT-meta-prod-equivI(*2*)
AOT-model-denotes-rel.rep-eq AOT-model-term-equiv-eps(*1*)
intro!: *AOT-model-irregular-eqI*)
ultimately have
‹*AOT-exe* Π (*κ*, *x*) = *AOT-model-irregular* (*λz. AOT-exe* Π (*κ*, *z*)) *x*›
unfolding *AOT-sem-exe-denoting*[*simplified AOT-sem-denotes*, *OF* Π-*denotes*]
by *auto*
}
ultimately have ‹*AOT-model-denotes* (*Abs-rel* (*λz. AOT-exe* Π (*κ*, *z*)))›
by (*simp add*: *Abs-rel-inverse AOT-model-denotes-rel.rep-eq*)
} note *AOT-meta-proj-denotes2* = *this*
{
fix $\kappa_1{}'\kappa_n{}'$ :: *'b* and Π :: ‹<*'a*×*'b*>›
assume Π-*denotes*: ‹*AOT-model-denotes* Π›
assume *β-denotes*: ‹*AOT-model-denotes* $\kappa_1{}'\kappa_n{}'$›
hence ‹*AOT-exe* Π (*x*, $\kappa_1{}'\kappa_n{}'$) = *AOT-exe* Π (*y*, $\kappa_1{}'\kappa_n{}'$)›
if ‹*AOT-model-term-equiv x y*› for *x y* :: *'a*
by (*simp add*: *AOT-meta-prod-equivI*(*2*) *AOT-sem-exe-equiv that*)
moreover have ‹*AOT-model-denotes κ*›
if ‹[*w* ⊨ [Π]*κ* $\kappa_1{}'...\kappa_n{}'$]› for *w κ*
by (*metis that AOT-model-denotes-prod-def AOT-sem-exe*
AOT-sem-denotes case-prodD)
moreover {
fix *x* :: *'a*
assume ‹¬*AOT-model-regular x*›
hence ‹*False*›
using *AOT-model-unary-regular* by *blast*
hence
‹*AOT-exe* Π (*x*,$\kappa_1{}'\kappa_n{}'$) = *AOT-model-irregular* (*λz. AOT-exe* Π (*z*,$\kappa_1{}'\kappa_n{}'$)) *x*›
unfolding *AOT-sem-exe-denoting*[*simplified AOT-sem-denotes*, *OF* Π-*denotes*]
by *auto*
}
ultimately have ‹*AOT-model-denotes* (*Abs-rel* (*λz. AOT-exe* Π (*z*,$\kappa_1{}'\kappa_n{}'$)))›
by (*simp add*: *Abs-rel-inverse AOT-model-denotes-rel.rep-eq*)
} note *AOT-meta-proj-denotes1* = *this*
{
assume Π-*denotes*: ‹*AOT-model-denotes* Π›
assume Π′-*denotes*: ‹*AOT-model-denotes* Π′›
have Π-*proj2-den*: ‹*AOT-model-denotes* (*Abs-rel* (*λz. Rep-rel* Π (*α*, *z*)))›
if ‹*AOT-model-denotes α*› for *α*
using *that AOT-meta-proj-denotes2*[*OF* Π-*denotes*]
AOT-sem-exe-denoting[*simplified AOT-sem-denotes*,*OF* Π-*denotes*] by *simp*
have Π′-*proj2-den*: ‹*AOT-model-denotes* (*Abs-rel* (*λz. Rep-rel* Π′ (*α*, *z*)))›
if ‹*AOT-model-denotes α*› for *α*
using *that AOT-meta-proj-denotes2*[*OF* Π′-*denotes*]
AOT-sem-exe-denoting[*simplified AOT-sem-denotes*,*OF* Π′-*denotes*] by *simp*
have Π-*proj1-den*: ‹*AOT-model-denotes* (*Abs-rel* (*λz. Rep-rel* Π (*z*, *α*)))›
if ‹*AOT-model-denotes α*› for *α*
using *that AOT-meta-proj-denotes1*[*OF* Π-*denotes*]
AOT-sem-exe-denoting[*simplified AOT-sem-denotes*,*OF* Π-*denotes*] by *simp*
have Π′-*proj1-den*: ‹*AOT-model-denotes* (*Abs-rel* (*λz. Rep-rel* Π′ (*z*, *α*)))›
if ‹*AOT-model-denotes α*› for *α*
using *that AOT-meta-proj-denotes1*[*OF* Π′-*denotes*]
AOT-sem-exe-denoting[*simplified AOT-sem-denotes*,*OF* Π′-*denotes*] by *simp*
{
fix *κ* :: *'a* and $\kappa_1{}'\kappa_n{}'$ :: *'b*
assume ‹Π = Π′›
assume ‹*AOT-model-denotes* (*κ*,$\kappa_1{}'\kappa_n{}'$)›

hence ‹*AOT-model-denotes* κ› **and** *beta-denotes*: ‹*AOT-model-denotes* $\kappa_1{}'\kappa_n{}'$›
  by (*auto simp*: *AOT-model-denotes-prod-def*)
have ‹*AOT-model-denotes* «[λz [Π]z $\kappa_1{}'...\kappa_n{}'$]»›
  by (*rule AOT-model-lambda-denotes*[*THEN iffD2*])
    (*metis AOT-sem-exe-denoting AOT-meta-prod-equivI*(*2*)
        AOT-model-denotes-rel.rep-eq AOT-sem-denotes
        AOT-sem-exe-denoting Π-*denotes*)
moreover have ‹«[λz [Π]z $\kappa_1{}'...\kappa_n{}'$]» = «[$\lambda z$ [Π']z $\kappa_1{}'...\kappa_n{}'$]»›
  by (*simp add*: ‹$\Pi = \Pi'$›)
moreover have ‹[v $\models$ «*AOT-sem-proj-id* $\kappa_1{}'\kappa_n{}'$ ($\lambda\kappa_1{}'\kappa_n{}'$. «[$\Pi$]$\kappa$ $\kappa_1{}'...\kappa_n{}'$»)
                    ($\lambda\kappa_1{}'\kappa_n{}'$. «[$\Pi'$]$\kappa$ $\kappa_1{}'...\kappa_n{}'$»)»]›
  unfolding ‹$\Pi = \Pi'$› **using** *beta-denotes*
  by (*rule AOT-sem-proj-id-refl*[*unfolded AOT-sem-denotes*];
      simp add: *AOT-sem-denotes AOT-sem-eq AOT-model-lambda-denotes*)
    (*metis AOT-meta-prod-equivI*(*1*) *AOT-model-denotes-rel.rep-eq*
        AOT-sem-exe AOT-sem-exe-denoting Π'-*denotes*)
ultimately have ‹[v $\models$ «*AOT-sem-proj-id* (κ,$\kappa_1{}'\kappa_n{}'$) (λ $\kappa_1\kappa_n$. «[Π]$\kappa_1...\kappa_n$»)
                    (λ $\kappa_1\kappa_n$. «[Π']$\kappa_1...\kappa_n$»)»]›
  unfolding *AOT-sem-proj-id-prod-def*
  by (*simp add*: *AOT-sem-denotes AOT-sem-conj AOT-sem-eq*)
}
moreover {
assume ‹$\bigwedge$ α . *AOT-model-denotes* α $\Longrightarrow$
  [v $\models$ «*AOT-sem-proj-id* α (λ $\kappa_1\kappa_n$. «[Π]$\kappa_1...\kappa_n$») (λ $\kappa_1\kappa_n$. «[Π']$\kappa_1...\kappa_n$»)»]›
hence *0*: ‹*AOT-model-denotes* κ $\Longrightarrow$ *AOT-model-denotes* $\kappa_1{}'\kappa_n{}'$ $\Longrightarrow$
      AOT-model-denotes «[λz [Π]z $\kappa_1{}'...\kappa_n{}'$]» $\wedge$
      AOT-model-denotes «[λz [Π']z $\kappa_1{}'...\kappa_n{}'$]» $\wedge$
      «[λz [Π]z $\kappa_1{}'...\kappa_n{}'$]» = «[$\lambda z$ [Π']z $\kappa_1{}'...\kappa_n{}'$]» $\wedge$
      [v $\models$ «*AOT-sem-proj-id* $\kappa_1{}'\kappa_n{}'$ ($\lambda\kappa_1\kappa_n$. «[Π]κ $\kappa_1...\kappa_n$»)
                ($\lambda\kappa_1\kappa_n$. «[Π']κ $\kappa_1...\kappa_n$»)»]› **for** κ $\kappa_1{}'\kappa_n{}'$
  unfolding *AOT-sem-proj-id-prod-def*
  by (*auto simp*: *AOT-sem-denotes AOT-sem-conj AOT-sem-eq*
        AOT-model-denotes-prod-def)
obtain α*den* :: $'a$ **and** β*den* :: $'b$ **where**
  α*den*: ‹*AOT-model-denotes* α*den*› **and** β*den*: ‹*AOT-model-denotes* β*den*›
  using *AOT-model-denoting-ex* **by** *metis*
{
  fix κ :: $'a$
  assume α*denotes*: ‹*AOT-model-denotes* κ›
  have *1*: ‹[v $\models$ «*AOT-sem-proj-id* $\kappa_1{}'\kappa_n{}'$ ($\lambda\kappa_1{}'\kappa_n{}'$. «[$\Pi$]$\kappa$ $\kappa_1{}'...\kappa_n{}'$»)
                ($\lambda\kappa_1{}'\kappa_n{}'$. «[$\Pi'$]$\kappa$ $\kappa_1{}'...\kappa_n{}'$»)»]›
    if ‹*AOT-model-denotes* $\kappa_1{}'\kappa_n{}'$› **for** $\kappa_1{}'\kappa_n{}'$
    using *that 0* **using** α*denotes* **by** *blast*
  hence ‹[v $\models$ «*AOT-sem-proj-id* β (λz. *Rep-rel* Π (κ, z))
              (λz. *Rep-rel* Π' (κ, z))»]›
    if ‹*AOT-model-denotes* β› **for** β
    using *that*
    unfolding *AOT-sem-exe-denoting*[*simplified AOT-sem-denotes*, *OF* Π-*denotes*]
        AOT-sem-exe-denoting[*simplified AOT-sem-denotes*, *OF* Π'-*denotes*]
    by *blast*
  hence ‹*Abs-rel* (λz. *Rep-rel* Π (κ, z)) = *Abs-rel* (λz. *Rep-rel* Π' (κ, z))›
    using *AOT-sem-proj-id-prop*[*of v* ‹*Abs-rel* (λz. *Rep-rel* Π (κ, z))›
              ‹*Abs-rel* (λz. *Rep-rel* Π' (κ, z))›,
        simplified AOT-sem-eq AOT-sem-conj AOT-sem-forall
            AOT-sem-denotes, *THEN iffD2*]
      Π-*proj2-den*[*OF* α*denotes*] Π'-*proj2-den*[*OF* α*denotes*]
    unfolding *AOT-sem-exe-denoting*[*simplified AOT-sem-denotes*, *OF* Π-*denotes*]
        AOT-sem-exe-denoting[*simplified AOT-sem-denotes*,
              OF Π-*proj2-den*[*OF* α*denotes*]]
        AOT-sem-exe-denoting[*simplified AOT-sem-denotes*,
              OF Π'-*proj2-den*[*OF* α*denotes*]]
    by (*metis Abs-rel-inverse UNIV-I*)
  hence *Rep-rel* Π (κ,β) = *Rep-rel* Π' (κ,β) **for** β

by (*simp add*: *Abs-rel-inject*[*simplified*]) *meson*
} **note** *αdenotes* = *this*
{
fix $\kappa_1{}'\kappa_n{}'$ :: *'b*
assume *βden*: ‹*AOT-model-denotes* $\kappa_1{}'\kappa_n{}'$›
have *1*: ‹«[λ*z* [Π]*z* $\kappa_1{}'...\kappa_n{}'$]» = «[λ*z* [Π']*z* $\kappa_1{}'...\kappa_n{}'$]»›
using *0 βden AOT-model-denoting-ex* **by** *blast*
hence ‹*Abs-rel* (λ*z*. *Rep-rel* Π (*z*, $\kappa_1{}'\kappa_n{}'$)) =
Abs-rel (λ*z*. *Rep-rel* Π' (*z*, $\kappa_1{}'\kappa_n{}'$))› (**is** ‹*?a* = *?b*›)
apply (*safe intro*!: *AOT-sem-proj-id-prop*[*of v* ‹*?a*› ‹*?b*›,
simplified AOT-sem-eq AOT-sem-conj AOT-sem-forall
AOT-sem-denotes, *THEN iffD2*, *THEN conjunct2*, *THEN conjunct2*]
Π-*proj1-den*[*OF βden*] Π'-*proj1-den*[*OF βden*])
unfolding *AOT-sem-exe-denoting*[*simplified AOT-sem-denotes*, *OF* Π-*denotes*]
AOT-sem-exe-denoting[*simplified AOT-sem-denotes*, *OF* Π'-*denotes*]
AOT-sem-exe-denoting[*simplified AOT-sem-denotes*,
OF Π-*proj1-den*[*OF βden*]]
AOT-sem-exe-denoting[*simplified AOT-sem-denotes*,
OF Π'-*proj1-den*[*OF βden*]]
by (*subst* (*0 1*) *Abs-rel-inverse*; *simp?*)
(*metis* (*no-types*, *lifting*) *AOT-model-denotes-rel.abs-eq*
AOT-model-lambda-denotes AOT-sem-denotes AOT-sem-eq
AOT-sem-unary-proj-id Π-*proj1-den*[*OF βden*])
hence ‹*Rep-rel* Π (*x*,$\kappa_1{}'\kappa_n{}'$) = *Rep-rel* Π' (*x*,$\kappa_1{}'\kappa_n{}'$)› **for** *x*
by (*simp add*: *Abs-rel-inject*)
metis
} **note** *βdenotes* = *this*
{
fix α :: *'a* **and** β :: *'b*
assume ‹*AOT-model-regular* (α, β)›
moreover {
assume ‹*AOT-model-denotes* α ∧ *AOT-model-regular* β›
hence ‹*Rep-rel* Π (α,β) = *Rep-rel* Π' (α,β)›
using *αdenotes* **by** *presburger*
}
moreover {
assume ‹¬*AOT-model-denotes* α ∧ *AOT-model-denotes* β›
hence ‹*Rep-rel* Π (α,β) = *Rep-rel* Π' (α,β)›
by (*simp add*: *βdenotes*)
}
ultimately have ‹*Rep-rel* Π (α,β) = *Rep-rel* Π' (α,β)›
by (*metis* (*no-types*, *lifting*) *AOT-model-regular-prod-def case-prodD*)
}
hence ‹*Rep-rel* Π = *Rep-rel* Π'›
using Π-*denotes*[*unfolded AOT-model-denotes-rel.rep-eq*,
THEN conjunct2, *THEN conjunct2*, *THEN spec*, *THEN mp*]
using Π'-*denotes*[*unfolded AOT-model-denotes-rel.rep-eq*,
THEN conjunct2, *THEN conjunct2*, *THEN spec*, *THEN mp*]
using *AOT-model-irregular-eqI*[*of* ‹*Rep-rel* Π› ‹*Rep-rel* Π'› ‹(-,-)›]
using *AOT-model-irregular-nondenoting* **by** *fastforce*
hence ‹Π = Π'›
by (*rule Rep-rel-inject*[*THEN iffD1*])
}
ultimately have ‹Π = Π' = (∀ κ . *AOT-model-denotes* κ ⟶
[*v* ⊨ «*AOT-sem-proj-id* κ (λ $\kappa_1\kappa_n$. «[Π]$\kappa_1...\kappa_n$»)
(λ $\kappa_1\kappa_n$. «[Π']$\kappa_1...\kappa_n$»)»])›
by *auto*
}
thus ‹[*v* ⊨ Π = Π'] = [*v* ⊨ Π↓ & Π'↓ &
∀α («*AOT-sem-proj-id* α (λ $\kappa_1\kappa_n$. «[Π]$\kappa_1...\kappa_n$») (λ $\kappa_1\kappa_n$. «[Π']$\kappa_1...\kappa_n$»)»)]›
by (*auto simp*: *AOT-sem-eq AOT-sem-denotes AOT-sem-forall AOT-sem-conj*)
next
fix *v* **and** φ :: ‹*'a*×*'b*⇒o› **and** τ :: ‹*'a*×*'b*›

assume ‹[$v \models \tau\downarrow$]›
moreover assume ‹[$v \models [\lambda z_1 ... z_n$ «φ $z_1 z_n$»] = [$\lambda z_1 ... z_n$ «φ $z_1 z_n$»]]›
ultimately show ‹[$v \models$ «*AOT-sem-proj-id* τ φ φ»]›
unfolding *AOT-sem-proj-id-prod-def*
using *AOT-sem-proj-id-refl*[*of v snd* τ $\lambda b.$ φ (*fst* τ, *b*)]
by (*auto simp*: *AOT-sem-eq AOT-sem-conj AOT-sem-denotes AOT-model-denotes-prod-def AOT-model-lambda-denotes AOT-meta-prod-equivI*)
qed
end

Sanity-check to verify that n-ary relation identity follows.

lemma ‹[$v \models \Pi = \Pi'$] = [$v \models \Pi\downarrow$ & $\Pi'\downarrow$ & $\forall x \forall y([\lambda z\ [\Pi]z\ y] = [\lambda z\ [\Pi']z\ y]$ & $[\lambda z\ [\Pi]x\ z] = [\lambda z\ [\Pi']x\ z])$]›
for Π :: ‹<$\kappa \times \kappa$>›
by (*auto simp*: *AOT-sem-proj-id-prop*[*of v* Π Π'] *AOT-sem-proj-id-prod-def AOT-sem-conj AOT-sem-denotes AOT-sem-forall AOT-sem-unary-proj-id AOT-model-denotes-prod-def*)
lemma ‹[$v \models \Pi = \Pi'$] = [$v \models \Pi\downarrow$ & $\Pi'\downarrow$ & $\forall x_1 \forall x_2 \forall x_3$ (
$[\lambda z\ [\Pi]z\ x_2\ x_3] = [\lambda z\ [\Pi']z\ x_2\ x_3]$ &
$[\lambda z\ [\Pi]x_1\ z\ x_3] = [\lambda z\ [\Pi']x_1\ z\ x_3]$ &
$[\lambda z\ [\Pi]x_1\ x_2\ z] = [\lambda z\ [\Pi']x_1\ x_2\ z]$)]›
for Π :: ‹<$\kappa \times \kappa \times \kappa$>›
by (*auto simp*: *AOT-sem-proj-id-prop*[*of v* Π Π'] *AOT-sem-proj-id-prod-def AOT-sem-conj AOT-sem-denotes AOT-sem-forall AOT-sem-unary-proj-id AOT-model-denotes-prod-def*)
lemma ‹[$v \models \Pi = \Pi'$] = [$v \models \Pi\downarrow$ & $\Pi'\downarrow$ & $\forall x_1 \forall x_2 \forall x_3 \forall x_4$ (
$[\lambda z\ [\Pi]z\ x_2\ x_3\ x_4] = [\lambda z\ [\Pi']z\ x_2\ x_3\ x_4]$ &
$[\lambda z\ [\Pi]x_1\ z\ x_3\ x_4] = [\lambda z\ [\Pi']x_1\ z\ x_3\ x_4]$ &
$[\lambda z\ [\Pi]x_1\ x_2\ z\ x_4] = [\lambda z\ [\Pi']x_1\ x_2\ z\ x_4]$ &
$[\lambda z\ [\Pi]x_1\ x_2\ x_3\ z] = [\lambda z\ [\Pi']x_1\ x_2\ x_3\ z]$)]›
for Π :: ‹<$\kappa \times \kappa \times \kappa \times \kappa$>›
by (*auto simp*: *AOT-sem-proj-id-prop*[*of v* Π Π'] *AOT-sem-proj-id-prod-def AOT-sem-conj AOT-sem-denotes AOT-sem-forall AOT-sem-unary-proj-id AOT-model-denotes-prod-def*)

n-ary Encoding is constructed using a similar mechanism as n-ary relation identity using an auxiliary notion of projection-encoding.

class *AOT-Enc* =
fixes *AOT-enc* :: ‹$'a \Rightarrow$ <$'a$::*AOT-IndividualTerm*> $\Rightarrow$ o›
and *AOT-proj-enc* :: ‹$'a \Rightarrow ('a \Rightarrow$ o) $\Rightarrow$ o›
assumes *AOT-sem-enc-denotes*:
‹[$v \models$ «*AOT-enc* $\kappa_1 \kappa_n$ Π»] $\Longrightarrow$ [$v \models \kappa_1 ... \kappa_n\downarrow$] $\wedge$ [$v \models \Pi\downarrow$]›
assumes *AOT-sem-enc-proj-enc*:
‹[$v \models$ «*AOT-enc* $\kappa_1 \kappa_n$ Π»] =
[$v \models \Pi\downarrow$ & «*AOT-proj-enc* $\kappa_1 \kappa_n$ (λ $\kappa_1 \kappa_n$. «$[\Pi]\kappa_1 ... \kappa_n$»)»]›
assumes *AOT-sem-proj-enc-denotes*:
‹[$v \models$ «*AOT-proj-enc* $\kappa_1 \kappa_n$ φ»] $\Longrightarrow$ [$v \models \kappa_1 ... \kappa_n\downarrow$]›
assumes *AOT-sem-enc-nec*:
‹[$v \models$ «*AOT-enc* $\kappa_1 \kappa_n$ Π»] $\Longrightarrow$ [$w \models$ «*AOT-enc* $\kappa_1 \kappa_n$ Π»]›
assumes *AOT-sem-proj-enc-nec*:
‹[$v \models$ «*AOT-proj-enc* $\kappa_1 \kappa_n$ φ»] $\Longrightarrow$ [$w \models$ «*AOT-proj-enc* $\kappa_1 \kappa_n$ φ»]›
assumes *AOT-sem-universal-encoder*:
‹$\exists$ $\kappa_1 \kappa_n$. [$v \models \kappa_1 ... \kappa_n\downarrow$] $\wedge$ ($\forall$ Π . [$v \models \Pi\downarrow$] $\longrightarrow$ [$v \models$ «*AOT-enc* $\kappa_1 \kappa_n$ Π»]) $\wedge$
($\forall$ φ . [$v \models [\lambda z_1 ... z_n\ \varphi\{z_1 ... z_n\}]\downarrow$] $\longrightarrow$ [$v \models$ «*AOT-proj-enc* $\kappa_1 \kappa_n$ φ»])›

AOT-syntax-print-translations
-AOT-enc (*-AOT-individual-term* κ) (*-AOT-relation-term* Π) <= *CONST AOT-enc* κ Π

context *AOT-meta-syntax*
begin
notation *AOT-enc* (‹⦃-,-⦄›)
end

context *AOT-no-meta-syntax*
begin
no-notation *AOT-enc* (‹⦃-,-⦄›)
end

Unary encoding additionally has to satisfy the axioms of unary encoding and the definition of property identity.

class *AOT-UnaryEnc* = *AOT-UnaryIndividualTerm* +
  assumes *AOT-sem-enc-eq*: ‹[v ⊨ Π↓ & Π'↓ & □∀ν (ν[Π] ≡ ν[Π']) → Π = Π']›
    and *AOT-sem-A-objects*: ‹[v ⊨ ∃x (¬◇[E!]x & ∀F (x[F] ≡ φ{F}))]›
    and *AOT-sem-unary-proj-enc*: ‹*AOT-proj-enc* x ψ = *AOT-enc* x «[λz ψ{z}]»›
    and *AOT-sem-nocoder*: ‹[v ⊨ [E!]κ] ⟹ ¬[w ⊨ «*AOT-enc* κ Π»]›
    and *AOT-sem-ind-eq*: ‹([v ⊨ κ↓] ∧ [v ⊨ κ'↓] ∧ κ = (κ')) =
     (([v ⊨ [λx ◇[E!]x]κ] ∧
      [v ⊨ [λx ◇[E!]x]κ'] ∧
      (∀ v Π . [v ⊨ Π↓] ⟶ [v ⊨ [Π]κ] = [v ⊨ [Π]κ']))
     ∨ ([v ⊨ [λx ¬◇[E!]x]κ] ∧
       [v ⊨ [λx ¬◇[E!]x]κ'] ∧
       (∀ v Π . [v ⊨ Π↓] ⟶ [v ⊨ κ[Π]] = [v ⊨ κ'[Π]])))›

    and *AOT-sem-enc-indistinguishable-all*:
      ‹*AOT-ExtendedModel* ⟹
      [v ⊨ [λx ¬◇[E!]x]κ] ⟹
      [v ⊨ [λx ¬◇[E!]x]κ'] ⟹
      (⋀ Π' . [v ⊨ Π'↓] ⟹ (⋀ w . [w ⊨ [Π']κ] = [w ⊨ [Π']κ'])) ⟹
      [v ⊨ Π↓] ⟹
      (⋀ Π' . [v ⊨ Π'↓] ⟹ (⋀ κ_0 . [v ⊨ [λx ◇[E!]x]κ_0] ⟹
        (⋀ w . [w ⊨ [Π']κ_0] = [w ⊨ [Π]κ_0])) ⟹ [v ⊨ κ[Π']]) ⟹
      (⋀ Π' . [v ⊨ Π'↓] ⟹ (⋀ κ_0 . [v ⊨ [λx ◇[E!]x]κ_0] ⟹
        (⋀ w . [w ⊨ [Π']κ_0] = [w ⊨ [Π]κ_0])) ⟹ [v ⊨ κ'[Π']])›
    and *AOT-sem-enc-indistinguishable-ex*:
      ‹*AOT-ExtendedModel* ⟹
      [v ⊨ [λx ¬◇[E!]x]κ] ⟹
      [v ⊨ [λx ¬◇[E!]x]κ'] ⟹
      (⋀ Π' . [v ⊨ Π'↓] ⟹ (⋀ w . [w ⊨ [Π']κ] = [w ⊨ [Π']κ'])) ⟹
      [v ⊨ Π↓] ⟹
      ∃ Π' . [v ⊨ Π'↓] ∧ [v ⊨ κ[Π']] ∧
          (∀ κ_0 . [v ⊨ [λx ◇[E!]x]κ_0] ⟶
              (∀ w . [w ⊨ [Π']κ_0] = [w ⊨ [Π]κ_0])) ⟹
      ∃ Π' . [v ⊨ Π'↓] ∧ [v ⊨ κ'[Π']] ∧
          (∀ κ_0 . [v ⊨ [λx ◇[E!]x]κ_0] ⟶
              (∀ w . [w ⊨ [Π']κ_0] = [w ⊨ [Π]κ_0]))›

We specify encoding to align with the model-construction of encoding.

consts *AOT-sem-enc-κ* :: ‹κ ⇒ <κ> ⇒ o›
specification(*AOT-sem-enc-κ*)
  AOT-sem-enc-κ:
  ‹[v ⊨ «*AOT-sem-enc-κ* κ Π»] =
   (*AOT-model-denotes* κ ∧ *AOT-model-denotes* Π ∧ *AOT-model-enc* κ Π)›
  by (*rule exI*[**where** x=‹λ κ Π . ε_o w . *AOT-model-denotes* κ ∧ *AOT-model-denotes* Π ∧
                    AOT-model-enc κ Π›])
    (*simp add*: *AOT-model-proposition-choice-simp AOT-model-enc-κ-def κ.case-eq-if*)

We show that κ satisfies the generic properties of n-ary encoding.

instantiation κ :: *AOT-Enc*
begin
definition *AOT-enc-κ* :: ‹κ ⇒ <κ> ⇒ o› **where**
  ‹*AOT-enc-κ* ≡ *AOT-sem-enc-κ*›
definition *AOT-proj-enc-κ* :: ‹κ ⇒ (κ ⇒ o) ⇒ o› **where**
  ‹*AOT-proj-enc-κ* ≡ λ κ φ . *AOT-enc* κ «[λz «φ z»]»›
lemma *AOT-enc-κ-meta*:
  ‹[v ⊨ κ[Π]] = (*AOT-model-denotes* κ ∧ *AOT-model-denotes* Π ∧ *AOT-model-enc* κ Π)›

```
  for κ::κ
  using AOT-sem-enc-κ unfolding AOT-enc-κ-def by auto
instance proof
  fix v and κ :: κ and Π
  show ‹[v ⊨ «AOT-enc κ Π»] ⟹ [v ⊨ κ↓] ∧ [v ⊨ Π↓]›
    unfolding AOT-sem-denotes
    using AOT-enc-κ-meta by blast
next
  fix v and κ :: κ and Π
  show ‹[v ⊨ κ[Π]] = [v ⊨ Π↓ & «AOT-proj-enc κ (λ κ'.  «[Π]κ'»)»]›
  proof
    assume enc: ‹[v ⊨ κ[Π]]›
    hence Π-denotes: ‹AOT-model-denotes Π›
      by (simp add: AOT-enc-κ-meta)
    hence Π-eta-denotes: ‹AOT-model-denotes «[λz [Π]z]»›
      using AOT-sem-denotes AOT-sem-eq AOT-sem-lambda-eta by metis
    show ‹[v ⊨ Π↓ & «AOT-proj-enc κ (λ κ.  «[Π]κ»)»]›
      using AOT-sem-lambda-eta[simplified AOT-sem-denotes AOT-sem-eq, OF Π-denotes]
      using Π-eta-denotes Π-denotes
      by (simp add: AOT-sem-conj AOT-sem-denotes AOT-proj-enc-κ-def enc)
  next
    assume ‹[v ⊨ Π↓ & «AOT-proj-enc κ (λ κ.  «[Π]κ»)»]›
    hence Π-denotes: AOT-model-denotes Π and eta-enc: [v ⊨ κ[λz [Π]z]]
      by (auto simp: AOT-sem-conj AOT-sem-denotes AOT-proj-enc-κ-def)
    thus ‹[v ⊨ κ[Π]]›
      using AOT-sem-lambda-eta[simplified AOT-sem-denotes AOT-sem-eq, OF Π-denotes]
      by auto
  qed
next
  show ‹[v ⊨ «AOT-proj-enc κ φ»] ⟹ [v ⊨ κ↓]› for v and κ :: κ and φ
    by (simp add: AOT-enc-κ-meta AOT-sem-denotes AOT-proj-enc-κ-def)
next
  fix v w and κ :: κ and Π
  assume ‹[v ⊨ κ[Π]]›
  thus ‹[w ⊨ κ[Π]]›
    by (simp add: AOT-enc-κ-meta)
next
  fix v w and κ :: κ and φ
  assume ‹[v ⊨ «AOT-proj-enc κ φ»]›
  thus ‹[w ⊨ «AOT-proj-enc κ φ»]›
    by (simp add: AOT-enc-κ-meta AOT-proj-enc-κ-def)
next
  show ‹∃ κ::κ. [v ⊨ κ↓] ∧ (∀ Π . [v ⊨ Π↓] ⟶  [v ⊨ κ[Π]]) ∧
             (∀ φ . [v ⊨ [λz φ{z}]↓] ⟶  [v ⊨ «AOT-proj-enc κ φ»])› for v
    by (rule exI[where x=‹ακ UNIV›])
       (simp add: AOT-sem-denotes AOT-enc-κ-meta AOT-model-enc-κ-def
                  AOT-model-denotes-κ-def  AOT-proj-enc-κ-def)
qed
end
```

We show that κ satisfies the properties of unary encoding.

```
instantiation κ :: AOT-UnaryEnc
begin
instance proof
  fix v and Π Π' :: ‹<κ>›
  show ‹[v ⊨ Π↓ & Π'↓ & □∀ν (ν[Π] ≡ ν[Π']) → Π = Π']›
    apply (simp add: AOT-sem-forall AOT-sem-eq AOT-sem-imp AOT-sem-equiv
                     AOT-enc-κ-meta AOT-sem-conj AOT-sem-denotes AOT-sem-box)
    using AOT-meta-A-objects-κ by fastforce
next
  fix v and φ:: ‹<κ> ⇒ o›
  show ‹[v ⊨ ∃ x (¬◇[E!]x & ∀ F (x[F] ≡ φ{F}))]›
    using AOT-model-A-objects[of λ Π . [v ⊨ φ{Π}]]
```

```
    by (auto simp: AOT-sem-denotes AOT-sem-exists AOT-sem-conj AOT-sem-not
                   AOT-sem-dia AOT-sem-concrete AOT-enc-κ-meta AOT-sem-equiv
                   AOT-sem-forall)
next
  show ‹AOT-proj-enc x ψ = AOT-enc x (AOT-lambda ψ)› for x :: κ and ψ
    by (simp add: AOT-proj-enc-κ-def)
next
  show ‹[v ⊨ [E!]κ] ⟹ ¬ [w ⊨ κ[Π]]› for v w and κ :: κ and Π
    by (simp add: AOT-enc-κ-meta AOT-sem-concrete AOT-model-nocoder)
next
  fix v and κ κ' :: κ
  show ‹([v ⊨ κ↓] ∧ [v ⊨ κ'↓] ∧ κ = κ') =
        (([v ⊨ [λx ◇[E!]x]κ] ∧
          [v ⊨ [λx ◇[E!]x]κ'] ∧
          (∀ v Π . [v ⊨ Π↓] ⟶ [v ⊨ [Π]κ] = [v ⊨ [Π]κ']))
         ∨ ([v ⊨ [λx ¬◇[E!]x]κ] ∧
            [v ⊨ [λx ¬◇[E!]x]κ'] ∧
            (∀ v Π . [v ⊨ Π↓] ⟶ [v ⊨ κ[Π]] = [v ⊨ κ'[Π]])))›
    (is ‹?lhs = (?ordeq ∨ ?abseq)›)
  proof -
  {
    assume 0: ‹[v ⊨ κ↓] ∧ [v ⊨ κ'↓] ∧ κ = κ'›
    {
      assume ‹is-ωκ κ'›
      hence ‹[v ⊨ [λx ◇[E!]x]κ']›
        apply (subst AOT-sem-lambda-beta[OF AOT-sem-ordinary-def-denotes, of v κ'])
         apply (simp add: 0)
        apply (simp add: AOT-sem-dia)
        using AOT-sem-concrete AOT-model-ω-concrete-in-some-world is-ωκ-def by force
      hence ‹?ordeq› unfolding 0[THEN conjunct2, THEN conjunct2] by auto
    }
    moreover {
      assume ‹is-ακ κ'›
      hence ‹[v ⊨ [λx ¬◇[E!]x]κ']›
        apply (subst AOT-sem-lambda-beta[OF AOT-sem-abstract-def-denotes, of v κ'])
         apply (simp add: 0)
        apply (simp add: AOT-sem-not AOT-sem-dia)
        using AOT-sem-concrete is-ακ-def by force
      hence ‹?abseq› unfolding 0[THEN conjunct2, THEN conjunct2] by auto
    }
    ultimately have ‹?ordeq ∨ ?abseq›
      by (meson 0 AOT-sem-denotes AOT-model-denotes-κ-def κ.exhaust-disc)
  }
  moreover {
    assume ordeq: ‹?ordeq›
    hence κ-denotes: ‹[v ⊨ κ↓]› and κ'-denotes: ‹[v ⊨ κ'↓]›
      by (simp add: AOT-sem-denotes AOT-sem-exe)+
    hence ‹is-ωκ κ› and ‹is-ωκ κ'›
      by (metis AOT-model-concrete-κ.simps(2) AOT-model-denotes-κ-def
              AOT-sem-concrete AOT-sem-denotes AOT-sem-dia AOT-sem-lambda-beta
              AOT-sem-ordinary-def-denotes κ.collapse(2) κ.exhaust-disc ordeq)+
    have denotes: ‹[v ⊨ [λz «εₒ w . κυ z = κυ κ»]↓]›
      unfolding AOT-sem-denotes AOT-model-lambda-denotes
      by (simp add: AOT-model-term-equiv-κ-def)
    hence [v ⊨ [λz «εₒ w . κυ z = κυ κ»]κ] = [v ⊨ [λz «εₒ w . κυ z = κυ κ»]κ']
      using ordeq by (simp add: AOT-sem-denotes)
    hence ‹[v ⊨ «κ»↓] ∧ [v ⊨ «κ'»↓] ∧ κ = κ'›
      unfolding AOT-sem-lambda-beta[OF denotes, OF κ-denotes]
              AOT-sem-lambda-beta[OF denotes, OF κ'-denotes]
      using κ'-denotes ‹is-ωκ κ'› ‹is-ωκ κ› is-ωκ-def
          AOT-model-proposition-choice-simp by force
  }
  moreover {
```

```
    assume 0: ‹?abseq›
    hence κ-denotes: ‹[v ⊨ κ↓]› and κ'-denotes: ‹[v ⊨ κ'↓]›
      by (simp add: AOT-sem-denotes AOT-sem-exe)+
    hence ‹¬is-ωκ κ› and ‹¬is-ωκ κ'›
      by (metis AOT-model-ω-concrete-in-some-world AOT-model-concrete-κ.simps(1)
              AOT-sem-concrete AOT-sem-dia AOT-sem-exe AOT-sem-lambda-beta
              AOT-sem-not κ.collapse(1) 0)+
    hence ‹is-ακ κ› and ‹is-ακ κ'›
      by (meson AOT-sem-denotes AOT-model-denotes-κ-def κ.exhaust-disc
              κ-denotes κ'-denotes)+
    then obtain x y where κ-def: ‹κ = ακ x› and κ'-def: ‹κ' = ακ y›
      using is-ακ-def by auto
    {
      fix r
      assume ‹r ∈ x›
      hence ‹[v ⊨ κ[«urrel-to-rel r»]]›
        unfolding κ-def
        unfolding AOT-enc-κ-meta
        unfolding AOT-model-enc-κ-def
        apply (simp add: AOT-model-denotes-κ-def)
        by (metis (mono-tags) AOT-rel-equiv-def Quotient-def urrel-quotient)
      hence ‹[v ⊨ κ'[«urrel-to-rel r»]]›
        using AOT-enc-κ-meta 0 by (metis AOT-sem-enc-denotes)
      hence ‹r ∈ y›
        unfolding κ'-def
        unfolding AOT-enc-κ-meta
        unfolding AOT-model-enc-κ-def
        apply (simp add: AOT-model-denotes-κ-def)
        using Quotient-abs-rep urrel-quotient by fastforce
    }
    moreover {
      fix r
      assume ‹r ∈ y›
      hence ‹[v ⊨ κ'[«urrel-to-rel r»]]›
        unfolding κ'-def
        unfolding AOT-enc-κ-meta
        unfolding AOT-model-enc-κ-def
        apply (simp add: AOT-model-denotes-κ-def)
        by (metis (mono-tags) AOT-rel-equiv-def Quotient-def urrel-quotient)
      hence ‹[v ⊨ κ[«urrel-to-rel r»]]›
        using AOT-enc-κ-meta 0 by (metis AOT-sem-enc-denotes)
      hence ‹r ∈ x›
        unfolding κ-def
        unfolding AOT-enc-κ-meta
        unfolding AOT-model-enc-κ-def
        apply (simp add: AOT-model-denotes-κ-def)
        using Quotient-abs-rep urrel-quotient by fastforce
    }
    ultimately have x = y by blast
    hence ‹[v ⊨ κ↓] ∧ [v ⊨ κ'↓] ∧ κ = κ'›
      using κ'-def κ'-denotes κ-def by blast
  }
  ultimately show ?thesis
    unfolding AOT-sem-denotes
    by auto
  qed

next
  fix v and κ κ' :: κ and Π Π' :: ‹<κ>›
  assume ext: ‹AOT-ExtendedModel›
  assume ‹[v ⊨ [λx ¬◇[E!]x]κ]›
  hence ‹is-ακ κ›
    by (metis AOT-model-ω-concrete-in-some-world AOT-model-concrete-κ.simps(1)
```

AOT-model-denotes-κ-def AOT-sem-concrete AOT-sem-denotes AOT-sem-dia
AOT-sem-exe AOT-sem-lambda-beta AOT-sem-not κ.collapse(1) κ.exhaust-disc)
hence *κ-abs*: ‹¬(∃ w . *AOT-model-concrete w κ*)›
using *is-ακ-def* **by** *fastforce*
have *κ-den*: ‹*AOT-model-denotes κ*›
by (*simp add*: *AOT-model-denotes-κ-def κ.distinct-disc*(*5*) ‹*is-ακ κ*›)
assume ‹[*v* ⊨ [λ*x* ¬◇[*E*!]*x*]κ']›
hence ‹*is-ακ κ'*›
by (*metis AOT-model-ω-concrete-in-some-world AOT-model-concrete-κ.simps*(*1*)
AOT-model-denotes-κ-def AOT-sem-concrete AOT-sem-denotes AOT-sem-dia
AOT-sem-exe AOT-sem-lambda-beta AOT-sem-not κ.collapse(*1*)
κ.exhaust-disc)
hence *κ'-abs*: ‹¬(∃ w . *AOT-model-concrete w κ'*)›
using *is-ακ-def* **by** *fastforce*
have *κ'-den*: ‹*AOT-model-denotes κ'*›
by (*meson AOT-model-denotes-κ-def κ.distinct-disc*(*6*) ‹*is-ακ κ'*›)
assume ‹[*v* ⊨ Π'↓] ⟹ [*w* ⊨ [Π']κ] = [*w* ⊨ [Π']κ']› **for** Π' *w*
hence *indist*: ‹[*v* ⊨ «*Rep-rel* Π' κ»] = [*v* ⊨ «*Rep-rel* Π' κ'»]›
if ‹*AOT-model-denotes* Π'› **for** Π' *v*
by (*metis AOT-sem-denotes AOT-sem-exe κ'-den κ-den that*)
assume *κ-enc-cond*: ‹[*v* ⊨ Π'↓] ⟹
(⋀ κ₀ *w*. [*v* ⊨ [λ*x* ◇[*E*!]*x*]κ₀] ⟹
[*w* ⊨ [Π']κ₀] = [*w* ⊨ [Π]κ₀]) ⟹
[*v* ⊨ κ[Π']]› **for** Π'
assume Π-*den'*: ‹[*v* ⊨ Π↓]›
hence Π-*den*: ‹*AOT-model-denotes* Π›
using *AOT-sem-denotes* **by** *blast*
{
fix Π' :: ‹<κ>›
assume Π'-*den*: ‹*AOT-model-denotes* Π'›
hence Π'-*den'*: ‹[*v* ⊨ Π'↓]›
by (*simp add*: *AOT-sem-denotes*)
assume *1*: ‹∃ *w*. *AOT-model-concrete w x* ⟹
[*v* ⊨ «*Rep-rel* Π' *x*»] = [*v* ⊨ «*Rep-rel* Π *x*»]› **for** *v x*
{
fix κ₀ :: κ **and** *w*
assume ‹[*v* ⊨ [λ*x* ◇[*E*!]*x*]κ₀]›
hence ‹*is-ωκ* κ₀›
by (*smt* (*z3*) *AOT-model-concrete-κ.simps*(*2*) *AOT-model-denotes-κ-def*
AOT-sem-concrete AOT-sem-denotes AOT-sem-dia AOT-sem-exe
AOT-sem-lambda-beta κ.exhaust-disc is-ακ-def)
then obtain *x* **where** *x-prop*: ‹κ₀ = ωκ *x*›
using *is-ωκ-def* **by** *blast*
have ‹∃ *w* . *AOT-model-concrete w* (ωκ *x*)›
by (*simp add*: *AOT-model-ω-concrete-in-some-world*)
hence ‹[*v* ⊨ «*Rep-rel* Π' (ωκ *x*)»] = [*v* ⊨ «*Rep-rel* Π (ωκ *x*)»]› **for** *v*
using *1* **by** *blast*
hence ‹[*w* ⊨ [Π']κ₀] = [*w* ⊨ [Π]κ₀]› **unfolding** *x-prop*
by (*simp add*: *AOT-sem-exe AOT-sem-denotes AOT-model-denotes-κ-def*
Π'-*den* Π-*den*)
} note *2* = *this*
have ‹[*v* ⊨ κ[Π']]›
using *κ-enc-cond*[*OF* Π'-*den'*, *OF 2*]
by *metis*
hence ‹*AOT-model-enc* κ Π'›
using *AOT-enc-κ-meta* **by** *blast*
} note *κ-enc-cond* = *this*
hence ‹*AOT-model-denotes* Π' ⟹
(⋀*v x*. ∃ *w*. *AOT-model-concrete w x* ⟹
[*v* ⊨ «*Rep-rel* Π' *x*»] = [*v* ⊨ «*Rep-rel* Π *x*»]) ⟹
AOT-model-enc κ Π'› **for** Π'
by *blast*
assume Π'-*den'*: ‹[*v* ⊨ Π'↓]›

hence Π'-*den*: ‹*AOT-model-denotes* Π'›
  using *AOT-sem-denotes* **by** *blast*
assume *ord-indist*: ‹$[v \models [\lambda x\ \Diamond[E!]x]\kappa_0] \Longrightarrow$
  $[w \models [\Pi'\rceil\kappa_0] = [w \models [\Pi]\kappa_0]$› **for** κ_0 w
{
  fix w **and** κ_0 :: κ
  assume *0*: ‹$\exists\, w$. *AOT-model-concrete* w κ_0›
  hence ‹$[v \models [\lambda x\ \Diamond[E!]x]\kappa_0]$›
    using *AOT-model-concrete-denotes AOT-sem-concrete AOT-sem-denotes AOT-sem-dia AOT-sem-lambda-beta AOT-sem-ordinary-def-denotes* **by** *blast*
  hence ‹$[w \models [\Pi'\rceil\kappa_0] = [w \models [\Pi]\kappa_0]$›
    using *ord-indist* **by** *metis*
  hence ‹$[w \models$ «*Rep-rel* Π' κ_0»$] = [w \models$ «*Rep-rel* Π κ_0»$]$›
    by (*metis AOT-model-concrete-denotes AOT-sem-denotes AOT-sem-exe* Π'-*den* Π-*den 0*)
} **note** *ord-indist* = *this*
have ‹*AOT-model-enc* κ' Π'›
  using *AOT-model-enc-indistinguishable-all*
    [*OF ext*, *OF* κ-*den*, *OF* κ-*abs*, *OF* κ'-*den*, *OF* κ'-*abs*, *OF* Π-*den*]
    indist κ-*enc-cond* Π'-*den* *ord-indist* **by** *blast*
thus ‹$[v \models \kappa'[\Pi'\rceil]]$›
  using *AOT-enc-*κ*-meta* Π'-*den* κ'-*den* **by** *blast*
next
fix v **and** κ κ' :: κ **and** Π Π' :: ‹<κ>›
assume *ext*: ‹*AOT-ExtendedModel*›
assume ‹$[v \models [\lambda x\ \neg\Diamond[E!]x]\kappa]$›
hence ‹*is-*$\alpha\kappa$ κ›
  by (*metis AOT-model-*ω*-concrete-in-some-world AOT-model-concrete-*κ*.simps*(*1*)
    AOT-model-denotes-κ*-def AOT-sem-concrete AOT-sem-denotes AOT-sem-dia*
    AOT-sem-exe AOT-sem-lambda-beta AOT-sem-not κ*.collapse*(*1*)
    κ*.exhaust-disc*)
hence κ-*abs*: ‹$\neg(\exists\ w$. *AOT-model-concrete* w κ)›
  using *is-*$\alpha\kappa$*-def* **by** *fastforce*
have κ-*den*: ‹*AOT-model-denotes* κ›
  by (*simp add*: *AOT-model-denotes-*κ*-def* κ*.distinct-disc*(*5*) ‹*is-*$\alpha\kappa$ κ›)
assume ‹$[v \models [\lambda x\ \neg\Diamond[E!]x]\kappa'\rceil]$›
hence ‹*is-*$\alpha\kappa$ κ'›
  by (*metis AOT-model-*ω*-concrete-in-some-world AOT-model-concrete-*κ*.simps*(*1*)
    AOT-model-denotes-κ*-def AOT-sem-concrete AOT-sem-denotes AOT-sem-dia*
    AOT-sem-exe AOT-sem-lambda-beta AOT-sem-not κ*.collapse*(*1*)
    κ*.exhaust-disc*)
hence κ'-*abs*: ‹$\neg(\exists\ w$. *AOT-model-concrete* w κ')›
  using *is-*$\alpha\kappa$*-def* **by** *fastforce*
have κ'-*den*: ‹*AOT-model-denotes* κ'›
  by (*meson AOT-model-denotes-*κ*-def* κ*.distinct-disc*(*6*) ‹*is-*$\alpha\kappa$ κ'›)
assume ‹$[v \models \Pi'\downarrow] \Longrightarrow [w \models [\Pi'\rceil\kappa] = [w \models [\Pi'\rceil\kappa'\rceil]$› **for** Π' w
hence *indist*: ‹$[v \models$ «*Rep-rel* Π' κ»$] = [v \models$ «*Rep-rel* Π' κ'»$]$›
  if ‹*AOT-model-denotes* Π'› **for** Π' v
  by (*metis AOT-sem-denotes AOT-sem-exe* κ'-*den* κ-*den that*)
assume Π-*den*$'$: ‹$[v \models \Pi\downarrow]$›
hence Π-*den*: ‹*AOT-model-denotes* Π›
  using *AOT-sem-denotes* **by** *blast*
assume ‹$\exists\,\Pi'$. $[v \models \Pi'\downarrow] \wedge [v \models \kappa[\Pi'\rceil]] \wedge$
  $(\forall\,\kappa_0$. $[v \models [\lambda x\ \Diamond[E!]x]\kappa_0] \longrightarrow$
    $(\forall\, w$. $[w \models [\Pi'\rceil\kappa_0] = [w \models [\Pi]\kappa_0])))$›
then obtain Π' **where**
  Π'-*den*: ‹$[v \models \Pi'\downarrow]$› **and**
  Π'-*enc*: ‹$[v \models \kappa[\Pi'\rceil]]$› **and**
  Π'-*prop*: ‹$\forall\,\kappa_0$. $[v \models [\lambda x\ \Diamond[E!]x]\kappa_0] \longrightarrow$
    $(\forall\, w$. $[w \models [\Pi'\rceil\kappa_0] = [w \models [\Pi]\kappa_0])$›
  by *blast*
have ‹*AOT-model-denotes* Π'›
  using *AOT-enc-*κ*-meta* Π'-*enc* **by** *force*
moreover have ‹*AOT-model-enc* κ Π'›

```
    using AOT-enc-κ-meta Π'-enc by blast
  moreover have ‹(∃ w. AOT-model-concrete w κ₀) ⟹
               [v ⊨ «Rep-rel Π' κ₀»] = [v ⊨ «Rep-rel Π κ₀»]› for κ₀ v
  proof –
    assume 0: ‹∃ w. AOT-model-concrete w κ₀›
    hence ‹[v ⊨ [λx ◇[E!]x]κ₀]› for v
      using AOT-model-concrete-denotes AOT-sem-concrete AOT-sem-denotes AOT-sem-dia
          AOT-sem-lambda-beta AOT-sem-ordinary-def-denotes by blast
    hence ‹∀ w. [w ⊨ [Π']κ₀] = [w ⊨ [Π]κ₀]› using Π'-prop by blast
    thus ‹[v ⊨ «Rep-rel Π' κ₀»] = [v ⊨ «Rep-rel Π κ₀»]›
      by (meson 0 AOT-model-concrete-denotes AOT-sem-denotes AOT-sem-exe Π-den
               calculation(1))
  qed
  ultimately have ‹∃ Π'. AOT-model-denotes Π' ∧ AOT-model-enc κ Π' ∧
                     (∀ v x. (∃ w. AOT-model-concrete w x) ⟶
                      [v ⊨ «Rep-rel Π' x»] = [v ⊨ «Rep-rel Π x»])›
    by blast
  hence ‹∃ Π'. AOT-model-denotes Π' ∧ AOT-model-enc κ' Π' ∧
            (∀ v x. (∃ w. AOT-model-concrete w x) ⟶
             [v ⊨ «Rep-rel Π' x»] = [v ⊨ «Rep-rel Π x»])›
    using AOT-model-enc-indistinguishable-ex
          [OF ext, OF κ-den, OF κ-abs, OF κ'-den, OF κ'-abs, OF Π-den]
        indist by blast
  then obtain Π'' where
      Π''-den: ‹AOT-model-denotes Π''›
      and Π''-enc: ‹AOT-model-enc κ' Π''›
      and Π''-prop: ‹(∃ w. AOT-model-concrete w x) ⟹
                   [v ⊨ «Rep-rel Π'' x»] = [v ⊨ «Rep-rel Π x»]› for v x
    by blast
  have ‹[v ⊨ Π''↓]›
    by (simp add: AOT-sem-denotes Π''-den)
  moreover have ‹[v ⊨ κ'[Π'']]›
    by (simp add: AOT-enc-κ-meta Π''-den Π''-enc κ'-den)
  moreover have ‹[v ⊨ [λx ◇[E!]x]κ₀] ⟹
               (∀ w. [w ⊨ [Π'']κ₀] = [w ⊨ [Π]κ₀])› for κ₀
  proof –
    assume ‹[v ⊨ [λx ◇[E!]x]κ₀]›
    hence ‹∃ w. AOT-model-concrete w κ₀›
      by (metis AOT-sem-concrete AOT-sem-dia AOT-sem-exe AOT-sem-lambda-beta)
    thus ‹∀ w. [w ⊨ [Π'']κ₀] = [w ⊨ [Π]κ₀]›
      using Π''-prop
      by (metis AOT-sem-denotes AOT-sem-exe Π''-den Π-den)
  qed
  ultimately show ‹∃ Π'. [v ⊨ Π'↓] ∧ [v ⊨ κ'[Π']] ∧
                     (∀ κ₀. [v ⊨ [λx ◇[E!]x]κ₀] ⟶
                          (∀ w. [w ⊨ [Π']κ₀] = [w ⊨ [Π]κ₀]))›
    by (safe intro!: exI[where x=Π'']) blast+
qed
end
```

Define encoding for products using projection-encoding.

```
instantiation prod :: (AOT-UnaryEnc, AOT-Enc) AOT-Enc
begin
definition AOT-proj-enc-prod :: ‹'a×'b ⇒ ('a×'b ⇒ o) ⇒ o› where
  ‹AOT-proj-enc-prod ≡ λ (κ,κ') φ . «κ[λν «φ (ν,κ')»] &
                                «AOT-proj-enc κ' (λν. φ (κ,ν))»»›
definition AOT-enc-prod :: ‹'a×'b ⇒ <'a×'b> ⇒ o› where
  ‹AOT-enc-prod ≡ λ κ Π . «Π↓ & «AOT-proj-enc κ (λ κ₁'κₙ'. «[Π]κ₁'...κₙ'»)»»›
instance proof
  show ‹[v ⊨ κ₁...κₙ[Π]] ⟹ [v ⊨ κ₁...κₙ↓] ∧ [v ⊨ Π↓]›
    for v and κ₁κₙ :: ‹'a×'b› and Π
    unfolding AOT-enc-prod-def
    apply (induct κ₁κₙ; simp add: AOT-sem-conj AOT-sem-denotes AOT-proj-enc-prod-def)
```

```
    by (metis AOT-sem-denotes AOT-model-denotes-prod-def AOT-sem-enc-denotes
              AOT-sem-proj-enc-denotes case-prodI)
next
  show ‹[v ⊨ κ1...κn[Π]] =
        [v ⊨ «Π»↓ & «AOT-proj-enc κ1κn (λ κ1κn.  «[Π]κ1...κn»)»]›
    for v and κ1κn :: ‹'a×'b› and Π
    unfolding AOT-enc-prod-def ..
next
  show ‹[v ⊨ «AOT-proj-enc κs φ»] ⟹ [v ⊨ «κs»↓]›
    for v and κs :: ‹'a×'b› and φ
    by (metis (mono-tags, lifting)
          AOT-sem-conj AOT-sem-denotes AOT-model-denotes-prod-def
          AOT-sem-enc-denotes AOT-sem-proj-enc-denotes
          AOT-proj-enc-prod-def case-prod-unfold)
next
  fix v w Π and κ1κn :: ‹'a×'b›
  show ‹[w ⊨ κ1...κn[Π]]› if ‹[v ⊨ κ1...κn[Π]]› for v w Π and κ1κn :: ‹'a×'b›
    by (metis (mono-tags, lifting)
          AOT-enc-prod-def AOT-sem-enc-proj-enc AOT-sem-conj AOT-sem-denotes
          AOT-sem-proj-enc-nec AOT-proj-enc-prod-def case-prod-unfold that)
next
  show ‹[w ⊨ «AOT-proj-enc κ1κn φ»]› if ‹[v ⊨ «AOT-proj-enc κ1κn φ»]›
    for v w φ and κ1κn :: ‹'a×'b›
    by (metis (mono-tags, lifting)
          that AOT-sem-enc-proj-enc AOT-sem-conj AOT-sem-denotes
          AOT-sem-proj-enc-nec AOT-proj-enc-prod-def case-prod-unfold)
next
  fix v
  obtain κ :: 'a where a-prop: ‹[v ⊨ κ↓] ∧ (∀ Π . [v ⊨ Π↓] ⟶  [v ⊨ κ[Π]])›
    using AOT-sem-universal-encoder by blast
  obtain κ1'κn' :: 'b where b-prop:
    ‹[v ⊨ κ1'...κn'↓] ∧ (∀ φ . [v ⊨ [λν1...νn «φ ν1νn»]↓] ⟶
                               [v ⊨ «AOT-proj-enc κ1'κn' φ»])›
    using AOT-sem-universal-encoder by blast
  have ‹AOT-model-denotes «[λν1...νn [«Π»]ν1...νn κ1'...κn']»›
    if ‹AOT-model-denotes Π› for Π :: ‹<'a×'b>›
    unfolding AOT-model-lambda-denotes
    by (metis AOT-meta-prod-equivI(2) AOT-sem-exe-equiv)
  moreover have ‹AOT-model-denotes  «[λν1...νn [«Π»]κ ν1...νn]»›
    if ‹AOT-model-denotes Π› for Π :: ‹<'a×'b>›
    unfolding AOT-model-lambda-denotes
    by (metis AOT-meta-prod-equivI(1) AOT-sem-exe-equiv)
  ultimately have 1: ‹[v ⊨ «(κ,κ1'κn')»↓]›
            and 2: ‹(∀ Π . [v ⊨ Π↓] ⟶  [v ⊨ κ κ1'...κn'[Π]])›
    using a-prop b-prop
    by (auto simp: AOT-sem-denotes AOT-enc-κ-meta AOT-model-enc-κ-def
                   AOT-model-denotes-κ-def AOT-model-denotes-prod-def
                   AOT-enc-prod-def AOT-proj-enc-prod-def AOT-sem-conj)
  have ‹AOT-model-denotes «[λz1...zn «φ (z1zn, κ1'κn')»]»›
    if ‹AOT-model-denotes «[λz1...zm φ{z1...zm}]»› for φ :: ‹'a×'b ⇒ o›
    using that
    unfolding AOT-model-lambda-denotes
    by (metis (no-types, lifting) AOT-sem-denotes AOT-model-denotes-prod-def
                                  AOT-meta-prod-equivI(2) b-prop case-prodI)
  moreover have ‹AOT-model-denotes «[λz1...zn «φ (κ, z1zn)»]»›
    if ‹AOT-model-denotes «[λz1...zm φ{z1...zm}]»› for φ :: ‹'a×'b ⇒ o›
    using that
    unfolding AOT-model-lambda-denotes
    by (metis (no-types, lifting) AOT-sem-denotes AOT-model-denotes-prod-def
                                  AOT-meta-prod-equivI(1) a-prop case-prodI)
  ultimately have 3:
    ‹[v ⊨ «(κ,κ1'κn')»↓] ∧ (∀ φ . [v ⊨ [λz1...zn φ{z1...zn}]↓] ⟶
                                  [v ⊨ «AOT-proj-enc (κ,κ1'κn') φ»])›
```

using *a-prop b-prop*
by (*auto simp*: *AOT-sem-denotes AOT-enc-κ-meta AOT-model-enc-κ-def AOT-model-denotes-κ-def AOT-enc-prod-def AOT-proj-enc-prod-def AOT-sem-conj AOT-model-denotes-prod-def*)
show ‹$\exists\, \kappa_1\kappa_n$::'a×'b. $[v \models \kappa_1...\kappa_n\downarrow] \wedge (\forall\ \Pi\ .\ [v \models \Pi\downarrow] \longrightarrow [v \models \kappa_1...\kappa_n[\Pi]]) \wedge$
$(\forall\ \varphi\ .\ [v \models [\lambda z_1...z_n$ «$\varphi\ z_1 z_n$»$]\downarrow] \longrightarrow$
$[v \models$ «*AOT-proj-enc* $\kappa_1\kappa_n\ \varphi$»$])$›
apply (*rule exI*[**where** x=‹$(\kappa,\kappa_1{}'\kappa_n{}')$›]) **using** *1 2 3* **by** *blast*
qed
end

Sanity-check to verify that n-ary encoding follows.

lemma ‹$[v \models \kappa_1\kappa_2[\Pi]] = [v \models \Pi\downarrow\ \&\ \kappa_1[\lambda\nu\ [\Pi]\nu\kappa_2]\ \&\ \kappa_2[\lambda\nu\ [\Pi]\kappa_1\nu]]$›
for κ_1 :: 'a::*AOT-UnaryEnc* **and** κ_2 :: 'b::*AOT-UnaryEnc*
by (*simp add*: *AOT-sem-conj AOT-enc-prod-def AOT-proj-enc-prod-def AOT-sem-unary-proj-enc*)
lemma ‹$[v \models \kappa_1\kappa_2\kappa_3[\Pi]] =$
$[v \models \Pi\downarrow\ \&\ \kappa_1[\lambda\nu\ [\Pi]\nu\kappa_2\kappa_3]\ \&\ \kappa_2[\lambda\nu\ [\Pi]\kappa_1\nu\kappa_3]\ \&\ \kappa_3[\lambda\nu\ [\Pi]\kappa_1\kappa_2\nu]]$›
for $\kappa_1\ \kappa_2\ \kappa_3$ :: 'a::*AOT-UnaryEnc*
by (*simp add*: *AOT-sem-conj AOT-enc-prod-def AOT-proj-enc-prod-def AOT-sem-unary-proj-enc*)

lemma *AOT-sem-vars-denote*: ‹$[v \models \alpha_1...\alpha_n\downarrow]$›
by *induct simp*

Combine the introduced type classes and register them as type constraints for individual terms.

class *AOT-κs* = *AOT-IndividualTerm* + *AOT-RelationProjection* + *AOT-Enc*
class *AOT-κ* = *AOT-κs* + *AOT-UnaryIndividualTerm* +
AOT-UnaryRelationProjection + *AOT-UnaryEnc*

instance κ :: *AOT-κ* **by** *standard*
instance *prod* :: (*AOT-κ*, *AOT-κs*) *AOT-κs* **by** *standard*

AOT-register-type-constraints
Individual: ‹-::*AOT-κ*› ‹-::*AOT-κs*› **and**
Relation: ‹<-::*AOT-κs*>›

We define semantic predicates to capture the conditions of cqt.2 (i.e. the base cases of denoting terms) on matrices of λ-expressions.

definition *AOT-instance-of-cqt-2* :: ‹('a::*AOT-κs* ⇒ o) ⇒ *bool*› **where**
‹*AOT-instance-of-cqt-2* ≡ λ φ . ∀ x y . *AOT-model-denotes* x ∧ *AOT-model-denotes* y ∧
AOT-model-term-equiv x y ⟶ φ x = φ y›
definition *AOT-instance-of-cqt-2-exe-arg* :: ‹('a::*AOT-κs* ⇒ 'b::*AOT-κs*) ⇒ *bool*› **where**
‹*AOT-instance-of-cqt-2-exe-arg* ≡ λ φ . ∀ x y .
AOT-model-denotes x ∧ *AOT-model-denotes* y ∧ *AOT-model-term-equiv* x y ⟶
AOT-model-term-equiv (φ x) (φ y)›

λ-expressions with a matrix that satisfies our predicate denote.

lemma *AOT-sem-cqt-2*:
assumes ‹*AOT-instance-of-cqt-2* φ›
shows ‹$[v \models [\lambda\nu_1...\nu_n\ \varphi\{\nu_1...\nu_n\}]\downarrow]$›
using *assms*
by (*metis AOT-instance-of-cqt-2-def AOT-model-lambda-denotes AOT-sem-denotes*)

syntax *AOT-instance-of-cqt-2* :: ‹*id-position* ⇒ *AOT-prop*›
(‹*INSTANCE'-OF'-CQT'-2'*(-')›)

Prove introduction rules for the predicates that match the natural language restrictions of the axiom.

named-theorems *AOT-instance-of-cqt-2-intro*
lemma *AOT-instance-of-cqt-2-intros-const*[*AOT-instance-of-cqt-2-intro*]:
‹*AOT-instance-of-cqt-2* (λα. φ)›
by (*simp add*: *AOT-instance-of-cqt-2-def AOT-sem-denotes AOT-model-lambda-denotes*)

lemma *AOT-instance-of-cqt-2-intros-not*[*AOT-instance-of-cqt-2-intro*]:
assumes ‹*AOT-instance-of-cqt-2* φ›
shows ‹*AOT-instance-of-cqt-2* ($\lambda\tau$. «$\neg\varphi\{\tau\}$»)›
using *assms*
by (*metis* (*no-types*, *lifting*) *AOT-instance-of-cqt-2-def*)
lemma *AOT-instance-of-cqt-2-intros-imp*[*AOT-instance-of-cqt-2-intro*]:
assumes ‹*AOT-instance-of-cqt-2* φ› **and** ‹*AOT-instance-of-cqt-2* ψ›
shows ‹*AOT-instance-of-cqt-2* ($\lambda\tau$. «$\varphi\{\tau\} \rightarrow \psi\{\tau\}$»)›
using *assms*
by (*auto simp*: *AOT-instance-of-cqt-2-def AOT-sem-denotes*
AOT-model-lambda-denotes AOT-sem-imp)
lemma *AOT-instance-of-cqt-2-intros-box*[*AOT-instance-of-cqt-2-intro*]:
assumes ‹*AOT-instance-of-cqt-2* φ›
shows ‹*AOT-instance-of-cqt-2* ($\lambda\tau$. «$\Box\varphi\{\tau\}$»)›
using *assms*
by (*auto simp*: *AOT-instance-of-cqt-2-def AOT-sem-denotes*
AOT-model-lambda-denotes AOT-sem-box)
lemma *AOT-instance-of-cqt-2-intros-act*[*AOT-instance-of-cqt-2-intro*]:
assumes ‹*AOT-instance-of-cqt-2* φ›
shows ‹*AOT-instance-of-cqt-2* ($\lambda\tau$. «$\boldsymbol{\mathcal{A}}\varphi\{\tau\}$»)›
using *assms*
by (*auto simp*: *AOT-instance-of-cqt-2-def AOT-sem-denotes*
AOT-model-lambda-denotes AOT-sem-act)
lemma *AOT-instance-of-cqt-2-intros-diamond*[*AOT-instance-of-cqt-2-intro*]:
assumes ‹*AOT-instance-of-cqt-2* φ›
shows ‹*AOT-instance-of-cqt-2* ($\lambda\tau$. «$\Diamond\varphi\{\tau\}$»)›
using *assms*
by (*auto simp*: *AOT-instance-of-cqt-2-def AOT-sem-denotes*
AOT-model-lambda-denotes AOT-sem-dia)
lemma *AOT-instance-of-cqt-2-intros-conj*[*AOT-instance-of-cqt-2-intro*]:
assumes ‹*AOT-instance-of-cqt-2* φ› **and** ‹*AOT-instance-of-cqt-2* ψ›
shows ‹*AOT-instance-of-cqt-2* ($\lambda\tau$. «$\varphi\{\tau\}$ & $\psi\{\tau\}$»)›
using *assms*
by (*auto simp*: *AOT-instance-of-cqt-2-def AOT-sem-denotes*
AOT-model-lambda-denotes AOT-sem-conj)
lemma *AOT-instance-of-cqt-2-intros-disj*[*AOT-instance-of-cqt-2-intro*]:
assumes ‹*AOT-instance-of-cqt-2* φ› **and** ‹*AOT-instance-of-cqt-2* ψ›
shows ‹*AOT-instance-of-cqt-2* ($\lambda\tau$. «$\varphi\{\tau\} \vee \psi\{\tau\}$»)›
using *assms*
by (*auto simp*: *AOT-instance-of-cqt-2-def AOT-sem-denotes*
AOT-model-lambda-denotes AOT-sem-disj)
lemma *AOT-instance-of-cqt-2-intros-equib*[*AOT-instance-of-cqt-2-intro*]:
assumes ‹*AOT-instance-of-cqt-2* φ› **and** ‹*AOT-instance-of-cqt-2* ψ›
shows ‹*AOT-instance-of-cqt-2* ($\lambda\tau$. «$\varphi\{\tau\} \equiv \psi\{\tau\}$»)›
using *assms*
by (*auto simp*: *AOT-instance-of-cqt-2-def AOT-sem-denotes*
AOT-model-lambda-denotes AOT-sem-equiv)
lemma *AOT-instance-of-cqt-2-intros-forall*[*AOT-instance-of-cqt-2-intro*]:
assumes ‹$\bigwedge$ α . *AOT-instance-of-cqt-2* (Φ α)›
shows ‹*AOT-instance-of-cqt-2* ($\lambda\tau$. «$\forall\,\alpha\ \Phi\{\alpha,\tau\}$»)›
using *assms*
by (*auto simp*: *AOT-instance-of-cqt-2-def AOT-sem-denotes*
AOT-model-lambda-denotes AOT-sem-forall)
lemma *AOT-instance-of-cqt-2-intros-exists*[*AOT-instance-of-cqt-2-intro*]:
assumes ‹$\bigwedge$ α . *AOT-instance-of-cqt-2* (Φ α)›
shows ‹*AOT-instance-of-cqt-2* ($\lambda\tau$. «$\exists\,\alpha\ \Phi\{\alpha,\tau\}$»)›
using *assms*
by (*auto simp*: *AOT-instance-of-cqt-2-def AOT-sem-denotes*
AOT-model-lambda-denotes AOT-sem-exists)
lemma *AOT-instance-of-cqt-2-intros-exe-arg-self*[*AOT-instance-of-cqt-2-intro*]:
‹*AOT-instance-of-cqt-2-exe-arg* ($\lambda x.\ x$)›
unfolding *AOT-instance-of-cqt-2-exe-arg-def AOT-instance-of-cqt-2-def*
AOT-sem-lambda-denotes

```
  by (auto simp: AOT-model-term-equiv-part-equivp equivp-reflp AOT-sem-denotes)
lemma AOT-instance-of-cqt-2-intros-exe-arg-const[AOT-instance-of-cqt-2-intro]:
    ‹AOT-instance-of-cqt-2-exe-arg (λx. κ)›
  unfolding AOT-instance-of-cqt-2-exe-arg-def AOT-instance-of-cqt-2-def
  by (auto simp: AOT-model-term-equiv-part-equivp equivp-reflp
              AOT-sem-denotes AOT-sem-lambda-denotes)
lemma AOT-instance-of-cqt-2-intros-exe-arg-fst[AOT-instance-of-cqt-2-intro]:
  ‹AOT-instance-of-cqt-2-exe-arg fst›
  unfolding AOT-instance-of-cqt-2-exe-arg-def AOT-instance-of-cqt-2-def
  by (simp add: AOT-model-term-equiv-prod-def case-prod-beta)
lemma AOT-instance-of-cqt-2-intros-exe-arg-snd[AOT-instance-of-cqt-2-intro]:
  ‹AOT-instance-of-cqt-2-exe-arg snd›
  unfolding AOT-instance-of-cqt-2-exe-arg-def AOT-instance-of-cqt-2-def
  by (simp add: AOT-model-term-equiv-prod-def AOT-sem-denotes AOT-sem-lambda-denotes)
lemma AOT-instance-of-cqt-2-intros-exe-arg-Pair[AOT-instance-of-cqt-2-intro]:
  assumes ‹AOT-instance-of-cqt-2-exe-arg φ› and ‹AOT-instance-of-cqt-2-exe-arg ψ›
  shows ‹AOT-instance-of-cqt-2-exe-arg (λτ. Pair (φ τ) (ψ τ))›
  using assms
  unfolding AOT-instance-of-cqt-2-exe-arg-def AOT-instance-of-cqt-2-def
          AOT-sem-denotes AOT-sem-lambda-denotes AOT-model-term-equiv-prod-def
          AOT-model-denotes-prod-def
  by auto
lemma AOT-instance-of-cqt-2-intros-desc[AOT-instance-of-cqt-2-intro]:
  assumes ‹⋀z :: 'a::AOT-κ. AOT-instance-of-cqt-2 (Φ z)›
  shows ‹AOT-instance-of-cqt-2-exe-arg (λ κ :: 'b::AOT-κ . «ιz(Φ{z,κ})»)›
proof -
  have 0: ‹⋀ κ κ'. AOT-model-denotes κ ∧ AOT-model-denotes κ' ∧
                AOT-model-term-equiv κ κ' ⟹
                Φ z κ = Φ z κ'› for z
    using assms
    unfolding AOT-instance-of-cqt-2-def
            AOT-sem-denotes AOT-model-lambda-denotes by force
  {
    fix κ κ'
    have ‹«ιz(Φ{z,κ})» = «ιz(Φ{z,κ'})»›
      if ‹AOT-model-denotes κ ∧ AOT-model-denotes κ' ∧ AOT-model-term-equiv κ κ'›
      using 0[OF that]
      by auto
    moreover have ‹AOT-model-term-equiv x x› for x :: ‹'a::AOT-κ›
      by (metis AOT-instance-of-cqt-2-exe-arg-def
               AOT-instance-of-cqt-2-intros-exe-arg-const
               AOT-model-A-objects AOT-model-term-equiv-denotes
               AOT-model-term-equiv-eps(1))
    ultimately have ‹AOT-model-term-equiv «ιz(Φ{z,κ})» «ιz(Φ{z,κ'})»›
      if ‹AOT-model-denotes κ ∧ AOT-model-denotes κ' ∧ AOT-model-term-equiv κ κ'›
      using that by simp
  }
  thus ?thesis using 0
    unfolding AOT-instance-of-cqt-2-exe-arg-def
    by simp
qed

lemma AOT-instance-of-cqt-2-intros-exe-const[AOT-instance-of-cqt-2-intro]:
  assumes ‹AOT-instance-of-cqt-2-exe-arg κs›
  shows ‹AOT-instance-of-cqt-2 (λx :: 'b::AOT-κs. AOT-exe Π (κs x))›
  using assms
  unfolding AOT-instance-of-cqt-2-def AOT-sem-denotes AOT-model-lambda-denotes
          AOT-sem-disj AOT-sem-conj
          AOT-sem-not AOT-sem-box AOT-sem-act AOT-instance-of-cqt-2-exe-arg-def
          AOT-sem-equiv AOT-sem-imp AOT-sem-forall AOT-sem-exists AOT-sem-dia
  by (auto intro!: AOT-sem-exe-equiv)
lemma AOT-instance-of-cqt-2-intros-exe-lam[AOT-instance-of-cqt-2-intro]:
  assumes ‹⋀ y . AOT-instance-of-cqt-2 (λx. φ x y)›
```

and ‹*AOT-instance-of-cqt-2-exe-arg κs*›
shows ‹*AOT-instance-of-cqt-2* (λκ₁κₙ :: ′*b*::*AOT-κs*.
«[λν₁...νₙ φ{κ₁...κₙ,ν₁...νₙ}]«κs κ₁κₙ»»)›
proof −
{
fix *x y* :: ′*b*
assume ‹*AOT-model-denotes x*›
moreover assume ‹*AOT-model-denotes y*›
moreover assume ‹*AOT-model-term-equiv x y*›
moreover have *1*: ‹φ *x* = φ *y*›
using *assms calculation* **unfolding** *AOT-instance-of-cqt-2-def* **by** *blast*
ultimately have ‹*AOT-exe* (*AOT-lambda* (φ *x*)) (κ*s x*) =
AOT-exe (*AOT-lambda* (φ *y*)) (κ*s y*)›
unfolding *1*
apply (*safe intro*!: *AOT-sem-exe-equiv*)
by (*metis AOT-instance-of-cqt-2-exe-arg-def assms*(*2*))
}
thus *?thesis*
unfolding *AOT-instance-of-cqt-2-def*
AOT-instance-of-cqt-2-exe-arg-def
by *blast*
qed
lemma *AOT-instance-of-cqt-2-intro-prod*[*AOT-instance-of-cqt-2-intro*]:
assumes ‹⋀ *x* . *AOT-instance-of-cqt-2* (φ *x*)›
and ‹⋀ *x* . *AOT-instance-of-cqt-2* (λ *z* . φ *z x*)›
shows ‹*AOT-instance-of-cqt-2* (λ(*x*,*y*) . φ *x y*)›
using *assms* **unfolding** *AOT-instance-of-cqt-2-def*
by (*auto simp add*: *AOT-model-lambda-denotes AOT-sem-denotes*
AOT-model-denotes-prod-def
AOT-model-term-equiv-prod-def)

The following are already derivable semantically, but not yet added to *AOT-instance-of-cqt-2-intro*. They will be added with the next planned extension of axiom cqt:2.

named-theorems *AOT-instance-of-cqt-2-intro-next*
definition *AOT-instance-of-cqt-2-enc-arg* :: ‹(′*a*::*AOT-κs* ⇒ ′*b*::*AOT-κs*) ⇒ *bool*› **where**
‹*AOT-instance-of-cqt-2-enc-arg* ≡ λ φ . ∀ *x y z* .
AOT-model-denotes x ∧ *AOT-model-denotes y* ∧ *AOT-model-term-equiv x y* ⟶
AOT-enc (φ *x*) *z* = *AOT-enc* (φ *y*) *z*›
definition *AOT-instance-of-cqt-2-enc-rel* :: ‹(′*a*::*AOT-κs* ⇒ <′*b*::*AOT-κs*>) ⇒ *bool*› **where**
‹*AOT-instance-of-cqt-2-enc-rel* ≡ λ φ . ∀ *x y z* .
AOT-model-denotes x ∧ *AOT-model-denotes y* ∧ *AOT-model-term-equiv x y* ⟶
AOT-enc z (φ *x*) = *AOT-enc z* (φ *y*)›
lemma *AOT-instance-of-cqt-2-intros-enc*[*AOT-instance-of-cqt-2-intro-next*]:
assumes ‹*AOT-instance-of-cqt-2-enc-rel* Π› **and** ‹*AOT-instance-of-cqt-2-enc-arg κs*›
shows ‹*AOT-instance-of-cqt-2* (λ*x* . *AOT-enc* (κ*s x*) «[«Π *x*»]»)›
using *assms*
unfolding *AOT-instance-of-cqt-2-def AOT-sem-denotes AOT-model-lambda-denotes*
AOT-instance-of-cqt-2-enc-rel-def AOT-sem-not AOT-sem-box AOT-sem-act
AOT-sem-dia AOT-sem-conj AOT-sem-disj AOT-sem-equiv AOT-sem-imp
AOT-sem-forall AOT-sem-exists AOT-instance-of-cqt-2-enc-arg-def
by *fastforce*+
lemma *AOT-instance-of-cqt-2-enc-arg-intro-const*[*AOT-instance-of-cqt-2-intro-next*]:
‹*AOT-instance-of-cqt-2-enc-arg* (λ*x*. *c*)›
unfolding *AOT-instance-of-cqt-2-enc-arg-def* **by** *simp*
lemma *AOT-instance-of-cqt-2-enc-arg-intro-desc*[*AOT-instance-of-cqt-2-intro-next*]:
assumes ‹⋀*z* :: ′*a*::*AOT-κ*. *AOT-instance-of-cqt-2* (Φ *z*)›
shows ‹*AOT-instance-of-cqt-2-enc-arg* (λ κ :: ′*b*::*AOT-κ* . «ι*z*(Φ{*z*,κ})»)›
proof −
have *0*: ‹⋀ κ κ′. *AOT-model-denotes* κ ∧ *AOT-model-denotes* κ′ ∧
AOT-model-term-equiv κ κ′ ⟹
Φ *z* κ = Φ *z* κ′› **for** *z*
using *assms*
unfolding *AOT-instance-of-cqt-2-def*

AOT-sem-denotes AOT-model-lambda-denotes **by** *force*
{
fix *κ κ'*
have ‹«*ιz*(Φ{*z*,*κ*})» = «*ιz*(Φ{*z*,*κ'*})»›
if ‹*AOT-model-denotes κ* ∧ *AOT-model-denotes κ'* ∧ *AOT-model-term-equiv κ κ'*›
using *0*[*OF that*]
by *auto*
}
thus *?thesis* **using** *0*
unfolding *AOT-instance-of-cqt-2-enc-arg-def* **by** *meson*
qed
lemma *AOT-instance-of-cqt-2-enc-rel-intro*[*AOT-instance-of-cqt-2-intro-next*]:
assumes ‹⋀ *κ* . *AOT-instance-of-cqt-2* (*λκ'* :: *'b*::*AOT-κs* . *φ κ κ'*)›
assumes ‹⋀ *κ'* . *AOT-instance-of-cqt-2* (*λκ* :: *'a*::*AOT-κs* . *φ κ κ'*)›
shows ‹*AOT-instance-of-cqt-2-enc-rel* (*λκ* :: *'a*::*AOT-κs*. *AOT-lambda* (*λκ'*. *φ κ κ'*))›
proof −
{
fix *x y* :: *'a* **and** *z* ::*'b*
assume ‹*AOT-model-term-equiv x y*›
moreover assume ‹*AOT-model-denotes x*›
moreover assume ‹*AOT-model-denotes y*›
ultimately have ‹*φ x* = *φ y*›
using *assms* **unfolding** *AOT-instance-of-cqt-2-def* **by** *blast*
hence ‹*AOT-enc z* (*AOT-lambda* (*φ x*)) = *AOT-enc z* (*AOT-lambda* (*φ y*))›
by *simp*
}
thus *?thesis*
unfolding *AOT-instance-of-cqt-2-enc-rel-def* **by** *auto*
qed

Further restrict unary individual variables to type *κ* (rather than class *AOT-κ* only) and define being ordinary and being abstract.

AOT-register-type-constraints
Individual: ‹*κ*› ‹-::*AOT-κs*›

AOT-define *AOT-ordinary* :: ‹Π› (‹*O*!›) ‹*O*! $=_{df}$ [*λx* ◇*E*!*x*]›
declare *AOT-ordinary*[*AOT del*, *AOT-defs del*]
AOT-define *AOT-abstract* :: ‹Π› (‹*A*!›) ‹*A*! $=_{df}$ [*λx* ¬◇*E*!*x*]›
declare *AOT-abstract*[*AOT del*, *AOT-defs del*]

context *AOT-meta-syntax*
begin
notation *AOT-ordinary* (‹**O!**›)
notation *AOT-abstract* (‹**A!**›)
end
context *AOT-no-meta-syntax*
begin
no-notation *AOT-ordinary* (‹**O!**›)
no-notation *AOT-abstract* (‹**A!**›)
end

no-translations
-AOT-concrete => *CONST AOT-term-of-var* (*CONST AOT-concrete*)
parse-translation‹
[(***syntax-const***‹*-AOT-concrete*›, *fn* - => *fn* [] =>
Const (***const-name***‹*AOT-term-of-var*›, *dummyT*)
$ *Const* (***const-name***‹*AOT-concrete*›, ***typ***‹<*κ*> *AOT-var*›))]
›

Auxiliary lemmata.

lemma *AOT-sem-ordinary*: «*O*!» = «[*λx* ◇*E*!*x*]»
using *AOT-ordinary*[*THEN AOT-sem-id-def0E1*] *AOT-sem-ordinary-def-denotes*
by (*auto simp*: *AOT-sem-eq*)

```
lemma AOT-sem-abstract: «A!» = «[λx ¬◇E!x]»
  using AOT-abstract[THEN AOT-sem-id-def0E1]  AOT-sem-abstract-def-denotes
  by (auto simp: AOT-sem-eq)
lemma AOT-sem-ordinary-denotes: ‹[w ⊨ O!↓]›
  by (simp add: AOT-sem-ordinary AOT-sem-ordinary-def-denotes)
lemma AOT-meta-abstract-denotes: ‹[w ⊨ A!↓]›
  by (simp add: AOT-sem-abstract AOT-sem-abstract-def-denotes)
lemma AOT-model-abstract-ακ: ‹∃ a . κ = ακ a› if ‹[v ⊨ A!κ]›
  using that[unfolded AOT-sem-abstract, simplified
      AOT-meta-abstract-denotes[unfolded AOT-sem-abstract, THEN AOT-sem-lambda-beta,
          OF that[simplified AOT-sem-exe, THEN conjunct2, THEN conjunct1]]]
  apply (simp add: AOT-sem-not AOT-sem-dia AOT-sem-concrete)
  by (metis AOT-model-ω-concrete-in-some-world AOT-model-concrete-κ.simps(1)
            AOT-model-denotes-κ-def AOT-sem-denotes AOT-sem-exe κ.exhaust-disc
            is-ακ-def is-ωκ-def that)
lemma AOT-model-ordinary-ωκ: ‹∃ a . κ = ωκ a› if ‹[v ⊨ O!κ]›
  using that[unfolded AOT-sem-ordinary, simplified
      AOT-sem-ordinary-denotes[unfolded AOT-sem-ordinary, THEN AOT-sem-lambda-beta,
        OF that[simplified AOT-sem-exe, THEN conjunct2, THEN conjunct1]]]
  apply (simp add: AOT-sem-dia AOT-sem-concrete)
  by (metis AOT-model-concrete-κ.simps(2) AOT-model-concrete-κ.simps(3)
            κ.exhaust-disc is-ακ-def is-ωκ-def is-nullκ-def)
lemma AOT-model-ωκ-ordinary: ‹[v ⊨ O!«ωκ x»]›
  by (metis AOT-model-abstract-ακ AOT-model-denotes-κ-def AOT-sem-abstract
            AOT-sem-denotes AOT-sem-ind-eq AOT-sem-ordinary κ.disc(7) κ.distinct(1))
lemma AOT-model-ακ-ordinary: ‹[v ⊨ A!«ακ x»]›
  by (metis AOT-model-denotes-κ-def AOT-model-ordinary-ωκ AOT-sem-abstract
            AOT-sem-denotes AOT-sem-ind-eq AOT-sem-ordinary κ.disc(8) κ.distinct(1))
AOT-theorem prod-denotesE: assumes ‹«(κ1,κ2)»↓› shows ‹κ1↓ & κ2↓›
  using assms by (simp add: AOT-sem-denotes AOT-sem-conj AOT-model-denotes-prod-def)
declare prod-denotesE[AOT del]
AOT-theorem prod-denotesI: assumes ‹κ1↓ & κ2↓› shows ‹«(κ1,κ2)»↓›
  using assms by (simp add: AOT-sem-denotes AOT-sem-conj AOT-model-denotes-prod-def)
declare prod-denotesI[AOT del]
```

Prepare the derivation of the additional axioms that are validated by our extended models.

```
locale AOT-ExtendedModel =
  assumes AOT-ExtendedModel: ‹AOT-ExtendedModel›
begin
lemma AOT-sem-indistinguishable-ord-enc-all:
  assumes Π-den: ‹[v ⊨ Π↓]›
  assumes Ax: ‹[v ⊨ A!x]›
  assumes Ay: ‹[v ⊨ A!y]›
  assumes indist: ‹[v ⊨ ∀ F □([F]x ≡ [F]y)]›
  shows
  ‹[v ⊨ ∀ G(∀ z(O!z → □([G]z ≡ [Π]z)) → x[G])] =
   [v ⊨ ∀ G(∀ z(O!z → □([G]z ≡ [Π]z)) → y[G])]›
proof –
    have 0: ‹[v ⊨ [λx ¬◇[E!]x]x]›
      using Ax by (simp add: AOT-sem-abstract)
    have 1: ‹[v ⊨ [λx ¬◇[E!]x]y]›
      using Ay by (simp add: AOT-sem-abstract)
    {
      assume ‹[v ⊨ ∀ G(∀ z (O!z → □([G]z ≡ [Π]z)) → x[G])]›
      hence 3: ‹[v ⊨ ∀ G(∀ z([λx ◇[E!]x]z → □([G]z ≡ [Π]z)) → x[G])]›
        by (simp add: AOT-sem-ordinary)
      {
        fix Π' :: ‹<κ>›
        assume 1: ‹[v ⊨ Π'↓]›
        assume 2: ‹[v ⊨ [λx ◇[E!]x]z → □([Π']z ≡ [Π]z)]› for z
        have ‹[v ⊨ x[Π']]›
          using 3
          by (auto simp: AOT-sem-forall AOT-sem-imp AOT-sem-box AOT-sem-denotes)
```

(*metis* (*no-types*, *lifting*) *1 2 AOT-term-of-var-cases AOT-sem-box*
AOT-sem-denotes AOT-sem-imp)
} **note** *3* = *this*
fix Π′ :: ‹<κ>›
assume Π-*den*: ‹[*v* ⊨ Π′↓]›
assume *4*: ‹[*v* ⊨ ∀ *z* (*O*!*z* → □([Π′]*z* ≡ [Π]*z*))]›
{
fix κ_0
assume ‹[*v* ⊨ [λ*x* ◊[*E*!]*x*]κ_0]›
hence ‹[*v* ⊨ *O*!κ_0]›
using *AOT-sem-ordinary* **by** *metis*
moreover have ‹[*v* ⊨ κ_0↓]›
using *calculation* **by** (*simp add*: *AOT-sem-exe*)
ultimately have ‹[*v* ⊨ □([Π′]κ_0 ≡ [Π]κ_0)]›
using *4* **by** (*auto simp*: *AOT-sem-forall AOT-sem-imp*)
} **note** *4* = *this*
{
fix Π′′ :: ‹<κ>›
assume ‹[*v* ⊨ Π′′↓]›
moreover assume ‹[*w* ⊨ [Π′′]κ_0] = [*w* ⊨ [Π′]κ_0]› **if** ‹[*v* ⊨ [λ*x* ◊*E*!*x*]κ_0]› **for** κ_0 *w*
ultimately have *5*: ‹[*v* ⊨ *x*[Π′′]]›
using *4 3*
by (*auto simp*: *AOT-sem-imp AOT-sem-equiv AOT-sem-box*)
} **note** *5* = *this*
have ‹[*v* ⊨ *y*[Π′]]›
apply (*rule AOT-sem-enc-indistinguishable-all*[*OF AOT-ExtendedModel*])
apply (*fact 0*)
by (*auto simp*: *5 0 1* Π-*den indist*[*simplified AOT-sem-forall*
AOT-sem-box AOT-sem-equiv])
}
moreover {
{
assume ‹[*v* ⊨ ∀ *G*(∀ *z* (*O*!*z* → □([*G*]*z* ≡ [Π]*z*)) → *y*[*G*])]›
hence *3*: ‹[*v* ⊨ ∀ *G*(∀ *z* ([λ*x* ◊[*E*!]*x*]*z* → □([*G*]*z* ≡ [Π]*z*)) → *y*[*G*])]›
by (*simp add*: *AOT-sem-ordinary*)
{
fix Π′ :: ‹<κ>›
assume *1*: ‹[*v* ⊨ Π′↓]›
assume *2*: ‹[*v* ⊨ [λ*x* ◊[*E*!]*x*]*z* → □([Π′]*z* ≡ [Π]*z*)]› **for** *z*
have ‹[*v* ⊨ *y*[Π′]]›
using *3*
apply (*simp add*: *AOT-sem-forall AOT-sem-imp AOT-sem-box AOT-sem-denotes*)
by (*metis* (*no-types*, *lifting*) *1 2 AOT-model.AOT-term-of-var-cases*
AOT-sem-box AOT-sem-denotes AOT-sem-imp)
} **note** *3* = *this*
fix Π′ :: ‹<κ>›
assume Π-*den*: ‹[*v* ⊨ Π′↓]›
assume *4*: ‹[*v* ⊨ ∀ *z* (*O*!*z* → □([Π′]*z* ≡ [Π]*z*))]›
{
fix κ_0
assume ‹[*v* ⊨ [λ*x* ◊[*E*!]*x*]κ_0]›
hence ‹[*v* ⊨ *O*!κ_0]›
using *AOT-sem-ordinary* **by** *metis*
moreover have ‹[*v* ⊨ κ_0↓]›
using *calculation* **by** (*simp add*: *AOT-sem-exe*)
ultimately have ‹[*v* ⊨ □([Π′]κ_0 ≡ [Π]κ_0)]›
using *4* **by** (*auto simp*: *AOT-sem-forall AOT-sem-imp*)
} **note** *4* = *this*
{
fix Π′′ :: ‹<κ>›
assume ‹[*v* ⊨ Π′′↓]›
moreover assume ‹[*w* ⊨ [Π′′]κ_0] = [*w* ⊨ [Π′]κ_0]› **if** ‹[*v* ⊨ [λ*x* ◊*E*!*x*]κ_0]› **for** *w* κ_0
ultimately have ‹[*v* ⊨ *y*[Π′′]]›

```
        using 3 4 by (auto simp: AOT-sem-imp AOT-sem-equiv AOT-sem-box)
      } note 5 = this
      have ‹[v ⊨ x[Π']]›
        apply (rule AOT-sem-enc-indistinguishable-all[OF AOT-ExtendedModel])
            apply (fact 1)
        by (auto simp: 5 0 1 Π-den indist[simplified AOT-sem-forall
                    AOT-sem-box AOT-sem-equiv])
    }
  }
  ultimately show ‹[v ⊨ ∀ G (∀ z (O!z → □([G]z ≡ [Π]z)) → x[G])] =
        [v ⊨ ∀ G (∀ z (O!z → □([G]z ≡ [Π]z)) → y[G])]›
    by (auto simp: AOT-sem-forall AOT-sem-imp)
qed

lemma AOT-sem-indistinguishable-ord-enc-ex:
  assumes Π-den: ‹[v ⊨ Π↓]›
  assumes Ax: ‹[v ⊨ A!x]›
  assumes Ay: ‹[v ⊨ A!y]›
  assumes indist: ‹[v ⊨ ∀ F □([F]x ≡ [F]y)]›
  shows ‹[v ⊨ ∃ G(∀ z (O!z → □([G]z ≡ [Π]z)) & x[G])] =
         [v ⊨ ∃ G(∀ z(O!z → □([G]z ≡ [Π]z)) & y[G])]›
proof -
  have Aux: ‹[v ⊨ [λx ◇[E!]x]κ] = ([v ⊨ [λx ◇[E!]x]κ] ∧ [v ⊨ κ↓])› for v κ
    using AOT-sem-exe by blast
  AOT-modally-strict {
    fix x y
    AOT-assume Π-den: ‹[Π]↓›
    AOT-assume 2: ‹∀ F □([F]x ≡ [F]y)›
    AOT-assume ‹A!x›
    AOT-hence 0: ‹[λx ¬◇[E!]x]x›
      by (simp add: AOT-sem-abstract)
    AOT-assume ‹A!y›
    AOT-hence 1: ‹[λx ¬◇[E!]x]y›
      by (simp add: AOT-sem-abstract)
    {
      AOT-assume ‹∃ G(∀ z (O!z → □([G]z ≡ [Π]z)) & x[G])›
      then AOT-obtain Π'
        where Π'-den: ‹Π'↓›
          and Π'-indist: ‹∀ z (O!z → □([Π']z ≡ [Π]z))›
          and x-enc-Π': ‹x[Π']›
        by (meson AOT-sem-conj AOT-sem-exists)
      {
        fix κ₀
        AOT-assume ‹[λx ◇[E!]x]κ₀›
        AOT-hence ‹□([Π']κ₀ ≡ [Π]κ₀)›
          using Π'-indist
          by (auto simp: AOT-sem-exe AOT-sem-imp AOT-sem-exists AOT-sem-conj
                    AOT-sem-ordinary AOT-sem-forall)
      } note 3 = this
      AOT-have ‹∀ z ([λx ◇[E!]x]z → □([Π']z ≡ [Π]z))›
        using Π'-indist by (simp add: AOT-sem-ordinary)
      AOT-obtain Π'' where
          Π''-den: ‹Π''↓› and
          Π''-indist: ‹[λx ◇[E!]x]κ₀ → □([Π'']κ₀ ≡ [Π]κ₀)› and
          y-enc-Π'': ‹y[Π'']› for κ₀
        using AOT-sem-enc-indistinguishable-ex[OF AOT-ExtendedModel,
              OF 0, OF 1, rotated, OF Π-den,
              OF exI[where x=Π'], OF conjI, OF Π'-den, OF conjI,
              OF x-enc-Π', OF allI, OF impI,
              OF 3[simplified AOT-sem-box AOT-sem-equiv], simplified, OF
              2[simplified AOT-sem-forall AOT-sem-equiv AOT-sem-box,
                THEN spec, THEN mp, THEN spec], simplified]
        unfolding AOT-sem-imp AOT-sem-box AOT-sem-equiv by blast
```

```
      {
        AOT-have ‹Π''↓›
            and ‹∀ x ([λx ◇[E!]x]x → □([Π'']x ≡ [Π]x))›
            and ‹y[Π'']›
          apply (simp add: Π''-den)
          apply (simp add: AOT-sem-forall Π''-indist)
          by (simp add: y-enc-Π'')
      } note 2 = this
      AOT-have ‹∃ G(∀ z (O!z → □([G]z ≡ [Π]z)) & y[G])›
        apply (simp add: AOT-sem-exists AOT-sem-ordinary
            AOT-sem-imp AOT-sem-box AOT-sem-forall AOT-sem-equiv AOT-sem-conj)
        using 2[simplified AOT-sem-box AOT-sem-equiv AOT-sem-imp AOT-sem-forall]
        by blast
    }
  } note 0 = this
  AOT-modally-strict {
    {
      fix x y
      AOT-assume Π-den: ‹[Π]↓›
      moreover AOT-assume ‹∀ F □([F]x ≡ [F]y)›
      moreover AOT-have ‹∀ F □([F]y ≡ [F]x)›
        using calculation(2)
        by (auto simp: AOT-sem-forall AOT-sem-box AOT-sem-equiv)
      moreover AOT-assume ‹A!x›
      moreover AOT-assume ‹A!y›
      ultimately AOT-have ‹∃ G (∀ z (O!z → □([G]z ≡ [Π]z)) & x[G]) ≡
                          ∃ G (∀ z (O!z → □([G]z ≡ [Π]z)) & y[G])›
        using 0 by (auto simp: AOT-sem-equiv)
    }
    have 1: ‹[v ⊨ ∀ F □([F]y ≡ [F]x)]›
      using indist
      by (auto simp: AOT-sem-forall AOT-sem-box AOT-sem-equiv)
    thus ‹[v ⊨ ∃ G (∀ z (O!z → □([G]z ≡ [Π]z)) & x[G])] =
        [v ⊨ ∃ G (∀ z (O!z → □([G]z ≡ [Π]z)) & y[G])]›
      using assms
      by (auto simp: AOT-sem-imp AOT-sem-conj AOT-sem-equiv 0)
  }
qed
end


setup‹setup-AOT-no-atp›
bundle AOT-no-atp begin declare AOT-no-atp[no-atp] end


theory AOT-Definitions
  imports AOT-semantics
begin
```

6 Definitions of AOT

```
AOT-theorem conventions:1: ‹φ & ψ ≡df ¬(φ → ¬ψ)›
  using AOT-conj.
AOT-theorem conventions:2: ‹φ ∨ ψ ≡df ¬φ → ψ›
  using AOT-disj.
AOT-theorem conventions:3: ‹φ ≡ ψ ≡df (φ → ψ) & (ψ → φ)›
  using AOT-equiv.
AOT-theorem conventions:4: ‹∃ α φ{α} ≡df ¬∀ α ¬φ{α}›
  using AOT-exists.
AOT-theorem conventions:5: ‹◇φ ≡df ¬□¬φ›
  using AOT-dia.
```

declare *conventions*:*1*[*AOT-defs*] *conventions*:*2*[*AOT-defs*]
conventions:*3*[*AOT-defs*] *conventions*:*4*[*AOT-defs*]
conventions:*5*[*AOT-defs*]

notepad
begin
fix $\varphi\ \psi\ \chi$

have *conventions3*[*1*]: ‹«$\varphi \rightarrow \psi \equiv \neg\psi \rightarrow \neg\varphi$» = «$(\varphi \rightarrow \psi) \equiv (\neg\psi \rightarrow \neg\varphi)$»›
by *blast*
have *conventions3*[*2*]: ‹«$\varphi\ \&\ \psi \rightarrow \chi$» = «$(\varphi\ \&\ \psi) \rightarrow \chi$»›
and ‹«$\varphi \vee \psi \rightarrow \chi$» = «$(\varphi \vee \psi) \rightarrow \chi$»›
by *blast*+
have *conventions3*[*3*]: ‹«$\varphi \vee \psi\ \&\ \chi$» = «$(\varphi \vee \psi)\ \&\ \chi$»›
and ‹«$\varphi\ \&\ \psi \vee \chi$» = «$(\varphi\ \&\ \psi) \vee \chi$»›
by *blast*+ — Note that PLM instead generally uses parenthesis in these cases.
end

AOT-theorem *existence*:*1*: ‹$\kappa\downarrow \equiv_{df} \exists F\ [F]\kappa$›
by (*simp add*: *AOT-sem-denotes AOT-sem-exists AOT-model-equiv-def*)
(*metis AOT-sem-denotes AOT-sem-exe AOT-sem-lambda-beta AOT-sem-lambda-denotes*)
AOT-theorem *existence*:*2*: ‹$\Pi\downarrow \equiv_{df} \exists x_1 ... \exists x_n\ x_1 ... x_n[\Pi]$›
using *AOT-sem-denotes AOT-sem-enc-denotes AOT-sem-universal-encoder*
by (*simp add*: *AOT-sem-denotes AOT-sem-exists AOT-model-equiv-def*) *blast*
AOT-theorem *existence*:*2*[*1*]: ‹$\Pi\downarrow \equiv_{df} \exists x\ x[\Pi]$›
using *existence*:*2*[*of* Π] **by** *simp*
AOT-theorem *existence*:*2*[*2*]: ‹$\Pi\downarrow \equiv_{df} \exists x \exists y\ xy[\Pi]$›
using *existence*:*2*[*of* Π]
by (*simp add*: *AOT-sem-denotes AOT-sem-exists AOT-model-equiv-def*
AOT-model-denotes-prod-def)
AOT-theorem *existence*:*2*[*3*]: ‹$\Pi\downarrow \equiv_{df} \exists x \exists y \exists z\ xyz[\Pi]$›
using *existence*:*2*[*of* Π]
by (*simp add*: *AOT-sem-denotes AOT-sem-exists AOT-model-equiv-def*
AOT-model-denotes-prod-def)
AOT-theorem *existence*:*2*[*4*]: ‹$\Pi\downarrow \equiv_{df} \exists x_1 \exists x_2 \exists x_3 \exists x_4\ x_1x_2x_3x_4[\Pi]$›
using *existence*:*2*[*of* Π]
by (*simp add*: *AOT-sem-denotes AOT-sem-exists AOT-model-equiv-def*
AOT-model-denotes-prod-def)

AOT-theorem *existence*:*3*: ‹$\varphi\downarrow \equiv_{df} [\lambda x\ \varphi]\downarrow$›
by (*simp add*: *AOT-sem-denotes AOT-model-denotes-*o*-def AOT-model-equiv-def*
AOT-model-lambda-denotes)

declare *existence*:*1*[*AOT-defs*] *existence*:*2*[*AOT-defs*] *existence*:*2*[*1*][*AOT-defs*]
existence:*2*[*2*][*AOT-defs*] *existence*:*2*[*3*][*AOT-defs*]
existence:*2*[*4*][*AOT-defs*] *existence*:*3*[*AOT-defs*]

AOT-theorem *oa*:*1*: ‹$O! =_{df} [\lambda x\ \Diamond E!x]$› **using** *AOT-ordinary* **.**
AOT-theorem *oa*:*2*: ‹$A! =_{df} [\lambda x\ \neg\Diamond E!x]$› **using** *AOT-abstract* **.**

declare *oa*:*1*[*AOT-defs*] *oa*:*2*[*AOT-defs*]

AOT-theorem *identity*:*1*:
‹$x = y \equiv_{df} ([O!]x\ \&\ [O!]y\ \&\ \Box\forall F\ ([F]x \equiv [F]y)) \vee$
$([A!]x\ \&\ [A!]y\ \&\ \Box\forall F\ (x[F] \equiv y[F]))$›
unfolding *AOT-model-equiv-def*
using *AOT-sem-ind-eq*[*of* - *x y*]
by (*simp add*: *AOT-sem-ordinary AOT-sem-abstract AOT-sem-conj*
AOT-sem-box AOT-sem-equiv AOT-sem-forall AOT-sem-disj AOT-sem-eq
AOT-sem-denotes)

AOT-theorem *identity*:*2*:

‹$F = G \equiv_{df} F\downarrow$ & $G\downarrow$ & $\Box\forall x(x[F] \equiv x[G])$›
  using *AOT-sem-enc-eq*[*of - F G*]
  by (*auto simp*: *AOT-model-equiv-def AOT-sem-imp AOT-sem-denotes AOT-sem-eq AOT-sem-conj AOT-sem-forall AOT-sem-box AOT-sem-equiv*)

AOT-theorem *identity*:*3*[*2*]:
‹$F = G \equiv_{df} F\downarrow$ & $G\downarrow$ & $\forall y([\lambda z\ [F]zy] = [\lambda z\ [G]zy]$ & $[\lambda z\ [F]yz] = [\lambda z\ [G]yz])$›
  by (*auto simp*: *AOT-model-equiv-def AOT-sem-proj-id-prop*[*of - F G*] *AOT-sem-proj-id-prod-def AOT-sem-conj AOT-sem-denotes AOT-sem-forall AOT-sem-unary-proj-id AOT-model-denotes-prod-def*)
AOT-theorem *identity*:*3*[*3*]:
‹$F = G \equiv_{df} F\downarrow$ & $G\downarrow$ & $\forall y_1\forall y_2([\lambda z\ [F]zy_1y_2] = [\lambda z\ [G]zy_1y_2]$ &
$[\lambda z\ [F]y_1zy_2] = [\lambda z\ [G]y_1zy_2]$ &
$[\lambda z\ [F]y_1y_2z] = [\lambda z\ [G]y_1y_2z])$›
  by (*auto simp*: *AOT-model-equiv-def AOT-sem-proj-id-prop*[*of - F G*] *AOT-sem-proj-id-prod-def AOT-sem-conj AOT-sem-denotes AOT-sem-forall AOT-sem-unary-proj-id AOT-model-denotes-prod-def*)
AOT-theorem *identity*:*3*[*4*]:
‹$F = G \equiv_{df} F\downarrow$ & $G\downarrow$ & $\forall y_1\forall y_2\forall y_3([\lambda z\ [F]zy_1y_2y_3] = [\lambda z\ [G]zy_1y_2y_3]$ &
$[\lambda z\ [F]y_1zy_2y_3] = [\lambda z\ [G]y_1zy_2y_3]$ &
$[\lambda z\ [F]y_1y_2zy_3] = [\lambda z\ [G]y_1y_2zy_3]$ &
$[\lambda z\ [F]y_1y_2y_3z] = [\lambda z\ [G]y_1y_2y_3z])$›
  by (*auto simp*: *AOT-model-equiv-def AOT-sem-proj-id-prop*[*of - F G*] *AOT-sem-proj-id-prod-def AOT-sem-conj AOT-sem-denotes AOT-sem-forall AOT-sem-unary-proj-id AOT-model-denotes-prod-def*)
AOT-theorem *identity*:*3*:
‹$F = G \equiv_{df} F\downarrow$ & $G\downarrow$ & $\forall x_1...\forall x_n$ «*AOT-sem-proj-id* x_1x_n ($\lambda\ \tau$. *AOT-exe F* τ)
($\lambda\ \tau$. *AOT-exe G* τ)»›
  by (*auto simp*: *AOT-model-equiv-def AOT-sem-proj-id-prop*[*of - F G*] *AOT-sem-proj-id-prod-def AOT-sem-conj AOT-sem-denotes AOT-sem-forall AOT-sem-unary-proj-id AOT-model-denotes-prod-def*)

AOT-theorem *identity*:*4*:
‹$p = q \equiv_{df} p\downarrow$ & $q\downarrow$ & $[\lambda x\ p] = [\lambda x\ q]$›
  by (*auto simp*: *AOT-model-equiv-def AOT-sem-eq AOT-sem-denotes AOT-sem-conj AOT-model-lambda-denotes AOT-sem-lambda-eq-prop-eq*)

declare *identity*:*1*[*AOT-defs*] *identity*:*2*[*AOT-defs*] *identity*:*3*[*2*][*AOT-defs*] *identity*:*3*[*3*][*AOT-defs*] *identity*:*3*[*4*][*AOT-defs*] *identity*:*3*[*AOT-defs*] *identity*:*4*[*AOT-defs*]

AOT-define *AOT-nonidentical* :: ‹$\tau \Rightarrow \tau \Rightarrow \varphi$› (**infixl** ‹$\neq$› *50*)
  $=-$*infix*: ‹$\tau \neq \sigma \equiv_{df} \neg(\tau = \sigma)$›

context *AOT-meta-syntax*
begin
notation *AOT-nonidentical* (**infixl** ‹$\neq$› *50*)
end
context *AOT-no-meta-syntax*
begin
no-notation *AOT-nonidentical* (**infixl** ‹$\neq$› *50*)
end

The following are purely technical pseudo-definitions required due to our internal implementation of n-ary relations and ellipses using tuples.

AOT-theorem *tuple-denotes*: ‹«(τ,τ')»$\downarrow \equiv_{df} \tau\downarrow$ & $\tau'\downarrow$›
  by (*simp add*: *AOT-model-denotes-prod-def AOT-model-equiv-def AOT-sem-conj AOT-sem-denotes*)
AOT-theorem *tuple-identity-1*: ‹«(τ,τ')» = «(σ, σ')» $\equiv_{df} (\tau = \sigma)$ & $(\tau' = \sigma')$›
  by (*auto simp*: *AOT-model-equiv-def AOT-sem-conj AOT-sem-eq AOT-model-denotes-prod-def AOT-sem-denotes*)
AOT-theorem *tuple-forall*: ‹$\forall \alpha_1...\forall \alpha_n\ \varphi\{\alpha_1...\alpha_n\} \equiv_{df} \forall \alpha_1(\forall \alpha_2...\forall \alpha_n\ \varphi\{$«$(\alpha_1, \alpha_2\alpha_n)$»$\})$›
  by (*auto simp*: *AOT-model-equiv-def AOT-sem-forall AOT-sem-denotes*

AOT-model-denotes-prod-def)
AOT-theorem *tuple-exists*: ‹$\exists\,\alpha_1 ... \exists\,\alpha_n\ \varphi\{\alpha_1 ... \alpha_n\} \equiv_{df} \exists\,\alpha_1 (\exists\,\alpha_2 ... \exists\,\alpha_n\ \varphi\{«(\alpha_1,\ \alpha_2 \alpha_n)»\})$›
by (*auto simp*: *AOT-model-equiv-def AOT-sem-exists AOT-sem-denotes AOT-model-denotes-prod-def*)
declare *tuple-denotes*[*AOT-defs*] *tuple-identity-1*[*AOT-defs*] *tuple-forall*[*AOT-defs*] *tuple-exists*[*AOT-defs*]

end

7 Axioms of PLM

AOT-axiom *pl*:*1*: ‹$\varphi \rightarrow (\psi \rightarrow \varphi)$›
by (*auto simp*: *AOT-sem-imp AOT-model-axiomI*)
AOT-axiom *pl*:*2*: ‹$(\varphi \rightarrow (\psi \rightarrow \chi)) \rightarrow ((\varphi \rightarrow \psi) \rightarrow (\varphi \rightarrow \chi))$›
by (*auto simp*: *AOT-sem-imp AOT-model-axiomI*)
AOT-axiom *pl*:*3*: ‹$(\neg\varphi \rightarrow \neg\psi) \rightarrow ((\neg\varphi \rightarrow \psi) \rightarrow \varphi)$›
by (*auto simp*: *AOT-sem-imp AOT-sem-not AOT-model-axiomI*)

AOT-axiom *cqt*:*1*: ‹$\forall\,\alpha\ \varphi\{\alpha\} \rightarrow (\tau\downarrow \rightarrow \varphi\{\tau\})$›
by (*auto simp*: *AOT-sem-denotes AOT-sem-forall AOT-sem-imp AOT-model-axiomI*)

AOT-axiom *cqt*:*2*[*const-var*]: ‹$\alpha\downarrow$›
using *AOT-sem-vars-denote* **by** (*rule AOT-model-axiomI*)
AOT-axiom *cqt*:*2*[*lambda*]:
assumes ‹*INSTANCE-OF-CQT-2*(φ)›
shows ‹$[\lambda\nu_1 ... \nu_n\ \varphi\{\nu_1 ... \nu_n\}]\downarrow$›
by (*auto intro*!: *AOT-model-axiomI AOT-sem-cqt-2*[*OF assms*])
AOT-axiom *cqt*:*2*[*lambda0*]:
shows ‹$[\lambda\ \varphi]\downarrow$›
by (*auto intro*!: *AOT-model-axiomI*
simp: *AOT-sem-lambda-denotes existence*:*3*[*unfolded AOT-model-equiv-def*])

AOT-axiom *cqt*:*3*: ‹$\forall\,\alpha\ (\varphi\{\alpha\} \rightarrow \psi\{\alpha\}) \rightarrow (\forall\,\alpha\ \varphi\{\alpha\} \rightarrow \forall\,\alpha\ \psi\{\alpha\})$›
by (*simp add*: *AOT-sem-forall AOT-sem-imp AOT-model-axiomI*)
AOT-axiom *cqt*:*4*: ‹$\varphi \rightarrow \forall\,\alpha\ \varphi$›
by (*simp add*: *AOT-sem-forall AOT-sem-imp AOT-model-axiomI*)
AOT-axiom *cqt*:*5*:*a*: ‹$[\Pi]\kappa_1 ... \kappa_n \rightarrow (\Pi\downarrow\ \&\ \kappa_1 ... \kappa_n\downarrow)$›
by (*simp add*: *AOT-sem-conj AOT-sem-denotes AOT-sem-exe AOT-sem-imp AOT-model-axiomI*)
AOT-axiom *cqt*:*5*:*a*[*1*]: ‹$[\Pi]\kappa \rightarrow (\Pi\downarrow\ \&\ \kappa\downarrow)$›
using *cqt*:*5*:*a AOT-model-axiomI* **by** *blast*
AOT-axiom *cqt*:*5*:*a*[*2*]: ‹$[\Pi]\kappa_1\kappa_2 \rightarrow (\Pi\downarrow\ \&\ \kappa_1\downarrow\ \&\ \kappa_2\downarrow)$›
by (*rule AOT-model-axiomI*)
(*metis AOT-model-denotes-prod-def AOT-sem-conj AOT-sem-denotes AOT-sem-exe AOT-sem-imp case-prodD*)
AOT-axiom *cqt*:*5*:*a*[*3*]: ‹$[\Pi]\kappa_1\kappa_2\kappa_3 \rightarrow (\Pi\downarrow\ \&\ \kappa_1\downarrow\ \&\ \kappa_2\downarrow\ \&\ \kappa_3\downarrow)$›
by (*rule AOT-model-axiomI*)
(*metis AOT-model-denotes-prod-def AOT-sem-conj AOT-sem-denotes AOT-sem-exe AOT-sem-imp case-prodD*)
AOT-axiom *cqt*:*5*:*a*[*4*]: ‹$[\Pi]\kappa_1\kappa_2\kappa_3\kappa_4 \rightarrow (\Pi\downarrow\ \&\ \kappa_1\downarrow\ \&\ \kappa_2\downarrow\ \&\ \kappa_3\downarrow\ \&\ \kappa_4\downarrow)$›
by (*rule AOT-model-axiomI*)
(*metis AOT-model-denotes-prod-def AOT-sem-conj AOT-sem-denotes AOT-sem-exe AOT-sem-imp case-prodD*)
AOT-axiom *cqt*:*5*:*b*: ‹$\kappa_1 ... \kappa_n[\Pi] \rightarrow (\Pi\downarrow\ \&\ \kappa_1 ... \kappa_n\downarrow)$›
using *AOT-sem-enc-denotes*
by (*auto intro*!: *AOT-model-axiomI simp*: *AOT-sem-conj AOT-sem-denotes AOT-sem-imp*)+
AOT-axiom *cqt*:*5*:*b*[*1*]: ‹$\kappa[\Pi] \rightarrow (\Pi\downarrow\ \&\ \kappa\downarrow)$›
using *cqt*:*5*:*b AOT-model-axiomI* **by** *blast*
AOT-axiom *cqt*:*5*:*b*[*2*]: ‹$\kappa_1\kappa_2[\Pi] \rightarrow (\Pi\downarrow\ \&\ \kappa_1\downarrow\ \&\ \kappa_2\downarrow)$›
by (*rule AOT-model-axiomI*)
(*metis AOT-model-denotes-prod-def AOT-sem-conj AOT-sem-denotes AOT-sem-enc-denotes AOT-sem-imp case-prodD*)
AOT-axiom *cqt*:*5*:*b*[*3*]: ‹$\kappa_1\kappa_2\kappa_3[\Pi] \rightarrow (\Pi\downarrow\ \&\ \kappa_1\downarrow\ \&\ \kappa_2\downarrow\ \&\ \kappa_3\downarrow)$›

by (*rule AOT-model-axiomI*)
  (*metis AOT-model-denotes-prod-def AOT-sem-conj AOT-sem-denotes*
    AOT-sem-enc-denotes AOT-sem-imp case-prodD)
AOT-axiom *cqt:5:b[4]*: ‹$\kappa_1\kappa_2\kappa_3\kappa_4[\Pi] \rightarrow (\Pi{\downarrow}\ \&\ \kappa_1{\downarrow}\ \&\ \kappa_2{\downarrow}\ \&\ \kappa_3{\downarrow}\ \&\ \kappa_4{\downarrow})$›
by (*rule AOT-model-axiomI*)
  (*metis AOT-model-denotes-prod-def AOT-sem-conj AOT-sem-denotes*
    AOT-sem-enc-denotes AOT-sem-imp case-prodD)

AOT-axiom *l−identity*: ‹$\alpha = \beta \rightarrow (\varphi\{\alpha\} \rightarrow \varphi\{\beta\})$›
by (*rule AOT-model-axiomI*)
  (*simp add*: *AOT-sem-eq AOT-sem-imp*)

AOT-act-axiom *logic−actual*: ‹$\boldsymbol{\mathcal{A}}\varphi \rightarrow \varphi$›
by (*rule AOT-model-act-axiomI*)
  (*simp add*: *AOT-sem-act AOT-sem-imp*)

AOT-axiom *logic−actual−nec:1*: ‹$\boldsymbol{\mathcal{A}}\neg\varphi \equiv \neg\boldsymbol{\mathcal{A}}\varphi$›
by (*rule AOT-model-axiomI*)
  (*simp add*: *AOT-sem-act AOT-sem-equiv AOT-sem-not*)
AOT-axiom *logic−actual−nec:2*: ‹$\boldsymbol{\mathcal{A}}(\varphi \rightarrow \psi) \equiv (\boldsymbol{\mathcal{A}}\varphi \rightarrow \boldsymbol{\mathcal{A}}\psi)$›
by (*rule AOT-model-axiomI*)
  (*simp add*: *AOT-sem-act AOT-sem-equiv AOT-sem-imp*)

AOT-axiom *logic−actual−nec:3*: ‹$\boldsymbol{\mathcal{A}}(\forall\,\alpha\ \varphi\{\alpha\}) \equiv \forall\,\alpha\ \boldsymbol{\mathcal{A}}\varphi\{\alpha\}$›
by (*rule AOT-model-axiomI*)
  (*simp add*: *AOT-sem-act AOT-sem-equiv AOT-sem-forall AOT-sem-denotes*)
AOT-axiom *logic−actual−nec:4*: ‹$\boldsymbol{\mathcal{A}}\varphi \equiv \boldsymbol{\mathcal{A}}\boldsymbol{\mathcal{A}}\varphi$›
by (*rule AOT-model-axiomI*)
  (*simp add*: *AOT-sem-act AOT-sem-equiv*)

AOT-axiom *qml:1*: ‹$\Box(\varphi \rightarrow \psi) \rightarrow (\Box\varphi \rightarrow \Box\psi)$›
by (*rule AOT-model-axiomI*)
  (*simp add*: *AOT-sem-box AOT-sem-imp*)
AOT-axiom *qml:2*: ‹$\Box\varphi \rightarrow \varphi$›
by (*rule AOT-model-axiomI*)
  (*simp add*: *AOT-sem-box AOT-sem-imp*)
AOT-axiom *qml:3*: ‹$\Diamond\varphi \rightarrow \Box\Diamond\varphi$›
by (*rule AOT-model-axiomI*)
  (*simp add*: *AOT-sem-box AOT-sem-dia AOT-sem-imp*)

AOT-axiom *qml:4*: ‹$\Diamond\exists\,x\ (E!x\ \&\ \neg\boldsymbol{\mathcal{A}}E!x)$›
using *AOT-sem-concrete AOT-model-contingent*
by (*auto intro*!: *AOT-model-axiomI*
    simp: *AOT-sem-box AOT-sem-dia AOT-sem-imp AOT-sem-exists*
      AOT-sem-denotes AOT-sem-conj AOT-sem-not AOT-sem-act
      AOT-sem-exe)+

AOT-axiom *qml−act:1*: ‹$\boldsymbol{\mathcal{A}}\varphi \rightarrow \Box\boldsymbol{\mathcal{A}}\varphi$›
by (*rule AOT-model-axiomI*)
  (*simp add*: *AOT-sem-act AOT-sem-box AOT-sem-imp*)
AOT-axiom *qml−act:2*: ‹$\Box\varphi \equiv \boldsymbol{\mathcal{A}}\Box\varphi$›
by (*rule AOT-model-axiomI*)
  (*simp add*: *AOT-sem-act AOT-sem-box AOT-sem-equiv*)

AOT-axiom *descriptions*: ‹$x = \iota x(\varphi\{x\}) \equiv \forall\,z(\boldsymbol{\mathcal{A}}\varphi\{z\} \equiv z = x)$›
proof (*rule AOT-model-axiomI*)
  AOT-modally-strict {
    AOT-show ‹$x = \iota x(\varphi\{x\}) \equiv \forall\,z(\boldsymbol{\mathcal{A}}\varphi\{z\} \equiv z = x)$›
      by (*induct*; *simp add*: *AOT-sem-equiv AOT-sem-forall AOT-sem-act AOT-sem-eq*)
        (*metis* (*no-types*, *opaque-lifting*) *AOT-sem-desc-denotes AOT-sem-desc-prop*
          AOT-sem-denotes)
  }
qed

AOT-axiom *lambda−predicates*:*1*:
‹$[\lambda\nu_1...\nu_n\ \varphi\{\nu_1...\nu_n\}]\downarrow \rightarrow [\lambda\nu_1...\nu_n\ \varphi\{\nu_1...\nu_n\}] = [\lambda\mu_1...\mu_n\ \varphi\{\mu_1...\mu_n\}]$›
by (*rule AOT-model-axiomI*)
(*simp add*: *AOT-sem-denotes AOT-sem-eq AOT-sem-imp*)
AOT-axiom *lambda−predicates*:*1*[*zero*]: ‹$[\lambda\ p]\downarrow \rightarrow [\lambda\ p] = [\lambda\ p]$›
by (*rule AOT-model-axiomI*)
(*simp add*: *AOT-sem-denotes AOT-sem-eq AOT-sem-imp*)
AOT-axiom *lambda−predicates*:*2*:
‹$[\lambda x_1...x_n\ \varphi\{x_1...x_n\}]\downarrow \rightarrow ([\lambda x_1...x_n\ \varphi\{x_1...x_n\}]x_1...x_n \equiv \varphi\{x_1...x_n\})$›
by (*rule AOT-model-axiomI*)
(*simp add*: *AOT-sem-equiv AOT-sem-imp AOT-sem-lambda-beta AOT-sem-vars-denote*)
AOT-axiom *lambda−predicates*:*3*: ‹$[\lambda x_1...x_n\ [F]x_1...x_n] = F$›
by (*rule AOT-model-axiomI*)
(*simp add*: *AOT-sem-lambda-eta AOT-sem-vars-denote*)
AOT-axiom *lambda−predicates*:*3*[*zero*]: ‹$[\lambda\ p] = p$›
by (*rule AOT-model-axiomI*)
(*simp add*: *AOT-sem-eq AOT-sem-lambda0 AOT-sem-vars-denote*)

AOT-axiom *safe−ext*:
‹$([\lambda\nu_1...\nu_n\ \varphi\{\nu_1...\nu_n\}]\downarrow\ \&\ \Box\forall\nu_1...\forall\nu_n\ (\varphi\{\nu_1...\nu_n\} \equiv \psi\{\nu_1...\nu_n\})) \rightarrow$
$[\lambda\nu_1...\nu_n\ \psi\{\nu_1...\nu_n\}]\downarrow$›
using *AOT-sem-lambda-coex*
by (*auto intro*!: *AOT-model-axiomI simp*: *AOT-sem-imp AOT-sem-denotes AOT-sem-conj AOT-sem-equiv AOT-sem-box AOT-sem-forall*)
AOT-axiom *safe−ext*[*2*]:
‹$([\lambda\nu_1\nu_2\ \varphi\{\nu_1,\nu_2\}]\downarrow\ \&\ \Box\forall\nu_1\forall\nu_2\ (\varphi\{\nu_1,\ \nu_2\} \equiv \psi\{\nu_1,\ \nu_2\})) \rightarrow$
$[\lambda\nu_1\nu_2\ \psi\{\nu_1,\nu_2\}]\downarrow$›
using *safe−ext*[**where** $\varphi=\lambda(x,y).\ \varphi\ x\ y$]
by (*simp add*: *AOT-model-axiom-def AOT-sem-denotes AOT-model-denotes-prod-def AOT-sem-forall AOT-sem-imp AOT-sem-conj AOT-sem-equiv AOT-sem-box*)
AOT-axiom *safe−ext*[*3*]:
‹$([\lambda\nu_1\nu_2\nu_3\ \varphi\{\nu_1,\nu_2,\nu_3\}]\downarrow\ \&\ \Box\forall\nu_1\forall\nu_2\forall\nu_3\ (\varphi\{\nu_1,\ \nu_2,\ \nu_3\} \equiv \psi\{\nu_1,\ \nu_2,\ \nu_3\})) \rightarrow$
$[\lambda\nu_1\nu_2\nu_3\ \psi\{\nu_1,\nu_2,\nu_3\}]\downarrow$›
using *safe−ext*[**where** $\varphi=\lambda(x,y,z).\ \varphi\ x\ y\ z$]
by (*simp add*: *AOT-model-axiom-def AOT-model-denotes-prod-def AOT-sem-forall AOT-sem-denotes AOT-sem-imp AOT-sem-conj AOT-sem-equiv AOT-sem-box*)
AOT-axiom *safe−ext*[*4*]:
‹$([\lambda\nu_1\nu_2\nu_3\nu_4\ \varphi\{\nu_1,\nu_2,\nu_3,\nu_4\}]\downarrow\ \&$
$\Box\forall\nu_1\forall\nu_2\forall\nu_3\forall\nu_4\ (\varphi\{\nu_1,\ \nu_2,\ \nu_3,\ \nu_4\} \equiv \psi\{\nu_1,\ \nu_2,\ \nu_3,\ \nu_4\})) \rightarrow$
$[\lambda\nu_1\nu_2\nu_3\nu_4\ \psi\{\nu_1,\nu_2,\nu_3,\nu_4\}]\downarrow$›
using *safe−ext*[**where** $\varphi=\lambda(x,y,z,w).\ \varphi\ x\ y\ z\ w$]
by (*simp add*: *AOT-model-axiom-def AOT-model-denotes-prod-def AOT-sem-forall AOT-sem-denotes AOT-sem-imp AOT-sem-conj AOT-sem-equiv AOT-sem-box*)

AOT-axiom *nary−encoding*[*2*]:
‹$x_1x_2[F] \equiv x_1[\lambda y\ [F]yx_2]\ \&\ x_2[\lambda y\ [F]x_1y]$›
by (*rule AOT-model-axiomI*)
(*simp add*: *AOT-sem-conj AOT-sem-equiv AOT-enc-prod-def AOT-proj-enc-prod-def AOT-sem-unary-proj-enc AOT-sem-vars-denote*)
AOT-axiom *nary−encoding*[*3*]:
‹$x_1x_2x_3[F] \equiv x_1[\lambda y\ [F]yx_2x_3]\ \&\ x_2[\lambda y\ [F]x_1yx_3]\ \&\ x_3[\lambda y\ [F]x_1x_2y]$›
by (*rule AOT-model-axiomI*)
(*simp add*: *AOT-sem-conj AOT-sem-equiv AOT-enc-prod-def AOT-proj-enc-prod-def AOT-sem-unary-proj-enc AOT-sem-vars-denote*)
AOT-axiom *nary−encoding*[*4*]:
‹$x_1x_2x_3x_4[F] \equiv x_1[\lambda y\ [F]yx_2x_3x_4]\ \&$
$x_2[\lambda y\ [F]x_1yx_3x_4]\ \&$
$x_3[\lambda y\ [F]x_1x_2yx_4]\ \&$
$x_4[\lambda y\ [F]x_1x_2x_3y]$›
by (*rule AOT-model-axiomI*)
(*simp add*: *AOT-sem-conj AOT-sem-equiv AOT-enc-prod-def AOT-proj-enc-prod-def AOT-sem-unary-proj-enc AOT-sem-vars-denote*)

AOT-axiom *encoding*: ‹$x[F] \to \Box x[F]$›
  using *AOT-sem-enc-nec*
  by (*auto intro*!: *AOT-model-axiomI simp*: *AOT-sem-imp AOT-sem-box*)

AOT-axiom *nocoder*: ‹$O!x \to \neg\exists F\ x[F]$›
  by (*auto intro*!: *AOT-model-axiomI*
          simp: *AOT-sem-imp AOT-sem-not AOT-sem-exists AOT-sem-ordinary*
                AOT-sem-dia
                AOT-sem-lambda-beta[*OF AOT-sem-ordinary-def-denotes*,
                                   OF AOT-sem-vars-denote])
     (*metis AOT-sem-nocoder*)

AOT-axiom *A−objects*: ‹$\exists x\ (A!x\ \&\ \forall F(x[F] \equiv \varphi\{F\}))$›
proof(*rule AOT-model-axiomI*)
  AOT-modally-strict {
    AOT-obtain κ **where** ‹$\kappa\downarrow\ \&\ \Box\neg E!\kappa\ \&\ \forall F\ (\kappa[F] \equiv \varphi\{F\})$›
      using *AOT-sem-A-objects*[*of - φ*]
      by (*auto simp*: *AOT-sem-imp AOT-sem-box AOT-sem-forall AOT-sem-exists*
                  AOT-sem-conj AOT-sem-not AOT-sem-dia AOT-sem-denotes
                  AOT-sem-equiv) *blast*
    AOT-thus ‹$\exists x\ (A!x\ \&\ \forall F(x[F] \equiv \varphi\{F\}))$›
      unfolding *AOT-sem-exists*
      by (*auto intro*!: *exI*[**where** $x{=}\kappa$]
              simp: *AOT-sem-lambda-beta*[*OF AOT-sem-abstract-def-denotes*]
                  AOT-sem-box AOT-sem-dia AOT-sem-not AOT-sem-denotes
                  AOT-var-of-term-inverse AOT-sem-conj
                  AOT-sem-equiv AOT-sem-forall AOT-sem-abstract)
  }
qed

AOT-theorem *universal-closure*:
  assumes ‹*for arbitrary* α: $\varphi\{\alpha\} \in \Lambda_\Box$›
  shows ‹$\forall\alpha\ \varphi\{\alpha\} \in \Lambda_\Box$›
  using *assms*
  by (*metis AOT-term-of-var-cases AOT-model-axiom-def AOT-sem-denotes AOT-sem-forall*)

AOT-theorem *act-closure*:
  assumes ‹$\varphi \in \Lambda_\Box$›
  shows ‹$\boldsymbol{\mathcal{A}}\varphi \in \Lambda_\Box$›
  using *assms* **by** (*simp add*: *AOT-model-axiom-def AOT-sem-act*)

AOT-theorem *nec-closure*:
  assumes ‹$\varphi \in \Lambda_\Box$›
  shows ‹$\Box\varphi \in \Lambda_\Box$›
  using *assms* **by** (*simp add*: *AOT-model-axiom-def AOT-sem-box*)

AOT-theorem *universal-closure-act*:
  assumes ‹*for arbitrary* α: $\varphi\{\alpha\} \in \Lambda$›
  shows ‹$\forall\alpha\ \varphi\{\alpha\} \in \Lambda$›
  using *assms*
  by (*metis AOT-term-of-var-cases AOT-model-act-axiom-def AOT-sem-denotes*
         AOT-sem-forall)

The following are not part of PLM and only hold in the extended models. They are a generalization of the predecessor axiom.

context *AOT-ExtendedModel*
begin
AOT-axiom *indistinguishable-ord-enc-all*:
  ‹$\Pi\downarrow\ \&\ A!x\ \&\ A!y\ \&\ \forall F\ \Box([F]x \equiv [F]y) \to$
  $((\forall G(\forall z(O!z \to \Box([G]z \equiv [\Pi]z)) \to x[G])) \equiv$
    $\forall G(\forall z(O!z \to \Box([G]z \equiv [\Pi]z)) \to y[G]))$›
  by (*rule AOT-model-axiomI*)

(*auto simp*: *AOT-sem-equiv AOT-sem-imp AOT-sem-conj*
AOT-sem-indistinguishable-ord-enc-all)
AOT-axiom *indistinguishable-ord-enc-ex*:
‹Π↓ & $A!x$ & $A!y$ & $\forall F\ \Box([F]x \equiv [F]y) \rightarrow$
$((\exists G(\forall z(O!z \rightarrow \Box([G]z \equiv [\Pi]z))\ \&\ x[G])) \equiv$
$\exists G(\forall z(O!z \rightarrow \Box([G]z \equiv [\Pi]z))\ \&\ y[G]))$›
by (*rule AOT-model-axiomI*)
(*auto simp*: *AOT-sem-equiv AOT-sem-imp AOT-sem-conj*
AOT-sem-indistinguishable-ord-enc-ex)
end

8 The Deductive System PLM

unbundle *AOT-no-atp*

8.1 Primitive Rule of PLM: Modus Ponens

AOT-theorem *modus−ponens*:
assumes ‹φ› **and** ‹φ → ψ›
shows ‹ψ›

using *assms* **by** (*simp add*: *AOT-sem-imp*)
lemmas *MP* = *modus−ponens*

8.2 (Modally Strict) Proofs and Derivations

AOT-theorem *non−con−thm−thm*:
assumes ‹$\vdash_\Box$ φ›
shows ‹⊢ φ›
using *assms* **by** *simp*

AOT-theorem *vdash−properties*:*1*[*1*]:
assumes ‹φ ∈ Λ›
shows ‹⊢ φ›

using *assms* **unfolding** *AOT-model-act-axiom-def* **by** *blast*

Convenience attribute for instantiating modally-fragile axioms.

attribute-setup *act-axiom-inst* =
‹*Scan.succeed* (*Thm.rule-attribute* []
(*K* (*fn thm* => *thm RS* @{*thm vdash−properties*:*1*[*1*]})))›
Instantiate modally fragile axiom as modally fragile theorem.

AOT-theorem *vdash−properties*:*1*[*2*]:
assumes ‹φ ∈ $\Lambda_\Box$›
shows ‹$\vdash_\Box$ φ›

using *assms* **unfolding** *AOT-model-axiom-def* **by** *blast*

Convenience attribute for instantiating modally-strict axioms.

attribute-setup *axiom-inst* =
‹*Scan.succeed* (*Thm.rule-attribute* []
(*K* (*fn thm* => *thm RS* @{*thm vdash−properties*:*1*[*2*]})))›
Instantiate axiom as theorem.

Convenience methods and theorem sets for applying "cqt:2".

method *cqt-2-lambda-inst-prover* =
(*fast intro*: *AOT-instance-of-cqt-2-intro*)
method *cqt*:*2*[*lambda*] =

(*rule cqt*:*2*[*lambda*][*axiom-inst*]; *cqt-2-lambda-inst-prover*)
lemmas *cqt*:*2* =
cqt:*2*[*const-var*][*axiom-inst*] *cqt*:*2*[*lambda*][*axiom-inst*]
AOT-instance-of-cqt-2-intro
method *cqt*:*2* = (*safe intro*!: *cqt*:*2*)

AOT-theorem *vdash−properties*:*3*:
assumes ‹$\vdash_\Box$ φ›
shows ‹Γ ⊢ φ›
using *assms* **by** *blast*

AOT-theorem *vdash−properties*:*5*:
assumes ‹Γ_1 ⊢ φ› **and** ‹Γ_2 ⊢ φ → ψ›
shows ‹Γ_1, Γ_2 ⊢ ψ›
using *MP assms* **by** *blast*

AOT-theorem *vdash−properties*:*6*:
assumes ‹φ› **and** ‹φ → ψ›
shows ‹ψ›
using *MP assms* **by** *blast*

AOT-theorem *vdash−properties*:*8*:
assumes ‹Γ ⊢ φ› **and** ‹φ ⊢ ψ›
shows ‹Γ ⊢ ψ›
using *assms* **by** *argo*

AOT-theorem *vdash−properties*:*9*:
assumes ‹φ›
shows ‹ψ → φ›
using *MP pl*:*1*[*axiom-inst*] *assms* **by** *blast*

AOT-theorem *vdash−properties*:*10*:
assumes ‹φ → ψ› **and** ‹φ›
shows ‹ψ›
using *MP assms* **by** *blast*
lemmas →*E* = *vdash−properties*:*10*

8.3 Two Fundamental Metarules: GEN and RN

AOT-theorem *rule−gen*:
assumes ‹*for arbitrary* α: φ{α}›
shows ‹∀ α φ{α}›

using *assms* **by** (*metis AOT-var-of-term-inverse AOT-sem-denotes AOT-sem-forall*)
lemmas *GEN* = *rule−gen*

AOT-theorem *RN*[*prem*]:
assumes ‹Γ $\vdash_\Box$ φ›
shows ‹□Γ $\vdash_\Box$ □φ›
by (*meson AOT-sem-box assms image-iff*)
AOT-theorem *RN*:
assumes ‹$\vdash_\Box$ φ›
shows ‹□φ›
using *RN*[*prem*] *assms* **by** *blast*

8.4 The Inferential Role of Definitions

AOT-axiom *df−rules−formulas*[*1*]:
assumes ‹φ $\equiv_{df}$ ψ›
shows ‹φ → ψ›

using *assms*
by (*auto simp*: *assms AOT-model-axiomI AOT-model-equiv-def AOT-sem-imp*)

AOT-axiom *df−rules−formulas*[*2*]:
assumes ‹$\varphi \equiv_{df} \psi$›
shows ‹$\psi \rightarrow \varphi$›

using *assms*
by (*auto simp*: *AOT-model-axiomI AOT-model-equiv-def AOT-sem-imp*)

AOT-theorem *df−rules−formulas*[*3*]:
assumes ‹$\varphi \equiv_{df} \psi$›
shows ‹$\varphi \rightarrow \psi$›
using *df−rules−formulas*[*1*][*axiom-inst*, *OF assms*].
AOT-theorem *df−rules−formulas*[*4*]:
assumes ‹$\varphi \equiv_{df} \psi$›
shows ‹$\psi \rightarrow \varphi$›
using *df−rules−formulas*[*2*][*axiom-inst*, *OF assms*].

AOT-axiom *df−rules−terms*[*1*]:
assumes ‹$\tau\{\alpha_1...\alpha_n\} =_{df} \sigma\{\alpha_1...\alpha_n\}$›
shows ‹$(\sigma\{\tau_1...\tau_n\}\downarrow \rightarrow \tau\{\tau_1...\tau_n\} = \sigma\{\tau_1...\tau_n\})$ &
$(\neg\sigma\{\tau_1...\tau_n\}\downarrow \rightarrow \neg\tau\{\tau_1...\tau_n\}\downarrow)$›

using *assms*
by (*simp add*: *AOT-model-axiomI AOT-sem-conj AOT-sem-imp AOT-sem-eq AOT-sem-not AOT-sem-denotes AOT-model-id-def*)
AOT-axiom *df−rules−terms*[*2*]:
assumes ‹$\tau =_{df} \sigma$›
shows ‹$(\sigma\downarrow \rightarrow \tau = \sigma)$ & $(\neg\sigma\downarrow \rightarrow \neg\tau\downarrow)$›
by (*metis df−rules−terms*[*1*] *case-unit-Unity assms*)

AOT-theorem *df−rules−terms*[*3*]:
assumes ‹$\tau\{\alpha_1...\alpha_n\} =_{df} \sigma\{\alpha_1...\alpha_n\}$›
shows ‹$(\sigma\{\tau_1...\tau_n\}\downarrow \rightarrow \tau\{\tau_1...\tau_n\} = \sigma\{\tau_1...\tau_n\})$ &
$(\neg\sigma\{\tau_1...\tau_n\}\downarrow \rightarrow \neg\tau\{\tau_1...\tau_n\}\downarrow)$›
using *df−rules−terms*[*1*][*axiom-inst*, *OF assms*].
AOT-theorem *df−rules−terms*[*4*]:
assumes ‹$\tau =_{df} \sigma$›
shows ‹$(\sigma\downarrow \rightarrow \tau = \sigma)$ & $(\neg\sigma\downarrow \rightarrow \neg\tau\downarrow)$›
using *df−rules−terms*[*2*][*axiom-inst*, *OF assms*].

8.5 The Theory of Negations and Conditionals

AOT-theorem *if−p−then−p*: ‹$\varphi \rightarrow \varphi$›
by (*meson pl*:*1*[*axiom-inst*] *pl*:*2*[*axiom-inst*] *MP*)

AOT-theorem *deduction−theorem*:
assumes ‹$\varphi \vdash \psi$›
shows ‹$\varphi \rightarrow \psi$›

using *assms* **by** (*simp add*: *AOT-sem-imp*)
lemmas *CP* = *deduction−theorem*
lemmas →*I* = *deduction−theorem*

AOT-theorem *ded−thm−cor*:*1*:
assumes ‹$\Gamma_1 \vdash \varphi \rightarrow \psi$› **and** ‹$\Gamma_2 \vdash \psi \rightarrow \chi$›
shows ‹$\Gamma_1, \Gamma_2 \vdash \varphi \rightarrow \chi$›
using →*E* →*I assms* **by** *blast*
AOT-theorem *ded−thm−cor*:*2*:
assumes ‹$\Gamma_1 \vdash \varphi \rightarrow (\psi \rightarrow \chi)$› **and** ‹$\Gamma_2 \vdash \psi$›
shows ‹$\Gamma_1, \Gamma_2 \vdash \varphi \rightarrow \chi$›
using →*E* →*I assms* **by** *blast*

AOT-theorem *ded−thm−cor*:*3*:

assumes ‹$\varphi \rightarrow \psi$› **and** ‹$\psi \rightarrow \chi$›
shows ‹$\varphi \rightarrow \chi$›
using $\rightarrow E$ $\rightarrow I$ *assms* **by** *blast*
declare *ded−thm−cor*:*3*[*trans*]
AOT-theorem *ded−thm−cor*:*4*:
assumes ‹$\varphi \rightarrow (\psi \rightarrow \chi)$› **and** ‹$\psi$›
shows ‹$\varphi \rightarrow \chi$›
using $\rightarrow E$ $\rightarrow I$ *assms* **by** *blast*

lemmas *Hypothetical Syllogism* = *ded−thm−cor*:*3*

AOT-theorem *useful−tautologies*:*1*: ‹$\neg\neg\varphi \rightarrow \varphi$›
by (*metis pl*:*3*[*axiom-inst*] $\rightarrow I$ *Hypothetical Syllogism*)
AOT-theorem *useful−tautologies*:*2*: ‹$\varphi \rightarrow \neg\neg\varphi$›
by (*metis pl*:*3*[*axiom-inst*] $\rightarrow I$ *ded−thm−cor*:*4*)
AOT-theorem *useful−tautologies*:*3*: ‹$\neg\varphi \rightarrow (\varphi \rightarrow \psi)$›
by (*meson ded−thm−cor*:*4 pl*:*3*[*axiom-inst*] $\rightarrow I$)
AOT-theorem *useful−tautologies*:*4*: ‹$(\neg\psi \rightarrow \neg\varphi) \rightarrow (\varphi \rightarrow \psi)$›
by (*meson pl*:*3*[*axiom-inst*] *Hypothetical Syllogism* $\rightarrow I$)
AOT-theorem *useful−tautologies*:*5*: ‹$(\varphi \rightarrow \psi) \rightarrow (\neg\psi \rightarrow \neg\varphi)$›
by (*metis useful−tautologies*:*4 Hypothetical Syllogism* $\rightarrow I$)

AOT-theorem *useful−tautologies*:*6*: ‹$(\varphi \rightarrow \neg\psi) \rightarrow (\psi \rightarrow \neg\varphi)$›
by (*metis* $\rightarrow I$ *MP useful−tautologies*:*4*)

AOT-theorem *useful−tautologies*:*7*: ‹$(\neg\varphi \rightarrow \psi) \rightarrow (\neg\psi \rightarrow \varphi)$›
by (*metis* $\rightarrow I$ *MP useful−tautologies*:*3 useful−tautologies*:*5*)

AOT-theorem *useful−tautologies*:*8*: ‹$\varphi \rightarrow (\neg\psi \rightarrow \neg(\varphi \rightarrow \psi))$›
by (*metis* $\rightarrow I$ *MP useful−tautologies*:*5*)

AOT-theorem *useful−tautologies*:*9*: ‹$(\varphi \rightarrow \psi) \rightarrow ((\neg\varphi \rightarrow \psi) \rightarrow \psi)$›
by (*metis* $\rightarrow I$ *MP useful−tautologies*:*6*)

AOT-theorem *useful−tautologies*:*10*: ‹$(\varphi \rightarrow \neg\psi) \rightarrow ((\varphi \rightarrow \psi) \rightarrow \neg\varphi)$›
by (*metis* $\rightarrow I$ *MP pl*:*3*[*axiom-inst*])

AOT-theorem *dn−i−e*:*1*:
assumes ‹φ›
shows ‹$\neg\neg\varphi$›
using *MP useful−tautologies*:*2 assms* **by** *blast*
lemmas $\neg\neg I$ = *dn−i−e*:*1*
AOT-theorem *dn−i−e*:*2*:
assumes ‹$\neg\neg\varphi$›
shows ‹φ›
using *MP useful−tautologies*:*1 assms* **by** *blast*
lemmas $\neg\neg E$ = *dn−i−e*:*2*

AOT-theorem *modus−tollens*:*1*:
assumes ‹$\varphi \rightarrow \psi$› **and** ‹$\neg\psi$›
shows ‹$\neg\varphi$›
using *MP useful−tautologies*:*5 assms* **by** *blast*
AOT-theorem *modus−tollens*:*2*:
assumes ‹$\varphi \rightarrow \neg\psi$› **and** ‹$\psi$›
shows ‹$\neg\varphi$›
using $\neg\neg I$ *modus−tollens*:*1 assms* **by** *blast*
lemmas *MT* = *modus−tollens*:*1 modus−tollens*:*2*

AOT-theorem *contraposition*:*1*[*1*]:
assumes ‹$\varphi \rightarrow \psi$›
shows ‹$\neg\psi \rightarrow \neg\varphi$›
using $\rightarrow I$ *MT*(*1*) *assms* **by** *blast*
AOT-theorem *contraposition*:*1*[*2*]:

assumes ‹¬ψ → ¬φ›
shows ‹φ → ψ›
using *→I ¬¬E MT(2) assms* **by** *blast*

AOT-theorem *contraposition:2*:
assumes ‹φ → ¬ψ›
shows ‹ψ → ¬φ›
using *→I MT(2) assms* **by** *blast*

AOT-theorem *reductio−aa:1*:
assumes ‹¬φ ⊢ ¬ψ› **and** ‹¬φ ⊢ ψ›
shows ‹φ›
using *→I ¬¬E MT(2) assms* **by** *blast*
AOT-theorem *reductio−aa:2*:
assumes ‹φ ⊢ ¬ψ› **and** ‹φ ⊢ ψ›
shows ‹¬φ›
using *reductio−aa:1 assms* **by** *blast*
lemmas *RAA* = *reductio−aa:1 reductio−aa:2*

AOT-theorem *exc−mid*: ‹φ ∨ ¬φ›
using *df−rules−formulas[4] if−p−then−p MP*
conventions:2 **by** *blast*

AOT-theorem *non−contradiction*: ‹¬(φ & ¬φ)›
using *df−rules−formulas[3] MT(2) useful−tautologies:2*
conventions:1 **by** *blast*

AOT-theorem *con−dis−taut:1*: ‹(φ & ψ) → φ›
by (*meson →I df−rules−formulas[3] MP RAA(1) conventions:1*)
AOT-theorem *con−dis−taut:2*: ‹(φ & ψ) → ψ›
by (*metis →I df−rules−formulas[3] MT(2) RAA(2)*
¬¬E conventions:1)
lemmas *Conjunction Simplification* = *con−dis−taut:1 con−dis−taut:2*

AOT-theorem *con−dis−taut:3*: ‹φ → (φ ∨ ψ)›
by (*meson contraposition:1[2] df−rules−formulas[4]*
MP →I conventions:2)
AOT-theorem *con−dis−taut:4*: ‹ψ → (φ ∨ ψ)›
using *Hypothetical Syllogism df−rules−formulas[4]*
pl:1[axiom-inst] conventions:2 **by** *blast*
lemmas *Disjunction Addition* = *con−dis−taut:3 con−dis−taut:4*

AOT-theorem *con−dis−taut:5*: ‹φ → (ψ → (φ & ψ))›
by (*metis contraposition:2 Hypothetical Syllogism →I*
df−rules−formulas[4] conventions:1)
lemmas *Adjunction* = *con−dis−taut:5*

AOT-theorem *con−dis−taut:6*: ‹(φ & φ) ≡ φ›
by (*metis Adjunction →I df−rules−formulas[4] MP*
Conjunction Simplification(1) conventions:3)
lemmas *Idempotence of* & = *con−dis−taut:6*

AOT-theorem *con−dis−taut:7*: ‹(φ ∨ φ) ≡ φ›
proof −
{
AOT-assume ‹φ ∨ φ›
AOT-hence ‹¬φ → φ›
using *conventions:2[THEN df−rules−formulas[3]] MP* **by** *blast*
AOT-hence ‹φ› **using** *if−p−then−p RAA(1) MP* **by** *blast*
}
moreover {
AOT-assume ‹φ›
AOT-hence ‹φ ∨ φ› **using** *Disjunction Addition(1) MP* **by** *blast*

}
ultimately **AOT-show** ‹(φ ∨ φ) ≡ φ›
using *conventions:3[THEN df−rules−formulas[4]] MP*
by (*metis Adjunction →I*)
qed
lemmas *Idempotence of ∨ = con−dis−taut:7*

AOT-theorem *con−dis−i−e:1*:
assumes ‹φ› **and** ‹ψ›
shows ‹φ & ψ›
using *Adjunction MP assms* **by** *blast*
lemmas *&I = con−dis−i−e:1*

AOT-theorem *con−dis−i−e:2:a*:
assumes ‹φ & ψ›
shows ‹φ›
using *Conjunction Simplification(1) MP assms* **by** *blast*
AOT-theorem *con−dis−i−e:2:b*:
assumes ‹φ & ψ›
shows ‹ψ›
using *Conjunction Simplification(2) MP assms* **by** *blast*
lemmas *&E = con−dis−i−e:2:a con−dis−i−e:2:b*

AOT-theorem *con−dis−i−e:3:a*:
assumes ‹φ›
shows ‹φ ∨ ψ›
using *Disjunction Addition(1) MP assms* **by** *blast*
AOT-theorem *con−dis−i−e:3:b*:
assumes ‹ψ›
shows ‹φ ∨ ψ›
using *Disjunction Addition(2) MP assms* **by** *blast*
AOT-theorem *con−dis−i−e:3:c*:
assumes ‹φ ∨ ψ› **and** ‹φ → χ› **and** ‹ψ → Θ›
shows ‹χ ∨ Θ›
by (*metis con−dis−i−e:3:a Disjunction Addition(2)*
df−rules−formulas[3] MT(1) RAA(1)
conventions:2 assms)
lemmas *∨I = con−dis−i−e:3:a con−dis−i−e:3:b con−dis−i−e:3:c*

AOT-theorem *con−dis−i−e:4:a*:
assumes ‹φ ∨ ψ› **and** ‹φ → χ› **and** ‹ψ → χ›
shows ‹χ›
by (*metis MP RAA(2) df−rules−formulas[3] conventions:2 assms*)
AOT-theorem *con−dis−i−e:4:b*:
assumes ‹φ ∨ ψ› **and** ‹¬φ›
shows ‹ψ›
using *con−dis−i−e:4:a RAA(1) →I assms* **by** *blast*
AOT-theorem *con−dis−i−e:4:c*:
assumes ‹φ ∨ ψ› **and** ‹¬ψ›
shows ‹φ›
using *con−dis−i−e:4:a RAA(1) →I assms* **by** *blast*
lemmas *∨E = con−dis−i−e:4:a con−dis−i−e:4:b con−dis−i−e:4:c*

AOT-theorem *raa−cor:1*:
assumes ‹¬φ ⊢ ψ & ¬ψ›
shows ‹φ›
using *&E ∨E(3) ∨I(2) RAA(2) assms* **by** *blast*
AOT-theorem *raa−cor:2*:
assumes ‹φ ⊢ ψ & ¬ψ›
shows ‹¬φ›
using *raa−cor:1 assms* **by** *blast*
AOT-theorem *raa−cor:3*:

assumes ‹φ› **and** ‹$\neg\psi \vdash \neg\varphi$›
shows ‹ψ›
using *RAA assms* **by** *blast*
AOT-theorem *raa−cor:4*:
assumes ‹$\neg\varphi$› **and** ‹$\neg\psi \vdash \varphi$›
shows ‹ψ›
using *RAA assms* **by** *blast*
AOT-theorem *raa−cor:5*:
assumes ‹φ› **and** ‹$\psi \vdash \neg\varphi$›
shows ‹$\neg\psi$›
using *RAA assms* **by** *blast*
AOT-theorem *raa−cor:6*:
assumes ‹$\neg\varphi$› **and** ‹$\psi \vdash \varphi$›
shows ‹$\neg\psi$›
using *RAA assms* **by** *blast*

AOT-theorem *oth−class−taut:1:a*: ‹$(\varphi \rightarrow \psi) \equiv \neg(\varphi \ \& \ \neg\psi)$›
by (*rule conventions:3*[*THEN df−rules−formulas*[*4*], *THEN* →*E*])
(*metis* &*E* &*I raa−cor:3* →*I MP*)
AOT-theorem *oth−class−taut:1:b*: ‹$\neg(\varphi \rightarrow \psi) \equiv (\varphi \ \& \ \neg\psi)$›
by (*rule conventions:3*[*THEN df−rules−formulas*[*4*], *THEN* →*E*])
(*metis* &*E* &*I raa−cor:3* →*I MP*)
AOT-theorem *oth−class−taut:1:c*: ‹$(\varphi \rightarrow \psi) \equiv (\neg\varphi \vee \psi)$›
by (*rule conventions:3*[*THEN df−rules−formulas*[*4*], *THEN* →*E*])
(*metis* &*I* ∨*I*(*1*, *2*) ∨*E*(*3*) →*I MP raa−cor:1*)

AOT-theorem *oth−class−taut:2:a*: ‹$(\varphi \ \& \ \psi) \equiv (\psi \ \& \ \varphi)$›
by (*rule conventions:3*[*THEN df−rules−formulas*[*4*], *THEN* →*E*])
(*meson* &*I* &*E* →*I*)
lemmas *Commutativity of* & = *oth−class−taut:2:a*
AOT-theorem *oth−class−taut:2:b*: ‹$(\varphi \ \& \ (\psi \ \& \ \chi)) \equiv ((\varphi \ \& \ \psi) \ \& \ \chi)$›
by (*rule conventions:3*[*THEN df−rules−formulas*[*4*], *THEN* →*E*])
(*metis* &*I* &*E* →*I*)
lemmas *Associativity of* & = *oth−class−taut:2:b*
AOT-theorem *oth−class−taut:2:c*: ‹$(\varphi \vee \psi) \equiv (\psi \vee \varphi)$›
by (*rule conventions:3*[*THEN df−rules−formulas*[*4*], *THEN* →*E*])
(*metis* &*I* ∨*I*(*1*, *2*) ∨*E*(*1*) →*I*)
lemmas *Commutativity of* ∨ = *oth−class−taut:2:c*
AOT-theorem *oth−class−taut:2:d*: ‹$(\varphi \vee (\psi \vee \chi)) \equiv ((\varphi \vee \psi) \vee \chi)$›
by (*rule conventions:3*[*THEN df−rules−formulas*[*4*], *THEN* →*E*])
(*metis* &*I* ∨*I*(*1*, *2*) ∨*E*(*1*) →*I*)
lemmas *Associativity of* ∨ = *oth−class−taut:2:d*
AOT-theorem *oth−class−taut:2:e*: ‹$(\varphi \equiv \psi) \equiv (\psi \equiv \varphi)$›
by (*rule conventions:3*[*THEN df−rules−formulas*[*4*], *THEN* →*E*]; *rule* &*I*;
metis &*I df−rules−formulas*[*4*] *conventions:3* &*E*
Hypothetical Syllogism →*I df−rules−formulas*[*3*])
lemmas *Commutativity of* ≡ = *oth−class−taut:2:e*
AOT-theorem *oth−class−taut:2:f*: ‹$(\varphi \equiv (\psi \equiv \chi)) \equiv ((\varphi \equiv \psi) \equiv \chi)$›
using *conventions:3*[*THEN df−rules−formulas*[*4*]]
conventions:3[*THEN df−rules−formulas*[*3*]]
→*I* →*E* &*E* &*I*
by *metis*
lemmas *Associativity of* ≡ = *oth−class−taut:2:f*

AOT-theorem *oth−class−taut:3:a*: ‹$\varphi \equiv \varphi$›
using &*I vdash−properties:6 if−p−then−p*
df−rules−formulas[*4*] *conventions:3* **by** *blast*
AOT-theorem *oth−class−taut:3:b*: ‹$\varphi \equiv \neg\neg\varphi$›
using &*I useful−tautologies:1 useful−tautologies:2* →*E*
df−rules−formulas[*4*] *conventions:3* **by** *blast*
AOT-theorem *oth−class−taut:3:c*: ‹$\neg(\varphi \equiv \neg\varphi)$›
by (*metis* &*E* →*E RAA df−rules−formulas*[*3*] *conventions:3*)

AOT-theorem *oth−class−taut:4:a*: ‹$(\varphi \rightarrow \psi) \rightarrow ((\psi \rightarrow \chi) \rightarrow (\varphi \rightarrow \chi))$›
 by (*metis* $\rightarrow E$ $\rightarrow I$)
AOT-theorem *oth−class−taut:4:b*: ‹$(\varphi \equiv \psi) \equiv (\neg\varphi \equiv \neg\psi)$›
 using *conventions:3[THEN df−rules−formulas[4]]*
  conventions:3[THEN df−rules−formulas[3]]
  $\rightarrow I$ $\rightarrow E$ &E &I *RAA* **by** *metis*
AOT-theorem *oth−class−taut:4:c*: ‹$(\varphi \equiv \psi) \rightarrow ((\varphi \rightarrow \chi) \equiv (\psi \rightarrow \chi))$›
 using *conventions:3[THEN df−rules−formulas[4]]*
  conventions:3[THEN df−rules−formulas[3]]
  $\rightarrow I$ $\rightarrow E$ &E &I **by** *metis*
AOT-theorem *oth−class−taut:4:d*: ‹$(\varphi \equiv \psi) \rightarrow ((\chi \rightarrow \varphi) \equiv (\chi \rightarrow \psi))$›
 using *conventions:3[THEN df−rules−formulas[4]]*
  conventions:3[THEN df−rules−formulas[3]]
  $\rightarrow I$ $\rightarrow E$ &E &I **by** *metis*
AOT-theorem *oth−class−taut:4:e*: ‹$(\varphi \equiv \psi) \rightarrow ((\varphi \,\&\, \chi) \equiv (\psi \,\&\, \chi))$›
 using *conventions:3[THEN df−rules−formulas[4]]*
  conventions:3[THEN df−rules−formulas[3]]
  $\rightarrow I$ $\rightarrow E$ &E &I **by** *metis*
AOT-theorem *oth−class−taut:4:f*: ‹$(\varphi \equiv \psi) \rightarrow ((\chi \,\&\, \varphi) \equiv (\chi \,\&\, \psi))$›
 using *conventions:3[THEN df−rules−formulas[4]]*
  conventions:3[THEN df−rules−formulas[3]]
  $\rightarrow I$ $\rightarrow E$ &E &I **by** *metis*
AOT-theorem *oth−class−taut:4:g*: ‹$(\varphi \equiv \psi) \equiv ((\varphi \,\&\, \psi) \vee (\neg\varphi \,\&\, \neg\psi))$›
proof(*safe intro*!: *conventions:3[THEN df−rules−formulas[4], THEN* $\rightarrow E$]
  &I $\rightarrow I$
  dest!: *conventions:3[THEN df−rules−formulas[3], THEN* $\rightarrow E$])
 AOT-show ‹$\varphi \,\&\, \psi \vee (\neg\varphi \,\&\, \neg\psi)$› **if** ‹$(\varphi \rightarrow \psi) \,\&\, (\psi \rightarrow \varphi)$›
  using &E $\vee I$ $\rightarrow E$ &I *raa−cor:1* $\rightarrow I$ $\vee E$ *that* **by** *metis*
next
 AOT-show ‹ψ› **if** ‹$\varphi \,\&\, \psi \vee (\neg\varphi \,\&\, \neg\psi)$› **and** ‹$\varphi$›
  using *that* $\vee E$ &E *raa−cor:3* **by** *blast*
next
 AOT-show ‹φ› **if** ‹$\varphi \,\&\, \psi \vee (\neg\varphi \,\&\, \neg\psi)$› **and** ‹$\psi$›
  using *that* $\vee E$ &E *raa−cor:3* **by** *blast*
qed
AOT-theorem *oth−class−taut:4:h*: ‹$\neg(\varphi \equiv \psi) \equiv ((\varphi \,\&\, \neg\psi) \vee (\neg\varphi \,\&\, \psi))$›
proof (*safe intro*!: *conventions:3[THEN df−rules−formulas[4], THEN* $\rightarrow E$]
  &I $\rightarrow I$)
 AOT-show ‹$\varphi \,\&\, \neg\psi \vee (\neg\varphi \,\&\, \psi)$› **if** ‹$\neg(\varphi \equiv \psi)$›
  by (*metis that* &I $\vee I(1, 2)$ $\rightarrow I$ *MT(1) df−rules−formulas[4]*
   raa−cor:3 conventions:3)
next
 AOT-show ‹$\neg(\varphi \equiv \psi)$› **if** ‹$\varphi \,\&\, \neg\psi \vee (\neg\varphi \,\&\, \psi)$›
  by (*metis that* &E $\vee E(2)$ $\rightarrow E$ *df−rules−formulas[3]*
   raa−cor:3 conventions:3)
qed
AOT-theorem *oth−class−taut:5:a*: ‹$(\varphi \,\&\, \psi) \equiv \neg(\neg\varphi \vee \neg\psi)$›
 using *conventions:3[THEN df−rules−formulas[4]]*
  $\rightarrow I$ $\rightarrow E$ &E &I $\vee I$ $\vee E$ *RAA* **by** *metis*
AOT-theorem *oth−class−taut:5:b*: ‹$(\varphi \vee \psi) \equiv \neg(\neg\varphi \,\&\, \neg\psi)$›
 using *conventions:3[THEN df−rules−formulas[4]]*
  $\rightarrow I$ $\rightarrow E$ &E &I $\vee I$ $\vee E$ *RAA* **by** *metis*
AOT-theorem *oth−class−taut:5:c*: ‹$\neg(\varphi \,\&\, \psi) \equiv (\neg\varphi \vee \neg\psi)$›
 using *conventions:3[THEN df−rules−formulas[4]]*
  $\rightarrow I$ $\rightarrow E$ &E &I $\vee I$ $\vee E$ *RAA* **by** *metis*
AOT-theorem *oth−class−taut:5:d*: ‹$\neg(\varphi \vee \psi) \equiv (\neg\varphi \,\&\, \neg\psi)$›
 using *conventions:3[THEN df−rules−formulas[4]]*
  $\rightarrow I$ $\rightarrow E$ &E &I $\vee I$ $\vee E$ *RAA* **by** *metis*

lemmas *DeMorgan* = *oth−class−taut:5:c oth−class−taut:5:d*

AOT-theorem *oth−class−taut:6:a*:
 ‹$(\varphi \,\&\, (\psi \vee \chi)) \equiv ((\varphi \,\&\, \psi) \vee (\varphi \,\&\, \chi))$›

using *conventions*:*3*[*THEN df−rules−formulas*[*4*]]
  →*I* →*E* &*E* &*I* ∨*I* ∨*E* *RAA* **by** *metis*
AOT-theorem *oth−class−taut*:*6*:*b*:
‹$(\varphi \vee (\psi \ \& \ \chi)) \equiv ((\varphi \vee \psi) \ \& \ (\varphi \vee \chi))$›
using *conventions*:*3*[*THEN df−rules−formulas*[*4*]]
  →*I* →*E* &*E* &*I* ∨*I* ∨*E* *RAA* **by** *metis*

AOT-theorem *oth−class−taut*:*7*:*a*: ‹$((\varphi \ \& \ \psi) \to \chi) \to (\varphi \to (\psi \to \chi))$›
by (*metis* &*I* →*E* →*I*)
lemmas *Exportation* = *oth−class−taut*:*7*:*a*
AOT-theorem *oth−class−taut*:*7*:*b*: ‹$(\varphi \to (\psi \to \chi)) \to ((\varphi \ \& \ \psi) \to \chi)$›
by (*metis* &*E* →*E* →*I*)
lemmas *Importation* = *oth−class−taut*:*7*:*b*

AOT-theorem *oth−class−taut*:*8*:*a*:
‹$(\varphi \to (\psi \to \chi)) \equiv (\psi \to (\varphi \to \chi))$›
using *conventions*:*3*[*THEN df−rules−formulas*[*4*]] →*I* →*E* &*E* &*I*
by *metis*
lemmas *Permutation* = *oth−class−taut*:*8*:*a*
AOT-theorem *oth−class−taut*:*8*:*b*:
‹$(\varphi \to \psi) \to ((\varphi \to \chi) \to (\varphi \to (\psi \ \& \ \chi)))$›
by (*metis* &*I* →*E* →*I*)
lemmas *Composition* = *oth−class−taut*:*8*:*b*
AOT-theorem *oth−class−taut*:*8*:*c*:
‹$(\varphi \to \chi) \to ((\psi \to \chi) \to ((\varphi \vee \psi) \to \chi))$›
by (*metis* ∨*E*(*2*) →*E* →*I* *RAA*(*1*))
AOT-theorem *oth−class−taut*:*8*:*d*:
‹$((\varphi \to \psi) \ \& \ (\chi \to \Theta)) \to ((\varphi \ \& \ \chi) \to (\psi \ \& \ \Theta))$›
by (*metis* &*E* &*I* →*E* →*I*)
lemmas *Double Composition* = *oth−class−taut*:*8*:*d*
AOT-theorem *oth−class−taut*:*8*:*e*:
‹$((\varphi \ \& \ \psi) \equiv (\varphi \ \& \ \chi)) \equiv (\varphi \to (\psi \equiv \chi))$›
by (*metis conventions*:*3*[*THEN df−rules−formulas*[*4*]]
  conventions:*3*[*THEN df−rules−formulas*[*3*]]
  →*I* →*E* &*E* &*I*)
AOT-theorem *oth−class−taut*:*8*:*f*:
‹$((\varphi \ \& \ \psi) \equiv (\chi \ \& \ \psi)) \equiv (\psi \to (\varphi \equiv \chi))$›
by (*metis conventions*:*3*[*THEN df−rules−formulas*[*4*]]
  conventions:*3*[*THEN df−rules−formulas*[*3*]]
  →*I* →*E* &*E* &*I*)
AOT-theorem *oth−class−taut*:*8*:*g*:
‹$(\psi \equiv \chi) \to ((\varphi \vee \psi) \equiv (\varphi \vee \chi))$›
by (*metis conventions*:*3*[*THEN df−rules−formulas*[*4*]]
  conventions:*3*[*THEN df−rules−formulas*[*3*]]
  →*I* →*E* &*E* &*I* ∨*I* ∨*E*(*1*))
AOT-theorem *oth−class−taut*:*8*:*h*:
‹$(\psi \equiv \chi) \to ((\psi \vee \varphi) \equiv (\chi \vee \varphi))$›
by (*metis conventions*:*3*[*THEN df−rules−formulas*[*4*]]
  conventions:*3*[*THEN df−rules−formulas*[*3*]]
  →*I* →*E* &*E* &*I* ∨*I* ∨*E*(*1*))
AOT-theorem *oth−class−taut*:*8*:*i*:
‹$(\varphi \equiv (\psi \ \& \ \chi)) \to (\psi \to (\varphi \equiv \chi))$›
by (*metis conventions*:*3*[*THEN df−rules−formulas*[*4*]]
  conventions:*3*[*THEN df−rules−formulas*[*3*]]
  →*I* →*E* &*E* &*I*)

AOT-theorem *intro−elim*:*1*:
assumes ‹$\varphi \vee \psi$› **and** ‹$\varphi \equiv \chi$› **and** ‹$\psi \equiv \Theta$›
shows ‹$\chi \vee \Theta$›
by (*metis assms* ∨*I*(*1*, *2*) ∨*E*(*1*) →*I* →*E* &*E*(*1*)
  conventions:*3*[*THEN df−rules−formulas*[*3*]])

AOT-theorem *intro−elim*:*2*:

```
  assumes ‹φ → ψ› and ‹ψ → φ›
  shows ‹φ ≡ ψ›
  by (meson &I conventions:3 df−rules−formulas[4] MP assms)
lemmas ≡I = intro−elim:2

AOT-theorem intro−elim:3:a:
  assumes ‹φ ≡ ψ› and ‹φ›
  shows ‹ψ›
  by (metis ∨I(1) →I ∨E(1) intro−elim:1 assms)
AOT-theorem intro−elim:3:b:
  assumes ‹φ ≡ ψ› and ‹ψ›
  shows ‹φ›
  using intro−elim:3:a Commutativity of ≡ assms by blast
AOT-theorem intro−elim:3:c:
  assumes ‹φ ≡ ψ› and ‹¬φ›
  shows ‹¬ψ›
  using intro−elim:3:b raa−cor:3 assms by blast
AOT-theorem intro−elim:3:d:
  assumes ‹φ ≡ ψ› and ‹¬ψ›
  shows ‹¬φ›
  using intro−elim:3:a raa−cor:3 assms by blast
AOT-theorem intro−elim:3:e:
  assumes ‹φ ≡ ψ› and ‹ψ ≡ χ›
  shows ‹φ ≡ χ›
  by (metis ≡I →I intro−elim:3:a intro−elim:3:b assms)
declare intro−elim:3:e[trans]
AOT-theorem intro−elim:3:f:
  assumes ‹φ ≡ ψ› and ‹φ ≡ χ›
  shows ‹χ ≡ ψ›
  by (metis ≡I →I intro−elim:3:a intro−elim:3:b assms)
lemmas ≡E = intro−elim:3:a intro−elim:3:b intro−elim:3:c
             intro−elim:3:d intro−elim:3:e intro−elim:3:f

declare Commutativity of ≡[THEN ≡E(1), sym]

AOT-theorem rule−eq−df:1:
  assumes ‹φ ≡df ψ›
  shows ‹φ ≡ ψ›
  by (simp add: ≡I df−rules−formulas[3] df−rules−formulas[4] assms)
lemmas ≡Df = rule−eq−df:1
AOT-theorem rule−eq−df:2:
  assumes ‹φ ≡df ψ› and ‹φ›
  shows ‹ψ›
  using ≡Df ≡E(1) assms by blast
lemmas ≡dfE = rule−eq−df:2
AOT-theorem rule−eq−df:3:
  assumes ‹φ ≡df ψ› and ‹ψ›
  shows ‹φ›
  using ≡Df ≡E(2) assms by blast
lemmas ≡dfI = rule−eq−df:3

AOT-theorem df−simplify:1:
  assumes ‹φ ≡ (ψ & χ)› and ‹ψ›
  shows ‹φ ≡ χ›
  by (metis &E(2) &I ≡E(1, 2) ≡I →I assms)

AOT-theorem df−simplify:2:
  assumes ‹φ ≡ (ψ & χ)› and ‹χ›
  shows ‹φ ≡ ψ›
  by (metis &E(1) &I ≡E(1, 2) ≡I →I assms)
lemmas ≡S = df−simplify:1 df−simplify:2
```

8.6 The Theory of Quantification

AOT-theorem *rule−ui*:*1*:
  assumes ‹∀ α φ{α}› **and** ‹τ↓›
  shows ‹φ{τ}›
  using →*E cqt*:*1*[*axiom-inst*] *assms* **by** *blast*
AOT-theorem *rule−ui*:*2*[*const-var*]:
  assumes ‹∀ α φ{α}›
  shows ‹φ{β}›
  by (*simp add*: *rule−ui*:*1 cqt*:*2*[*const-var*][*axiom-inst*] *assms*)
AOT-theorem *rule−ui*:*2*[*lambda*]:
  assumes ‹∀ *F* φ{*F*}› **and** ‹*INSTANCE-OF-CQT-2*(ψ)›
  shows ‹φ{[λν1...νn ψ{ν1...νn}]}›
  by (*simp add*: *rule−ui*:*1 cqt*:*2*[*lambda*][*axiom-inst*] *assms*)
AOT-theorem *rule−ui*:*3*:
  assumes ‹∀ α φ{α}›
  shows ‹φ{α}›
  by (*simp add*: *rule−ui*:*2*[*const-var*] *assms*)
lemmas ∀ *E* = *rule−ui*:*1 rule−ui*:*2*[*const-var*]
    rule−ui:*2*[*lambda*] *rule−ui*:*3*

AOT-theorem *cqt−orig*:*1*[*const-var*]: ‹∀ α φ{α} → φ{β}›
  by (*simp add*: ∀ *E*(*2*) →*I*)
AOT-theorem *cqt−orig*:*1*[*lambda*]:
  assumes ‹*INSTANCE-OF-CQT-2*(ψ)›
  shows ‹∀ *F* φ{*F*} → φ{[λν1...νn ψ{ν1...νn}]}›
  by (*simp add*: ∀ *E*(*3*) →*I assms*)
AOT-theorem *cqt−orig*:*2*: ‹∀ α (φ → ψ{α}) → (φ → ∀ α ψ{α})›
  by (*metis* →*I GEN vdash−properties*:*6* ∀ *E*(*4*))
AOT-theorem *cqt−orig*:*3*: ‹∀ α φ{α} → φ{α}›
  using *cqt−orig*:*1*[*const-var*]**.**

AOT-theorem *universal*:
  assumes ‹*for arbitrary* β: φ{β}›
  shows ‹∀ α φ{α}›
  using *GEN assms* **.**
lemmas ∀ *I* = *universal*

ML‹
```
fun get-instantiated-allI ctxt varname thm = let
val trm = Thm.concl-of thm
val trm =
  case trm of (@{const Trueprop} $ (@{const AOT-model-valid-in} $ - $ x)) => x
  | - => raise Term.TERM (Expected simple theorem., [trm])
fun extractVars (Const (const-name ‹AOT-term-of-var›, -) $ Var v) =
    (if fst (fst v) = fst varname then [Var v] else [])
  | extractVars (t1 $ t2) = extractVars t1 @ extractVars t2
  | extractVars (Abs (-, -, t)) = extractVars t
  | extractVars - = []
val vars = extractVars trm
val vars = fold Term.add-vars vars []
val var = hd vars
val trmty =
  case (snd var) of (Type (type-name ‹AOT-var›, [t])) => (t)
  | - => raise Term.TYPE (Expected variable type., [snd var], [Var var])
val trm = Abs (Term.string-of-vname (fst var), trmty, Term.abstract-over (
     Const (const-name ‹AOT-term-of-var›, Type (fun, [snd var, trmty]))
      $ Var var, trm))
val trm = Thm.cterm-of (Context.proof-of ctxt) trm
val ty = hd (Term.add-tvars (Thm.prop-of @{thm ∀ I}) [])
val typ = Thm.ctyp-of (Context.proof-of ctxt) trmty
val allthm = Drule.instantiate-normalize (TVars.make [(ty, typ)], Vars.empty) @{thm ∀ I}
```

```
val phi = hd (Term.add-vars (Thm.prop-of allthm) [])
val allthm = Drule.instantiate-normalize (TVars.empty, Vars.make [(phi,trm)]) allthm
in
allthm
end
›

attribute-setup ∀I =
  ‹Scan.lift (Scan.repeat1 Args.var) >> (fn args => Thm.rule-attribute []
  (fn ctxt => fn thm => fold (fn arg => fn thm =>
    thm RS get-instantiated-allI ctxt arg thm) args thm))›
  Quantify over a variable in a theorem using GEN.

attribute-setup unvarify =
  ‹Scan.lift (Scan.repeat1 Args.var) >> (fn args => Thm.rule-attribute []
  (fn ctxt => fn thm =>
    let
      fun get-inst-allI arg thm = thm RS get-instantiated-allI ctxt arg thm
      val thm = fold get-inst-allI args thm
      val thm = fold (K (fn thm => thm RS @{thm ∀E(1)})) args thm
    in
     thm
    end))›
  Generalize a statement about variables to a statement about denoting terms.


AOT-theorem cqt−basic:1: ‹∀α∀β φ{α,β} ≡ ∀β∀α φ{α,β}›
  by (metis ≡I ∀E(2) ∀I →I)

AOT-theorem cqt−basic:2:
  ‹∀α(φ{α} ≡ ψ{α}) ≡ (∀α(φ{α} → ψ{α}) & ∀α(ψ{α} → φ{α}))›
proof (rule ≡I; rule →I)
  AOT-assume ‹∀α(φ{α} ≡ ψ{α})›
  AOT-hence ‹φ{α} ≡ ψ{α}› for α using ∀E(2) by blast
  AOT-hence ‹φ{α} → ψ{α}› and ‹ψ{α} → φ{α}› for α
    using ≡E(1,2) →I by blast+
  AOT-thus ‹∀α(φ{α} → ψ{α}) & ∀α(ψ{α} → φ{α})›
    by (auto intro: &I ∀I)
next
  AOT-assume ‹∀α(φ{α} → ψ{α}) & ∀α(ψ{α} → φ{α})›
  AOT-hence ‹φ{α} → ψ{α}› and ‹ψ{α} → φ{α}› for α
    using ∀E(2) &E by blast+
  AOT-hence ‹φ{α} ≡ ψ{α}› for α
    using ≡I by blast
  AOT-thus ‹∀α(φ{α} ≡ ψ{α})› by (auto intro: ∀I)
qed

AOT-theorem cqt−basic:3: ‹∀α(φ{α} ≡ ψ{α}) → (∀α φ{α} ≡ ∀α ψ{α})›
proof(rule →I)
  AOT-assume ‹∀α(φ{α} ≡ ψ{α})›
  AOT-hence 1: ‹φ{α} ≡ ψ{α}› for α using ∀E(2) by blast
  {
    AOT-assume ‹∀α φ{α}›
    AOT-hence ‹∀α ψ{α}› using 1 ∀I ∀E(4) ≡E by metis
  }
  moreover {
    AOT-assume ‹∀α ψ{α}›
    AOT-hence ‹∀α φ{α}› using 1 ∀I ∀E(4) ≡E by metis
  }
  ultimately AOT-show ‹∀α φ{α} ≡ ∀α ψ{α}›
    using ≡I →I by auto
qed
```

```
AOT-theorem cqt−basic:4: ‹∀α(φ{α} & ψ{α}) → (∀α φ{α} & ∀α ψ{α})›
proof(rule →I)
  AOT-assume 0: ‹∀α(φ{α} & ψ{α})›
  AOT-have ‹φ{α}› and ‹ψ{α}› for α using ∀E(2) 0 &E by blast+
  AOT-thus ‹∀α φ{α} & ∀α ψ{α}›
    by (auto intro: ∀I &I)
qed

AOT-theorem cqt−basic:5: ‹(∀α_1...∀α_n(φ{α_1...α_n})) → φ{α_1...α_n}›
  using cqt−orig:3 by blast

AOT-theorem cqt−basic:6: ‹∀α∀α φ{α} ≡ ∀α φ{α}›
  by (meson ≡I →I GEN cqt−orig:1[const-var])

AOT-theorem cqt−basic:7: ‹(φ → ∀α ψ{α}) ≡ ∀α(φ → ψ{α})›
  by (metis →I vdash−properties:6 rule−ui:3 ≡I GEN)

AOT-theorem cqt−basic:8: ‹(∀α φ{α} ∨ ∀α ψ{α}) → ∀α (φ{α} ∨ ψ{α})›
  by (simp add: ∨I(3) →I GEN cqt−orig:1[const-var])

AOT-theorem cqt−basic:9:
  ‹(∀α (φ{α} → ψ{α}) & ∀α (ψ{α} → χ{α})) → ∀α(φ{α} → χ{α})›
proof −
  {
    AOT-assume ‹∀α (φ{α} → ψ{α})›
    moreover AOT-assume ‹∀α (ψ{α} → χ{α})›
    ultimately AOT-have ‹φ{α} → ψ{α}› and ‹ψ{α} → χ{α}› for α
      using ∀E by blast+
    AOT-hence ‹φ{α} → χ{α}› for α by (metis →E →I)
    AOT-hence ‹∀α(φ{α} → χ{α})› using ∀I by fast
  }
  thus ?thesis using &I →I &E by meson
qed

AOT-theorem cqt−basic:10:
  ‹(∀α(φ{α} ≡ ψ{α}) & ∀α(ψ{α} ≡ χ{α})) → ∀α (φ{α} ≡ χ{α})›
proof(rule →I; rule ∀I)
  fix β
  AOT-assume ‹∀α(φ{α} ≡ ψ{α}) & ∀α(ψ{α} ≡ χ{α})›
  AOT-hence ‹φ{β} ≡ ψ{β}› and ‹ψ{β} ≡ χ{β}› using &E ∀E by blast+
  AOT-thus ‹φ{β} ≡ χ{β}› using ≡I ≡E by blast
qed

AOT-theorem cqt−basic:11: ‹∀α(φ{α} ≡ ψ{α}) ≡ ∀α (ψ{α} ≡ φ{α})›
proof (rule ≡I; rule →I)
  AOT-assume 0: ‹∀α(φ{α} ≡ ψ{α})›
  {
    fix α
    AOT-have ‹φ{α} ≡ ψ{α}› using 0 ∀E by blast
    AOT-hence ‹ψ{α} ≡ φ{α}› using ≡I ≡E →I →E by metis
  }
  AOT-thus ‹∀α(ψ{α} ≡ φ{α})› using ∀I by fast
next
  AOT-assume 0: ‹∀α(ψ{α} ≡ φ{α})›
  {
    fix α
    AOT-have ‹ψ{α} ≡ φ{α}› using 0 ∀E by blast
    AOT-hence ‹φ{α} ≡ ψ{α}› using ≡I ≡E →I →E by metis
  }
  AOT-thus ‹∀α(φ{α} ≡ ψ{α})› using ∀I by fast
qed
```

AOT-theorem *cqt−basic*:*12*: ‹$\forall\alpha\ \varphi\{\alpha\} \rightarrow \forall\alpha\ (\psi\{\alpha\} \rightarrow \varphi\{\alpha\})$›
by (*simp add*: $\forall E(2)$ $\rightarrow I$ *GEN*)

AOT-theorem *cqt−basic*:*13*: ‹$\forall\alpha\ \varphi\{\alpha\} \equiv \forall\beta\ \varphi\{\beta\}$›
using $\equiv I$ $\rightarrow I$ **by** *blast*

AOT-theorem *cqt−basic*:*14*:
‹$(\forall\alpha_1...\forall\alpha_n\ (\varphi\{\alpha_1...\alpha_n\} \rightarrow \psi\{\alpha_1...\alpha_n\})) \rightarrow$
$((\forall\alpha_1...\forall\alpha_n\ \varphi\{\alpha_1...\alpha_n\}) \rightarrow (\forall\alpha_1...\forall\alpha_n\ \psi\{\alpha_1...\alpha_n\}))$›
using *cqt*:*3*[*axiom-inst*] **by** *auto*

AOT-theorem *cqt−basic*:*15*:
‹$(\forall\alpha_1...\forall\alpha_n\ (\varphi \rightarrow \psi\{\alpha_1...\alpha_n\})) \rightarrow (\varphi \rightarrow (\forall\alpha_1...\forall\alpha_n\ \psi\{\alpha_1...\alpha_n\}))$›
using *cqt−orig*:*2* **by** *auto*

AOT-theorem *universal−cor*:
assumes ‹*for arbitrary* β: $\varphi\{\beta\}$›
shows ‹$\forall\alpha\ \varphi\{\alpha\}$›
using *GEN assms* **.**

AOT-theorem *existential*:*1*:
assumes ‹$\varphi\{\tau\}$› **and** ‹$\tau\downarrow$›
shows ‹$\exists\alpha\ \varphi\{\alpha\}$›
proof(*rule raa−cor*:*1*)
AOT-assume ‹$\neg\exists\alpha\ \varphi\{\alpha\}$›
AOT-hence ‹$\forall\alpha\ \neg\varphi\{\alpha\}$›
using $\equiv_{df}I$ *conventions*:*4* *RAA* &*I* **by** *blast*
AOT-hence ‹$\neg\varphi\{\tau\}$› **using** *assms*(*2*) $\forall E(1)$ $\rightarrow E$ **by** *blast*
AOT-thus ‹$\varphi\{\tau\}\ \&\ \neg\varphi\{\tau\}$› **using** *assms*(*1*) &*I* **by** *blast*
qed

AOT-theorem *existential*:*2*[*const-var*]:
assumes ‹$\varphi\{\beta\}$›
shows ‹$\exists\alpha\ \varphi\{\alpha\}$›
using *existential*:*1* *cqt*:*2*[*const-var*][*axiom-inst*] *assms* **by** *blast*

AOT-theorem *existential*:*2*[*lambda*]:
assumes ‹$\varphi\{[\lambda\nu_1...\nu_n\ \psi\{\nu_1...\nu_n\}]\}$› **and** ‹*INSTANCE-OF-CQT-2*(ψ)›
shows ‹$\exists\alpha\ \varphi\{\alpha\}$›
using *existential*:*1* *cqt*:*2*[*lambda*][*axiom-inst*] *assms* **by** *blast*
lemmas $\exists I$ = *existential*:*1* *existential*:*2*[*const-var*]
existential:*2*[*lambda*]

AOT-theorem *instantiation*:
assumes ‹*for arbitrary* β: $\varphi\{\beta\} \vdash \psi$› **and** ‹$\exists\alpha\ \varphi\{\alpha\}$›
shows ‹ψ›
by (*metis* (*no-types*, *lifting*) $\equiv_{df}E$ *GEN raa−cor*:*3* *conventions*:*4* *assms*)
lemmas $\exists E$ = *instantiation*

AOT-theorem *cqt−further*:*1*: ‹$\forall\alpha\ \varphi\{\alpha\} \rightarrow \exists\alpha\ \varphi\{\alpha\}$›
using $\forall E(4)$ $\exists I(2)$ $\rightarrow I$ **by** *metis*

AOT-theorem *cqt−further*:*2*: ‹$\neg\forall\alpha\ \varphi\{\alpha\} \rightarrow \exists\alpha\ \neg\varphi\{\alpha\}$›
using $\forall I$ $\exists I(2)$ $\rightarrow I$ *RAA* **by** *metis*

AOT-theorem *cqt−further*:*3*: ‹$\forall\alpha\ \varphi\{\alpha\} \equiv \neg\exists\alpha\ \neg\varphi\{\alpha\}$›
using $\forall E(4)$ $\exists E$ $\rightarrow I$ *RAA*
by (*metis cqt−further*:*2* $\equiv I$ *modus−tollens*:*1*)

AOT-theorem *cqt−further*:*4*: ‹$\neg\exists\alpha\ \varphi\{\alpha\} \rightarrow \forall\alpha\ \neg\varphi\{\alpha\}$›
using $\forall I$ $\exists I(2)$$\rightarrow I$ *RAA* **by** *metis*

AOT-theorem *cqt−further*:*5*: ‹$\exists\alpha\ (\varphi\{\alpha\}\ \&\ \psi\{\alpha\}) \rightarrow (\exists\alpha\ \varphi\{\alpha\}\ \&\ \exists\alpha\ \psi\{\alpha\})$›

by (*metis* (*no-types*, *lifting*) &*E* &*I* ∃ *E* ∃ *I*(*2*) →*I*)

AOT-theorem *cqt−further*:*6*: ‹∃ α (φ{α} ∨ ψ{α}) → (∃ α φ{α} ∨ ∃ α ψ{α})›
by (*metis* (*mono-tags*, *lifting*) ∃ *E* ∃ *I*(*2*) ∨*E*(*3*) ∨*I*(*1*, *2*) →*I* *RAA*(*2*))

AOT-theorem *cqt−further*:*7*: ‹∃ α φ{α} ≡ ∃ β φ{β}›
by (*simp add*: *oth−class−taut*:*3*:*a*)

AOT-theorem *cqt−further*:*8*:
‹(∀ α φ{α} & ∀ α ψ{α}) → ∀ α (φ{α} ≡ ψ{α})›
by (*metis* (*mono-tags*, *lifting*) &*E* ≡*I* ∀ *E*(*2*) →*I* *GEN*)

AOT-theorem *cqt−further*:*9*:
‹(¬∃ α φ{α} & ¬∃ α ψ{α}) → ∀ α (φ{α} ≡ ψ{α})›
by (*metis* (*mono-tags*, *lifting*) &*E* ≡*I* ∃ *I*(*2*) →*I* *GEN* *raa−cor*:*4*)

AOT-theorem *cqt−further*:*10*:
‹(∃ α φ{α} & ¬∃ α ψ{α}) → ¬∀ α (φ{α} ≡ ψ{α})›
proof(*rule* →*I*; *rule raa−cor*:*2*)
AOT-assume *0*: ‹∃ α φ{α} & ¬∃ α ψ{α}›
then AOT-obtain α **where** ‹φ{α}› **using** ∃ *E* &*E*(*1*) **by** *metis*
moreover AOT-assume ‹∀ α (φ{α} ≡ ψ{α})›
ultimately AOT-have ‹ψ{α}› **using** ∀ *E*(*4*) ≡*E*(*1*) **by** *blast*
AOT-hence ‹∃ α ψ{α}› **using** ∃ *I* **by** *blast*
AOT-thus ‹∃ α ψ{α} & ¬∃ α ψ{α}› **using** *0* &*E*(*2*) &*I* **by** *blast*
qed

AOT-theorem *cqt−further*:*11*: ‹∃ α∃ β φ{α,β} ≡ ∃ β∃ α φ{α,β}›
using ≡*I* →*I* ∃ *I*(*2*) ∃ *E* **by** *metis*

8.7 Logical Existence, Identity, and Truth

AOT-theorem *log−prop−prop*:*1*: ‹[λ φ]↓›
using *cqt*:*2*[*lambda0*][*axiom-inst*] **by** *auto*

AOT-theorem *log−prop−prop*:*2*: ‹φ↓›
by (*rule* $\equiv_{df}$*I*[*OF existence*:*3*]) *cqt*:*2*[*lambda*]

AOT-theorem *exist−nec*: ‹τ↓ → □τ↓›
proof −
AOT-have ‹∀ β □β↓›
by (*simp add*: *GEN RN cqt*:*2*[*const-var*][*axiom-inst*])
AOT-thus ‹τ↓ → □τ↓›
using *cqt*:*1*[*axiom-inst*] →*E* **by** *blast*
qed

class *AOT-Term-id* = *AOT-Term* +
assumes *t*=*t−proper*:*1*[*AOT*]: ‹[*v* ⊨ τ = τ′ → τ↓]›
and *t*=*t−proper*:*2*[*AOT*]: ‹[*v* ⊨ τ = τ′ → τ′↓]›

instance κ :: *AOT-Term-id*
proof
AOT-modally-strict {
AOT-show ‹κ = κ′ → κ↓› **for** κ κ′
proof(*rule* →*I*)
AOT-assume ‹κ = κ′›
AOT-hence ‹*O*!κ ∨ *A*!κ›
by (*rule* ∨*I*(*3*)[*OF* $\equiv_{df}$*E*[*OF identity*:*1*]])
(*meson* →*I* ∨*I*(*1*) &*E*(*1*))+
AOT-thus ‹κ↓›
by (*rule* ∨*E*(*1*))

```
      (metis cqt:5:a[axiom-inst] →I →E &E(2))+
  qed
 }
next
 AOT-modally-strict {
  AOT-show ‹κ = κ' → κ'↓› for κ κ'
  proof(rule →I)
   AOT-assume ‹κ = κ'›
   AOT-hence ‹O!κ' ∨ A!κ'›
    by (rule ∨I(3)[OF ≡df E[OF identity:1]])
      (meson →I ∨I &E)+
   AOT-thus ‹κ'↓›
    by (rule ∨E(1))
      (metis cqt:5:a[axiom-inst] →I →E &E(2))+
  qed
 }
qed

instance rel :: (AOT-κs) AOT-Term-id
proof
 AOT-modally-strict {
  AOT-show ‹Π = Π' → Π↓› for Π Π' :: ‹<'a>›
  proof(rule →I)
   AOT-assume ‹Π = Π'›
   AOT-thus ‹Π↓› using ≡df E[OF identity:3[of Π Π']] &E by blast
  qed
 }
next
 AOT-modally-strict {
  AOT-show ‹Π = Π' → Π'↓› for Π Π' :: ‹<'a>›
  proof(rule →I)
   AOT-assume ‹Π = Π'›
   AOT-thus ‹Π'↓› using ≡df E[OF identity:3[of Π Π']] &E by blast
  qed
 }
qed

instance o :: AOT-Term-id
proof
 AOT-modally-strict {
  fix φ ψ
  AOT-show ‹φ = ψ → φ↓›
  proof(rule →I)
   AOT-assume ‹φ = ψ›
   AOT-thus ‹φ↓› using ≡df E[OF identity:4[of φ ψ]] &E by blast
  qed
 }
next
 AOT-modally-strict {
  fix φ ψ
  AOT-show ‹φ = ψ → ψ↓›
  proof(rule →I)
   AOT-assume ‹φ = ψ›
   AOT-thus ‹ψ↓› using ≡df E[OF identity:4[of φ ψ]] &E by blast
  qed
 }
qed

instance prod :: (AOT-Term-id, AOT-Term-id) AOT-Term-id
proof
 AOT-modally-strict {
  fix τ τ' :: ‹'a×'b›
  AOT-show ‹τ = τ' → τ↓›
```

```
    proof (induct τ; induct τ'; rule →I)
      fix τ₁ τ₁' :: 'a and τ₂  τ₂' :: 'b
      AOT-assume ‹«(τ₁, τ₂)» = «(τ₁', τ₂')»›
      AOT-hence ‹(τ₁ = τ₁') & (τ₂ = τ₂')› by (metis ≡df E tuple-identity-1)
      AOT-hence ‹τ₁↓› and ‹τ₂↓›
        using t=t−proper:1 &E vdash−properties:10 by blast+
      AOT-thus ‹«(τ₁, τ₂)»↓› by (metis ≡df I &I tuple-denotes)
    qed
  }
next
  AOT-modally-strict {
    fix τ τ' :: ‹'a×'b›
    AOT-show ‹τ = τ' → τ'↓›
    proof (induct τ; induct τ'; rule →I)
      fix τ₁ τ₁' :: 'a and τ₂  τ₂' :: 'b
      AOT-assume ‹«(τ₁, τ₂)» = «(τ₁', τ₂')»›
      AOT-hence ‹(τ₁ = τ₁') & (τ₂ = τ₂')› by (metis ≡df E tuple-identity-1)
      AOT-hence ‹τ₁'↓› and ‹τ₂'↓›
        using t=t−proper:2 &E vdash−properties:10 by blast+
      AOT-thus ‹«(τ₁', τ₂')»↓› by (metis ≡df I &I tuple-denotes)
    qed
  }
qed


AOT-register-type-constraints
  Term: ‹-::AOT-Term-id› ‹-::AOT-Term-id›
AOT-register-type-constraints
  Individual: ‹κ› ‹-::{AOT-κs, AOT-Term-id}›
AOT-register-type-constraints
  Relation: ‹<-::{AOT-κs, AOT-Term-id}>›

AOT-theorem id−rel−nec−equiv:1:
  ‹Π = Π' → □∀x₁...∀xₙ ([Π]x₁...xₙ ≡ [Π']x₁...xₙ)›
proof(rule →I)
  AOT-assume assumption: ‹Π = Π'›
  AOT-hence ‹Π↓› and ‹Π'↓›
    using t=t−proper:1 t=t−proper:2 MP by blast+
  moreover AOT-have ‹∀F∀G (F = G → ((□∀x₁...∀xₙ ([F]x₁...xₙ ≡ [F]x₁...xₙ)) →
                                  □∀x₁...∀xₙ ([F]x₁...xₙ ≡ [G]x₁...xₙ))))›
    apply (rule GEN)+ using l−identity[axiom-inst] by force
  ultimately AOT-have ‹Π = Π' → ((□∀x₁...∀xₙ ([Π]x₁...xₙ ≡ [Π]x₁...xₙ)) →
                                □∀x₁...∀xₙ ([Π]x₁...xₙ ≡ [Π']x₁...xₙ)))›
    using ∀E(1) by blast
  AOT-hence ‹(□∀x₁...∀xₙ ([Π]x₁...xₙ ≡ [Π]x₁...xₙ)) →
            □∀x₁...∀xₙ ([Π]x₁...xₙ ≡ [Π']x₁...xₙ)›
    using assumption →E by blast
  moreover AOT-have ‹□∀x₁...∀xₙ ([Π]x₁...xₙ ≡ [Π]x₁...xₙ)›
    by (simp add: RN oth−class−taut:3:a universal−cor)
  ultimately AOT-show ‹□∀x₁...∀xₙ ([Π]x₁...xₙ ≡ [Π']x₁...xₙ)›
    using →E by blast
qed

AOT-theorem id−rel−nec−equiv:2: ‹φ = ψ → □(φ ≡ ψ)›
proof(rule →I)
  AOT-assume assumption: ‹φ = ψ›
  AOT-hence ‹φ↓› and ‹ψ↓›
    using t=t−proper:1 t=t−proper:2 MP by blast+
  moreover AOT-have ‹∀p∀q (p = q → ((□(p ≡ p) → □(p ≡ q))))›
    apply (rule GEN)+ using l−identity[axiom-inst] by force
  ultimately AOT-have ‹φ = ψ → (□(φ ≡ φ) → □(φ ≡ ψ))›
    using ∀E(1) by blast
  AOT-hence ‹□(φ ≡ φ) → □(φ ≡ ψ)›
```

using *assumption* →*E* **by** *blast*
moreover AOT-have ‹□(φ ≡ φ)›
by (*simp add*: *RN oth−class−taut*:*3*:*a universal−cor*)
ultimately AOT-show ‹□(φ ≡ ψ)›
using →*E* **by** *blast*
qed

AOT-theorem *rule*=*E*:
assumes ‹φ{τ}› **and** ‹τ = σ›
shows ‹φ{σ}›
proof −
AOT-have ‹τ↓› **and** ‹σ↓›
using *assms*(*2*) *t*=*t−proper*:*1 t*=*t−proper*:*2* →*E* **by** *blast*+
moreover AOT-have ‹∀α∀β(α = β → (φ{α} → φ{β}))›
apply (*rule GEN*)+ **using** *l−identity*[*axiom-inst*] **by** *blast*
ultimately AOT-have ‹τ = σ → (φ{τ} → φ{σ})›
using ∀*E*(*1*) **by** *blast*
AOT-thus ‹φ{σ}› **using** *assms* →*E* **by** *blast*
qed

AOT-theorem *propositions−lemma*:*1*: ‹[λ φ] = φ›
proof −
AOT-have ‹φ↓› **by** (*simp add*: *log−prop−prop*:*2*)
moreover AOT-have ‹∀*p* [λ *p*] = *p*›
using *lambda−predicates*:*3*[*zero*][*axiom-inst*] ∀*I* **by** *fast*
ultimately AOT-show ‹[λ φ] = φ›
using ∀*E* **by** *blast*
qed

AOT-theorem *propositions−lemma*:*2*: ‹[λ φ] ≡ φ›
proof −
AOT-have ‹[λ φ] ≡ [λ φ]› **by** (*simp add*: *oth−class−taut*:*3*:*a*)
AOT-thus ‹[λ φ] ≡ φ› **using** *propositions−lemma*:*1 rule*=*E* **by** *blast*
qed

propositions-lemma:3 through propositions-lemma:5 hold implicitly

AOT-theorem *propositions−lemma*:*6*: ‹(φ ≡ ψ) ≡ ([λ φ] ≡ [λ ψ])›
by (*metis* ≡*E*(*1*) ≡*E*(*5*) *Associativity of* ≡ *propositions−lemma*:*2*)

dr-alphabetic-rules holds implicitly

AOT-theorem *oa−exist*:*1*: ‹*O*!↓›
proof −
AOT-have ‹[λ*x* ◊[*E*!]*x*]↓› **by** *cqt*:*2*[*lambda*]
AOT-hence *1*: ‹*O*! = [λ*x* ◊[*E*!]*x*]›
using *df−rules−terms*[*4*][*OF oa*:*1*, *THEN* &*E*(*1*)] →*E* **by** *blast*
AOT-show ‹*O*!↓› **using** *t*=*t−proper*:*1*[*THEN* →*E*, *OF 1*] **by** *simp*
qed

AOT-theorem *oa−exist*:*2*: ‹*A*!↓›
proof −
AOT-have ‹[λ*x* ¬◊[*E*!]*x*]↓› **by** *cqt*:*2*[*lambda*]
AOT-hence *1*: ‹*A*! = [λ*x* ¬◊[*E*!]*x*]›
using *df−rules−terms*[*4*][*OF oa*:*2*, *THEN* &*E*(*1*)] →*E* **by** *blast*
AOT-show ‹*A*!↓› **using** *t*=*t−proper*:*1*[*THEN* →*E*, *OF 1*] **by** *simp*
qed

AOT-theorem *oa−exist*:*3*: ‹*O*!*x* ∨ *A*!*x*›
proof(*rule raa−cor*:*1*)
AOT-assume ‹¬(*O*!*x* ∨ *A*!*x*)›
AOT-hence *A*: ‹¬*O*!*x*› **and** *B*: ‹¬*A*!*x*›
using *Disjunction Addition*(*1*) *modus−tollens*:*1*
∨*I*(*2*) *raa−cor*:*5* **by** *blast*+
AOT-have *C*: ‹*O*! = [λ*x* ◊[*E*!]*x*]›

by (*rule df−rules−terms*[*4*][*OF oa*:*1*, *THEN* &*E*(*1*), *THEN* →*E*]) *cqt*:*2*
AOT-have *D*: ‹$A! = [\lambda x\ \neg\Diamond[E!]x]$›
by (*rule df−rules−terms*[*4*][*OF oa*:*2*, *THEN* &*E*(*1*), *THEN* →*E*]) *cqt*:*2*
AOT-have *E*: ‹$\neg[\lambda x\ \Diamond[E!]x]x$›
using *A C rule*=*E* **by** *fast*
AOT-have *F*: ‹$\neg[\lambda x\ \neg\Diamond[E!]x]x$›
using *B D rule*=*E* **by** *fast*
AOT-have *G*: ‹$[\lambda x\ \Diamond[E!]x]x \equiv \Diamond[E!]x$›
by (*rule lambda−predicates*:*2*[*axiom-inst*, *THEN* →*E*]) *cqt*:*2*
AOT-have *H*: ‹$[\lambda x\ \neg\Diamond[E!]x]x \equiv \neg\Diamond[E!]x$›
by (*rule lambda−predicates*:*2*[*axiom-inst*, *THEN* →*E*]) *cqt*:*2*
AOT-show ‹$\neg\Diamond[E!]x\ \&\ \neg\neg\Diamond[E!]x$› **using** *G E* ≡*E H F* ≡*E* &*I* **by** *metis*
qed

AOT-theorem *p−identity−thm2*:*1*: ‹$F = G \equiv \Box\forall x(x[F] \equiv x[G])$›
proof −
AOT-have ‹$F = G \equiv F\downarrow\ \&\ G\downarrow\ \&\ \Box\forall x(x[F] \equiv x[G])$›
using *identity*:*2 df−rules−formulas*[*3*] *df−rules−formulas*[*4*]
→*E* &*E* ≡*I* →*I* **by** *blast*
moreover AOT-have ‹$F\downarrow$› **and** ‹$G\downarrow$›
by (*auto simp*: *cqt*:*2*[*const-var*][*axiom-inst*])
ultimately AOT-show ‹$F = G \equiv \Box\forall x(x[F] \equiv x[G])$›
using ≡*S*(*1*) &*I* **by** *blast*
qed

AOT-theorem *p−identity−thm2*:*2*[*2*]:
‹$F = G \equiv \forall y_1([\lambda x\ [F]xy_1] = [\lambda x\ [G]xy_1]\ \&\ [\lambda x\ [F]y_1x] = [\lambda x\ [G]y_1x])$›
proof −
AOT-have ‹$F = G \equiv F\downarrow\ \&\ G\downarrow\ \&$
$\forall y_1([\lambda x\ [F]xy_1] = [\lambda x\ [G]xy_1]\ \&\ [\lambda x\ [F]y_1x] = [\lambda x\ [G]y_1x])$›
using *identity*:*3*[*2*] *df−rules−formulas*[*3*] *df−rules−formulas*[*4*]
→*E* &*E* ≡*I* →*I* **by** *blast*
moreover AOT-have ‹$F\downarrow$› **and** ‹$G\downarrow$›
by (*auto simp*: *cqt*:*2*[*const-var*][*axiom-inst*])
ultimately show *?thesis*
using ≡*S*(*1*) &*I* **by** *blast*
qed

AOT-theorem *p−identity−thm2*:*2*[*3*]:
‹$F = G \equiv \forall y_1\forall y_2([\lambda x\ [F]xy_1y_2] = [\lambda x\ [G]xy_1y_2]\ \&$
$[\lambda x\ [F]y_1xy_2] = [\lambda x\ [G]y_1xy_2]\ \&$
$[\lambda x\ [F]y_1y_2x] = [\lambda x\ [G]y_1y_2x])$›
proof −
AOT-have ‹$F = G \equiv F\downarrow\ \&\ G\downarrow\ \&\ \forall y_1\forall y_2([\lambda x\ [F]xy_1y_2] = [\lambda x\ [G]xy_1y_2]\ \&$
$[\lambda x\ [F]y_1xy_2] = [\lambda x\ [G]y_1xy_2]\ \&$
$[\lambda x\ [F]y_1y_2x] = [\lambda x\ [G]y_1y_2x])$›
using *identity*:*3*[*3*] *df−rules−formulas*[*3*] *df−rules−formulas*[*4*]
→*E* &*E* ≡*I* →*I* **by** *blast*
moreover AOT-have ‹$F\downarrow$› **and** ‹$G\downarrow$›
by (*auto simp*: *cqt*:*2*[*const-var*][*axiom-inst*])
ultimately show *?thesis*
using ≡*S*(*1*) &*I* **by** *blast*
qed

AOT-theorem *p−identity−thm2*:*2*[*4*]:
‹$F = G \equiv \forall y_1\forall y_2\forall y_3([\lambda x\ [F]xy_1y_2y_3] = [\lambda x\ [G]xy_1y_2y_3]\ \&$
$[\lambda x\ [F]y_1xy_2y_3] = [\lambda x\ [G]y_1xy_2y_3]\ \&$
$[\lambda x\ [F]y_1y_2xy_3] = [\lambda x\ [G]y_1y_2xy_3]\ \&$
$[\lambda x\ [F]y_1y_2y_3x] = [\lambda x\ [G]y_1y_2y_3x])$›
proof −
AOT-have ‹$F = G \equiv F\downarrow\ \&\ G\downarrow\ \&\ \forall y_1\forall y_2\forall y_3([\lambda x\ [F]xy_1y_2y_3] = [\lambda x\ [G]xy_1y_2y_3]\ \&$
$[\lambda x\ [F]y_1xy_2y_3] = [\lambda x\ [G]y_1xy_2y_3]\ \&$
$[\lambda x\ [F]y_1y_2xy_3] = [\lambda x\ [G]y_1y_2xy_3]\ \&$

```
                                [λx [F]y1y2y3x] = [λx [G]y1y2y3x])›
    using identity:3[4] df−rules−formulas[3] df−rules−formulas[4]
          →E &E ≡I →I by blast
  moreover AOT-have ‹F↓› and ‹G↓›
    by (auto simp: cqt:2[const-var][axiom-inst])
  ultimately show ?thesis
    using ≡S(1) &I by blast
qed

AOT-theorem p−identity−thm2:2:
  ‹F = G ≡ ∀x1...∀xn «AOT-sem-proj-id x1xn (λ τ . «[F]τ») (λ τ . «[G]τ»)»›
proof −
  AOT-have ‹F = G ≡ F↓ & G↓ &
              ∀x1...∀xn «AOT-sem-proj-id x1xn (λ τ . «[F]τ») (λ τ . «[G]τ»)»›
    using identity:3 df−rules−formulas[3] df−rules−formulas[4]
          →E &E ≡I →I by blast
  moreover AOT-have ‹F↓› and ‹G↓›
    by (auto simp: cqt:2[const-var][axiom-inst])
  ultimately show ?thesis
    using ≡S(1) &I by blast
qed

AOT-theorem p−identity−thm2:3:
  ‹p = q ≡ [λx p] = [λx q]›
proof −
  AOT-have ‹p = q ≡ p↓ & q↓ & [λx p] = [λx q]›
    using identity:4 df−rules−formulas[3] df−rules−formulas[4]
          →E &E ≡I →I by blast
  moreover AOT-have ‹p↓› and ‹q↓›
    by (auto simp: cqt:2[const-var][axiom-inst])
  ultimately show ?thesis
    using ≡S(1) &I by blast
qed

class AOT-Term-id-2 = AOT-Term-id + assumes id−eq:1: ‹[v ⊨ α = α]›

instance κ :: AOT-Term-id-2
proof
  AOT-modally-strict {
    fix x
    {
      AOT-assume ‹O!x›
      moreover AOT-have ‹□∀F([F]x ≡ [F]x)›
        using RN GEN oth−class−taut:3:a by fast
      ultimately AOT-have ‹O!x & O!x & □∀F([F]x ≡ [F]x)› using &I by simp
    }
    moreover {
      AOT-assume ‹A!x›
      moreover AOT-have ‹□∀F(x[F] ≡ x[F])›
        using RN GEN oth−class−taut:3:a by fast
      ultimately AOT-have ‹A!x & A!x & □∀F(x[F] ≡ x[F])› using &I by simp
    }
    ultimately AOT-have ‹(O!x & O!x & □∀F([F]x ≡ [F]x)) ∨
                         (A!x & A!x & □∀F(x[F] ≡ x[F]))›
      using oa−exist:3 ∨I(1) ∨I(2) ∨E(3) raa−cor:1 by blast
    AOT-thus ‹x = x›
      using identity:1[THEN df−rules−formulas[4]] →E by blast
  }
qed

instance rel :: ({AOT-κs,AOT-Term-id-2}) AOT-Term-id-2
proof
  AOT-modally-strict {
```

```
    fix F :: <'a> AOT-var
    AOT-have 0: ‹[λx1...xn [F]x1...xn] = F›
      by (simp add: lambda−predicates:3[axiom-inst])
    AOT-have ‹[λx1...xn [F]x1...xn]↓›
      by cqt:2[lambda]
    AOT-hence ‹[λx1...xn [F]x1...xn] = [λx1...xn [F]x1...xn]›
      using lambda−predicates:1[axiom-inst] →E by blast
    AOT-show ‹F = F› using rule=E 0 by force
  }
qed

instance o :: AOT-Term-id-2
proof
  AOT-modally-strict {
    fix p
    AOT-have 0: ‹[λ p] = p›
      by (simp add: lambda−predicates:3[zero][axiom-inst])
    AOT-have ‹[λ p]↓›
      by (rule cqt:2[lambda0][axiom-inst])
    AOT-hence ‹[λ p] = [λ p]›
      using lambda−predicates:1[zero][axiom-inst] →E by blast
    AOT-show ‹p = p› using rule=E 0 by force
  }
qed

instance prod :: (AOT-Term-id-2, AOT-Term-id-2) AOT-Term-id-2
proof
  AOT-modally-strict {
    fix α :: ‹('a×'b) AOT-var›
    AOT-show ‹α = α›
    proof (induct)
      AOT-show ‹τ = τ› if ‹τ↓› for τ :: ‹'a×'b›
        using that
      proof (induct τ)
        fix τ1 :: 'a and τ2 :: 'b
        AOT-assume ‹«(τ1,τ2)»↓›
        AOT-hence ‹τ1↓› and ‹τ2↓›
          using ≡df E &E tuple-denotes by blast+
        AOT-hence ‹τ1 = τ1› and ‹τ2 = τ2›
          using id−eq:1[unvarify α] by blast+
        AOT-thus ‹«(τ1, τ2)» = «(τ1, τ2)»›
          by (metis ≡df I &I tuple-identity-1)
      qed
    qed
  }
qed

AOT-register-type-constraints
  Term: ‹-::AOT-Term-id-2› ‹-::AOT-Term-id-2›
AOT-register-type-constraints
  Individual: ‹κ› ‹-::{AOT-κs, AOT-Term-id-2}›
AOT-register-type-constraints
  Relation: ‹<-::{AOT-κs, AOT-Term-id-2}>›

AOT-theorem id−eq:2: ‹α = β → β = α›
  by (meson rule=E deduction−theorem)

AOT-theorem id−eq:3: ‹α = β & β = γ → α = γ›
  using rule=E →I &E by blast

AOT-theorem id−eq:4: ‹α = β ≡ ∀γ (α = γ ≡ β = γ)›
proof (rule ≡I; rule →I)
  AOT-assume 0: ‹α = β›
```

```
  AOT-hence 1: ‹β = α› using id−eq:2 →E by blast
  AOT-show ‹∀γ (α = γ ≡ β = γ)›
    by (rule GEN) (metis ≡I →I 0 1 rule=E)
next
  AOT-assume ‹∀γ (α = γ ≡ β = γ)›
  AOT-hence ‹α = α ≡ β = α› using ∀E(2) by blast
  AOT-hence ‹α = α → β = α› using ≡E(1) →I by blast
  AOT-hence ‹β = α› using id−eq:1 →E by blast
  AOT-thus ‹α = β› using id−eq:2 →E by blast
qed

AOT-theorem rule=I:1:
  assumes ‹τ↓›
  shows ‹τ = τ›
proof −
  AOT-have ‹∀α (α = α)›
    by (rule GEN) (metis id−eq:1)
  AOT-thus ‹τ = τ› using assms ∀E by blast
qed

AOT-theorem rule=I:2[const-var]: α = α
  using id−eq:1.

AOT-theorem rule=I:2[lambda]:
  assumes ‹INSTANCE-OF-CQT-2(φ)›
  shows [λν1...νn φ{ν1...νn}] = [λν1...νn φ{ν1...νn}]
proof −
  AOT-have ‹∀α (α = α)›
    by (rule GEN) (metis id−eq:1)
  moreover AOT-have ‹[λν1...νn φ{ν1...νn}]↓›
    using assms by (rule cqt:2[lambda][axiom-inst])
  ultimately AOT-show ‹[λν1...νn φ{ν1...νn}] = [λν1...νn φ{ν1...νn}]›
    using assms ∀E by blast
qed

lemmas =I = rule=I:1 rule=I:2[const-var] rule=I:2[lambda]

AOT-theorem rule−id−df:1:
  assumes ‹τ{α1...αn} =df σ{α1...αn}› and ‹σ{τ1...τn}↓›
  shows ‹τ{τ1...τn} = σ{τ1...τn}›
proof −
  AOT-have ‹σ{τ1...τn}↓ → τ{τ1...τn} = σ{τ1...τn}›
    using df−rules−terms[3] assms(1) &E by blast
  AOT-thus ‹τ{τ1...τn} = σ{τ1...τn}›
    using assms(2) →E by blast
qed

AOT-theorem rule−id−df:1[zero]:
  assumes ‹τ =df σ› and ‹σ↓›
  shows ‹τ = σ›
proof −
  AOT-have ‹σ↓ → τ = σ›
    using df−rules−terms[4] assms(1) &E by blast
  AOT-thus ‹τ = σ›
    using assms(2) →E by blast
qed

AOT-theorem rule−id−df:2:a:
  assumes ‹τ{α1...αn} =df σ{α1...αn}› and ‹σ{τ1...τn}↓› and ‹φ{τ{τ1...τn}}›
  shows ‹φ{σ{τ1...τn}}›
proof −
  AOT-have ‹τ{τ1...τn} = σ{τ1...τn}› using rule−id−df:1 assms(1,2) by blast
  AOT-thus ‹φ{σ{τ1...τn}}› using assms(3) rule=E by blast
```

qed

AOT-theorem *rule–id–df:2:a*[*2*]:
assumes ‹$\tau\{«(\alpha_1,\alpha_2)»\} =_{df} \sigma\{«(\alpha_1,\alpha_2)»\}$›
and ‹$\sigma\{«(\tau_1,\tau_2)»\}\downarrow$›
and ‹$\varphi\{\tau\{«(\tau_1,\tau_2)»\}\}$›
shows ‹$\varphi\{\sigma\{«(\tau_1::'a::AOT\text{-}Term\text{-}id\text{-}2,\tau_2::'b::AOT\text{-}Term\text{-}id\text{-}2)»\}\}$›
proof –
AOT-have ‹$\tau\{«(\tau_1,\tau_2)»\} = \sigma\{«(\tau_1,\tau_2)»\}$›
using *rule–id–df:1 assms*(*1*,*2*) **by** *auto*
AOT-thus ‹$\varphi\{\sigma\{«(\tau_1,\tau_2)»\}\}$› **using** *assms*(*3*) *rule=E* **by** *blast*
qed

AOT-theorem *rule–id–df:2:a*[*zero*]:
assumes ‹$\tau =_{df} \sigma$› **and** ‹$\sigma\downarrow$› **and** ‹$\varphi\{\tau\}$›
shows ‹$\varphi\{\sigma\}$›
proof –
AOT-have ‹$\tau = \sigma$› **using** *rule–id–df:1*[*zero*] *assms*(*1*,*2*) **by** *blast*
AOT-thus ‹$\varphi\{\sigma\}$› **using** *assms*(*3*) *rule=E* **by** *blast*
qed

lemmas $=_{df}E$ = *rule–id–df:2:a rule–id–df:2:a*[*zero*]

AOT-theorem *rule–id–df:2:b*:
assumes ‹$\tau\{\alpha_1...\alpha_n\} =_{df} \sigma\{\alpha_1...\alpha_n\}$› **and** ‹$\sigma\{\tau_1...\tau_n\}\downarrow$› **and** ‹$\varphi\{\sigma\{\tau_1...\tau_n\}\}$›
shows ‹$\varphi\{\tau\{\tau_1...\tau_n\}\}$›
proof –
AOT-have ‹$\tau\{\tau_1...\tau_n\} = \sigma\{\tau_1...\tau_n\}$›
using *rule–id–df:1 assms*(*1*,*2*) **by** *blast*
AOT-hence ‹$\sigma\{\tau_1...\tau_n\} = \tau\{\tau_1...\tau_n\}$›
using *rule=E =I*(*1*) *t=t–proper:1* $\rightarrow E$ **by** *fast*
AOT-thus ‹$\varphi\{\tau\{\tau_1...\tau_n\}\}$› **using** *assms*(*3*) *rule=E* **by** *blast*
qed

AOT-theorem *rule–id–df:2:b*[*2*]:
assumes ‹$\tau\{«(\alpha_1,\alpha_2)»\} =_{df} \sigma\{«(\alpha_1,\alpha_2)»\}$›
and ‹$\sigma\{«(\tau_1,\tau_2)»\}\downarrow$›
and ‹$\varphi\{\sigma\{«(\tau_1,\tau_2)»\}\}$›
shows ‹$\varphi\{\tau\{«(\tau_1::'a::AOT\text{-}Term\text{-}id\text{-}2,\tau_2::'b::AOT\text{-}Term\text{-}id\text{-}2)»\}\}$›
proof –
AOT-have ‹$\tau\{«(\tau_1,\tau_2)»\} = \sigma\{«(\tau_1,\tau_2)»\}$›
using *=I*(*1*) *rule–id–df:2:a*[*2*] *RAA*(*1*) *assms*(*1*,*2*) $\rightarrow I$ **by** *metis*
AOT-hence ‹$\sigma\{«(\tau_1,\tau_2)»\} = \tau\{«(\tau_1,\tau_2)»\}$›
using *rule=E =I*(*1*) *t=t–proper:1* $\rightarrow E$ **by** *fast*
AOT-thus ‹$\varphi\{\tau\{«(\tau_1,\tau_2)»\}\}$› **using** *assms*(*3*) *rule=E* **by** *blast*
qed

AOT-theorem *rule–id–df:2:b*[*zero*]:
assumes ‹$\tau =_{df} \sigma$› **and** ‹$\sigma\downarrow$› **and** ‹$\varphi\{\sigma\}$›
shows ‹$\varphi\{\tau\}$›
proof –
AOT-have ‹$\tau = \sigma$› **using** *rule–id–df:1*[*zero*] *assms*(*1*,*2*) **by** *blast*
AOT-hence ‹$\sigma = \tau$›
using *rule=E =I*(*1*) *t=t–proper:1* $\rightarrow E$ **by** *fast*
AOT-thus ‹$\varphi\{\tau\}$› **using** *assms*(*3*) *rule=E* **by** *blast*
qed

lemmas $=_{df}I$ = *rule–id–df:2:b rule–id–df:2:b*[*zero*]

AOT-theorem *free–thms:1*: ‹$\tau\downarrow \equiv \exists\beta\ (\beta = \tau)$›
by (*metis* $\exists E$ *rule=I:1 t=t–proper:2* $\rightarrow I$ $\exists I$(*1*) $\equiv I$ $\rightarrow E$)

AOT-theorem *free–thms:2*: ‹$\forall\alpha\ \varphi\{\alpha\} \rightarrow (\exists\beta\ (\beta = \tau) \rightarrow \varphi\{\tau\})$›

by (*metis* $\exists E$ *rule=E cqt:2[const-var][axiom-inst]* $\rightarrow I$ $\forall E(1)$)

AOT-theorem *free−thms:3[const-var]*: ‹$\exists\beta\ (\beta = \alpha)$›
by (*meson* $\exists I(2)$ *id−eq:1*)

AOT-theorem *free−thms:3[lambda]*:
assumes ‹*INSTANCE-OF-CQT-2*(φ)›
shows ‹$\exists\beta\ (\beta = [\lambda\nu_1...\nu_n\ \varphi\{\nu_1...\nu_n\}])$›
by (*meson* $=I(3)$ *assms cqt:2[lambda][axiom-inst] existential:1*)

AOT-theorem *free−thms:4[rel]*:
‹$([\Pi]\kappa_1...\kappa_n \vee \kappa_1...\kappa_n[\Pi]) \rightarrow \exists\beta\ (\beta = \Pi)$›
by (*metis rule=I:1* $\&E(1)$ $\vee E(1)$ *cqt:5:a[axiom-inst]*
cqt:5:b[axiom-inst] $\rightarrow I$ $\exists I(1)$)

AOT-theorem *free−thms:4[vars]*:
‹$([\Pi]\kappa_1...\kappa_n \vee \kappa_1...\kappa_n[\Pi]) \rightarrow \exists\beta_1...\exists\beta_n\ (\beta_1...\beta_n = \kappa_1...\kappa_n)$›
by (*metis rule=I:1* $\&E(2)$ $\vee E(1)$ *cqt:5:a[axiom-inst]*
cqt:5:b[axiom-inst] $\rightarrow I$ $\exists I(1)$)

AOT-theorem *free−thms:4[1,rel]*:
‹$([\Pi]\kappa \vee \kappa[\Pi]) \rightarrow \exists\beta\ (\beta = \Pi)$›
by (*metis rule=I:1* $\&E(1)$ $\vee E(1)$ *cqt:5:a[axiom-inst]*
cqt:5:b[axiom-inst] $\rightarrow I$ $\exists I(1)$)
AOT-theorem *free−thms:4[1,1]*:
‹$([\Pi]\kappa \vee \kappa[\Pi]) \rightarrow \exists\beta\ (\beta = \kappa)$›
by (*metis rule=I:1* $\&E(2)$ $\vee E(1)$ *cqt:5:a[axiom-inst]*
cqt:5:b[axiom-inst] $\rightarrow I$ $\exists I(1)$)

AOT-theorem *free−thms:4[2,rel]*:
‹$([\Pi]\kappa_1\kappa_2 \vee \kappa_1\kappa_2[\Pi]) \rightarrow \exists\beta\ (\beta = \Pi)$›
by (*metis rule=I:1* $\&E(1)$ $\vee E(1)$ *cqt:5:a[2][axiom-inst]*
cqt:5:b[2][axiom-inst] $\rightarrow I$ $\exists I(1)$)
AOT-theorem *free−thms:4[2,1]*:
‹$([\Pi]\kappa_1\kappa_2 \vee \kappa_1\kappa_2[\Pi]) \rightarrow \exists\beta\ (\beta = \kappa_1)$›
by (*metis rule=I:1* $\&E$ $\vee E(1)$ *cqt:5:a[2][axiom-inst]*
cqt:5:b[2][axiom-inst] $\rightarrow I$ $\exists I(1)$)
AOT-theorem *free−thms:4[2,2]*:
‹$([\Pi]\kappa_1\kappa_2 \vee \kappa_1\kappa_2[\Pi]) \rightarrow \exists\beta\ (\beta = \kappa_2)$›
by (*metis rule=I:1* $\&E(2)$ $\vee E(1)$ *cqt:5:a[2][axiom-inst]*
cqt:5:b[2][axiom-inst] $\rightarrow I$ $\exists I(1)$)
AOT-theorem *free−thms:4[3,rel]*:
‹$([\Pi]\kappa_1\kappa_2\kappa_3 \vee \kappa_1\kappa_2\kappa_3[\Pi]) \rightarrow \exists\beta\ (\beta = \Pi)$›
by (*metis rule=I:1* $\&E(1)$ $\vee E(1)$ *cqt:5:a[3][axiom-inst]*
cqt:5:b[3][axiom-inst] $\rightarrow I$ $\exists I(1)$)
AOT-theorem *free−thms:4[3,1]*:
‹$([\Pi]\kappa_1\kappa_2\kappa_3 \vee \kappa_1\kappa_2\kappa_3[\Pi]) \rightarrow \exists\beta\ (\beta = \kappa_1)$›
by (*metis rule=I:1* $\&E$ $\vee E(1)$ *cqt:5:a[3][axiom-inst]*
cqt:5:b[3][axiom-inst] $\rightarrow I$ $\exists I(1)$)
AOT-theorem *free−thms:4[3,2]*:
‹$([\Pi]\kappa_1\kappa_2\kappa_3 \vee \kappa_1\kappa_2\kappa_3[\Pi]) \rightarrow \exists\beta\ (\beta = \kappa_2)$›
by (*metis rule=I:1* $\&E$ $\vee E(1)$ *cqt:5:a[3][axiom-inst]*
cqt:5:b[3][axiom-inst] $\rightarrow I$ $\exists I(1)$)
AOT-theorem *free−thms:4[3,3]*:
‹$([\Pi]\kappa_1\kappa_2\kappa_3 \vee \kappa_1\kappa_2\kappa_3[\Pi]) \rightarrow \exists\beta\ (\beta = \kappa_3)$›
by (*metis rule=I:1* $\&E(2)$ $\vee E(1)$ *cqt:5:a[3][axiom-inst]*
cqt:5:b[3][axiom-inst] $\rightarrow I$ $\exists I(1)$)
AOT-theorem *free−thms:4[4,rel]*:
‹$([\Pi]\kappa_1\kappa_2\kappa_3\kappa_4 \vee \kappa_1\kappa_2\kappa_3\kappa_4[\Pi]) \rightarrow \exists\beta\ (\beta = \Pi)$›
by (*metis rule=I:1* $\&E(1)$ $\vee E(1)$ *cqt:5:a[4][axiom-inst]*
cqt:5:b[4][axiom-inst] $\rightarrow I$ $\exists I(1)$)
AOT-theorem *free−thms:4[4,1]*:
‹$([\Pi]\kappa_1\kappa_2\kappa_3\kappa_4 \vee \kappa_1\kappa_2\kappa_3\kappa_4[\Pi]) \rightarrow \exists\beta\ (\beta = \kappa_1)$›

by (*metis rule*=*I*:*1* &*E* ∨*E*(*1*) *cqt*:*5*:*a*[*4*][*axiom-inst*]
cqt:*5*:*b*[*4*][*axiom-inst*] →*I* ∃*I*(*1*))
AOT-theorem *free−thms*:*4*[*4*,*2*]:
‹([Π]$\kappa_1\kappa_2\kappa_3\kappa_4$ ∨ $\kappa_1\kappa_2\kappa_3\kappa_4$[Π]) → ∃ β (β = κ_2)›
by (*metis rule*=*I*:*1* &*E* ∨*E*(*1*) *cqt*:*5*:*a*[*4*][*axiom-inst*]
cqt:*5*:*b*[*4*][*axiom-inst*] →*I* ∃*I*(*1*))
AOT-theorem *free−thms*:*4*[*4*,*3*]:
‹([Π]$\kappa_1\kappa_2\kappa_3\kappa_4$ ∨ $\kappa_1\kappa_2\kappa_3\kappa_4$[Π]) → ∃ β (β = κ_3)›
by (*metis rule*=*I*:*1* &*E* ∨*E*(*1*) *cqt*:*5*:*a*[*4*][*axiom-inst*]
cqt:*5*:*b*[*4*][*axiom-inst*] →*I* ∃*I*(*1*))
AOT-theorem *free−thms*:*4*[*4*,*4*]:
‹([Π]$\kappa_1\kappa_2\kappa_3\kappa_4$ ∨ $\kappa_1\kappa_2\kappa_3\kappa_4$[Π]) → ∃ β (β = κ_4)›
by (*metis rule*=*I*:*1* &*E*(*2*) ∨*E*(*1*) *cqt*:*5*:*a*[*4*][*axiom-inst*]
cqt:*5*:*b*[*4*][*axiom-inst*] →*I* ∃*I*(*1*))

AOT-theorem *ex*:*1*:*a*: ‹∀ α α↓›
by (*rule GEN*) (*fact cqt*:*2*[*const-var*][*axiom-inst*])
AOT-theorem *ex*:*1*:*b*: ‹∀ α∃ β(β = α)›
by (*rule GEN*) (*fact free−thms*:*3*[*const-var*])

AOT-theorem *ex*:*2*:*a*: ‹□α↓›
by (*rule RN*) (*fact cqt*:*2*[*const-var*][*axiom-inst*])
AOT-theorem *ex*:*2*:*b*: ‹□∃ β(β = α)›
by (*rule RN*) (*fact free−thms*:*3*[*const-var*])

AOT-theorem *ex*:*3*:*a*: ‹□∀ α α↓›
by (*rule RN*) (*fact ex*:*1*:*a*)
AOT-theorem *ex*:*3*:*b*: ‹□∀ α∃ β(β = α)›
by (*rule RN*) (*fact ex*:*1*:*b*)

AOT-theorem *ex*:*4*:*a*: ‹∀ α □α↓›
by (*rule GEN*; *rule RN*) (*fact cqt*:*2*[*const-var*][*axiom-inst*])
AOT-theorem *ex*:*4*:*b*: ‹∀ α□∃ β(β = α)›
by (*rule GEN*; *rule RN*) (*fact free−thms*:*3*[*const-var*])

AOT-theorem *ex*:*5*:*a*: ‹□∀ α □α↓›
by (*rule RN*) (*simp add*: *ex*:*4*:*a*)
AOT-theorem *ex*:*5*:*b*: ‹□∀ α□∃ β(β = α)›
by (*rule RN*) (*simp add*: *ex*:*4*:*b*)

AOT-theorem *all−self*=:*1*: ‹□∀ α(α = α)›
by (*rule RN*; *rule GEN*) (*fact id−eq*:*1*)
AOT-theorem *all−self*=:*2*: ‹∀ α□(α = α)›
by (*rule GEN*; *rule RN*) (*fact id−eq*:*1*)

AOT-theorem *id−nec*:*1*: ‹α = β → □(α = β)›
proof(*rule* →*I*)
AOT-assume ‹α = β›
moreover AOT-have ‹□(α = α)›
by (*rule RN*) (*fact id−eq*:*1*)
ultimately AOT-show ‹□(α = β)› **using** *rule*=*E* **by** *fast*
qed

AOT-theorem *id−nec*:*2*: ‹τ = σ → □(τ = σ)›
proof(*rule* →*I*)
AOT-assume *asm*: ‹τ = σ›
moreover AOT-have ‹τ↓›
using *calculation t*=*t−proper*:*1* →*E* **by** *blast*
moreover AOT-have ‹□(τ = τ)›
using *calculation all−self*=:*2* ∀ *E*(*1*) **by** *blast*
ultimately AOT-show ‹□(τ = σ)› **using** *rule*=*E* **by** *fast*
qed

AOT-theorem *term−out:1*: ‹φ{α} ≡ ∃β (β = α & φ{β})›
proof (*rule* ≡*I*; *rule* →*I*)
  AOT-assume *asm*: ‹φ{α}›
  AOT-show ‹∃β (β = α & φ{β})›
    by (*rule* ∃*I*(*2*)[**where** β=α]; *rule* &*I*)
      (*auto simp*: *id−eq*:*1 asm*)
next
  AOT-assume *0*: ‹∃β (β = α & φ{β})›
  AOT-obtain β **where** ‹β = α & φ{β}›
    using ∃*E*[*rotated*, *OF 0*] **by** *blast*
  AOT-thus ‹φ{α}› **using** &*E rule*=*E* **by** *blast*
qed

AOT-theorem *term−out:2*: ‹τ↓ → (φ{τ} ≡ ∃α(α = τ & φ{α}))›
proof(*rule* →*I*)
  AOT-assume ‹τ↓›
  moreover AOT-have ‹∀α (φ{α} ≡ ∃β (β = α & φ{β}))›
    by (*rule GEN*) (*fact term−out*:*1*)
  ultimately AOT-show ‹φ{τ} ≡ ∃α(α = τ & φ{α})›
    using ∀*E* **by** *blast*
qed

AOT-theorem *term−out:3*:
  ‹(φ{α} & ∀β(φ{β} → β = α)) ≡ ∀β(φ{β} ≡ β = α)›
  apply (*rule* ≡*I*; *rule* →*I*)
   apply (*frule* &*E*(*1*))
   apply (*drule* &*E*(*2*))
   apply (*rule GEN*; *rule* ≡*I*; *rule* →*I*)
  using *rule−ui*:*2*[*const-var*] *vdash−properties*:*5*
    apply *blast*
   apply (*meson rule*=*E id−eq*:*1*)
  apply (*rule* &*I*)
  using *id−eq*:*1* ≡*E*(*2*) *rule−ui*:*3*
   apply *blast*
  apply (*rule GEN*; *rule* →*I*)
  using ≡*E*(*1*) *rule−ui*:*2*[*const-var*]
  by *blast*

AOT-theorem *term−out:4*:
  ‹(φ{β} & ∀α(φ{α} → α = β)) ≡ ∀α(φ{α} ≡ α = β)›
  using *term−out*:*3* **.**

AOT-define *AOT-exists-unique* :: ‹α ⇒ φ ⇒ φ› *uniqueness*:*1*:
  ‹«*AOT-exists-unique* φ» $\equiv_{df}$ ∃α (φ{α} & ∀β (φ{β} → β = α))›
syntax (*input*) *-AOT-exists-unique* :: ‹α ⇒ φ ⇒ φ› (‹∃!- -› [*1*,*40*])
syntax (**output**) *-AOT-exists-unique* :: ‹α ⇒ φ ⇒ φ› (‹∃!-′(-′)› [*1*,*40*])
AOT-syntax-print-translations
  -AOT-exists-unique τ φ <= *CONST AOT-exists-unique* (*-abs* τ φ)
syntax
   -AOT-exists-unique-ellipse :: ‹*id-position* ⇒ *id-position* ⇒ φ ⇒ φ›
   (‹∃!-...∃!- -› [*1*,*40*])
parse-ast-translation‹
[(***syntax-const***‹*-AOT-exists-unique-ellipse*›,
  fn ctx => *fn* [*a*,*b*,*c*] => *Ast.mk-appl* (*Ast.Constant AOT-exists-unique*)
  [*parseEllipseList -AOT-vars ctx* [*a*,*b*],*c*]),
 (***syntax-const***‹*-AOT-exists-unique*›,
  AOT-restricted-binder
    const-name‹*AOT-exists-unique*›
    const-syntax‹*AOT-conj*›)]›
print-translation‹*AOT-syntax-print-translations* [
  AOT-preserve-binder-abs-tr′

    const-syntax ‹*AOT-exists-unique*›
    syntax-const ‹*-AOT-exists-unique*›
    (***syntax-const*** ‹*-AOT-exists-unique-ellipse*›, *true*)
    const-name ‹*AOT-conj*›,
  AOT-binder-trans
    @{*theory*}
    @{*binding AOT-exists-unique-binder*}
    syntax-const ‹*-AOT-exists-unique*›
]›

context *AOT-meta-syntax*
begin
notation *AOT-exists-unique* (**binder** ‹∃ **!**› *20*)
end
context *AOT-no-meta-syntax*
begin
no-notation *AOT-exists-unique* (**binder** ‹∃ **!**› *20*)
end

AOT-theorem *uniqueness:2*: ‹$\exists!\alpha\ \varphi\{\alpha\} \equiv \exists\alpha\forall\beta(\varphi\{\beta\} \equiv \beta = \alpha)$›
proof(*rule* ≡*I*; *rule* →*I*)
    AOT-assume ‹$\exists!\alpha\ \varphi\{\alpha\}$›
    AOT-hence ‹$\exists\alpha\ (\varphi\{\alpha\}\ \&\ \forall\beta\ (\varphi\{\beta\} \rightarrow \beta = \alpha))$›
      using *uniqueness:1* $\equiv_{df}E$ **by** *blast*
    then AOT-obtain α **where** ‹$\varphi\{\alpha\}\ \&\ \forall\beta\ (\varphi\{\beta\} \rightarrow \beta = \alpha)$›
      using *instantiation*[*rotated*] **by** *blast*
    AOT-hence ‹$\forall\beta(\varphi\{\beta\} \equiv \beta = \alpha)$›
      using *term−out:3* ≡*E* **by** *blast*
    AOT-thus ‹$\exists\alpha\forall\beta(\varphi\{\beta\} \equiv \beta = \alpha)$›
      using ∃ *I* **by** *fast*
next
    AOT-assume ‹$\exists\alpha\forall\beta(\varphi\{\beta\} \equiv \beta = \alpha)$›
    then AOT-obtain α **where** ‹$\forall\beta\ (\varphi\{\beta\} \equiv \beta = \alpha)$›
      using *instantiation*[*rotated*] **by** *blast*
    AOT-hence ‹$\varphi\{\alpha\}\ \&\ \forall\beta\ (\varphi\{\beta\} \rightarrow \beta = \alpha)$›
      using *term−out:3* ≡*E* **by** *blast*
    AOT-hence ‹$\exists\alpha\ (\varphi\{\alpha\}\ \&\ \forall\beta\ (\varphi\{\beta\} \rightarrow \beta = \alpha))$›
      using ∃ *I* **by** *fast*
    AOT-thus ‹$\exists!\alpha\ \varphi\{\alpha\}$›
      using *uniqueness:1* $\equiv_{df}I$ **by** *blast*
qed

AOT-theorem *uni−most*: ‹$\exists!\alpha\ \varphi\{\alpha\} \rightarrow \forall\beta\forall\gamma((\varphi\{\beta\}\ \&\ \varphi\{\gamma\}) \rightarrow \beta = \gamma)$›
proof(*rule* →*I*; *rule GEN*; *rule GEN*; *rule* →*I*)
  fix $\beta\ \gamma$
  AOT-assume ‹$\exists!\alpha\ \varphi\{\alpha\}$›
  AOT-hence ‹$\exists\alpha\forall\beta(\varphi\{\beta\} \equiv \beta = \alpha)$›
    using *uniqueness:2* ≡*E* **by** *blast*
  then AOT-obtain α **where** ‹$\forall\beta(\varphi\{\beta\} \equiv \beta = \alpha)$›
    using *instantiation*[*rotated*] **by** *blast*
  moreover AOT-assume ‹$\varphi\{\beta\}\ \&\ \varphi\{\gamma\}$›
  ultimately AOT-have ‹$\beta = \alpha$› **and** ‹$\gamma = \alpha$›
    using ∀ *E*(*2*) &*E* ≡*E*(*1*,*2*) **by** *blast+*
  AOT-thus ‹$\beta = \gamma$›
    by (*metis rule*=*E id−eq:2* →*E*)
qed

AOT-theorem *nec−exist−!*: ‹$\forall\alpha(\varphi\{\alpha\} \rightarrow \Box\varphi\{\alpha\}) \rightarrow (\exists!\alpha\ \varphi\{\alpha\} \rightarrow \exists!\alpha\ \Box\varphi\{\alpha\})$›
proof (*rule* →*I*; *rule* →*I*)
  AOT-assume *a*: ‹$\forall\alpha(\varphi\{\alpha\} \rightarrow \Box\varphi\{\alpha\})$›
  AOT-assume ‹$\exists!\alpha\ \varphi\{\alpha\}$›
  AOT-hence ‹$\exists\alpha\ (\varphi\{\alpha\}\ \&\ \forall\beta\ (\varphi\{\beta\} \rightarrow \beta = \alpha))$›

using *uniqueness:1* $\equiv_{df}E$ **by** *blast*
then AOT-obtain α **where** ξ: ‹$\varphi\{\alpha\}$ & $\forall\beta$ ($\varphi\{\beta\} \rightarrow \beta = \alpha$)›
using *instantiation*[*rotated*] **by** *blast*
AOT-have ‹$\Box\varphi\{\alpha\}$›
using ξ *a* &*E* $\forall$ *E* $\rightarrow$*E* **by** *fast*
moreover AOT-have ‹$\forall\beta$ ($\Box\varphi\{\beta\} \rightarrow \beta = \alpha$)›
apply (*rule GEN*; *rule* $\rightarrow$*I*)
using ξ[*THEN* &*E*(*2*), *THEN* $\forall$ *E*(*2*), *THEN* $\rightarrow$*E*]
qml:*2*[*axiom-inst*, *THEN* $\rightarrow$*E*] **by** *blast*
ultimately AOT-have ‹($\Box\varphi\{\alpha\}$ & $\forall\beta$ ($\Box\varphi\{\beta\} \rightarrow \beta = \alpha$))›
using &*I* **by** *blast*
AOT-thus ‹$\exists!\alpha\ \Box\varphi\{\alpha\}$›
using *uniqueness:1* $\equiv_{df}I$ $\exists$ *I* **by** *fast*
qed

8.8 The Theory of Actuality and Descriptions

AOT-theorem *act−cond*: ‹$\mathcal{A}(\varphi \rightarrow \psi) \rightarrow (\mathcal{A}\varphi \rightarrow \mathcal{A}\psi)$›
using $\rightarrow$*I* $\equiv$*E*(*1*) *logic−actual−nec*:*2*[*axiom-inst*] **by** *blast*

AOT-theorem *nec−imp−act*: ‹$\Box\varphi \rightarrow \mathcal{A}\varphi$›
by (*metis act−cond contraposition*:*1*[*2*] $\equiv$*E*(*4*)
qml:*2*[*THEN act-closure*, *axiom-inst*]
qml−act:*2*[*axiom-inst*] *RAA*(*1*) $\rightarrow$*E* $\rightarrow$*I*)

AOT-theorem *act−conj−act*:*1*: ‹$\mathcal{A}(\mathcal{A}\varphi \rightarrow \varphi)$›
using $\rightarrow$*I* $\equiv$*E*(*2*) *logic−actual−nec*:*2*[*axiom-inst*]
logic−actual−nec:*4*[*axiom-inst*] **by** *blast*

AOT-theorem *act−conj−act*:*2*: ‹$\mathcal{A}(\varphi \rightarrow \mathcal{A}\varphi)$›
by (*metis* $\rightarrow$*I* $\equiv$*E*(*2*, *4*) *logic−actual−nec*:*2*[*axiom-inst*]
logic−actual−nec:*4*[*axiom-inst*] *RAA*(*1*))

AOT-theorem *act−conj−act*:*3*: ‹$(\mathcal{A}\varphi \ \&\ \mathcal{A}\psi) \rightarrow \mathcal{A}(\varphi \ \&\ \psi)$›
proof −
AOT-have ‹$\Box(\varphi \rightarrow (\psi \rightarrow (\varphi \ \&\ \psi)))$›
by (*rule RN*) (*fact Adjunction*)
AOT-hence ‹$\mathcal{A}(\varphi \rightarrow (\psi \rightarrow (\varphi \ \&\ \psi)))$›
using *nec−imp−act* $\rightarrow$*E* **by** *blast*
AOT-hence ‹$\mathcal{A}\varphi \rightarrow \mathcal{A}(\psi \rightarrow (\varphi \ \&\ \psi))$›
using *act−cond* $\rightarrow$*E* **by** *blast*
moreover AOT-have ‹$\mathcal{A}(\psi \rightarrow (\varphi \ \&\ \psi)) \rightarrow (\mathcal{A}\psi \rightarrow \mathcal{A}(\varphi \ \&\ \psi))$›
by (*fact act−cond*)
ultimately AOT-have ‹$\mathcal{A}\varphi \rightarrow (\mathcal{A}\psi \rightarrow \mathcal{A}(\varphi \ \&\ \psi))$›
using $\rightarrow$*I* $\rightarrow$*E* **by** *metis*
AOT-thus ‹$(\mathcal{A}\varphi \ \&\ \mathcal{A}\psi) \rightarrow \mathcal{A}(\varphi \ \&\ \psi)$›
by (*metis Importation* $\rightarrow$*E*)
qed

AOT-theorem *act−conj−act*:*4*: ‹$\mathcal{A}(\mathcal{A}\varphi \equiv \varphi)$›
proof −
AOT-have ‹$(\mathcal{A}(\mathcal{A}\varphi \rightarrow \varphi) \ \&\ \mathcal{A}(\varphi \rightarrow \mathcal{A}\varphi)) \rightarrow \mathcal{A}((\mathcal{A}\varphi \rightarrow \varphi) \ \&\ (\varphi \rightarrow \mathcal{A}\varphi))$›
by (*fact act−conj−act*:*3*)
moreover AOT-have ‹$\mathcal{A}(\mathcal{A}\varphi \rightarrow \varphi) \ \&\ \mathcal{A}(\varphi \rightarrow \mathcal{A}\varphi)$›
using &*I act−conj−act*:*1 act−conj−act*:*2* **by** *simp*
ultimately AOT-have ζ: ‹$\mathcal{A}((\mathcal{A}\varphi \rightarrow \varphi) \ \&\ (\varphi \rightarrow \mathcal{A}\varphi))$›
using $\rightarrow$*E* **by** *blast*
AOT-have ‹$\mathcal{A}(((\mathcal{A}\varphi \rightarrow \varphi) \ \&\ (\varphi \rightarrow \mathcal{A}\varphi)) \rightarrow (\mathcal{A}\varphi \equiv \varphi))$›
using *conventions*:*3*[*THEN df−rules−formulas*[*2*],
THEN act-closure, *axiom-inst*] **by** *blast*
AOT-hence ‹$\mathcal{A}((\mathcal{A}\varphi \rightarrow \varphi) \ \&\ (\varphi \rightarrow \mathcal{A}\varphi)) \rightarrow \mathcal{A}(\mathcal{A}\varphi \equiv \varphi)$›
using *act−cond* $\rightarrow$*E* **by** *blast*
AOT-thus ‹$\mathcal{A}(\mathcal{A}\varphi \equiv \varphi)$› **using** ζ $\rightarrow$*E* **by** *blast*

qed

inductive *arbitrary-actualization* **for** φ **where**
‹*arbitrary-actualization* φ «$\mathcal{A}\varphi$»›
| ‹*arbitrary-actualization* φ «$\mathcal{A}\psi$»› **if** ‹*arbitrary-actualization* φ ψ›
declare *arbitrary-actualization.cases*[*AOT*]
arbitrary-actualization.induct[*AOT*]
arbitrary-actualization.simps[*AOT*]
arbitrary-actualization.intros[*AOT*]
syntax *arbitrary-actualization* :: ‹$\varphi' \Rightarrow \varphi' \Rightarrow$ *AOT-prop*›
(‹*ARBITRARY′-ACTUALIZATION′*(-,-′)›)

notepad
begin
AOT-modally-strict {
fix φ
AOT-have ‹*ARBITRARY-ACTUALIZATION*($\mathcal{A}\varphi \equiv \varphi$, $\mathcal{A}(\mathcal{A}\varphi \equiv \varphi)$)›
using *AOT-PLM.arbitrary-actualization.intros* **by** *metis*
AOT-have ‹*ARBITRARY-ACTUALIZATION*($\mathcal{A}\varphi \equiv \varphi$, $\mathcal{A}\mathcal{A}(\mathcal{A}\varphi \equiv \varphi)$)›
using *AOT-PLM.arbitrary-actualization.intros* **by** *metis*
AOT-have ‹*ARBITRARY-ACTUALIZATION*($\mathcal{A}\varphi \equiv \varphi$, $\mathcal{A}\mathcal{A}\mathcal{A}(\mathcal{A}\varphi \equiv \varphi)$)›
using *AOT-PLM.arbitrary-actualization.intros* **by** *metis*
}
end

AOT-theorem *closure−act*:*1*:
assumes ‹*ARBITRARY-ACTUALIZATION*($\mathcal{A}\varphi \equiv \varphi$, ψ)›
shows ‹ψ›
using *assms* **proof**(*induct*)
case *1*
AOT-show ‹$\mathcal{A}(\mathcal{A}\varphi \equiv \varphi)$›
by (*simp add*: *act−conj−act*:*4*)
next
case (*2* ψ)
AOT-thus ‹$\mathcal{A}\psi$›
by (*metis arbitrary-actualization.simps* ≡*E*(*1*)
logic−actual−nec:*4*[*axiom-inst*])
qed

AOT-theorem *closure−act*:*2*: ‹$\forall\alpha\ \mathcal{A}(\mathcal{A}\varphi\{\alpha\} \equiv \varphi\{\alpha\})$›
by (*simp add*: *act−conj−act*:*4* ∀*I*)

AOT-theorem *closure−act*:*3*: ‹$\mathcal{A}\forall\alpha\ \mathcal{A}(\mathcal{A}\varphi\{\alpha\} \equiv \varphi\{\alpha\})$›
by (*metis* (*no-types*, *lifting*) *act−conj−act*:*4* ≡*E*(*1*,*2*) ∀*I*
logic−actual−nec:*3*[*axiom-inst*]
logic−actual−nec:*4*[*axiom-inst*])

AOT-theorem *closure−act*:*4*: ‹$\mathcal{A}\forall\alpha_1...\forall\alpha_n\ \mathcal{A}(\mathcal{A}\varphi\{\alpha_1...\alpha_n\} \equiv \varphi\{\alpha_1...\alpha_n\})$›
using *closure−act*:*3* **.**

AOT-act-theorem *RA*[*1*]:
assumes ‹$\vdash \varphi$›
shows ‹$\vdash \mathcal{A}\varphi$›
— While this proof is rejected in PLM, we merely state it as modally-fragile rule, which addresses the concern in PLM.
using ¬¬*E assms* ≡*E*(*3*) *logic−actual*[*act-axiom-inst*]
logic−actual−nec:*1*[*axiom-inst*] *modus−tollens*:*2* **by** *blast*
AOT-theorem *RA*[*2*]:
assumes ‹$\vdash_\Box \varphi$›
shows ‹$\vdash_\Box \mathcal{A}\varphi$›
— This rule is in fact a consequence of RN and does not require an appeal to the semantics itself.

using *RN assms nec−imp−act vdash−properties*:*5* **by** *blast*
AOT-theorem *RA*[*3*]:
assumes ‹$\Gamma \vdash_\Box \varphi$›
shows ‹$\mathcal{A}\Gamma \vdash_\Box \mathcal{A}\varphi$›

This rule is only derivable from the semantics, but apparently no proof actually relies on it. If this turns out to be required, it is valid to derive it from the semantics just like RN, but we refrain from doing so, unless necessary.

oops — discard the rule

AOT-act-theorem *ANeg*:*1*: ‹$\neg\mathcal{A}\varphi \equiv \neg\varphi$›
by (*simp add*: *RA*[*1*] *contraposition*:*1*[*1*] *deduction−theorem* $\equiv$*I logic−actual*[*act-axiom-inst*])

AOT-act-theorem *ANeg*:*2*: ‹$\neg\mathcal{A}\neg\varphi \equiv \varphi$›
using *ANeg*:*1* $\equiv$*I* $\equiv$*E*(*5*) *useful−tautologies*:*1* *useful−tautologies*:*2* **by** *blast*

AOT-theorem *Act−Basic*:*1*: ‹$\mathcal{A}\varphi \vee \mathcal{A}\neg\varphi$›
by (*meson* $\vee$*I*(*1*,*2*) $\equiv$*E*(*2*) *logic−actual−nec*:*1*[*axiom-inst*] *raa−cor*:*1*)

AOT-theorem *Act−Basic*:*2*: ‹$\mathcal{A}(\varphi \;\&\; \psi) \equiv (\mathcal{A}\varphi \;\&\; \mathcal{A}\psi)$›
proof (*rule* $\equiv$*I*; *rule* $\rightarrow$*I*)
AOT-assume ‹$\mathcal{A}(\varphi \;\&\; \psi)$›
moreover AOT-have ‹$\mathcal{A}((\varphi \;\&\; \psi) \rightarrow \varphi)$›
by (*simp add*: *RA*[*2*] *Conjunction Simplification*(*1*))
moreover AOT-have ‹$\mathcal{A}((\varphi \;\&\; \psi) \rightarrow \psi)$›
by (*simp add*: *RA*[*2*] *Conjunction Simplification*(*2*))
ultimately AOT-show ‹$\mathcal{A}\varphi \;\&\; \mathcal{A}\psi$›
using *act−cond*[*THEN* $\rightarrow$*E*, *THEN* $\rightarrow$*E*] &*I* **by** *metis*
next
AOT-assume ‹$\mathcal{A}\varphi \;\&\; \mathcal{A}\psi$›
AOT-thus ‹$\mathcal{A}(\varphi \;\&\; \psi)$›
using *act−conj−act*:*3* *vdash−properties*:*6* **by** *blast*
qed

AOT-theorem *Act−Basic*:*3*: ‹$\mathcal{A}(\varphi \equiv \psi) \equiv (\mathcal{A}(\varphi \rightarrow \psi) \;\&\; \mathcal{A}(\psi \rightarrow \varphi))$›
proof (*rule* $\equiv$*I*; *rule* $\rightarrow$*I*)
AOT-assume ‹$\mathcal{A}(\varphi \equiv \psi)$›
moreover AOT-have ‹$\mathcal{A}((\varphi \equiv \psi) \rightarrow (\varphi \rightarrow \psi))$›
by (*simp add*: *RA*[*2*] *deduction−theorem* $\equiv$*E*(*1*))
moreover AOT-have ‹$\mathcal{A}((\varphi \equiv \psi) \rightarrow (\psi \rightarrow \varphi))$›
by (*simp add*: *RA*[*2*] *deduction−theorem* $\equiv$*E*(*2*))
ultimately AOT-show ‹$\mathcal{A}(\varphi \rightarrow \psi) \;\&\; \mathcal{A}(\psi \rightarrow \varphi)$›
using *act−cond*[*THEN* $\rightarrow$*E*, *THEN* $\rightarrow$*E*] &*I* **by** *metis*
next
AOT-assume ‹$\mathcal{A}(\varphi \rightarrow \psi) \;\&\; \mathcal{A}(\psi \rightarrow \varphi)$›
AOT-hence ‹$\mathcal{A}((\varphi \rightarrow \psi) \;\&\; (\psi \rightarrow \varphi))$›
by (*metis act−conj−act*:*3* *vdash−properties*:*10*)
moreover AOT-have ‹$\mathcal{A}(((\varphi \rightarrow \psi) \;\&\; (\psi \rightarrow \varphi)) \rightarrow (\varphi \equiv \psi))$›
by (*simp add*: *conventions*:*3* *RA*[*2*] *df−rules−formulas*[*2*] *vdash−properties*:*1*[*2*])
ultimately AOT-show ‹$\mathcal{A}(\varphi \equiv \psi)$›
using *act−cond*[*THEN* $\rightarrow$*E*, *THEN* $\rightarrow$*E*] **by** *metis*
qed

AOT-theorem *Act−Basic*:*4*: ‹$(\mathcal{A}(\varphi \rightarrow \psi) \;\&\; \mathcal{A}(\psi \rightarrow \varphi)) \equiv (\mathcal{A}\varphi \equiv \mathcal{A}\psi)$›
proof (*rule* $\equiv$*I*; *rule* $\rightarrow$*I*)
AOT-assume *0*: ‹$\mathcal{A}(\varphi \rightarrow \psi) \;\&\; \mathcal{A}(\psi \rightarrow \varphi)$›
AOT-show ‹$\mathcal{A}\varphi \equiv \mathcal{A}\psi$›
using *0* &*E* *act−cond*[*THEN* $\rightarrow$*E*, *THEN* $\rightarrow$*E*] $\equiv$*I* $\rightarrow$*I* **by** *metis*
next
AOT-assume ‹$\mathcal{A}\varphi \equiv \mathcal{A}\psi$›

```
  AOT-thus ‹𝓐(φ → ψ) & 𝓐(ψ → φ)›
    by (metis →I logic−actual−nec:2[axiom-inst] ≡E(1,2) &I)
qed

AOT-theorem Act−Basic:5: ‹𝓐(φ ≡ ψ) ≡ (𝓐φ ≡ 𝓐ψ)›
  using Act−Basic:3 Act−Basic:4 ≡E(5) by blast

AOT-theorem Act−Basic:6: ‹𝓐φ ≡ □𝓐φ›
  by (simp add: ≡I qml:2[axiom-inst] qml−act:1[axiom-inst])

AOT-theorem Act−Basic:7: ‹𝓐□φ → □𝓐φ›
  by (metis Act−Basic:6 →I →E ≡E(1,2) nec−imp−act
          qml−act:2[axiom-inst])

AOT-theorem Act−Basic:8: ‹□φ → □𝓐φ›
  using Hypothetical Syllogism nec−imp−act qml−act:1[axiom-inst] by blast

AOT-theorem Act−Basic:9: ‹𝓐(φ ∨ ψ) ≡ (𝓐φ ∨ 𝓐ψ)›
proof (rule ≡I; rule →I)
  AOT-assume ‹𝓐(φ ∨ ψ)›
  AOT-thus ‹𝓐φ ∨ 𝓐ψ›
  proof (rule raa−cor:3)
    AOT-assume ‹¬(𝓐φ ∨ 𝓐ψ)›
    AOT-hence ‹¬𝓐φ & ¬𝓐ψ›
      by (metis ≡E(1) oth−class−taut:5:d)
    AOT-hence ‹𝓐¬φ & 𝓐¬ψ›
      using logic−actual−nec:1[axiom-inst, THEN ≡E(2)] &E &I by metis
    AOT-hence ‹𝓐(¬φ & ¬ψ)›
      using ≡E Act−Basic:2 by metis
    moreover AOT-have ‹𝓐((¬φ & ¬ψ) ≡ ¬(φ ∨ ψ))›
      using RA[2] ≡E(6) oth−class−taut:3:a oth−class−taut:5:d by blast
    moreover AOT-have ‹𝓐(¬φ & ¬ψ) ≡ 𝓐(¬(φ ∨ ψ))›
      using calculation(2) by (metis Act−Basic:5 ≡E(1))
    ultimately AOT-have ‹𝓐(¬(φ ∨ ψ))› using ≡E by blast
    AOT-thus ‹¬𝓐(φ ∨ ψ)›
      using logic−actual−nec:1[axiom-inst, THEN ≡E(1)] by auto
  qed
next
  AOT-assume ‹𝓐φ ∨ 𝓐ψ›
  AOT-thus ‹𝓐(φ ∨ ψ)›
    by (meson RA[2] act−cond ∨I(1) ∨E(1) Disjunction Addition(1,2))
qed

AOT-theorem Act−Basic:10: ‹𝓐∃α φ{α} ≡ ∃α 𝓐φ{α}›
proof −
  AOT-have ϑ: ‹¬𝓐∀α ¬φ{α} ≡ ¬∀α 𝓐¬φ{α}›
    by (rule oth−class−taut:4:b[THEN ≡E(1)])
       (metis logic−actual−nec:3[axiom-inst])
  AOT-have ξ: ‹¬∀α 𝓐¬φ{α} ≡ ¬∀α ¬𝓐φ{α}›
    by (rule oth−class−taut:4:b[THEN ≡E(1)])
       (rule logic−actual−nec:1[THEN universal-closure,
            axiom-inst, THEN cqt−basic:3[THEN →E]])
  AOT-have ‹𝓐(∃α φ{α}) ≡ 𝓐(¬∀α ¬φ{α})›
    using conventions:4[THEN df−rules−formulas[1],
                        THEN act-closure, axiom-inst]
          conventions:4[THEN df−rules−formulas[2],
                        THEN act-closure, axiom-inst]
    Act−Basic:4[THEN ≡E(1)] &I Act−Basic:5[THEN ≡E(2)] by metis
  also AOT-have ‹... ≡ ¬𝓐∀α ¬φ{α}›
    by (simp add: logic−actual−nec:1 vdash−properties:1[2])
  also AOT-have ‹... ≡ ¬∀α 𝓐 ¬φ{α}› using ϑ by blast
  also AOT-have ‹... ≡ ¬∀α ¬𝓐 φ{α}› using ξ by blast
  also AOT-have ‹... ≡ ∃α 𝓐 φ{α}›
```

using *conventions*:*4*[*THEN* ≡*Df*] **by** (*metis* ≡*E*(*6*) *oth*−*class*−*taut*:*3*:*a*)
finally AOT-show ‹$\mathcal{A}\exists\,\alpha\ \varphi\{\alpha\} \equiv \exists\,\alpha\ \mathcal{A}\varphi\{\alpha\}$› **.**
qed

AOT-theorem *Act*−*Basic*:*11*:
‹$\mathcal{A}\forall\,\alpha(\varphi\{\alpha\} \equiv \psi\{\alpha\}) \equiv \forall\,\alpha(\mathcal{A}\varphi\{\alpha\} \equiv \mathcal{A}\psi\{\alpha\})$›
proof(*rule* ≡*I*; *rule* →*I*)
AOT-assume ‹$\mathcal{A}\forall\,\alpha(\varphi\{\alpha\} \equiv \psi\{\alpha\})$›
AOT-hence ‹$\forall\,\alpha\mathcal{A}(\varphi\{\alpha\} \equiv \psi\{\alpha\})$›
using *logic*−*actual*−*nec*:*3*[*axiom-inst*, *THEN* ≡*E*(*1*)] **by** *blast*
AOT-hence ‹$\mathcal{A}(\varphi\{\alpha\} \equiv \psi\{\alpha\})$› **for** α **using** ∀*E* **by** *blast*
AOT-hence ‹$\mathcal{A}\varphi\{\alpha\} \equiv \mathcal{A}\psi\{\alpha\}$› **for** α **by** (*metis Act*−*Basic*:*5* ≡*E*(*1*))
AOT-thus ‹$\forall\,\alpha(\mathcal{A}\varphi\{\alpha\} \equiv \mathcal{A}\psi\{\alpha\})$› **by** (*rule* ∀*I*)
next
AOT-assume ‹$\forall\,\alpha(\mathcal{A}\varphi\{\alpha\} \equiv \mathcal{A}\psi\{\alpha\})$›
AOT-hence ‹$\mathcal{A}\varphi\{\alpha\} \equiv \mathcal{A}\psi\{\alpha\}$› **for** α **using** ∀*E* **by** *blast*
AOT-hence ‹$\mathcal{A}(\varphi\{\alpha\} \equiv \psi\{\alpha\})$› **for** α **by** (*metis Act*−*Basic*:*5* ≡*E*(*2*))
AOT-hence ‹$\forall\,\alpha\ \mathcal{A}(\varphi\{\alpha\} \equiv \psi\{\alpha\})$› **by** (*rule* ∀*I*)
AOT-thus ‹$\mathcal{A}\forall\,\alpha(\varphi\{\alpha\} \equiv \psi\{\alpha\})$›
using *logic*−*actual*−*nec*:*3*[*axiom-inst*, *THEN* ≡*E*(*2*)] **by** *fast*
qed

AOT-act-theorem *act*−*quant*−*uniq*:
‹$\forall\,\beta(\mathcal{A}\varphi\{\beta\} \equiv \beta = \alpha) \equiv \forall\,\beta(\varphi\{\beta\} \equiv \beta = \alpha)$›
proof(*rule* ≡*I*; *rule* →*I*)
AOT-assume ‹$\forall\,\beta(\mathcal{A}\varphi\{\beta\} \equiv \beta = \alpha)$›
AOT-hence ‹$\mathcal{A}\varphi\{\beta\} \equiv \beta = \alpha$› **for** β **using** ∀*E* **by** *blast*
AOT-hence ‹$\varphi\{\beta\} \equiv \beta = \alpha$› **for** β
using ≡*I* →*I RA*[*1*] ≡*E*(*1*,*2*) *logic*−*actual*[*act-axiom-inst*] →*E*
by *metis*
AOT-thus ‹$\forall\,\beta(\varphi\{\beta\} \equiv \beta = \alpha)$› **by** (*rule* ∀*I*)
next
AOT-assume ‹$\forall\,\beta(\varphi\{\beta\} \equiv \beta = \alpha)$›
AOT-hence ‹$\varphi\{\beta\} \equiv \beta = \alpha$› **for** β **using** ∀*E* **by** *blast*
AOT-hence ‹$\mathcal{A}\varphi\{\beta\} \equiv \beta = \alpha$› **for** β
using ≡*I* →*I RA*[*1*] ≡*E*(*1*,*2*) *logic*−*actual*[*act-axiom-inst*] →*E*
by *metis*
AOT-thus ‹$\forall\,\beta(\mathcal{A}\varphi\{\beta\} \equiv \beta = \alpha)$› **by** (*rule* ∀*I*)
qed

AOT-act-theorem *fund*−*cont*−*desc*: ‹$x = \iota x(\varphi\{x\}) \equiv \forall\,z(\varphi\{z\} \equiv z = x)$›
using *descriptions*[*axiom-inst*] *act*−*quant*−*uniq* ≡*E*(*5*) **by** *fast*

AOT-act-theorem *hintikka*: ‹$x = \iota x(\varphi\{x\}) \equiv (\varphi\{x\}\ \&\ \forall\,z\ (\varphi\{z\} \rightarrow z = x))$›
using *Commutativity of* ≡[*THEN* ≡*E*(*1*)] *term*−*out*:*3*
fund−*cont*−*desc* ≡*E*(*5*) **by** *blast*

locale *russell-axiom* =
fixes ψ
assumes *ψ-denotes-asm*: $[v \models \psi\{\kappa\}] \Longrightarrow [v \models \kappa\downarrow]$
begin
AOT-act-theorem *russell*−*axiom*:
‹$\psi\{\iota x\ \varphi\{x\}\} \equiv \exists\,x(\varphi\{x\}\ \&\ \forall\,z(\varphi\{z\} \rightarrow z = x)\ \&\ \psi\{x\})$›
proof −
AOT-have *b*: ‹$\forall\,x\ (x = \iota x\ \varphi\{x\} \equiv (\varphi\{x\}\ \&\ \forall\,z(\varphi\{z\} \rightarrow z = x)))$›
using *hintikka* ∀*I* **by** *fast*
show *?thesis*
proof(*rule* ≡*I*; *rule* →*I*)
AOT-assume *c*: ‹$\psi\{\iota x\ \varphi\{x\}\}$›
AOT-hence *d*: ‹$\iota x\ \varphi\{x\}\downarrow$›
using *ψ-denotes-asm* **by** *blast*

```
    AOT-hence ‹∃ y (y = ιx φ{x})›
      by (metis rule=I:1 existential:1)
    then AOT-obtain a where a-def: ‹a = ιx φ{x}›
      using instantiation[rotated] by blast
    moreover AOT-have ‹a = ιx φ{x} ≡ (φ{a} & ∀ z(φ{z} → z = a))›
      using b ∀ E by blast
    ultimately AOT-have ‹φ{a} & ∀ z(φ{z} → z = a)›
      using ≡E by blast
    moreover AOT-have ‹ψ{a}›
    proof –
      AOT-have 1: ‹∀ x∀ y(x = y → y = x)›
        by (simp add: id–eq:2 universal–cor)
      AOT-have ‹a = ιx φ{x} →  ιx φ{x} = a›
        by (rule ∀ E(1)[where τ=«ιx φ{x}»]; rule ∀ E(2)[where β=a])
           (auto simp: 1 d universal–cor)
      AOT-thus ‹ψ{a}›
        using a-def c rule=E →E by blast
    qed
    ultimately AOT-have ‹φ{a} & ∀ z(φ{z} → z = a) & ψ{a}› by (rule &I)
    AOT-thus ‹∃ x(φ{x} & ∀ z(φ{z} → z = x) & ψ{x})› by (rule ∃ I)
  next
    AOT-assume ‹∃ x(φ{x} & ∀ z(φ{z} → z = x) & ψ{x})›
    then AOT-obtain b where g: ‹φ{b} & ∀ z(φ{z} → z = b) & ψ{b}›
      using instantiation[rotated] by blast
    AOT-hence h: ‹b = ιx φ{x} ≡ (φ{b} & ∀ z(φ{z} → z = b))›
      using b ∀ E by blast
    AOT-have ‹φ{b} & ∀ z(φ{z} → z = b)› and j: ‹ψ{b}›
      using g &E by blast+
    AOT-hence ‹b = ιx φ{x}› using h ≡E by blast
    AOT-thus ‹ψ{ιx φ{x}}› using j rule=E by blast
  qed
qed
end

interpretation russell–axiom[exe,1]: russell-axiom ‹λ κ . «[Π]κ»›
  by standard (metis cqt:5:a[1][axiom-inst, THEN →E] &E(2))
interpretation russell–axiom[exe,2,1,1]: russell-axiom ‹λ κ . «[Π]κκ'»›
  by standard (metis cqt:5:a[2][axiom-inst, THEN →E] &E)
interpretation russell–axiom[exe,2,1,2]: russell-axiom ‹λ κ . «[Π]κ'κ»›
  by standard (metis cqt:5:a[2][axiom-inst, THEN →E] &E(2))
interpretation russell–axiom[exe,2,2]: russell-axiom ‹λ κ . «[Π]κκ»›
  by standard (metis cqt:5:a[2][axiom-inst, THEN →E] &E(2))
interpretation russell–axiom[exe,3,1,1]: russell-axiom ‹λ κ . «[Π]κκ'κ''»›
  by standard (metis cqt:5:a[3][axiom-inst, THEN →E] &E)
interpretation russell–axiom[exe,3,1,2]: russell-axiom ‹λ κ . «[Π]κ'κκ''»›
  by standard (metis cqt:5:a[3][axiom-inst, THEN →E] &E)
interpretation russell–axiom[exe,3,1,3]: russell-axiom ‹λ κ . «[Π]κ'κ''κ»›
  by standard (metis cqt:5:a[3][axiom-inst, THEN →E] &E(2))
interpretation russell–axiom[exe,3,2,1]: russell-axiom ‹λ κ . «[Π]κκκ'»›
  by standard (metis cqt:5:a[3][axiom-inst, THEN →E] &E)
interpretation russell–axiom[exe,3,2,2]: russell-axiom ‹λ κ . «[Π]κκ'κ»›
  by standard (metis cqt:5:a[3][axiom-inst, THEN →E] &E(2))
interpretation russell–axiom[exe,3,2,3]: russell-axiom ‹λ κ . «[Π]κ'κκ»›
  by standard (metis cqt:5:a[3][axiom-inst, THEN →E] &E(2))
interpretation russell–axiom[exe,3,3]: russell-axiom ‹λ κ . «[Π]κκκ»›
  by standard (metis cqt:5:a[3][axiom-inst, THEN →E] &E(2))

interpretation russell–axiom[enc,1]: russell-axiom ‹λ κ . «κ[Π]»›
  by standard (metis cqt:5:b[1][axiom-inst, THEN →E] &E(2))
interpretation russell–axiom[enc,2,1]: russell-axiom ‹λ κ . «κκ'[Π]»›
  by standard (metis cqt:5:b[2][axiom-inst, THEN →E] &E)
interpretation russell–axiom[enc,2,2]: russell-axiom ‹λ κ . «κ'κ[Π]»›
  by standard (metis cqt:5:b[2][axiom-inst, THEN →E] &E(2))
```

```
interpretation russell−axiom[enc,2,3]: russell-axiom ‹λ κ . «κκ[Π]»›
  by standard (metis cqt:5:b[2][axiom-inst, THEN →E] &E(2))
interpretation russell−axiom[enc,3,1,1]: russell-axiom ‹λ κ . «κκ'κ''[Π]»›
  by standard (metis cqt:5:b[3][axiom-inst, THEN →E] &E)
interpretation russell−axiom[enc,3,1,2]: russell-axiom ‹λ κ . «κ'κκ''[Π]»›
  by standard (metis cqt:5:b[3][axiom-inst, THEN →E] &E)
interpretation russell−axiom[enc,3,1,3]: russell-axiom ‹λ κ . «κ'κ''κ[Π]»›
  by standard (metis cqt:5:b[3][axiom-inst, THEN →E] &E(2))
interpretation russell−axiom[enc,3,2,1]: russell-axiom ‹λ κ . «κκκ'[Π]»›
  by standard (metis cqt:5:b[3][axiom-inst, THEN →E] &E)
interpretation russell−axiom[enc,3,2,2]: russell-axiom ‹λ κ . «κκ'κ[Π]»›
  by standard (metis cqt:5:b[3][axiom-inst, THEN →E] &E(2))
interpretation russell−axiom[enc,3,2,3]: russell-axiom ‹λ κ . «κ'κκ[Π]»›
  by standard (metis cqt:5:b[3][axiom-inst, THEN →E] &E(2))
interpretation russell−axiom[enc,3,3]: russell-axiom ‹λ κ . «κκκ[Π]»›
  by standard (metis cqt:5:b[3][axiom-inst, THEN →E] &E(2))

AOT-act-theorem !−exists:1: ‹ιx φ{x}↓ ≡ ∃!x φ{x}›
proof(rule ≡I; rule →I)
  AOT-assume ‹ιx φ{x}↓›
  AOT-hence ‹∃ y (y = ιx φ{x})› by (metis rule=I:1 existential:1)
  then AOT-obtain a where ‹a = ιx φ{x}›
    using instantiation[rotated] by blast
  AOT-hence ‹φ{a} & ∀ z (φ{z} → z = a)›
    using hintikka ≡E by blast
  AOT-hence ‹∃ x (φ{x} & ∀ z (φ{z} → z = x))›
    by (rule ∃ I)
  AOT-thus ‹∃!x φ{x}›
    using uniqueness:1[THEN ≡df I] by blast
next
  AOT-assume ‹∃!x φ{x}›
  AOT-hence ‹∃ x (φ{x} & ∀ z (φ{z} → z = x))›
    using uniqueness:1[THEN ≡df E] by blast
  then AOT-obtain b where ‹φ{b} & ∀ z (φ{z} → z = b)›
    using instantiation[rotated] by blast
  AOT-hence ‹b = ιx φ{x}›
    using hintikka ≡E by blast
  AOT-thus ‹ιx φ{x}↓›
    by (metis t=t−proper:2 vdash−properties:6)
qed

AOT-act-theorem !−exists:2: ‹∃ y(y=ιx φ{x}) ≡ ∃!x φ{x}›
  using !−exists:1 free−thms:1 ≡E(6) by blast

AOT-act-theorem y−in:1: ‹x = ιx φ{x} → φ{x}›
  using &E(1) →I hintikka ≡E(1) by blast


AOT-act-theorem y−in:2: ‹z = ιx φ{x} → φ{z}› using y−in:1.

AOT-act-theorem y−in:3: ‹ιx φ{x}↓ → φ{ιx φ{x}}›
proof(rule →I)
  AOT-assume ‹ιx φ{x}↓›
  AOT-hence ‹∃ y (y = ιx φ{x})›
    by (metis rule=I:1 existential:1)
  then AOT-obtain a where ‹a = ιx φ{x}›
    using instantiation[rotated] by blast
  moreover AOT-have ‹φ{a}›
    using calculation hintikka ≡E(1) &E by blast
  ultimately AOT-show ‹φ{ιx φ{x}}› using rule=E by blast
qed

AOT-act-theorem y−in:4: ‹∃ y (y = ιx φ{x}) → φ{ιx φ{x}}›
```

using *y−in:3*[*THEN* →*E*] *free−thms:1*[*THEN* ≡*E*(*2*)] →*I* **by** *blast*

AOT-theorem *act−quant−nec*:
‹∀β ($\mathcal{A}\varphi\{\beta\}$ ≡ β = α) ≡ ∀β($\mathcal{A}\mathcal{A}\varphi\{\beta\}$ ≡ β = α)›
proof(*rule* ≡*I*; *rule* →*I*)
AOT-assume ‹∀β ($\mathcal{A}\varphi\{\beta\}$ ≡ β = α)›
AOT-hence ‹$\mathcal{A}\varphi\{\beta\}$ ≡ β = α› **for** β **using** ∀*E* **by** *blast*
AOT-hence ‹$\mathcal{A}\mathcal{A}\varphi\{\beta\}$ ≡ β = α› **for** β
by (*metis Act−Basic:5 act−conj−act:4* ≡*E*(*1*) ≡*E*(*5*))
AOT-thus ‹∀β($\mathcal{A}\mathcal{A}\varphi\{\beta\}$ ≡ β = α)›
by (*rule* ∀*I*)
next
AOT-assume ‹∀β($\mathcal{A}\mathcal{A}\varphi\{\beta\}$ ≡ β = α)›
AOT-hence ‹$\mathcal{A}\mathcal{A}\varphi\{\beta\}$ ≡ β = α› **for** β **using** ∀*E* **by** *blast*
AOT-hence ‹$\mathcal{A}\varphi\{\beta\}$ ≡ β = α› **for** β
by (*metis Act−Basic:5 act−conj−act:4* ≡*E*(*1*) ≡*E*(*6*))
AOT-thus ‹∀β ($\mathcal{A}\varphi\{\beta\}$ ≡ β = α)›
by (*rule* ∀*I*)
qed

AOT-theorem *equi−desc−descA:1*: ‹x = ιx φ{x} ≡ x = ιx($\mathcal{A}\varphi\{x\}$)›
proof −
AOT-have ‹x = ιx φ{x} ≡ ∀z ($\mathcal{A}\varphi\{z\}$ ≡ z = x)›
using *descriptions*[*axiom-inst*] **by** *blast*
also AOT-have ‹... ≡ ∀z ($\mathcal{A}\mathcal{A}\varphi\{z\}$ ≡ z = x)›
proof(*rule* ≡*I*; *rule* →*I*; *rule* ∀*I*)
AOT-assume ‹∀z ($\mathcal{A}\varphi\{z\}$ ≡ z = x)›
AOT-hence ‹$\mathcal{A}\varphi\{a\}$ ≡ a = x› **for** *a*
using ∀*E* **by** *blast*
AOT-thus ‹$\mathcal{A}\mathcal{A}\varphi\{a\}$ ≡ a = x› **for** *a*
by (*metis Act−Basic:5 act−conj−act:4* ≡*E*(*1*) ≡*E*(*5*))
next
AOT-assume ‹∀z ($\mathcal{A}\mathcal{A}\varphi\{z\}$ ≡ z = x)›
AOT-hence ‹$\mathcal{A}\mathcal{A}\varphi\{a\}$ ≡ a = x› **for** *a*
using ∀*E* **by** *blast*
AOT-thus ‹$\mathcal{A}\varphi\{a\}$ ≡ a = x› **for** *a*
by (*metis Act−Basic:5 act−conj−act:4* ≡*E*(*1*) ≡*E*(*6*))
qed
also AOT-have ‹... ≡ x = ιx($\mathcal{A}\varphi\{x\}$)›
using *Commutativity of* ≡[*THEN* ≡*E*(*1*)] *descriptions*[*axiom-inst*] **by** *fast*
finally show *?thesis* **.**
qed

AOT-theorem *equi−desc−descA:2*: ‹ιx φ{x}↓ → ιx φ{x} = ιx($\mathcal{A}\varphi\{x\}$)›
proof(*rule* →*I*)
AOT-assume ‹ιx φ{x}↓›
AOT-hence ‹∃y (y = ιx φ{x})›
by (*metis rule=I:1 existential:1*)
then AOT-obtain *a* **where** ‹a = ιx φ{x}›
using *instantiation*[*rotated*] **by** *blast*
moreover AOT-have ‹a = ιx($\mathcal{A}\varphi\{x\}$)›
using *calculation equi−desc−descA:1*[*THEN* ≡*E*(*1*)] **by** *blast*
ultimately AOT-show ‹ιx φ{x} = ιx($\mathcal{A}\varphi\{x\}$)›
using *rule=E* **by** *fast*
qed

AOT-theorem *nec−hintikka−scheme*:
‹x = ιx φ{x} ≡ $\mathcal{A}\varphi\{x\}$ & ∀z($\mathcal{A}\varphi\{z\}$ → z = x)›
proof −
AOT-have ‹x = ιx φ{x} ≡ ∀z($\mathcal{A}\varphi\{z\}$ ≡ z = x)›
using *descriptions*[*axiom-inst*] **by** *blast*
also AOT-have ‹. . . ≡ ($\mathcal{A}\varphi\{x\}$ & ∀z($\mathcal{A}\varphi\{z\}$ → z = x))›

```
    using Commutativity of ≡[THEN ≡E(1)] term−out:3 by fast
  finally show ?thesis.
qed

AOT-theorem equiv−desc−eq:1:
  ‹𝒜∀x(φ{x} ≡ ψ{x}) → ∀x (x = ιx φ{x} ≡ x = ιx ψ{x})›
proof(rule →I; rule ∀I)
  fix β
  AOT-assume ‹𝒜∀x(φ{x} ≡ ψ{x})›
  AOT-hence ‹𝒜(φ{x} ≡ ψ{x})› for x
    using logic−actual−nec:3[axiom-inst, THEN ≡E(1)] ∀E(2) by blast
  AOT-hence 0: ‹𝒜φ{x} ≡ 𝒜ψ{x}› for x
    by (metis Act−Basic:5 ≡E(1))
  AOT-have ‹β = ιx φ{x} ≡ 𝒜φ{β} & ∀z(𝒜φ{z} → z = β)›
    using nec−hintikka−scheme by blast
  also AOT-have ‹... ≡ 𝒜ψ{β} & ∀z(𝒜ψ{z} → z = β)›
  proof (rule ≡I; rule →I)
    AOT-assume 1: ‹𝒜φ{β} & ∀z(𝒜φ{z} → z = β)›
    AOT-hence ‹𝒜φ{z} → z = β› for z
      using &E ∀E by blast
    AOT-hence ‹𝒜ψ{z} → z = β› for z
      using 0 ≡E →I →E by metis
    AOT-hence ‹∀z(𝒜ψ{z} → z = β)›
      using ∀I by fast
    moreover AOT-have ‹𝒜ψ{β}›
      using &E 0[THEN ≡E(1)] 1 by blast
    ultimately AOT-show ‹𝒜ψ{β} & ∀z(𝒜ψ{z} → z = β)›
      using &I by blast
  next
    AOT-assume 1: ‹𝒜ψ{β} & ∀z(𝒜ψ{z} → z = β)›
    AOT-hence ‹𝒜ψ{z} → z = β› for z
      using &E ∀E by blast
    AOT-hence ‹𝒜φ{z} → z = β› for z
      using 0 ≡E →I →E by metis
    AOT-hence ‹∀z(𝒜φ{z} → z = β)›
      using ∀I by fast
    moreover AOT-have ‹𝒜φ{β}›
      using &E 0[THEN ≡E(2)] 1 by blast
    ultimately AOT-show ‹𝒜φ{β} & ∀z(𝒜φ{z} → z = β)›
      using &I by blast
  qed
  also AOT-have ‹... ≡ β = ιx ψ{x}›
    using Commutativity of ≡[THEN ≡E(1)] nec−hintikka−scheme by blast
  finally AOT-show ‹β = ιx φ{x} ≡ β = ιx ψ{x}› .
qed

AOT-theorem equiv−desc−eq:2:
  ‹ιx φ{x}↓ & 𝒜∀x(φ{x} ≡ ψ{x}) → ιx φ{x} = ιx ψ{x}›
proof(rule →I)
  AOT-assume ‹ιx φ{x}↓ & 𝒜∀x(φ{x} ≡ ψ{x})›
  AOT-hence 0: ‹∃y (y = ιx φ{x})› and
            1: ‹∀x (x = ιx φ{x} ≡ x = ιx ψ{x})›
    using &E free−thms:1[THEN ≡E(1)] equiv−desc−eq:1 →E by blast+
  then AOT-obtain a where ‹a = ιx φ{x}›
    using instantiation[rotated] by blast
  moreover AOT-have ‹a = ιx ψ{x}›
    using calculation 1 ∀E ≡E(1) by fast
  ultimately AOT-show ‹ιx φ{x} = ιx ψ{x}›
    using rule=E by fast
qed

AOT-theorem equiv−desc−eq:3:
  ‹ιx φ{x}↓ & □∀x(φ{x} ≡ ψ{x}) → ιx φ{x} = ιx ψ{x}›
```

```
  using →I equiv−desc−eq:2[THEN →E, OF &I] &E
        nec−imp−act[THEN →E] by metis


AOT-theorem equiv−desc−eq:4: ‹ιx φ{x}↓ → □ιx φ{x}↓›
proof(rule →I)
  AOT-assume ‹ιx φ{x}↓›
  AOT-hence ‹∃ y (y = ιx φ{x})›
    by (metis rule=I:1 existential:1)
  then AOT-obtain a where ‹a = ιx φ{x}›
    using instantiation[rotated] by blast
  AOT-thus ‹□ιx φ{x}↓›
    using ex:2:a rule=E by fast
qed

AOT-theorem equiv−desc−eq:5: ‹ιx φ{x}↓ → ∃ y □(y = ιx φ{x})›
proof(rule →I)
  AOT-assume ‹ιx φ{x}↓›
  AOT-hence ‹∃ y (y = ιx φ{x})›
    by (metis rule=I:1 existential:1)
  then AOT-obtain a where ‹a = ιx φ{x}›
    using instantiation[rotated] by blast
  AOT-hence ‹□(a = ιx φ{x})›
    by (metis id−nec:2 vdash−properties:10)
  AOT-thus ‹∃ y □(y = ιx φ{x})›
    by (rule ∃ I)
qed

AOT-act-theorem equiv−desc−eq2:1:
  ‹∀ x (φ{x} ≡ ψ{x}) → ∀ x (x = ιx φ{x} ≡ x = ιx ψ{x})›
  using →I logic−actual[act-axiom-inst, THEN →E]
        equiv−desc−eq:1[THEN →E]
        RA[1] deduction−theorem by blast

AOT-act-theorem equiv−desc−eq2:2:
  ‹ιx φ{x}↓ & ∀ x (φ{x} ≡ ψ{x}) → ιx φ{x} = ιx ψ{x}›
  using →I logic−actual[act-axiom-inst, THEN →E]
        equiv−desc−eq:2[THEN →E, OF &I]
        RA[1] deduction−theorem &E by metis

context russell-axiom
begin
AOT-theorem nec−russell−axiom:
  ‹ψ{ιx φ{x}} ≡ ∃ x(𝓐φ{x} & ∀ z(𝓐φ{z} → z = x) & ψ{x})›
proof −
  AOT-have b: ‹∀ x (x = ιx φ{x} ≡ (𝓐φ{x} & ∀ z(𝓐φ{z} → z = x)))›
    using nec−hintikka−scheme ∀ I by fast
  show ?thesis
  proof(rule ≡I; rule →I)
    AOT-assume c: ‹ψ{ιx φ{x}}›
    AOT-hence d: ‹ιx φ{x}↓›
      using ψ-denotes-asm by blast
    AOT-hence ‹∃ y (y = ιx φ{x})›
      by (metis rule=I:1 existential:1)
    then AOT-obtain a where a-def: ‹a = ιx φ{x}›
      using instantiation[rotated] by blast
    moreover AOT-have ‹a = ιx φ{x} ≡ (𝓐φ{a} & ∀ z(𝓐φ{z} → z = a))›
      using b ∀ E by blast
    ultimately AOT-have ‹𝓐φ{a} & ∀ z(𝓐φ{z} → z = a)›
      using ≡E by blast
    moreover AOT-have ‹ψ{a}›
    proof −
      AOT-have 1: ‹∀ x∀ y(x = y → y = x)›
```

```
        by (simp add: id−eq:2 universal−cor)
      AOT-have ‹a = ιx φ{x} →  ιx φ{x} = a›
        by (rule ∀E(1)[where τ=«ιx φ{x}»]; rule ∀E(2)[where β=a])
          (auto simp: d universal−cor 1)
      AOT-thus ‹ψ{a}›
        using a-def c rule=E →E by metis
    qed
    ultimately AOT-have ‹𝓐φ{a} & ∀z(𝓐φ{z} → z = a) & ψ{a}›
      by (rule &I)
    AOT-thus ‹∃x(𝓐φ{x} & ∀z(𝓐φ{z} → z = x) & ψ{x})›
      by (rule ∃I)
  next
    AOT-assume ‹∃x(𝓐φ{x} & ∀z(𝓐φ{z} → z = x) & ψ{x})›
    then AOT-obtain b where g: ‹𝓐φ{b} & ∀z(𝓐φ{z} → z = b) & ψ{b}›
      using instantiation[rotated] by blast
    AOT-hence h: ‹b = ιx φ{x} ≡ (𝓐φ{b} & ∀z(𝓐φ{z} → z = b))›
      using b ∀E by blast
    AOT-have ‹𝓐φ{b} & ∀z(𝓐φ{z} → z = b)› and j: ‹ψ{b}›
      using g &E by blast+
    AOT-hence ‹b = ιx φ{x}›
      using h ≡E by blast
    AOT-thus ‹ψ{ιx φ{x}}›
      using j rule=E by blast
  qed
qed
end

AOT-theorem actual−desc:1: ‹ιx φ{x}↓ ≡ ∃!x 𝓐φ{x}›
proof (rule ≡I; rule →I)
  AOT-assume ‹ιx φ{x}↓›
  AOT-hence ‹∃y (y = ιx φ{x})›
    by (metis rule=I:1 existential:1)
  then AOT-obtain a where ‹a = ιx φ{x}›
    using instantiation[rotated] by blast
  moreover AOT-have ‹a = ιx φ{x} ≡ ∀z(𝓐φ{z} ≡ z = a)›
    using descriptions[axiom-inst] by blast
  ultimately AOT-have ‹∀z(𝓐φ{z} ≡ z = a)›
    using ≡E by blast
  AOT-hence ‹∃x∀z(𝓐φ{z} ≡ z = x)› by (rule ∃I)
  AOT-thus ‹∃!x 𝓐φ{x}›
    using uniqueness:2[THEN ≡E(2)] by fast
next
  AOT-assume ‹∃!x 𝓐φ{x}›
  AOT-hence ‹∃x∀z(𝓐φ{z} ≡ z = x)›
    using uniqueness:2[THEN ≡E(1)] by fast
  then AOT-obtain a where ‹∀z(𝓐φ{z} ≡ z = a)›
    using instantiation[rotated] by blast
  moreover AOT-have ‹a = ιx φ{x} ≡ ∀z(𝓐φ{z} ≡ z = a)›
    using descriptions[axiom-inst] by blast
  ultimately AOT-have ‹a = ιx φ{x}›
    using ≡E by blast
  AOT-thus ‹ιx φ{x}↓›
    by (metis t=t−proper:2 vdash−properties:6)
qed

AOT-theorem actual−desc:2: ‹x = ιx φ{x} → 𝓐φ{x}›
  using &E(1) contraposition:1[2] ≡E(1) nec−hintikka−scheme
      reductio−aa:2 vdash−properties:9 by blast


AOT-theorem actual−desc:3: ‹z = ιx φ{x} → 𝓐φ{z}›
  using actual−desc:2.
```

AOT-theorem *actual−desc:4*: ‹$\iota x\ \varphi\{x\}\downarrow \rightarrow \mathcal{A}\varphi\{\iota x\ \varphi\{x\}\}$›
proof(*rule* →*I*)
  AOT-assume ‹$\iota x\ \varphi\{x\}\downarrow$›
  AOT-hence ‹$\exists\, y\ (y = \iota x\ \varphi\{x\})$› **by** (*metis rule=I:1 existential:1*)
  then AOT-obtain *a* **where** ‹$a = \iota x\ \varphi\{x\}$› **using** *instantiation*[*rotated*] **by** *blast*
  AOT-thus ‹$\mathcal{A}\varphi\{\iota x\ \varphi\{x\}\}$›
    using *actual−desc:2 rule=E* →*E* **by** *fast*
qed

AOT-theorem *actual−desc:5*: ‹$\iota x\ \varphi\{x\} = \iota x\ \psi\{x\} \rightarrow \mathcal{A}\forall\, x(\varphi\{x\} \equiv \psi\{x\})$›
proof(*rule* →*I*)
  AOT-assume *0*: ‹$\iota x\ \varphi\{x\} = \iota x\ \psi\{x\}$›
  AOT-hence *φ-down*: ‹$\iota x\ \varphi\{x\}\downarrow$› **and** *ψ-down*: ‹$\iota x\ \psi\{x\}\downarrow$›
    using *t=t−proper:1 t=t−proper:2 vdash−properties:6* **by** *blast+*
  AOT-hence ‹$\exists\, y\ (y = \iota x\ \varphi\{x\})$› **and** ‹$\exists\, y\ (y = \iota x\ \psi\{x\})$›
    by (*metis rule=I:1 existential:1*)+
  then AOT-obtain *a* **and** *b* **where** *a-eq*: ‹$a = \iota x\ \varphi\{x\}$› **and** *b-eq*: ‹$b = \iota x\ \psi\{x\}$›
    using *instantiation*[*rotated*] **by** *metis*

  AOT-have ‹$\forall\,\alpha\forall\,\beta\ (\alpha = \beta \rightarrow \beta = \alpha)$›
    by (*rule* ∀*I*; *rule* ∀*I*; *rule id−eq:2*)
  AOT-hence ‹$\forall\,\beta\ (\iota x\ \varphi\{x\} = \beta \rightarrow \beta = \iota x\ \varphi\{x\})$›
    using ∀*E φ-down* **by** *blast*
  AOT-hence ‹$\iota x\ \varphi\{x\} = \iota x\ \psi\{x\} \rightarrow \iota x\ \psi\{x\} = \iota x\ \varphi\{x\}$›
    using ∀*E ψ-down* **by** *blast*
  AOT-hence *1*: ‹$\iota x\ \psi\{x\} = \iota x\ \varphi\{x\}$› **using** *0*
    →*E* **by** *blast*

  AOT-have ‹$\mathcal{A}\varphi\{x\} \equiv \mathcal{A}\psi\{x\}$› **for** *x*
  proof(*rule* ≡*I*; *rule* →*I*)
    AOT-assume ‹$\mathcal{A}\varphi\{x\}$›
    moreover AOT-have ‹$\mathcal{A}\varphi\{x\} \rightarrow x = a$› **for** *x*
      using *nec−hintikka−scheme*[*THEN* ≡*E*(*1*), *OF a-eq*, *THEN* &*E*(*2*)]
          ∀*E* **by** *blast*
    ultimately AOT-have ‹$x = a$›
      using →*E* **by** *blast*
    AOT-hence ‹$x = \iota x\ \varphi\{x\}$›
      using *a-eq rule=E* **by** *blast*
    AOT-hence ‹$x = \iota x\ \psi\{x\}$›
      using *0 rule=E* **by** *blast*
    AOT-thus ‹$\mathcal{A}\psi\{x\}$›
      by (*metis actual−desc:3 vdash−properties:6*)
  next
    AOT-assume ‹$\mathcal{A}\psi\{x\}$›
    moreover AOT-have ‹$\mathcal{A}\psi\{x\} \rightarrow x = b$› **for** *x*
      using *nec−hintikka−scheme*[*THEN* ≡*E*(*1*), *OF b-eq*, *THEN* &*E*(*2*)]
          ∀*E* **by** *blast*
    ultimately AOT-have ‹$x = b$›
      using →*E* **by** *blast*
    AOT-hence ‹$x = \iota x\ \psi\{x\}$›
      using *b-eq rule=E* **by** *blast*
    AOT-hence ‹$x = \iota x\ \varphi\{x\}$›
      using *1 rule=E* **by** *blast*
    AOT-thus ‹$\mathcal{A}\varphi\{x\}$›
      by (*metis actual−desc:3 vdash−properties:6*)
  qed
  AOT-hence ‹$\mathcal{A}(\varphi\{x\} \equiv \psi\{x\})$› **for** *x*
    by (*metis Act−Basic:5* ≡*E*(*2*))
  AOT-hence ‹$\forall\, x\ \mathcal{A}(\varphi\{x\} \equiv \psi\{x\})$›
    by (*rule* ∀*I*)
  AOT-thus ‹$\mathcal{A}\forall\, x\ (\varphi\{x\} \equiv \psi\{x\})$›
    using *logic−actual−nec:3*[*axiom-inst*, *THEN* ≡*E*(*2*)] **by** *fast*
qed

```
AOT-theorem !box-desc:1: ‹∃!x □φ{x} → ∀ y (y = ιx φ{x} → φ{y})›
proof(rule →I)
  AOT-assume ‹∃!x □φ{x}›
  AOT-hence ζ: ‹∃ x (□φ{x} & ∀ z (□φ{z} → z = x))›
    using uniqueness:1[THEN ≡df E] by blast
  then AOT-obtain b where ϑ: ‹□φ{b} & ∀ z (□φ{z} → z = b)›
    using instantiation[rotated] by blast
  AOT-show ‹∀ y (y = ιx φ{x} → φ{y})›
  proof(rule GEN; rule →I)
    fix y
    AOT-assume ‹y = ιx φ{x}›
    AOT-hence ‹𝓐φ{y} & ∀ z (𝓐φ{z} → z = y)›
      using nec-hintikka-scheme[THEN ≡E(1)] by blast
    AOT-hence ‹𝓐φ{b} → b = y›
      using &E ∀ E by blast
    moreover AOT-have ‹𝓐φ{b}›
      using ϑ[THEN &E(1)]  by (metis nec-imp-act →E)
    ultimately AOT-have ‹b = y›
      using →E by blast
    moreover AOT-have ‹φ{b}›
      using ϑ[THEN &E(1)]  by (metis qml:2[axiom-inst] →E)
    ultimately AOT-show ‹φ{y}›
      using rule=E by blast
  qed
qed

AOT-theorem !box-desc:2:
  ‹∀ x (φ{x} → □φ{x}) → (∃!x φ{x} → ∀ y (y = ιx φ{x} → φ{y}))›
proof(rule →I; rule →I)
  AOT-assume ‹∀ x (φ{x} → □φ{x})›
  moreover AOT-assume ‹∃!x φ{x}›
  ultimately AOT-have ‹∃!x □φ{x}›
    using nec-exist-![THEN →E, THEN →E] by blast
  AOT-thus ‹∀ y (y = ιx φ{x} → φ{y})›
    using !box-desc:1 →E by blast
qed


AOT-theorem dr-alphabetic-thm: ‹ιν φ{ν}↓ → ιν φ{ν} = ιμ φ{μ}›
  by (simp add: rule=I:1 →I)
```

8.9 The Theory of Necessity

```
AOT-theorem RM:1[prem]:
  assumes ‹Γ ⊢□ φ → ψ›
  shows ‹□Γ ⊢□ □φ → □ψ›
proof -
  AOT-have ‹□Γ ⊢□ □(φ → ψ)›
    using RN[prem] assms by blast
  AOT-thus ‹□Γ ⊢□ □φ → □ψ›
    by (metis qml:1[axiom-inst] →E)
qed

AOT-theorem RM:1:
  assumes ‹⊢□ φ → ψ›
  shows ‹⊢□ □φ → □ψ›
  using RM:1[prem] assms by blast

lemmas RM = RM:1

AOT-theorem RM:2[prem]:
  assumes ‹Γ ⊢□ φ → ψ›
```

shows ‹$\Box\Gamma \vdash_\Box \Diamond\varphi \rightarrow \Diamond\psi$›
proof −
AOT-have ‹$\Gamma \vdash_\Box \neg\psi \rightarrow \neg\varphi$›
using *assms*
by (*simp add*: *contraposition*:*1*[*1*])
AOT-hence ‹$\Box\Gamma \vdash_\Box \Box\neg\psi \rightarrow \Box\neg\varphi$›
using *RM*:*1*[*prem*] **by** *blast*
AOT-thus ‹$\Box\Gamma \vdash_\Box \Diamond\varphi \rightarrow \Diamond\psi$›
by (*meson* $\equiv_{df}E$ $\equiv_{df}I$ *conventions*:*5* $\rightarrow I$ *modus−tollens*:*1*)
qed

AOT-theorem *RM*:*2*:
assumes ‹$\vdash_\Box \varphi \rightarrow \psi$›
shows ‹$\vdash_\Box \Diamond\varphi \rightarrow \Diamond\psi$›
using *RM*:*2*[*prem*] *assms* **by** *blast*

lemmas *RM*◇ = *RM*:*2*

AOT-theorem *RM*:*3*[*prem*]:
assumes ‹$\Gamma \vdash_\Box \varphi \equiv \psi$›
shows ‹$\Box\Gamma \vdash_\Box \Box\varphi \equiv \Box\psi$›
proof −
AOT-have ‹$\Gamma \vdash_\Box \varphi \rightarrow \psi$› **and** ‹$\Gamma \vdash_\Box \psi \rightarrow \varphi$›
using *assms* $\equiv E$ $\rightarrow I$ **by** *metis+*
AOT-hence ‹$\Box\Gamma \vdash_\Box \Box\varphi \rightarrow \Box\psi$› **and** ‹$\Box\Gamma \vdash_\Box \Box\psi \rightarrow \Box\varphi$›
using *RM*:*1*[*prem*] **by** *metis+*
AOT-thus ‹$\Box\Gamma \vdash_\Box \Box\varphi \equiv \Box\psi$›
by (*simp add*: $\equiv I$)
qed

AOT-theorem *RM*:*3*:
assumes ‹$\vdash_\Box \varphi \equiv \psi$›
shows ‹$\vdash_\Box \Box\varphi \equiv \Box\psi$›
using *RM*:*3*[*prem*] *assms* **by** *blast*

lemmas *RE* = *RM*:*3*

AOT-theorem *RM*:*4*[*prem*]:
assumes ‹$\Gamma \vdash_\Box \varphi \equiv \psi$›
shows ‹$\Box\Gamma \vdash_\Box \Diamond\varphi \equiv \Diamond\psi$›
proof −
AOT-have ‹$\Gamma \vdash_\Box \varphi \rightarrow \psi$› **and** ‹$\Gamma \vdash_\Box \psi \rightarrow \varphi$›
using *assms* $\equiv E$ $\rightarrow I$ **by** *metis+*
AOT-hence ‹$\Box\Gamma \vdash_\Box \Diamond\varphi \rightarrow \Diamond\psi$› **and** ‹$\Box\Gamma \vdash_\Box \Diamond\psi \rightarrow \Diamond\varphi$›
using *RM*:*2*[*prem*] **by** *metis+*
AOT-thus ‹$\Box\Gamma \vdash_\Box \Diamond\varphi \equiv \Diamond\psi$›
by (*simp add*: $\equiv I$)
qed

AOT-theorem *RM*:*4*:
assumes ‹$\vdash_\Box \varphi \equiv \psi$›
shows ‹$\vdash_\Box \Diamond\varphi \equiv \Diamond\psi$›
using *RM*:*4*[*prem*] *assms* **by** *blast*

lemmas *RE*◇ = *RM*:*4*

AOT-theorem *KBasic*:*1*: ‹$\Box\varphi \rightarrow \Box(\psi \rightarrow \varphi)$›
by (*simp add*: *RM pl*:*1*[*axiom-inst*])

AOT-theorem *KBasic*:*2*: ‹$\Box\neg\varphi \rightarrow \Box(\varphi \rightarrow \psi)$›
by (*simp add*: *RM useful−tautologies*:*3*)

AOT-theorem *KBasic*:*3*: ‹$\Box(\varphi \;\&\; \psi) \equiv (\Box\varphi \;\&\; \Box\psi)$›

```
proof (rule ≡I; rule →I)
  AOT-assume ‹□(φ & ψ)›
  AOT-thus ‹□φ & □ψ›
    by (meson RM &I Conjunction Simplification(1, 2) →E)
next
  AOT-have ‹□φ → □(ψ → (φ & ψ))›
    by (simp add: RM:1 Adjunction)
  AOT-hence ‹□φ → (□ψ → □(φ & ψ))›
    by (metis Hypothetical Syllogism qml:1[axiom-inst])
  moreover AOT-assume ‹□φ & □ψ›
  ultimately AOT-show ‹□(φ & ψ)›
    using →E &E by blast
qed

AOT-theorem KBasic:4: ‹□(φ ≡ ψ) ≡ (□(φ → ψ) & □(ψ → φ))›
proof –
  AOT-have ϑ: ‹□((φ → ψ) & (ψ → φ)) ≡ (□(φ → ψ) & □(ψ → φ))›
    by (fact KBasic:3)
  AOT-modally-strict {
    AOT-have ‹(φ ≡ ψ) ≡ ((φ → ψ) & (ψ → φ))›
      by (fact conventions:3[THEN ≡Df])
  }
  AOT-hence ξ: ‹□(φ ≡ ψ) ≡ □((φ → ψ) & (ψ → φ))›
    by (rule RE)
  with ξ and ϑ AOT-show ‹□(φ ≡ ψ) ≡ (□(φ → ψ) & □(ψ → φ))›
    using ≡E(5) by blast
qed

AOT-theorem KBasic:5: ‹(□(φ → ψ) & □(ψ → φ)) → (□φ ≡ □ψ)›
proof –
  AOT-have ‹□(φ → ψ) → (□φ → □ψ)›
    by (fact qml:1[axiom-inst])
  moreover AOT-have ‹□(ψ → φ) → (□ψ → □φ)›
    by (fact qml:1[axiom-inst])
  ultimately AOT-have ‹(□(φ → ψ) & □(ψ → φ)) → ((□φ → □ψ) & (□ψ → □φ))›
    by (metis &I MP Double Composition)
  moreover AOT-have ‹((□φ → □ψ) & (□ψ → □φ)) → (□φ ≡ □ψ)›
    using conventions:3[THEN ≡_df I] →I by blast
  ultimately AOT-show ‹(□(φ → ψ) & □(ψ → φ)) → (□φ ≡ □ψ)›
    by (metis Hypothetical Syllogism)
qed

AOT-theorem KBasic:6: ‹□(φ ≡ ψ) → (□φ ≡ □ψ)›
  using KBasic:4 KBasic:5 deduction−theorem ≡E(1) →E by blast
AOT-theorem KBasic:7: ‹((□φ & □ψ) ∨ (□¬φ & □¬ψ)) → □(φ ≡ ψ)›
proof (rule →I; drule ∨E(1); (rule →I)?)
  AOT-assume ‹□φ & □ψ›
  AOT-hence ‹□φ› and ‹□ψ› using &E by blast+
  AOT-hence ‹□(φ → ψ)› and ‹□(ψ → φ)› using KBasic:1 →E by blast+
  AOT-hence ‹□(φ → ψ) & □(ψ → φ)› using &I by blast
  AOT-thus ‹□(φ ≡ ψ)›  by (metis KBasic:4 ≡E(2))
next
  AOT-assume ‹□¬φ & □¬ψ›
  AOT-hence 0: ‹□(¬φ & ¬ψ)› using KBasic:3[THEN ≡E(2)] by blast
  AOT-modally-strict {
    AOT-have ‹(¬φ & ¬ψ) → (φ ≡ ψ)›
      by (metis &E(1) &E(2) deduction−theorem ≡I reductio−aa:1)
  }
  AOT-hence ‹□(¬φ & ¬ψ) → □(φ ≡ ψ)›
    by (rule RM)
  AOT-thus ‹□(φ ≡ ψ)› using 0 →E by blast
qed(auto)
```

AOT-theorem *KBasic:8*: ‹$\Box(\varphi \ \&\ \psi) \rightarrow \Box(\varphi \equiv \psi)$›
  by (*meson RM:1 &E(1) &E(2) deduction−theorem ≡I*)
AOT-theorem *KBasic:9*: ‹$\Box(\neg\varphi \ \&\ \neg\psi) \rightarrow \Box(\varphi \equiv \psi)$›
  by (*metis RM:1 &E(1) &E(2) deduction−theorem ≡I raa−cor:4*)
AOT-theorem *KBasic:10*: ‹$\Box\varphi \equiv \Box\neg\neg\varphi$›
  by (*simp add*: *RM:3 oth−class−taut:3:b*)
AOT-theorem *KBasic:11*: ‹$\neg\Box\varphi \equiv \Diamond\neg\varphi$›
proof (*rule ≡I*; *rule →I*)
  AOT-show ‹$\Diamond\neg\varphi$› **if** ‹$\neg\Box\varphi$›
    using *that* $\equiv_{df}I$ *conventions:5 KBasic:10 ≡E(3)* **by** *blast*
next
  AOT-show ‹$\neg\Box\varphi$› **if** ‹$\Diamond\neg\varphi$›
    using $\equiv_{df}E$ *conventions:5 KBasic:10 ≡E(4) that* **by** *blast*
qed
AOT-theorem *KBasic:12*: ‹$\Box\varphi \equiv \neg\Diamond\neg\varphi$›
proof (*rule ≡I*; *rule →I*)
  AOT-show ‹$\neg\Diamond\neg\varphi$› **if** ‹$\Box\varphi$›
    using *¬¬I KBasic:11 ≡E(3) that* **by** *blast*
next
  AOT-show ‹$\Box\varphi$› **if** ‹$\neg\Diamond\neg\varphi$›
  using *KBasic:11 ≡E(1) reductio−aa:1 that* **by** *blast*
qed
AOT-theorem *KBasic:13*: ‹$\Box(\varphi \rightarrow \psi) \rightarrow (\Diamond\varphi \rightarrow \Diamond\psi)$›
proof −
  AOT-have ‹$\varphi \rightarrow \psi \vdash_\Box \varphi \rightarrow \psi$› **by** *blast*
  AOT-hence ‹$\Box(\varphi \rightarrow \psi) \vdash_\Box \Diamond\varphi \rightarrow \Diamond\psi$›
    using *RM:2[prem]* **by** *blast*
  AOT-thus ‹$\Box(\varphi \rightarrow \psi) \rightarrow (\Diamond\varphi \rightarrow \Diamond\psi)$› **using** *→I* **by** *blast*
qed
lemmas *K◇* = *KBasic:13*
AOT-theorem *KBasic:14*: ‹$\Diamond\Box\varphi \equiv \neg\Box\Diamond\neg\varphi$›
  by (*meson RE◇ KBasic:11 KBasic:12 ≡E(6) oth−class−taut:3:a*)
AOT-theorem *KBasic:15*: ‹$(\Box\varphi \vee \Box\psi) \rightarrow \Box(\varphi \vee \psi)$›
proof −
  AOT-modally-strict {
    AOT-have ‹$\varphi \rightarrow (\varphi \vee \psi)$› **and** ‹$\psi \rightarrow (\varphi \vee \psi)$›
      by (*auto simp*: *Disjunction Addition(1) Disjunction Addition(2)*)
  }
  AOT-hence ‹$\Box\varphi \rightarrow \Box(\varphi \vee \psi)$› **and** ‹$\Box\psi \rightarrow \Box(\varphi \vee \psi)$›
    using *RM* **by** *blast+*
  AOT-thus ‹$(\Box\varphi \vee \Box\psi) \rightarrow \Box(\varphi \vee \psi)$›
    by (*metis ∨E(1) deduction−theorem*)
qed

AOT-theorem *KBasic:16*: ‹$(\Box\varphi \ \&\ \Diamond\psi) \rightarrow \Diamond(\varphi \ \&\ \psi)$›
  by (*meson KBasic:13 RM:1 Adjunction Hypothetical Syllogism*
          Importation →E)

AOT-theorem *rule−sub−lem:1:a*:
  assumes ‹$\vdash_\Box \Box(\psi \equiv \chi)$›
  shows ‹$\vdash_\Box \neg\psi \equiv \neg\chi$›
  using *qml:2[axiom-inst, THEN →E, OF assms]*
      ≡E(1) oth−class−taut:4:b **by** *blast*

AOT-theorem *rule−sub−lem:1:b*:
  assumes ‹$\vdash_\Box \Box(\psi \equiv \chi)$›
  shows ‹$\vdash_\Box (\psi \rightarrow \Theta) \equiv (\chi \rightarrow \Theta)$›
  using *qml:2[axiom-inst, THEN →E, OF assms]*
  using *oth−class−taut:4:c vdash−properties:6* **by** *blast*

AOT-theorem *rule−sub−lem:1:c*:
  assumes ‹$\vdash_\Box \Box(\psi \equiv \chi)$›
  shows ‹$\vdash_\Box (\Theta \rightarrow \psi) \equiv (\Theta \rightarrow \chi)$›

using *qml*:*2*[*axiom-inst*, *THEN* →*E*, *OF assms*]
using *oth−class−taut*:*4*:*d vdash−properties*:*6* **by** *blast*

AOT-theorem *rule−sub−lem*:*1*:*d*:
assumes ‹*for arbitrary* α: ⊢□ □(ψ{α} ≡ χ{α})›
shows ‹⊢□ ∀ α ψ{α} ≡ ∀ α χ{α}›
proof −
AOT-modally-strict {
AOT-have ‹∀ α (ψ{α} ≡ χ{α})›
using *qml*:*2*[*axiom-inst*, *THEN* →*E*, *OF assms*] ∀ *I* **by** *fast*
AOT-hence *0*: ‹ψ{α} ≡ χ{α}› **for** α **using** ∀ *E* **by** *blast*
AOT-show ‹∀ α ψ{α} ≡ ∀ α χ{α}›
proof (*rule* ≡*I*; *rule* →*I*)
AOT-assume ‹∀ α ψ{α}›
AOT-hence ‹ψ{α}› **for** α **using** ∀ *E* **by** *blast*
AOT-hence ‹χ{α}› **for** α **using** *0* ≡*E* **by** *blast*
AOT-thus ‹∀ α χ{α}› **by** (*rule* ∀ *I*)
next
AOT-assume ‹∀ α χ{α}›
AOT-hence ‹χ{α}› **for** α **using** ∀ *E* **by** *blast*
AOT-hence ‹ψ{α}› **for** α **using** *0* ≡*E* **by** *blast*
AOT-thus ‹∀ α ψ{α}› **by** (*rule* ∀ *I*)
qed
}
qed

AOT-theorem *rule−sub−lem*:*1*:*e*:
assumes ‹⊢□ □(ψ ≡ χ)›
shows ‹⊢□ [λ ψ] ≡ [λ χ]›
using *qml*:*2*[*axiom-inst*, *THEN* →*E*, *OF assms*]
using ≡*E*(*1*) *propositions−lemma*:*6* **by** *blast*

AOT-theorem *rule−sub−lem*:*1*:*f*:
assumes ‹⊢□ □(ψ ≡ χ)›
shows ‹⊢□ ***𝒜***ψ ≡ ***𝒜***χ›
using *qml*:*2*[*axiom-inst*, *THEN* →*E*, *OF assms*, *THEN RA*[*2*]]
by (*metis Act−Basic*:*5* ≡*E*(*1*))

AOT-theorem *rule−sub−lem*:*1*:*g*:
assumes ‹⊢□ □(ψ ≡ χ)›
shows ‹⊢□ □ψ ≡ □χ›
using *KBasic*:*6 assms vdash−properties*:*6* **by** *blast*

Note that instead of deriving *rule−sub−lem*:*2*, *rule−sub−lem*:*3*, *rule−sub−lem*:*4*, and *rule−sub−nec*, we construct substitution methods instead.

class *AOT-subst* =
fixes *AOT-subst* :: (′*a* ⇒ o) ⇒ *bool*
and *AOT-subst-cond* :: ′*a* ⇒ ′*a* ⇒ *bool*
assumes *AOT-subst*:
AOT-subst φ ⟹ *AOT-subst-cond* ψ χ ⟹ [*v* ⊨ «φ ψ» ≡ «φ χ»]

named-theorems *AOT-substI*

instantiation o :: *AOT-subst*
begin

inductive *AOT-subst*-o **where**
AOT-subst-o-*id*[*AOT-substI*]:
‹*AOT-subst*-o (λφ. φ)›
| *AOT-subst*-o-*const*[*AOT-substI*]:
‹*AOT-subst*-o (λφ. ψ)›
| *AOT-subst*-o-*not*[*AOT-substI*]:
‹*AOT-subst*-o Θ ⟹ *AOT-subst*-o (λ φ. «¬Θ{φ}»)›

```
  | AOT-subst-o-imp[AOT-substI]:
    ‹AOT-subst-o Θ ⟹ AOT-subst-o Ξ ⟹ AOT-subst-o (λ φ. «Θ{φ} → Ξ{φ}»)›
  | AOT-subst-o-lambda0[AOT-substI]:
    ‹AOT-subst-o Θ ⟹ AOT-subst-o (λ φ. (AOT-lambda0 (Θ φ)))›
  | AOT-subst-o-act[AOT-substI]:
    ‹AOT-subst-o Θ ⟹ AOT-subst-o (λ φ. «𝓐Θ{φ}»)›
  | AOT-subst-o-box[AOT-substI]:
    ‹AOT-subst-o Θ ⟹ AOT-subst-o (λ φ. «□Θ{φ}»)›
  | AOT-subst-o-by-def[AOT-substI]:
    ‹(⋀ ψ . AOT-model-equiv-def (Θ ψ) (Ξ ψ)) ⟹
      AOT-subst-o Ξ ⟹ AOT-subst-o Θ›


definition AOT-subst-cond-o where
  ‹AOT-subst-cond-o ≡ λ ψ χ . ∀ v . [v ⊨ ψ ≡ χ]›

instance
proof
  fix ψ χ :: o and φ :: ‹o ⇒ o›
  assume cond: ‹AOT-subst-cond ψ χ›
  assume ‹AOT-subst φ›
  moreover AOT-have ‹⊢□ ψ ≡ χ›
    using cond unfolding AOT-subst-cond-o-def by blast
  ultimately AOT-show ‹⊢□ φ{ψ} ≡ φ{χ}›
  proof (induct arbitrary: ψ χ)
    case AOT-subst-o-id
    thus ?case
      using ≡E(2) oth−class−taut:4:b rule−sub−lem:1:a by blast
  next
    case (AOT-subst-o-const ψ)
    thus ?case
      by (simp add: oth−class−taut:3:a)
  next
    case (AOT-subst-o-not Θ)
    thus ?case
      by (simp add: RN rule−sub−lem:1:a)
  next
    case (AOT-subst-o-imp Θ Ξ)
    thus ?case
      by (meson RN ≡E(5) rule−sub−lem:1:b rule−sub−lem:1:c)
  next
    case (AOT-subst-o-lambda0 Θ)
    thus ?case
      by (simp add: RN rule−sub−lem:1:e)
  next
    case (AOT-subst-o-act Θ)
    thus ?case
      by (simp add: RN rule−sub−lem:1:f)
  next
    case (AOT-subst-o-box Θ)
    thus ?case
      by (simp add: RN rule−sub−lem:1:g)
  next
    case (AOT-subst-o-by-def Θ Ξ)
    AOT-modally-strict {
      AOT-have ‹Ξ{ψ} ≡ Ξ{χ}›
        using AOT-subst-o-by-def by simp
      AOT-thus ‹Θ{ψ} ≡ Θ{χ}›
        using ≡Df[OF AOT-subst-o-by-def(1), of - ψ]
              ≡Df[OF AOT-subst-o-by-def(1), of - χ]
        by (metis ≡E(6) oth−class−taut:3:a)
    }
  qed
```

```
qed
end

instantiation fun :: (AOT-Term-id-2, AOT-subst) AOT-subst
begin

definition AOT-subst-cond-fun :: ‹('a ⇒ 'b) ⇒ ('a ⇒ 'b) ⇒ bool› where
  ‹AOT-subst-cond-fun ≡ λ φ ψ . ∀ α . AOT-subst-cond (φ (AOT-term-of-var α))
                                                (ψ (AOT-term-of-var α))›

inductive AOT-subst-fun :: ‹(('a ⇒ 'b) ⇒ o) ⇒ bool› where
  AOT-subst-fun-const[AOT-substI]:
    ‹AOT-subst-fun (λφ. ψ)›
  | AOT-subst-fun-id[AOT-substI]:
    ‹AOT-subst Ψ ⟹ AOT-subst-fun (λφ. Ψ (φ (AOT-term-of-var α)))›
  | AOT-subst-fun-all[AOT-substI]:
    ‹AOT-subst Ψ ⟹ (⋀ α . AOT-subst-fun (Θ (AOT-term-of-var α))) ⟹
     AOT-subst-fun (λφ :: 'a ⇒ 'b. Ψ «∀ α «Θ (α::'a) φ»»)›
  | AOT-subst-fun-not[AOT-substI]:
    ‹AOT-subst Ψ ⟹ AOT-subst-fun (λφ. «¬«Ψ φ»»)›
  | AOT-subst-fun-imp[AOT-substI]:
    ‹AOT-subst Ψ ⟹ AOT-subst Θ ⟹ AOT-subst-fun (λφ. ««Ψ φ» → «Θ φ»»)›
  | AOT-subst-fun-lambda0[AOT-substI]:
    ‹AOT-subst Θ ⟹ AOT-subst-fun (λ φ. (AOT-lambda0 (Θ φ)))›
  | AOT-subst-fun-act[AOT-substI]:
    ‹AOT-subst Θ ⟹ AOT-subst-fun (λ φ. «𝓐«Θ φ»»)›
  | AOT-subst-fun-box[AOT-substI]:
    ‹AOT-subst Θ ⟹ AOT-subst-fun (λ φ. «□«Θ φ»»)›
  | AOT-subst-fun-def[AOT-substI]:
    ‹(⋀ φ . AOT-model-equiv-def (Θ φ) (Ψ φ)) ⟹
     AOT-subst-fun Ψ ⟹ AOT-subst-fun Θ›

instance proof
  fix ψ χ :: ‹'a ⇒ 'b› and φ :: ‹('a ⇒ 'b) ⇒ o›
  assume ‹AOT-subst φ›
  moreover assume cond: ‹AOT-subst-cond ψ χ›
  ultimately AOT-show ‹⊢□ «φ ψ» ≡ «φ χ»›
  proof(induct)
    case (AOT-subst-fun-const ψ)
    then show ?case by (simp add: oth−class−taut:3:a)
  next
  case (AOT-subst-fun-id Ψ x)
  then show ?case by (simp add: AOT-subst AOT-subst-cond-fun-def)
  next
  next
  case (AOT-subst-fun-all Ψ Θ)
  AOT-have ‹⊢□ □(Θ{α, «ψ»} ≡ Θ{α, «χ»})› for α
    using AOT-subst-fun-all.hyps(3) AOT-subst-fun-all.prems RN by presburger
  thus ?case using AOT-subst[OF AOT-subst-fun-all(1)]
    by (simp add: RN rule−sub−lem:1:d
             AOT-subst-cond-fun-def AOT-subst-cond-o-def)
  next
  case (AOT-subst-fun-not Ψ)
  then show ?case by (simp add: RN rule−sub−lem:1:a)
  next
  case (AOT-subst-fun-imp Ψ Θ)
  then show ?case
    unfolding AOT-subst-cond-fun-def AOT-subst-cond-o-def
    by (meson ≡E(5) oth−class−taut:4:c oth−class−taut:4:d →E)
  next
  case (AOT-subst-fun-lambda0 Θ)
  then show ?case by (simp add: RN rule−sub−lem:1:e)
  next
```

```
    case (AOT-subst-fun-act Θ)
    then show ?case by (simp add: RN rule−sub−lem:1:f)
    next
    case (AOT-subst-fun-box Θ)
    then show ?case by (simp add: RN rule−sub−lem:1:g)
    next
    case (AOT-subst-fun-def Θ Ψ)
    then show ?case
      by (meson df−rules−formulas[3] df−rules−formulas[4] ≡I ≡E(5))
    qed
  qed
  end

ML‹
fun prove-AOT-subst-tac ctxt = REPEAT (SUBGOAL (fn (trm,-) => let
  fun findHeadConst (Const x) = SOME x
    | findHeadConst (A $ -) = findHeadConst A
    | findHeadConst - = NONE
  fun findDef (Const (const-name‹AOT-model-equiv-def›, -) $ lhs $ -)
      = findHeadConst lhs
    | findDef (A $ B) = (case findDef A of SOME x => SOME x | - => findDef B)
    | findDef (Abs (-,-,c)) = findDef c
    | findDef - = NONE
  val const-opt = (findDef trm)
  val defs = case const-opt of SOME const => List.filter (fn thm => let
      val concl = Thm.concl-of thm
      val thmconst = (findDef concl)
      in case thmconst of SOME (c,-) => fst const = c | - => false end)
      (AOT-Definitions.get ctxt)
      | - => []
  val tac = case defs of
            [] => safe-step-tac (ctxt addSIs @{thms AOT-substI}) 1
            | - => resolve-tac ctxt defs 1
  in tac end) 1)
fun getSubstThm ctxt reversed phi p q = let
val p-ty = Term.type-of p
val abs = HOLogic.mk-Trueprop (@{const AOT-subst(-)} $ phi)
val abs = Syntax.check-term ctxt abs
val substThm = Goal.prove ctxt [] [] abs
  (fn {context=ctxt, prems=-} => prove-AOT-subst-tac ctxt)
val substThm = substThm RS @{thm AOT-subst}
in if reversed then let
  val substThm = Drule.instantiate-normalize
         (TVars.empty, Vars.make [(((χ, 0), p-ty), Thm.cterm-of ctxt p),
         (((ψ, 0), p-ty), Thm.cterm-of ctxt q)]) substThm
  val substThm = substThm RS @{thm ≡E(1)}
  in substThm end
else
  let
  val substThm = Drule.instantiate-normalize
         (TVars.empty, Vars.make [(((ψ, 0), p-ty), Thm.cterm-of ctxt p),
         (((χ, 0), p-ty), Thm.cterm-of ctxt q)]) substThm
  val substThm = substThm RS @{thm ≡E(2)}
  in substThm end end
›

method-setup AOT-subst = ‹
Scan.option (Scan.lift (Args.parens (Args.$$$ reverse))) −−
Scan.lift (Parse.embedded-inner-syntax −− Parse.embedded-inner-syntax) −−
Scan.option (Scan.lift (Args.$$$ for −− Args.colon) |−−
Scan.repeat1 (Scan.lift (Parse.embedded-inner-syntax) −−
Scan.option (Scan.lift (Args.$$$ :: |−− Parse.embedded-inner-syntax))))
>> (fn ((reversed,(raw-p,raw-q)),raw-bounds) => (fn ctxt =>
```

```
(Method.SIMPLE-METHOD (Subgoal.FOCUS (fn {context = ctxt, params = -,
  prems = prems, asms = asms, concl = concl, schematics = -} =>
let
val thms = prems
val ctxt' = ctxt
val ctxt = Context-Position.set-visible false ctxt
val raw-bounds = case raw-bounds of SOME bounds => bounds | - => []

val ctxt = (fold (fn (bound, ty) => fn ctxt =>
  let
    val bound = AOT-read-term @{nonterminal τ'} ctxt bound
    val ty = Option.map (Syntax.read-typ ctxt) ty
    val ctxt = case ty of SOME ty => let
        val bound = Const (-type-constraint-, Type (fun, [ty,ty])) $ bound
        val bound = Syntax.check-term ctxt bound
      in Variable.declare-term bound ctxt end | - => ctxt
  in ctxt end)) raw-bounds ctxt

val p = AOT-read-term @{nonterminal φ'} ctxt raw-p
val p = Syntax.check-term ctxt p
val ctxt = Variable.declare-term p ctxt
val q = AOT-read-term @{nonterminal φ'} ctxt raw-q
val q = Syntax.check-term ctxt q
val ctxt = Variable.declare-term q ctxt

val bounds = (map (fn (bound, -) =>
  Syntax.check-term ctxt (AOT-read-term @{nonterminal τ'} ctxt bound)
)) raw-bounds
val p = fold (fn bound => fn p =>
  Term.abs (α, Term.type-of bound) (Term.abstract-over (bound,p)))
  bounds p
val p = Syntax.check-term ctxt p
val p-ty = Term.type-of p

val pat = @{const Trueprop} $
  (@{const AOT-model-valid-in} $ Var ((w,0), @{typ w}) $
   (Var ((φ,0), Type (type-name ‹fun›, [p-ty, @{typ o}])) $ p))
val univ = Unify.matchers (Context.Proof ctxt) [(pat, Thm.term-of concl)]
val univ = hd (Seq.list-of univ) (* TODO: consider all matches *)
val phi = the (Envir.lookup univ
  ((φ,0), Type (type-name ‹fun›, [p-ty, @{typ o}])))

val q = fold (fn bound => fn q =>
  Term.abs (α, Term.type-of bound) (Term.abstract-over (bound,q))) bounds q
val q = Syntax.check-term ctxt q

(* Reparse to report bounds as fixes. *)
val ctxt = Context-Position.restore-visible ctxt' ctxt
val ctxt' = ctxt
fun unsource str = fst (Input.source-content (Syntax.read-input str))
val (-,ctxt') = Proof-Context.add-fixes (map (fn (str,-) =>
  (Binding.make (unsource str, Position.none), NONE, Mixfix.NoSyn)) raw-bounds)
  ctxt'
val - = (map (fn (x,-) =>
  Syntax.check-term ctxt (AOT-read-term @{nonterminal τ'} ctxt' x)))
  raw-bounds
val - = AOT-read-term @{nonterminal φ'} ctxt' raw-p
val - = AOT-read-term @{nonterminal φ'} ctxt' raw-q
val reversed = case reversed of SOME - => true | - => false
val simpThms = [@{thm AOT-subst-cond-o-def}, @{thm AOT-subst-cond-fun-def}]
in
resolve-tac ctxt [getSubstThm ctxt reversed phi p q] 1
THEN simp-tac (ctxt |> Simplifier.add-simps simpThms) 1
```

```
THEN (REPEAT (resolve-tac ctxt [@{thm allI}] 1))
THEN (TRY (resolve-tac ctxt thms 1))
end
) ctxt 1))))
›

method-setup AOT-subst-def = ‹
Scan.option (Scan.lift (Args.parens (Args.$$$ reverse))) --
Attrib.thm
>> (fn (reversed,fact) => (fn ctxt =>
(Method.SIMPLE-METHOD (Subgoal.FOCUS (fn {context = ctxt, params = -,
  prems = prems, asms = asms, concl = concl, schematics = -} =>
let
val c = Thm.concl-of fact
val (lhs, rhs) = case c of (const‹Trueprop› $
    (const‹AOT-model-equiv-def› $ lhs $ rhs)) => (lhs, rhs)
  | - => raise Fail Definition expected.
val substCond = HOLogic.mk-Trueprop
  (Const (const-name‹AOT-subst-cond›, dummyT) $ lhs $ rhs)
val substCond = Syntax.check-term
  (Proof-Context.set-mode Proof-Context.mode-schematic ctxt)
  substCond
val simpThms = [@{thm AOT-subst-cond-o-def},
  @{thm AOT-subst-cond-fun-def},
  fact RS @{thm ≡Df}]
val substCondThm = Goal.prove ctxt [] [] substCond
  (fn {context=ctxt, prems=prems} =>
      (SUBGOAL (fn (trm,int) =>
        auto-tac (ctxt |> Simplifier.add-simps simpThms)) 1))
val substThm = substCondThm RSN (2,@{thm AOT-subst})
in
resolve-tac ctxt [substThm RS
  (case reversed of NONE => @{thm ≡E(2)} | - => @{thm ≡E(1)})] 1
THEN prove-AOT-subst-tac ctxt
THEN (TRY (resolve-tac ctxt prems 1))
end
) ctxt 1))))
›

method-setup AOT-subst-thm = ‹
Scan.option (Scan.lift (Args.parens (Args.$$$ reverse))) --
Attrib.thm
>> (fn (reversed,fact) => (fn ctxt =>
(Method.SIMPLE-METHOD (Subgoal.FOCUS (fn {context = ctxt, params = -,
  prems = prems, asms = asms, concl = concl, schematics = -} =>
let
val c = Thm.concl-of fact
val (lhs, rhs) = case c of
  (const‹Trueprop› $
   (const‹AOT-model-valid-in› $ - $
    (const‹AOT-equiv› $ lhs $ rhs))) => (lhs, rhs)
  | - => raise Fail Equivalence expected.

val substCond = HOLogic.mk-Trueprop
  (Const (const-name‹AOT-subst-cond›, dummyT) $ lhs $ rhs)
val substCond = Syntax.check-term
  (Proof-Context.set-mode Proof-Context.mode-schematic ctxt)
  substCond
val simpThms = [@{thm AOT-subst-cond-o-def},
  @{thm AOT-subst-cond-fun-def},
  fact]
val substCondThm = Goal.prove ctxt [] [] substCond
  (fn {context=ctxt, prems=prems} =>
```

(*SUBGOAL* (*fn* (*trm*,*int*) => *auto-tac* (*ctxt* |> *Simplifier.add-simps simpThms*)) *1*))
val substThm = *substCondThm RSN* (*2*,@{*thm AOT-subst*})
in
resolve-tac ctxt [*substThm RS*
 (*case reversed of NONE* => @{*thm* ≡*E*(*2*)} | - => @{*thm* ≡*E*(*1*)})] *1*
THEN prove-AOT-subst-tac ctxt
THEN (*TRY* (*resolve-tac ctxt prems 1*))
end
) *ctxt 1*))))
›

AOT-theorem *rule−sub−remark*:*1*[*1*]:
 assumes ‹$\vdash_\Box$ $A!x \equiv \neg\Diamond E!x$› **and** ‹$\neg A!x$›
 shows ‹$\neg\neg\Diamond E!x$›
 by (*AOT-subst* (*reverse*) ‹$\neg\Diamond E!x$› ‹$A!x$›)
  (*auto simp*: *assms*)

AOT-theorem *rule−sub−remark*:*1*[*2*]:
 assumes ‹$\vdash_\Box$ $A!x \equiv \neg\Diamond E!x$› **and** ‹$\neg\neg\Diamond E!x$›
 shows ‹$\neg A!x$›
 by (*AOT-subst* ‹$A!x$› ‹$\neg\Diamond E!x$›)
  (*auto simp*: *assms*)

AOT-theorem *rule−sub−remark*:*2*[*1*]:
 assumes ‹$\vdash_\Box$ $[R]xy \equiv ([R]xy\ \&\ ([Q]a \vee \neg[Q]a))$›
  and ‹$p \rightarrow [R]xy$›
 shows ‹$p \rightarrow [R]xy\ \&\ ([Q]a \vee \neg[Q]a)$›
 by (*AOT-subst-thm* (*reverse*) *assms*(*1*)) (*simp add*: *assms*(*2*))

AOT-theorem *rule−sub−remark*:*2*[*2*]:
 assumes ‹$\vdash_\Box$ $[R]xy \equiv ([R]xy\ \&\ ([Q]a \vee \neg[Q]a))$›
  and ‹$p \rightarrow [R]xy\ \&\ ([Q]a \vee \neg[Q]a)$›
 shows ‹$p \rightarrow [R]xy$›
 by (*AOT-subst-thm assms*(*1*)) (*simp add*: *assms*(*2*))

AOT-theorem *rule−sub−remark*:*3*[*1*]:
 assumes ‹*for arbitrary x*: $\vdash_\Box$ $A!x \equiv \neg\Diamond E!x$›
  and ‹$\exists\, x\ A!x$›
  shows ‹$\exists\, x\ \neg\Diamond E!x$›
 by (*AOT-subst* (*reverse*) ‹$\neg\Diamond E!x$› ‹$A!x$› **for**: *x*)
  (*auto simp*: *assms*)

AOT-theorem *rule−sub−remark*:*3*[*2*]:
 assumes ‹*for arbitrary x*: $\vdash_\Box$ $A!x \equiv \neg\Diamond E!x$›
  and ‹$\exists\, x\ \neg\Diamond E!x$›
  shows ‹$\exists\, x\ A!x$›
 by (*AOT-subst* ‹$A!x$› ‹$\neg\Diamond E!x$› **for**: *x*)
  (*auto simp*: *assms*)

AOT-theorem *rule−sub−remark*:*4*[*1*]:
 assumes ‹$\vdash_\Box$ $\neg\neg[P]x \equiv [P]x$› **and** ‹$\boldsymbol{\mathcal{A}}\neg\neg[P]x$›
 shows ‹$\boldsymbol{\mathcal{A}}[P]x$›
 by (*AOT-subst-thm* (*reverse*) *assms*(*1*)) (*simp add*: *assms*(*2*))

AOT-theorem *rule−sub−remark*:*4*[*2*]:
 assumes ‹$\vdash_\Box$ $\neg\neg[P]x \equiv [P]x$› **and** ‹$\boldsymbol{\mathcal{A}}[P]x$›
 shows ‹$\boldsymbol{\mathcal{A}}\neg\neg[P]x$›
 by (*AOT-subst-thm assms*(*1*)) (*simp add*: *assms*(*2*))

AOT-theorem *rule−sub−remark*:*5*[*1*]:
 assumes ‹$\vdash_\Box$ $(\varphi \rightarrow \psi) \equiv (\neg\psi \rightarrow \neg\varphi)$› **and** ‹$\Box(\varphi \rightarrow \psi)$›
 shows ‹$\Box(\neg\psi \rightarrow \neg\varphi)$›
 by (*AOT-subst-thm* (*reverse*) *assms*(*1*)) (*simp add*: *assms*(*2*))

AOT-theorem *rule−sub−remark*:*5*[*2*]:
assumes ‹$\vdash_\Box$ (φ → ψ) ≡ (¬ψ → ¬φ)› **and** ‹□(¬ψ → ¬φ)›
shows ‹□(φ → ψ)›
by (*AOT-subst-thm assms*(*1*)) (*simp add*: *assms*(*2*))

AOT-theorem *rule−sub−remark*:*6*[*1*]:
assumes ‹$\vdash_\Box$ ψ ≡ χ› **and** ‹□(φ → ψ)›
shows ‹□(φ → χ)›
by (*AOT-subst-thm* (*reverse*) *assms*(*1*)) (*simp add*: *assms*(*2*))

AOT-theorem *rule−sub−remark*:*6*[*2*]:
assumes ‹$\vdash_\Box$ ψ ≡ χ› **and** ‹□(φ → χ)›
shows ‹□(φ → ψ)›
by (*AOT-subst-thm assms*(*1*)) (*simp add*: *assms*(*2*))

AOT-theorem *rule−sub−remark*:*7*[*1*]:
assumes ‹$\vdash_\Box$ φ ≡ ¬¬φ› **and** ‹□(φ → φ)›
shows ‹□(¬¬φ → φ)›
by (*AOT-subst-thm* (*reverse*) *assms*(*1*)) (*simp add*: *assms*(*2*))

AOT-theorem *rule−sub−remark*:*7*[*2*]:
assumes ‹$\vdash_\Box$ φ ≡ ¬¬φ› **and** ‹□(¬¬φ → φ)›
shows ‹□(φ → φ)›
by (*AOT-subst-thm assms*(*1*)) (*simp add*: *assms*(*2*))

AOT-theorem *KBasic2*:*1*: ‹□¬φ ≡ ¬◇φ›
by (*meson conventions*:*5 contraposition*:*2*
Hypothetical Syllogism df−rules−formulas[*3*]
df−rules−formulas[*4*] ≡*I useful−tautologies*:*1*)

AOT-theorem *KBasic2*:*2*: ‹◇(φ ∨ ψ) ≡ (◇φ ∨ ◇ψ)›
proof −
AOT-have ‹◇(φ ∨ ψ) ≡ ◇¬(¬φ & ¬ψ)›
by (*simp add*: *RE*◇ *oth−class−taut*:*5*:*b*)
also AOT-have ‹. . . ≡ ¬□(¬φ & ¬ψ)›
using *KBasic*:*11* ≡*E*(*6*) *oth−class−taut*:*3*:*a* **by** *blast*
also AOT-have ‹. . . ≡ ¬(□¬φ & □¬ψ)›
using *KBasic*:*3* ≡*E*(*1*) *oth−class−taut*:*4*:*b* **by** *blast*
also AOT-have ‹. . . ≡ ¬(¬◇φ & ¬◇ψ)›
using *KBasic2*:*1*
by (*AOT-subst* ‹□¬φ› ‹¬◇φ›; *AOT-subst* ‹□¬ψ› ‹¬◇ψ›;
auto simp: *oth−class−taut*:*3*:*a*)
also AOT-have ‹. . . ≡ ¬¬(◇φ ∨ ◇ψ)›
using ≡*E*(*6*) *oth−class−taut*:*3*:*b oth−class−taut*:*5*:*b* **by** *blast*
also AOT-have ‹. . . ≡ ◇φ ∨ ◇ψ›
by (*simp add*: ≡*I useful−tautologies*:*1 useful−tautologies*:*2*)
finally show *?thesis* **.**
qed

AOT-theorem *KBasic2*:*3*: ‹◇(φ & ψ) → (◇φ & ◇ψ)›
by (*metis RM*◇ &*I Conjunction Simplification*(*1*,*2*)
→*I modus−tollens*:*1 reductio−aa*:*1*)

AOT-theorem *KBasic2*:*4*: ‹◇(φ → ψ) ≡ (□φ → ◇ψ)›
proof −
AOT-have ‹◇(φ → ψ) ≡ ◇(¬φ ∨ ψ)›
by (*AOT-subst* ‹φ → ψ› ‹¬φ ∨ ψ›)
(*auto simp*: *oth−class−taut*:*1*:*c oth−class−taut*:*3*:*a*)
also AOT-have ‹... ≡ ◇¬φ ∨ ◇ψ›
by (*simp add*: *KBasic2*:*2*)
also AOT-have ‹... ≡ ¬□φ ∨ ◇ψ›
by (*AOT-subst* ‹¬□φ› ‹◇¬φ›)

(*auto simp*: *KBasic*:*11* *oth−class−taut*:*3*:*a*)
also AOT-have ‹... ≡ □φ → ◊ψ›
using ≡*E*(*6*) *oth−class−taut*:*1*:*c* *oth−class−taut*:*3*:*a* **by** *blast*
finally show *?thesis* **.**
qed

AOT-theorem *KBasic2*:*5*: ‹◊◊φ ≡ ¬□□¬φ›
using *conventions*:*5*[*THEN* ≡*Df*]
by (*AOT-subst* ‹◊φ› ‹¬□¬φ›;
AOT-subst ‹◊¬□¬φ› ‹¬□¬¬□¬φ›;
AOT-subst (*reverse*) ‹¬¬□¬φ› ‹□¬φ›)
(*auto simp*: *oth−class−taut*:*3*:*b* *oth−class−taut*:*3*:*a*)

AOT-theorem *KBasic2*:*6*: ‹□(φ ∨ ψ) → (□φ ∨ ◊ψ)›
proof(*rule* →*I*; *rule raa−cor*:*1*)
AOT-assume ‹□(φ ∨ ψ)›
AOT-hence ‹□(¬φ → ψ)›
using *conventions*:*2*[*THEN* ≡*Df*]
by (*AOT-subst* (*reverse*) ‹¬φ → ψ› ‹φ ∨ ψ›) *simp*
AOT-hence *1*: ‹◊¬φ → ◊ψ›
using *KBasic*:*13* *vdash−properties*:*10* **by** *blast*
AOT-assume ‹¬(□φ ∨ ◊ψ)›
AOT-hence ‹¬□φ› **and** ‹¬◊ψ›
using &*E* ≡*E*(*1*) *oth−class−taut*:*5*:*d* **by** *blast*+
AOT-thus ‹◊ψ & ¬◊ψ›
using &*I*(*1*) *1*[*THEN* →*E*] *KBasic*:*11* ≡*E*(*4*) *raa−cor*:*3* **by** *blast*
qed

AOT-theorem *KBasic2*:*7*: ‹(□(φ ∨ ψ) & ◊¬φ) → ◊ψ›
proof(*rule* →*I*; *frule* &*E*(*1*); *drule* &*E*(*2*))
AOT-assume ‹□(φ ∨ ψ)›
AOT-hence *1*: ‹□φ ∨ ◊ψ›
using *KBasic2*:*6* ∨*I*(*2*) ∨*E*(*1*) **by** *blast*
AOT-assume ‹◊¬φ›
AOT-hence ‹¬□φ› **using** *KBasic*:*11* ≡*E*(*2*) **by** *blast*
AOT-thus ‹◊ψ› **using** *1* ∨*E*(*2*) **by** *blast*
qed

AOT-theorem *T−S5−fund*:*1*: ‹φ → ◊φ›
by (*meson* $\equiv_{df}$*I* *conventions*:*5* *contraposition*:*2*
Hypothetical Syllogism →*I* *qml*:*2*[*axiom-inst*])
lemmas *T*◊ = *T−S5−fund*:*1*

AOT-theorem *T−S5−fund*:*2*: ‹◊□φ → □φ›
proof(*rule* →*I*)
AOT-assume ‹◊□φ›
AOT-hence ‹¬□◊¬φ›
using *KBasic*:*14* ≡*E*(*4*) *raa−cor*:*3* **by** *blast*
moreover AOT-have ‹◊¬φ → □◊¬φ›
by (*fact qml*:*3*[*axiom-inst*])
ultimately AOT-have ‹¬◊¬φ›
using *modus−tollens*:*1* **by** *blast*
AOT-thus ‹□φ› **using** *KBasic*:*12* ≡*E*(*2*) **by** *blast*
qed
lemmas *5*◊ = *T−S5−fund*:*2*

AOT-theorem *Act−Sub*:*1*: ‹𝓐φ ≡ ¬𝓐¬φ›
by (*AOT-subst* ‹𝓐¬φ› ‹¬𝓐φ›)
(*auto simp*: *logic−actual−nec*:*1*[*axiom-inst*] *oth−class−taut*:*3*:*b*)

AOT-theorem *Act−Sub*:*2*: ‹◊φ ≡ 𝓐◊φ›
using *conventions*:*5*[*THEN* ≡*Df*]

by (*AOT-subst* ‹$\Diamond\varphi$› ‹$\neg\Box\neg\varphi$›)
(*metis deduction−theorem* $\equiv I$ $\equiv E(1)$ $\equiv E(2)$ $\equiv E(3)$
logic−actual−nec:*1*[*axiom-inst*] *qml−act*:*2*[*axiom-inst*])

AOT-theorem *Act−Sub*:*3*: ‹$\boldsymbol{\mathcal{A}}\varphi \rightarrow \Diamond\varphi$›
using *conventions*:*5*[*THEN* $\equiv Df$]
by (*AOT-subst* ‹$\Diamond\varphi$› ‹$\neg\Box\neg\varphi$›)
(*metis Act−Sub*:*1* $\rightarrow I$ $\equiv E(4)$ *nec−imp−act reductio−aa*:*2* $\rightarrow E$)

AOT-theorem *Act−Sub*:*4*: ‹$\boldsymbol{\mathcal{A}}\varphi \equiv \Diamond\boldsymbol{\mathcal{A}}\varphi$›
proof (*rule* $\equiv I$; *rule* $\rightarrow I$)
AOT-assume ‹$\boldsymbol{\mathcal{A}}\varphi$›
AOT-thus ‹$\Diamond\boldsymbol{\mathcal{A}}\varphi$› **using** *T*$\Diamond$ *vdash−properties*:*10* **by** *blast*
next
AOT-assume ‹$\Diamond\boldsymbol{\mathcal{A}}\varphi$›
AOT-hence ‹$\neg\Box\neg\boldsymbol{\mathcal{A}}\varphi$›
using $\equiv_{df}E$ *conventions*:*5* **by** *blast*
AOT-hence ‹$\neg\Box\boldsymbol{\mathcal{A}}\neg\varphi$›
by (*AOT-subst* ‹$\boldsymbol{\mathcal{A}}\neg\varphi$› ‹$\neg\boldsymbol{\mathcal{A}}\varphi$›)
(*simp add*: *logic−actual−nec*:*1*[*axiom-inst*])
AOT-thus ‹$\boldsymbol{\mathcal{A}}\varphi$›
using *Act−Basic*:*1* *Act−Basic*:*6* $\vee E(3)$ $\equiv E(4)$
reductio−aa:*1* **by** *blast*
qed

AOT-theorem *Act−Sub*:*5*: ‹$\Diamond\boldsymbol{\mathcal{A}}\varphi \rightarrow \boldsymbol{\mathcal{A}}\Diamond\varphi$›
by (*metis Act−Sub*:*2* *Act−Sub*:*3* *Act−Sub*:*4* $\rightarrow I$ $\equiv E(1)$ $\equiv E(2)$ $\rightarrow E$)

AOT-theorem *S5Basic*:*1*: ‹$\Diamond\varphi \equiv \Box\Diamond\varphi$›
by (*simp add*: $\equiv I$ *qml*:*2*[*axiom-inst*] *qml*:*3*[*axiom-inst*])

AOT-theorem *S5Basic*:*2*: ‹$\Box\varphi \equiv \Diamond\Box\varphi$›
by (*simp add*: *T*$\Diamond$ *5*$\Diamond$ $\equiv I$)

AOT-theorem *S5Basic*:*3*: ‹$\varphi \rightarrow \Box\Diamond\varphi$›
using *T*$\Diamond$ *Hypothetical Syllogism qml*:*3*[*axiom-inst*] **by** *blast*
lemmas *B* = *S5Basic*:*3*

AOT-theorem *S5Basic*:*4*: ‹$\Diamond\Box\varphi \rightarrow \varphi$›
using *5*$\Diamond$ *Hypothetical Syllogism qml*:*2*[*axiom-inst*] **by** *blast*
lemmas *B*$\Diamond$ = *S5Basic*:*4*

AOT-theorem *S5Basic*:*5*: ‹$\Box\varphi \rightarrow \Box\Box\varphi$›
using *RM*:*1* *B* *5*$\Diamond$ *Hypothetical Syllogism* **by** *blast*
lemmas *4* = *S5Basic*:*5*

AOT-theorem *S5Basic*:*6*: ‹$\Box\varphi \equiv \Box\Box\varphi$›
by (*simp add*: *4* $\equiv I$ *qml*:*2*[*axiom-inst*])

AOT-theorem *S5Basic*:*7*: ‹$\Diamond\Diamond\varphi \rightarrow \Diamond\varphi$›
using *conventions*:*5*[*THEN* $\equiv Df$] *oth−class−taut*:*3*:*b*
by (*AOT-subst* ‹$\Diamond\Diamond\varphi$› ‹$\neg\Box\neg\Diamond\varphi$›;
AOT-subst ‹$\Diamond\varphi$› ‹$\neg\Box\neg\varphi$›;
AOT-subst (*reverse*) ‹$\neg\neg\Box\neg\varphi$› ‹$\Box\neg\varphi$›;
AOT-subst (*reverse*) ‹$\Box\Box\neg\varphi$› ‹$\Box\neg\varphi$›)
(*auto simp*: *S5Basic*:*6* *if−p−then−p*)

lemmas *4*$\Diamond$ = *S5Basic*:*7*

AOT-theorem *S5Basic*:*8*: ‹$\Diamond\Diamond\varphi \equiv \Diamond\varphi$›
by (*simp add*: *4*$\Diamond$ *T*$\Diamond$ $\equiv I$)

AOT-theorem *S5Basic*:*9*: ‹$\Box(\varphi \vee \Box\psi) \equiv (\Box\varphi \vee \Box\psi)$›

apply (*rule* ≡*I*; *rule* →*I*)
using *KBasic2*:*6* *5*◇ ∨*I*(*3*) *if−p−then−p vdash−properties*:*10*
apply *blast*
by (*meson KBasic*:*15* *4* ∨*I*(*3*) ∨*E*(*1*) *Disjunction Addition*(*1*)
con−dis−taut:*7* *intro−elim*:*1* *Commutativity of* ∨)

AOT-theorem *S5Basic*:*10*: ‹□(φ ∨ ◇ψ) ≡ (□φ ∨ ◇ψ)›
proof(*rule* ≡*I*; *rule* →*I*)
AOT-assume ‹□(φ ∨ ◇ψ)›
AOT-hence ‹□φ ∨ ◇◇ψ›
by (*meson KBasic2*:*6* ∨*I*(*2*) ∨*E*(*1*))
AOT-thus ‹□φ ∨ ◇ψ›
by (*meson B*◇ *4* *4*◇ *T*◇ ∨*I*(*3*))
next
AOT-assume ‹□φ ∨ ◇ψ›
AOT-hence ‹□φ ∨ □◇ψ›
by (*meson S5Basic*:*1* *B*◇ *S5Basic*:*6* *T*◇ *5*◇ ∨*I*(*3*) *intro−elim*:*1*)
AOT-thus ‹□(φ ∨ ◇ψ)›
by (*meson KBasic*:*15* ∨*I*(*3*) ∨*E*(*1*) *Disjunction Addition*(*1*,*2*))
qed

AOT-theorem *S5Basic*:*11*: ‹◇(φ & ◇ψ) ≡ (◇φ & ◇ψ)›
proof −
AOT-have ‹◇(φ & ◇ψ) ≡ ◇¬(¬φ ∨ ¬◇ψ)›
by (*AOT-subst* ‹φ & ◇ψ› ‹¬(¬φ ∨ ¬◇ψ)›)
(*auto simp*: *oth−class−taut*:*5*:*a* *oth−class−taut*:*3*:*a*)
also AOT-have ‹. . . ≡ ◇¬(¬φ ∨ □¬ψ)›
by (*AOT-subst* ‹□¬ψ› ‹¬◇ψ›)
(*auto simp*: *KBasic2*:*1* *oth−class−taut*:*3*:*a*)
also AOT-have ‹. . . ≡ ¬□(¬φ ∨ □¬ψ)›
using *KBasic*:*11* ≡*E*(*6*) *oth−class−taut*:*3*:*a* **by** *blast*
also AOT-have ‹. . . ≡ ¬(□¬φ ∨ □¬ψ)›
using *S5Basic*:*9* ≡*E*(*1*) *oth−class−taut*:*4*:*b* **by** *blast*
also AOT-have ‹. . . ≡ ¬(¬◇φ ∨ ¬◇ψ)›
using *KBasic2*:*1*
by (*AOT-subst* ‹□¬φ› ‹¬◇φ›; *AOT-subst* ‹□¬ψ› ‹¬◇ψ›)
(*auto simp*: *oth−class−taut*:*3*:*a*)
also AOT-have ‹. . . ≡ ◇φ & ◇ψ›
using ≡*E*(*6*) *oth−class−taut*:*3*:*a* *oth−class−taut*:*5*:*a* **by** *blast*
finally show *?thesis* **.**
qed

AOT-theorem *S5Basic*:*12*: ‹◇(φ & □ψ) ≡ (◇φ & □ψ)›
proof (*rule* ≡*I*; *rule* →*I*)
AOT-assume ‹◇(φ & □ψ)›
AOT-hence ‹◇φ & ◇□ψ›
using *KBasic2*:*3* *vdash−properties*:*6* **by** *blast*
AOT-thus ‹◇φ & □ψ›
using *5*◇ &*I* &*E*(*1*) &*E*(*2*) *vdash−properties*:*6* **by** *blast*
next
AOT-assume ‹◇φ & □ψ›
moreover AOT-have ‹(□□ψ & ◇φ) → ◇(φ & □ψ)›
by (*AOT-subst* ‹φ & □ψ› ‹□ψ & φ›)
(*auto simp*: *Commutativity of* & *KBasic*:*16*)
ultimately AOT-show ‹◇(φ & □ψ)›
by (*metis* *4* &*I* *Conjunction Simplification*(*1*,*2*) →*E*)
qed

AOT-theorem *S5Basic*:*13*: ‹□(φ → □ψ) ≡ □(◇φ → ψ)›
proof (*rule* ≡*I*)
AOT-modally-strict {
AOT-have ‹□(φ → □ψ) → (◇φ → ψ)›
by (*meson KBasic*:*13* *B*◇ *Hypothetical Syllogism* →*I*)

```
  }
  AOT-hence ‹□□(φ → □ψ) → □(◇φ → ψ)›
    by (rule RM)
  AOT-thus ‹□(φ → □ψ) → □(◇φ → ψ)›
    using 4 Hypothetical Syllogism by blast
next
  AOT-modally-strict {
    AOT-have ‹□(◇φ → ψ) → (φ → □ψ)›
      by (meson B Hypothetical Syllogism →I qml:1[axiom-inst])
  }
  AOT-hence ‹□□(◇φ → ψ) → □(φ → □ψ)›
    by (rule RM)
  AOT-thus ‹□(◇φ → ψ) → □(φ → □ψ)›
    using 4 Hypothetical Syllogism by blast
qed

AOT-theorem derived−S5−rules:1:
  assumes ‹Γ ⊢□ ◇φ → ψ›
  shows ‹□Γ ⊢□ φ → □ψ›
proof −
  AOT-have ‹□Γ ⊢□ □◇φ → □ψ›
    using assms by (rule RM:1[prem])
  AOT-thus ‹□Γ ⊢□ φ → □ψ›
    using B Hypothetical Syllogism by blast
qed

AOT-theorem derived−S5−rules:2:
  assumes ‹Γ ⊢□ φ → □ψ›
  shows ‹□Γ ⊢□ ◇φ → ψ›
proof −
  AOT-have ‹□Γ ⊢□ ◇φ → ◇□ψ›
    using assms by (rule RM:2[prem])
  AOT-thus ‹□Γ ⊢□ ◇φ → ψ›
    using B◇ Hypothetical Syllogism by blast
qed

AOT-theorem BFs:1: ‹∀α □φ{α} → □∀α φ{α}›
proof −
  AOT-modally-strict {
    AOT-have ‹◇∀α □φ{α} → ◇□φ{α}› for α
      using cqt−orig:3 by (rule RM◇)
    AOT-hence ‹◇∀α □φ{α} → ∀α φ{α}›
      using B◇ ∀I →E →I by metis
  }
  thus ?thesis
    using derived−S5−rules:1 by blast
qed
lemmas BF = BFs:1

AOT-theorem BFs:2: ‹□∀α φ{α} → ∀α □φ{α}›
proof −
  AOT-have ‹□∀α φ{α} → □φ{α}› for α
    using RM cqt−orig:3 by metis
  thus ?thesis
    using cqt−orig:2[THEN →E] ∀I by metis
qed
lemmas CBF = BFs:2

AOT-theorem BFs:3: ‹◇∃α φ{α} → ∃α ◇φ{α}›
proof(rule →I)
  AOT-modally-strict {
    AOT-have ‹□∀α ¬φ{α} ≡ ∀α □¬φ{α}›
      using BF CBF ≡I by blast
```

```
  } note ϑ = this

  AOT-assume ‹◇∃ α φ{α}›
  AOT-hence ‹¬□¬(∃ α φ{α})›
    using ≡_{df}E conventions:5 by blast
  AOT-hence ‹¬□∀ α ¬φ{α}›
    apply (AOT-subst ‹∀ α ¬φ{α}› ‹¬(∃ α φ{α})›)
    using ≡_{df}I conventions:3 conventions:4 &I
          contraposition:2 cqt-further:4
          df-rules-formulas[3] by blast
  AOT-hence ‹¬∀ α □¬φ{α}›
    apply (AOT-subst (reverse) ‹∀ α □¬φ{α}› ‹□∀ α ¬φ{α}›)
    using ϑ by blast
  AOT-hence ‹¬∀ α ¬¬□¬φ{α}›
    by (AOT-subst (reverse) ‹¬¬□¬φ{α}› ‹□¬φ{α}› for: α)
       (simp add: oth-class-taut:3:b)
  AOT-hence ‹∃ α ¬□¬φ{α}›
    by (rule conventions:4[THEN ≡_{df}I])
  AOT-thus ‹∃ α ◇φ{α}›
    using conventions:5[THEN ≡Df]
    by (AOT-subst ‹◇φ{α}› ‹¬□¬φ{α}› for: α)
qed
lemmas BF◇ = BFs:3

AOT-theorem BFs:4: ‹∃ α ◇φ{α} → ◇∃ α φ{α}›
proof(rule →I)
  AOT-assume ‹∃ α ◇φ{α}›
  AOT-hence ‹¬∀ α ¬◇φ{α}›
    using conventions:4[THEN ≡_{df}E] by blast
  AOT-hence ‹¬∀ α □¬φ{α}›
    using KBasic2:1
    by (AOT-subst ‹□¬φ{α}› ‹¬◇φ{α}› for: α)
  moreover AOT-have ‹∀ α □¬φ{α} ≡ □∀ α ¬φ{α}›
    using ≡I BF CBF by metis
  ultimately AOT-have 1: ‹¬□∀ α ¬φ{α}›
    using ≡E(3) by blast
  AOT-show ‹◇∃ α φ{α}›
    apply (rule conventions:5[THEN ≡_{df}I])
    apply (AOT-subst ‹∃ α φ{α}› ‹¬∀ α ¬φ{α}›)
     apply (simp add: conventions:4 ≡Df)
    apply (AOT-subst ‹¬¬∀ α ¬φ{α}› ‹∀ α ¬φ{α}›)
    by (auto simp: 1 ≡I useful-tautologies:1 useful-tautologies:2)
qed
lemmas CBF◇ = BFs:4

AOT-theorem sign-S5-thm:1: ‹∃ α □φ{α} → □∃ α φ{α}›
proof(rule →I)
  AOT-assume ‹∃ α □φ{α}›
  then AOT-obtain α where ‹□φ{α}› using ∃ E by metis
  moreover AOT-have ‹□α↓›
    by (simp add: ex:1:a rule-ui:2[const-var] RN)
  moreover AOT-have ‹□φ{τ}, □τ↓ ⊢_□ □∃ α φ{α}› for τ
  proof -
    AOT-have ‹φ{τ}, τ↓ ⊢_□ ∃ α φ{α}› using existential:1 by blast
    AOT-thus ‹□φ{τ}, □τ↓ ⊢_□ □∃ α φ{α}›
      using RN[prem][where Γ={φ τ, «τ↓»}, simplified] by blast
  qed
  ultimately AOT-show ‹□∃ α φ{α}› by blast
qed
lemmas Buridan = sign-S5-thm:1

AOT-theorem sign-S5-thm:2: ‹◇∀ α φ{α} → ∀ α ◇φ{α}›
proof -
```

AOT-have ‹$\forall\,\alpha$ ($\Diamond\forall\,\alpha$ $\varphi\{\alpha\}$ $\rightarrow$ $\Diamond\varphi\{\alpha\}$)›
  by (*simp add*: *RM*$\Diamond$ *cqt−orig*:*3* $\forall$ *I*)
AOT-thus ‹$\Diamond\forall\,\alpha$ $\varphi\{\alpha\}$ $\rightarrow$ $\forall\,\alpha$ $\Diamond\varphi\{\alpha\}$›
  using $\forall$ *E*(*4*) $\forall$ *I* $\rightarrow$*E* $\rightarrow$*I* **by** *metis*
qed
lemmas *Buridan*$\Diamond$ = *sign−S5−thm*:*2*

AOT-theorem *sign−S5−thm*:*3*:
‹$\Diamond\exists\,\alpha$ ($\varphi\{\alpha\}$ & $\psi\{\alpha\}$) $\rightarrow$ $\Diamond$($\exists\,\alpha$ $\varphi\{\alpha\}$ & $\exists\,\alpha$ $\psi\{\alpha\}$)›
apply (*rule RM*:*2*)
by (*metis* (*no-types*, *lifting*) $\exists$ *E* &*I* &*E*(*1*) &*E*(*2*) $\rightarrow$*I* $\exists$ *I*(*2*))

AOT-theorem *sign−S5−thm*:*4*: ‹$\Diamond\exists\,\alpha$ ($\varphi\{\alpha\}$ & $\psi\{\alpha\}$) $\rightarrow$ $\Diamond\exists\,\alpha$ $\varphi\{\alpha\}$›
apply (*rule RM*:*2*)
by (*meson instantiation* &*E*(*1*) $\rightarrow$*I* $\exists$ *I*(*2*))

AOT-theorem *sign−S5−thm*:*5*:
‹($\Box\forall\,\alpha$ ($\varphi\{\alpha\}$ $\rightarrow$ $\psi\{\alpha\}$) & $\Box\forall\,\alpha$ ($\psi\{\alpha\}$ $\rightarrow$ $\chi\{\alpha\}$)) $\rightarrow$ $\Box\forall\,\alpha$ ($\varphi\{\alpha\}$ $\rightarrow$ $\chi\{\alpha\}$)›
proof −
{
  fix φ' ψ' χ'
  AOT-assume ‹$\vdash_\Box$ φ' & ψ' $\rightarrow$ χ'›
  AOT-hence ‹$\Box\varphi'$ & $\Box\psi'$ $\rightarrow$ $\Box\chi'$›
    using *RN*[*prem*][**where** Γ={φ', ψ'}] **apply** *simp*
    using &*E* &*I* $\rightarrow$*E* $\rightarrow$*I* **by** *metis*
} **note** *R* = *this*
show *?thesis* **by** (*rule R*; *fact AOT*)
qed

AOT-theorem *sign−S5−thm*:*6*:
‹($\Box\forall\,\alpha$ ($\varphi\{\alpha\}$ $\equiv$ $\psi\{\alpha\}$) & $\Box\forall\,\alpha$($\psi\{\alpha\}$ $\equiv$ $\chi\{\alpha\}$)) $\rightarrow$ $\Box\forall\,\alpha$($\varphi\{\alpha\}$ $\equiv$ $\chi\{\alpha\}$)›
proof −
{
  fix φ' ψ' χ'
  AOT-assume ‹$\vdash_\Box$ φ' & ψ' $\rightarrow$ χ'›
  AOT-hence ‹$\Box\varphi'$ & $\Box\psi'$ $\rightarrow$ $\Box\chi'$›
    using *RN*[*prem*][**where** Γ={φ', ψ'}] **apply** *simp*
    using &*E* &*I* $\rightarrow$*E* $\rightarrow$*I* **by** *metis*
} **note** *R* = *this*
show *?thesis* **by** (*rule R*; *fact AOT*)
qed

AOT-theorem *exist−nec2*:*1*: ‹$\Diamond\tau\downarrow$ $\rightarrow$ $\tau\downarrow$›
using *B*$\Diamond$ *RM*$\Diamond$ *Hypothetical Syllogism exist−nec* **by** *blast*

AOT-theorem *exists−nec2*:*2*: ‹$\Diamond\tau\downarrow$ $\equiv$ $\Box\tau\downarrow$›
by (*meson Act−Sub*:*3 Hypothetical Syllogism exist−nec*
    exist−nec2:*1* $\equiv$*I nec−imp−act*)

AOT-theorem *exists−nec2*:*3*: ‹$\neg\tau\downarrow$ $\rightarrow$ $\Box\neg\tau\downarrow$›
using *KBasic2*:*1* $\rightarrow$*I exist−nec2*:*1* $\equiv$*E*(*2*) *modus−tollens*:*1* **by** *blast*

AOT-theorem *exists−nec2*:*4*: ‹$\Diamond\neg\tau\downarrow$ $\equiv$ $\Box\neg\tau\downarrow$›
by (*metis Act−Sub*:*3 KBasic*:*12* $\rightarrow$*I exist−nec exists−nec2*:*3*
    $\equiv$*I* $\equiv$*E*(*4*) *nec−imp−act reductio−aa*:*1*)

AOT-theorem *id−nec2*:*1*: ‹$\Diamond\alpha = \beta$ $\rightarrow$ $\alpha = \beta$›
using *B*$\Diamond$ *RM*$\Diamond$ *Hypothetical Syllogism id−nec*:*1* **by** *blast*

AOT-theorem *id−nec2*:*2*: ‹$\alpha \neq \beta$ $\rightarrow$ $\Box\alpha \neq \beta$›
apply (*AOT-subst* ‹$\alpha \neq \beta$› ‹$\neg(\alpha = \beta)$›)
using =−*infix*[*THEN* $\equiv$*Df*] **apply** *blast*
using *KBasic2*:*1* $\rightarrow$*I id−nec2*:*1* $\equiv$*E*(*2*) *modus−tollens*:*1* **by** *blast*

```
AOT-theorem id−nec2:3: ‹◇α ≠ β → α ≠ β›
  apply (AOT-subst ‹α ≠ β› ‹¬(α = β)›)
  using =−infix[THEN ≡Df] apply blast
  by (metis KBasic:11 →I id−nec:2 ≡E(3) reductio−aa:2 →E)

AOT-theorem id−nec2:4: ‹◇α = β → □α = β›
  using Hypothetical Syllogism id−nec2:1 id−nec:1 by blast

AOT-theorem id−nec2:5: ‹◇α ≠ β → □α ≠ β›
  using id−nec2:3 id−nec2:2 →I →E by metis

AOT-theorem sc−eq−box−box:1: ‹□(φ → □φ) ≡ (◇φ → □φ)›
  apply (rule ≡I; rule →I)
  using KBasic:13 5◇ Hypothetical Syllogism →E apply blast
  by (metis KBasic2:1 KBasic:1 KBasic:2 S5Basic:13 ≡E(2)
          raa−cor:5 →E)

AOT-theorem sc−eq−box−box:2: ‹(□(φ → □φ) ∨ (◇φ → □φ)) → (◇φ ≡ □φ)›
  by (metis Act−Sub:3 KBasic:13 5◇ ∨E(2) →I ≡I
          nec−imp−act raa−cor:2 →E)

AOT-theorem sc−eq−box−box:3: ‹□(φ → □φ) → (¬□φ ≡ □¬φ)›
proof (rule →I; rule ≡I; rule →I)
  AOT-assume ‹□(φ → □φ)›
  AOT-hence ‹◇φ → □φ› using sc−eq−box−box:1 ≡E by blast
  moreover AOT-assume ‹¬□φ›
  ultimately AOT-have ‹¬◇φ›
    using modus−tollens:1 by blast
  AOT-thus ‹□¬φ›
    using KBasic2:1 ≡E(2) by blast
next
  AOT-assume ‹□(φ → □φ)›
  moreover AOT-assume ‹□¬φ›
  ultimately AOT-show ‹¬□φ›
    using modus−tollens:1 qml:2[axiom-inst] →E by blast
qed

AOT-theorem sc−eq−box−box:4:
  ‹(□(φ → □φ) & □(ψ → □ψ)) → ((□φ ≡ □ψ) → □(φ ≡ ψ))›
proof(rule →I; rule →I)
  AOT-assume ϑ: ‹□(φ → □φ) & □(ψ → □ψ)›
  AOT-assume ξ: ‹□φ ≡ □ψ›
  AOT-hence ‹(□φ & □ψ) ∨ (¬□φ & ¬□ψ)›
    using ≡E(4) oth−class−taut:4:g raa−cor:3 by blast
  moreover {
    AOT-assume ‹□φ & □ψ›
    AOT-hence ‹□(φ ≡ ψ)›
      using KBasic:3 KBasic:8 ≡E(2) vdash−properties:10 by blast
  }
  moreover {
    AOT-assume ‹¬□φ & ¬□ψ›
    moreover AOT-have ‹¬□φ ≡ □¬φ› and ‹¬□ψ ≡ □¬ψ›
      using ϑ Conjunction Simplification(1,2)
          sc−eq−box−box:3 →E by metis+
    ultimately AOT-have ‹□¬φ & □¬ψ›
      by (metis &I Conjunction Simplification(1,2)
              ≡E(4) modus−tollens:1 raa−cor:3)
    AOT-hence ‹□(φ ≡ ψ)›
      using KBasic:3 KBasic:9 ≡E(2) →E by blast
  }
  ultimately AOT-show ‹□(φ ≡ ψ)›
    using ∨E(2) reductio−aa:1 by blast
```

qed

AOT-theorem *sc−eq−box−box:5*:
‹(□(φ → □φ) & □(ψ → □ψ)) → □((φ ≡ ψ) → □(φ ≡ ψ))›
proof (*rule* →*I*)
  AOT-assume ‹(□(φ → □φ) & □(ψ → □ψ))›
  AOT-hence ‹□(□(φ → □φ) & □(ψ → □ψ))›
    using *4*[*THEN* →*E*] &*E* &*I* *KBasic:3* ≡*E*(*2*) **by** *metis*
  moreover AOT-have ‹□(□(φ → □φ) & □(ψ → □ψ)) → □((φ ≡ ψ) → □(φ ≡ ψ))›
  proof (*rule RM*; *rule* →*I*; *rule* →*I*)
    AOT-modally-strict {
      AOT-assume *A*: ‹(□(φ → □φ) & □(ψ → □ψ))›
      AOT-hence ‹φ → □φ› **and** ‹ψ → □ψ›
        using &*E* *qml:2*[*axiom-inst*] →*E* **by** *blast+*
      moreover AOT-assume ‹φ ≡ ψ›
      ultimately AOT-have ‹□φ ≡ □ψ›
        using →*E* *qml:2*[*axiom-inst*] ≡*E* ≡*I* **by** *meson*
      moreover AOT-have ‹(□φ ≡ □ψ) → □(φ ≡ ψ)›
        using *A* *sc−eq−box−box:4* →*E* **by** *blast*
      ultimately AOT-show ‹□(φ ≡ ψ)› **using** →*E* **by** *blast*
    }
  qed
  ultimately AOT-show ‹□((φ ≡ ψ) → □(φ ≡ ψ))› **using** →*E* **by** *blast*
qed

AOT-theorem *sc−eq−box−box:6*: ‹□(φ → □φ) → ((φ → □ψ) → □(φ → ψ))›
proof (*rule* →*I*; *rule* →*I*; *rule raa−cor:1*)
  AOT-assume ‹¬□(φ → ψ)›
  AOT-hence ‹◇¬(φ → ψ)›
    by (*metis KBasic:11* ≡*E*(*1*))
  AOT-hence ‹◇(φ & ¬ψ)›
    by (*AOT-subst* ‹φ & ¬ψ› ‹¬(φ → ψ)›)
      (*meson Commutativity of* ≡ ≡*E*(*1*) *oth−class−taut:1:b*)
  AOT-hence ‹◇φ› **and** *2*: ‹◇¬ψ›
    using *KBasic2:3*[*THEN* →*E*] &*E* **by** *blast+*
  moreover AOT-assume ‹□(φ → □φ)›
  ultimately AOT-have ‹□φ›
    by (*metis* ≡*E*(*1*) *sc−eq−box−box:1* →*E*)
  AOT-hence φ
    using *qml:2*[*axiom-inst*, *THEN* →*E*] **by** *blast*
  moreover AOT-assume ‹φ → □ψ›
  ultimately AOT-have ‹□ψ›
    using →*E* **by** *blast*
  moreover AOT-have ‹¬□ψ›
    using *2 KBasic:12* ¬¬*I* *intro−elim:3:d* **by** *blast*
  ultimately AOT-show ‹□ψ & ¬□ψ›
    using &*I* **by** *blast*
qed

AOT-theorem *sc−eq−box−box:7*: ‹□(φ → □φ) → ((φ → 𝓐ψ) → 𝓐(φ → ψ))›
proof (*rule* →*I*; *rule* →*I*; *rule raa−cor:1*)
  AOT-assume ‹¬𝓐(φ → ψ)›
  AOT-hence ‹𝓐¬(φ → ψ)›
    by (*metis Act−Basic:1* ∨*E*(*2*))
  AOT-hence ‹𝓐(φ & ¬ψ)›
    by (*AOT-subst* ‹φ & ¬ψ› ‹¬(φ → ψ)›)
      (*meson Commutativity of* ≡ ≡*E*(*1*) *oth−class−taut:1:b*)
  AOT-hence ‹𝓐φ› **and** *2*: ‹𝓐¬ψ›
    using *Act−Basic:2*[*THEN* ≡*E*(*1*)] &*E* **by** *blast+*
  AOT-hence ‹◇φ›
    by (*metis Act−Sub:3* →*E*)
  moreover AOT-assume ‹□(φ → □φ)›
  ultimately AOT-have ‹□φ›

by (*metis* $\equiv E(1)$ *sc−eq−box−box*:*1* $\rightarrow E$)
AOT-hence φ
using *qml*:*2*[*axiom-inst*, *THEN* $\rightarrow E$] **by** *blast*
moreover AOT-assume ‹$\varphi \rightarrow \mathcal{A}\psi$›
ultimately AOT-have ‹$\mathcal{A}\psi$›
using $\rightarrow E$ **by** *blast*
moreover AOT-have ‹$\neg\mathcal{A}\psi$›
using *2* **by** (*meson* *Act−Sub*:*1* $\equiv E(4)$ *raa−cor*:*3*)
ultimately AOT-show ‹$\mathcal{A}\psi$ & $\neg\mathcal{A}\psi$›
using &*I* **by** *blast*
qed

AOT-theorem *sc−eq−fur*:*1*: ‹$\Diamond\mathcal{A}\varphi \equiv \Box\mathcal{A}\varphi$›
using *Act−Basic*:*6* *Act−Sub*:*4* $\equiv E(6)$ **by** *blast*

AOT-theorem *sc−eq−fur*:*2*: ‹$\Box(\varphi \rightarrow \Box\varphi) \rightarrow (\mathcal{A}\varphi \equiv \varphi)$›
by (*metis* *B*$\Diamond$ *Act−Sub*:*3* *KBasic*:*13* *T*$\Diamond$ *Hypothetical Syllogism* $\rightarrow I$ $\equiv I$ *nec−imp−act*)

AOT-theorem *sc−eq−fur*:*3*:
‹$\Box\forall x\ (\varphi\{x\} \rightarrow \Box\varphi\{x\}) \rightarrow (\exists!x\ \varphi\{x\} \rightarrow \iota x\ \varphi\{x\}\downarrow)$›
proof (*rule* $\rightarrow I$; *rule* $\rightarrow I$)
AOT-assume ‹$\Box\forall x\ (\varphi\{x\} \rightarrow \Box\varphi\{x\})$›
AOT-hence *A*: ‹$\forall x\ \Box(\varphi\{x\} \rightarrow \Box\varphi\{x\})$›
using *CBF* $\rightarrow E$ **by** *blast*
AOT-assume ‹$\exists!x\ \varphi\{x\}$›
then AOT-obtain *a* **where** *a-def*: ‹$\varphi\{a\}$ & $\forall y\ (\varphi\{y\} \rightarrow y = a)$›
using $\exists E$[*rotated 1*, *OF uniqueness*:*1*[*THEN* $\equiv_{df} E$]] **by** *blast*
moreover AOT-have ‹$\Box\varphi\{a\}$›
using *calculation A* $\forall E(2)$ *qml*:*2*[*axiom-inst*] $\rightarrow E$ &$E(1)$ **by** *blast*
AOT-hence ‹$\mathcal{A}\varphi\{a\}$›
using *nec−imp−act* $\rightarrow E$ **by** *blast*
moreover AOT-have ‹$\forall y\ (\mathcal{A}\varphi\{y\} \rightarrow y = a)$›
proof (*rule* $\forall I$; *rule* $\rightarrow I$)
fix *b*
AOT-assume ‹$\mathcal{A}\varphi\{b\}$›
AOT-hence ‹$\Diamond\varphi\{b\}$›
using *Act−Sub*:*3* $\rightarrow E$ **by** *blast*
moreover {
AOT-have ‹$\Box(\varphi\{b\} \rightarrow \Box\varphi\{b\})$›
using *A* $\forall E(2)$ **by** *blast*
AOT-hence ‹$\Diamond\varphi\{b\} \rightarrow \Box\varphi\{b\}$›
using *KBasic*:*13* *5*$\Diamond$ *Hypothetical Syllogism* $\rightarrow E$ **by** *blast*
}
ultimately AOT-have ‹$\Box\varphi\{b\}$›
using $\rightarrow E$ **by** *blast*
AOT-hence ‹$\varphi\{b\}$›
using *qml*:*2*[*axiom-inst*] $\rightarrow E$ **by** *blast*
AOT-thus ‹$b = a$›
using *a-def*[*THEN* &$E(2)$] $\forall E(2)$ $\rightarrow E$ **by** *blast*
qed
ultimately AOT-have ‹$\mathcal{A}\varphi\{a\}$ & $\forall y\ (\mathcal{A}\varphi\{y\} \rightarrow y = a)$›
using &*I* **by** *blast*
AOT-hence ‹$\exists x\ (\mathcal{A}\varphi\{x\}$ & $\forall y\ (\mathcal{A}\varphi\{y\} \rightarrow y = x))$›
using $\exists I$ **by** *fast*
AOT-hence ‹$\exists!x\ \mathcal{A}\varphi\{x\}$›
using *uniqueness*:*1*[*THEN* $\equiv_{df} I$] **by** *fast*
AOT-thus ‹$\iota x\ \varphi\{x\}\downarrow$›
using *actual−desc*:*1*[*THEN* $\equiv E(2)$] **by** *blast*
qed

AOT-theorem *sc−eq−fur*:*4*:
‹$\Box\forall x\ (\varphi\{x\} \rightarrow \Box\varphi\{x\}) \rightarrow (x = \iota x\ \varphi\{x\} \equiv (\varphi\{x\}$ & $\forall z\ (\varphi\{z\} \rightarrow z = x)))$›

```
proof (rule →I)
  AOT-assume ‹□∀ x (φ{x} → □φ{x})›
  AOT-hence ‹∀ x □(φ{x} → □φ{x})›
    using CBF →E by blast
  AOT-hence A: ‹𝓐φ{α} ≡ φ{α}› for α
    using sc−eq−fur:2 ∀ E →E by fast
  AOT-show ‹x = ιx φ{x} ≡ (φ{x} & ∀ z (φ{z} → z = x))›
  proof (rule ≡I; rule →I)
    AOT-assume ‹x = ιx φ{x}›
    AOT-hence B: ‹𝓐φ{x} & ∀ z (𝓐φ{z} → z = x)›
      using nec−hintikka−scheme[THEN ≡E(1)] by blast
    AOT-show ‹φ{x} & ∀ z (φ{z} → z = x)›
    proof (rule &I; (rule ∀ I; rule →I)?)
      AOT-show ‹φ{x}›
        using A B[THEN &E(1)] ≡E(1) by blast
    next
      AOT-show ‹z = x› if ‹φ{z}› for z
        using that B[THEN &E(2)] ∀ E(2) →E A[THEN ≡E(2)] by blast
    qed
  next
    AOT-assume B: ‹φ{x} & ∀ z (φ{z} → z = x)›
    AOT-have ‹𝓐φ{x} & ∀ z (𝓐φ{z} → z = x)›
    proof(rule &I; (rule ∀ I; rule →I)?)
      AOT-show ‹𝓐φ{x}›
        using B[THEN &E(1)] A[THEN ≡E(2)] by blast
    next
      AOT-show ‹b = x› if ‹𝓐φ{b}› for b
        using A[THEN ≡E(1)] that
              B[THEN &E(2), THEN ∀ E(2), THEN →E] by blast
    qed
    AOT-thus ‹x = ιx φ{x}›
      using nec−hintikka−scheme[THEN ≡E(2)] by blast
  qed
qed

AOT-theorem id−act:1: ‹α = β ≡ 𝓐α = β›
  by (meson Act−Sub:3 Hypothetical Syllogism
          id−nec2:1 id−nec:2 ≡I nec−imp−act)

AOT-theorem id−act:2: ‹α ≠ β ≡ 𝓐α ≠ β›
proof (AOT-subst ‹α ≠ β› ‹¬(α = β)›)
  AOT-modally-strict {
    AOT-show ‹α ≠ β ≡ ¬(α = β)›
      by (simp add: =−infix ≡Df)
  }
next
  AOT-show ‹¬(α = β) ≡ 𝓐¬(α = β)›
  proof (safe intro!: ≡I →I)
    AOT-assume ‹¬α = β›
    AOT-hence ‹¬𝓐α = β› using id−act:1 ≡E(3) by blast
    AOT-thus ‹𝓐¬α = β›
      using ¬¬E Act−Sub:1 ≡E(3) by blast
  next
    AOT-assume ‹𝓐¬α = β›
    AOT-hence ‹¬𝓐α = β›
      using ¬¬I Act−Sub:1 ≡E(4) by blast
    AOT-thus ‹¬α = β›
      using id−act:1 ≡E(4) by blast
  qed
qed

AOT-theorem A−Exists:1: ‹𝓐∃ !α φ{α} ≡ ∃ !α 𝓐φ{α}›
proof −
```

```
  AOT-have ‹𝓐∃!α φ{α} ≡ 𝓐∃α∀β (φ{β} ≡ β = α)›
    by (AOT-subst ‹∃!α φ{α}› ‹∃α∀β (φ{β} ≡ β = α)›)
       (auto simp add: oth−class−taut:3:a uniqueness:2)
  also AOT-have ‹... ≡ ∃α 𝓐∀β (φ{β} ≡ β = α)›
    by (simp add: Act−Basic:10)
  also AOT-have ‹... ≡ ∃α∀β 𝓐(φ{β} ≡ β = α)›
    by (AOT-subst ‹𝓐∀β (φ{β} ≡ β = α)› ‹∀β 𝓐(φ{β} ≡ β = α)› for: α)
       (auto simp: logic−actual−nec:3[axiom-inst] oth−class−taut:3:a)
  also AOT-have ‹... ≡ ∃α∀β (𝓐φ{β} ≡ 𝓐β = α)›
    by (AOT-subst (reverse) ‹𝓐φ{β} ≡ 𝓐β = α›
                           ‹𝓐(φ{β} ≡ β = α)› for: α β :: 'a)
       (auto simp: Act−Basic:5 cqt−further:7)
  also AOT-have ‹... ≡ ∃α∀β (𝓐φ{β} ≡ β = α)›
    by (AOT-subst (reverse) ‹𝓐β = α› ‹β = α› for: α β :: 'a)
       (auto simp: id−act:1 cqt−further:7)
  also AOT-have ‹... ≡ ∃!α 𝓐φ{α}›
    using uniqueness:2 Commutativity of ≡[THEN ≡E(1)] by fast
  finally show ?thesis.
qed

AOT-theorem A−Exists:2: ‹ιx φ{x}↓ ≡ 𝓐∃!x φ{x}›
  by (AOT-subst ‹𝓐∃!x φ{x}› ‹∃!x 𝓐φ{x}›)
     (auto simp: actual−desc:1 A−Exists:1)

AOT-theorem id−act−desc:1: ‹ιx (x = y)↓›
proof(rule existence:1[THEN ≡df I]; rule ∃I)
  AOT-show ‹[λx E!x → E!x]ιx (x = y)›
  proof (rule russell−axiom[exe,1].nec−russell−axiom[THEN ≡E(2)];
         rule ∃I; (rule &I)+)
    AOT-show ‹𝓐y = y› by (simp add: RA[2] id−eq:1)
  next
    AOT-show ‹∀z (𝓐z = y → z = y)›
      apply (rule ∀I)
      using id−act:1[THEN ≡E(2)] →I by blast
  next
    AOT-show ‹[λx E!x → E!x]y›
    proof (rule lambda−predicates:2[axiom-inst, THEN →E, THEN ≡E(2)])
      AOT-show ‹[λx E!x → E!x]↓›
        by cqt:2[lambda]
    next
      AOT-show ‹E!y → E!y›
        by (simp add: if−p−then−p)
    qed
  qed
next
  AOT-show ‹[λx E!x → E!x]↓›
    by cqt:2[lambda]
qed

AOT-theorem id−act−desc:2: ‹y = ιx (x = y)›
  by (rule descriptions[axiom-inst, THEN ≡E(2)];
      rule ∀I; rule id−act:1[symmetric])

AOT-theorem pre−en−eq:1[1]: ‹x₁[F] → □x₁[F]›
  by (simp add: encoding vdash−properties:1[2])

AOT-theorem pre−en−eq:1[2]: ‹x₁x₂[F] → □x₁x₂[F]›
proof (rule →I)
  AOT-assume ‹x₁x₂[F]›
  AOT-hence ‹x₁[λy [F]yx₂]› and ‹x₂[λy [F]x₁y]›
    using nary−encoding[2][axiom-inst, THEN ≡E(1)] &E by blast+
  moreover AOT-have ‹[λy [F]yx₂]↓› by cqt:2
  moreover AOT-have ‹[λy [F]x₁y]↓› by cqt:2
```

  ultimately AOT-have ‹$\Box x_1[\lambda y\ [F]yx_2]$› **and** ‹$\Box x_2[\lambda y\ [F]x_1y]$›
    using *encoding*[*axiom-inst*, *unvarify F*] $\rightarrow E$ $\&I$ **by** *blast*+
  note $A = this$
  AOT-hence ‹$\Box(x_1[\lambda y\ [F]yx_2]\ \&\ x_2[\lambda y\ [F]x_1y])$›
    using *KBasic*:*3*[*THEN* $\equiv E(2)$] $\&I$ **by** *blast*
  AOT-thus ‹$\Box x_1x_2[F]$›
    by (*rule nary−encoding*[*2*][*axiom-inst*, *THEN RN*,
                                   THEN KBasic:*6*[*THEN* $\rightarrow E$],
                                   THEN $\equiv E(2)$]])
qed

AOT-theorem *pre−en−eq*:*1*[*3*]: ‹$x_1x_2x_3[F] \rightarrow \Box x_1x_2x_3[F]$›
proof (*rule* $\rightarrow I$)
  AOT-assume ‹$x_1x_2x_3[F]$›
  AOT-hence ‹$x_1[\lambda y\ [F]yx_2x_3]$›
        and ‹$x_2[\lambda y\ [F]x_1yx_3]$›
        and ‹$x_3[\lambda y\ [F]x_1x_2y]$›
    using *nary−encoding*[*3*][*axiom-inst*, *THEN* $\equiv E(1)$] $\&E$ **by** *blast*+
  moreover AOT-have ‹$[\lambda y\ [F]yx_2x_3]\downarrow$› **by** *cqt*:*2*
  moreover AOT-have ‹$[\lambda y\ [F]x_1yx_3]\downarrow$› **by** *cqt*:*2*
  moreover AOT-have ‹$[\lambda y\ [F]x_1x_2y]\downarrow$› **by** *cqt*:*2*
  ultimately AOT-have ‹$\Box x_1[\lambda y\ [F]yx_2x_3]$›
                  and ‹$\Box x_2[\lambda y\ [F]x_1yx_3]$›
                  and ‹$\Box x_3[\lambda y\ [F]x_1x_2y]$›
    using *encoding*[*axiom-inst*, *unvarify F*] $\rightarrow E$ **by** *blast*+
  note $A = this$
  AOT-have B: ‹$\Box(x_1[\lambda y\ [F]yx_2x_3]\ \&\ x_2[\lambda y\ [F]x_1yx_3]\ \&\ x_3[\lambda y\ [F]x_1x_2y])$›
    by (*rule KBasic*:*3*[*THEN* $\equiv E(2)$] $\&I$ A)+
  AOT-thus ‹$\Box x_1x_2x_3[F]$›
    by (*rule nary−encoding*[*3*][*axiom-inst*, *THEN RN*,
              THEN KBasic:*6*[*THEN* $\rightarrow E$], *THEN* $\equiv E(2)$]])
qed

AOT-theorem *pre−en−eq*:*1*[*4*]: ‹$x_1x_2x_3x_4[F] \rightarrow \Box x_1x_2x_3x_4[F]$›
proof (*rule* $\rightarrow I$)
  AOT-assume ‹$x_1x_2x_3x_4[F]$›
  AOT-hence ‹$x_1[\lambda y\ [F]yx_2x_3x_4]$›
        and ‹$x_2[\lambda y\ [F]x_1yx_3x_4]$›
        and ‹$x_3[\lambda y\ [F]x_1x_2yx_4]$›
        and ‹$x_4[\lambda y\ [F]x_1x_2x_3y]$›
    using *nary−encoding*[*4*][*axiom-inst*, *THEN* $\equiv E(1)$] $\&E$ **by** *metis*+
  moreover AOT-have ‹$[\lambda y\ [F]yx_2x_3x_4]\downarrow$› **by** *cqt*:*2*
  moreover AOT-have ‹$[\lambda y\ [F]x_1yx_3x_4]\downarrow$› **by** *cqt*:*2*
  moreover AOT-have ‹$[\lambda y\ [F]x_1x_2yx_4]\downarrow$› **by** *cqt*:*2*
  moreover AOT-have ‹$[\lambda y\ [F]x_1x_2x_3y]\downarrow$› **by** *cqt*:*2*
  ultimately AOT-have ‹$\Box x_1[\lambda y\ [F]yx_2x_3x_4]$›
                  and ‹$\Box x_2[\lambda y\ [F]x_1yx_3x_4]$›
                  and ‹$\Box x_3[\lambda y\ [F]x_1x_2yx_4]$›
                  and ‹$\Box x_4[\lambda y\ [F]x_1x_2x_3y]$›
    using $\rightarrow E$ *encoding*[*axiom-inst*, *unvarify F*] **by** *blast*+
  note $A = this$
  AOT-have B: ‹$\Box(x_1[\lambda y\ [F]yx_2x_3x_4]\ \&$
                  $x_2[\lambda y\ [F]x_1yx_3x_4]\ \&$
                  $x_3[\lambda y\ [F]x_1x_2yx_4]\ \&$
                  $x_4[\lambda y\ [F]x_1x_2x_3y])$›
    by (*rule KBasic*:*3*[*THEN* $\equiv E(2)$] $\&I$ A)+
  AOT-thus ‹$\Box x_1x_2x_3x_4[F]$›
    by (*rule nary−encoding*[*4*][*axiom-inst*, *THEN RN*,
             THEN KBasic:*6*[*THEN* $\rightarrow E$], *THEN* $\equiv E(2)$]])
qed

AOT-theorem *pre−en−eq*:*2*[*1*]: ‹$\neg x_1[F] \rightarrow \Box\neg x_1[F]$›
proof (*rule* $\rightarrow I$; *rule raa−cor*:*1*)

AOT-assume ‹$\neg\Box\neg x_1[F]$›
AOT-hence ‹$\Diamond x_1[F]$›
by (*rule conventions*:*5*[*THEN* $\equiv_{df}I$])
AOT-hence ‹$x_1[F]$›
by(*rule S5Basic*:*13*[*THEN* $\equiv E(1)$, *OF pre−en−eq*:*1*[*1*][*THEN RN*],
THEN qml:*2*[*axiom-inst*, *THEN* $\rightarrow E$], *THEN* $\rightarrow E$])
moreover AOT-assume ‹$\neg x_1[F]$›
ultimately AOT-show ‹$x_1[F]$ & $\neg x_1[F]$› **by** (*rule* &*I*)
qed
AOT-theorem *pre−en−eq*:*2*[*2*]: ‹$\neg x_1x_2[F] \rightarrow \Box\neg x_1x_2[F]$›
proof (*rule* $\rightarrow I$; *rule raa−cor*:*1*)
AOT-assume ‹$\neg\Box\neg x_1x_2[F]$›
AOT-hence ‹$\Diamond x_1x_2[F]$›
by (*rule conventions*:*5*[*THEN* $\equiv_{df}I$])
AOT-hence ‹$x_1x_2[F]$›
by(*rule S5Basic*:*13*[*THEN* $\equiv E(1)$, *OF pre−en−eq*:*1*[*2*][*THEN RN*],
THEN qml:*2*[*axiom-inst*, *THEN* $\rightarrow E$], *THEN* $\rightarrow E$])
moreover AOT-assume ‹$\neg x_1x_2[F]$›
ultimately AOT-show ‹$x_1x_2[F]$ & $\neg x_1x_2[F]$› **by** (*rule* &*I*)
qed

AOT-theorem *pre−en−eq*:*2*[*3*]: ‹$\neg x_1x_2x_3[F] \rightarrow \Box\neg x_1x_2x_3[F]$›
proof (*rule* $\rightarrow I$; *rule raa−cor*:*1*)
AOT-assume ‹$\neg\Box\neg x_1x_2x_3[F]$›
AOT-hence ‹$\Diamond x_1x_2x_3[F]$›
by (*rule conventions*:*5*[*THEN* $\equiv_{df}I$])
AOT-hence ‹$x_1x_2x_3[F]$›
by(*rule S5Basic*:*13*[*THEN* $\equiv E(1)$, *OF pre−en−eq*:*1*[*3*][*THEN RN*],
THEN qml:*2*[*axiom-inst*, *THEN* $\rightarrow E$], *THEN* $\rightarrow E$])
moreover AOT-assume ‹$\neg x_1x_2x_3[F]$›
ultimately AOT-show ‹$x_1x_2x_3[F]$ & $\neg x_1x_2x_3[F]$› **by** (*rule* &*I*)
qed

AOT-theorem *pre−en−eq*:*2*[*4*]: ‹$\neg x_1x_2x_3x_4[F] \rightarrow \Box\neg x_1x_2x_3x_4[F]$›
proof (*rule* $\rightarrow I$; *rule raa−cor*:*1*)
AOT-assume ‹$\neg\Box\neg x_1x_2x_3x_4[F]$›
AOT-hence ‹$\Diamond x_1x_2x_3x_4[F]$›
by (*rule conventions*:*5*[*THEN* $\equiv_{df}I$])
AOT-hence ‹$x_1x_2x_3x_4[F]$›
by(*rule S5Basic*:*13*[*THEN* $\equiv E(1)$, *OF pre−en−eq*:*1*[*4*][*THEN RN*],
THEN qml:*2*[*axiom-inst*, *THEN* $\rightarrow E$], *THEN* $\rightarrow E$])
moreover AOT-assume ‹$\neg x_1x_2x_3x_4[F]$›
ultimately AOT-show ‹$x_1x_2x_3x_4[F]$ & $\neg x_1x_2x_3x_4[F]$› **by** (*rule* &*I*)
qed

AOT-theorem *en−eq*:*1*[*1*]: ‹$\Diamond x_1[F] \equiv \Box x_1[F]$›
using *pre−en−eq*:*1*[*1*][*THEN RN*] *sc−eq−box−box*:*2* $\vee I$ $\rightarrow E$ **by** *metis*
AOT-theorem *en−eq*:*1*[*2*]: ‹$\Diamond x_1x_2[F] \equiv \Box x_1x_2[F]$›
using *pre−en−eq*:*1*[*2*][*THEN RN*] *sc−eq−box−box*:*2* $\vee I$ $\rightarrow E$ **by** *metis*
AOT-theorem *en−eq*:*1*[*3*]: ‹$\Diamond x_1x_2x_3[F] \equiv \Box x_1x_2x_3[F]$›
using *pre−en−eq*:*1*[*3*][*THEN RN*] *sc−eq−box−box*:*2* $\vee I$ $\rightarrow E$ **by** *fast*
AOT-theorem *en−eq*:*1*[*4*]: ‹$\Diamond x_1x_2x_3x_4[F] \equiv \Box x_1x_2x_3x_4[F]$›
using *pre−en−eq*:*1*[*4*][*THEN RN*] *sc−eq−box−box*:*2* $\vee I$ $\rightarrow E$ **by** *fast*

AOT-theorem *en−eq*:*2*[*1*]: ‹$x_1[F] \equiv \Box x_1[F]$›
by (*simp add*: $\equiv I$ *pre−en−eq*:*1*[*1*] *qml*:*2*[*axiom-inst*])
AOT-theorem *en−eq*:*2*[*2*]: ‹$x_1x_2[F] \equiv \Box x_1x_2[F]$›
by (*simp add*: $\equiv I$ *pre−en−eq*:*1*[*2*] *qml*:*2*[*axiom-inst*])
AOT-theorem *en−eq*:*2*[*3*]: ‹$x_1x_2x_3[F] \equiv \Box x_1x_2x_3[F]$›
by (*simp add*: $\equiv I$ *pre−en−eq*:*1*[*3*] *qml*:*2*[*axiom-inst*])
AOT-theorem *en−eq*:*2*[*4*]: ‹$x_1x_2x_3x_4[F] \equiv \Box x_1x_2x_3x_4[F]$›
by (*simp add*: $\equiv I$ *pre−en−eq*:*1*[*4*] *qml*:*2*[*axiom-inst*])

AOT-theorem *en−eq:3[1]*: ‹$\Diamond x_1[F] \equiv x_1[F]$›
using *T*$\Diamond$ *derived−S5−rules:2[OF pre−en−eq:1[1]]* $\equiv I$ **by** *blast*
AOT-theorem *en−eq:3[2]*: ‹$\Diamond x_1x_2[F] \equiv x_1x_2[F]$›
using *T*$\Diamond$ *derived−S5−rules:2[OF pre−en−eq:1[2]]* $\equiv I$ **by** *blast*
AOT-theorem *en−eq:3[3]*: ‹$\Diamond x_1x_2x_3[F] \equiv x_1x_2x_3[F]$›
using *T*$\Diamond$ *derived−S5−rules:2[OF pre−en−eq:1[3]]* $\equiv I$ **by** *blast*
AOT-theorem *en−eq:3[4]*: ‹$\Diamond x_1x_2x_3x_4[F] \equiv x_1x_2x_3x_4[F]$›
using *T*$\Diamond$ *derived−S5−rules:2[OF pre−en−eq:1[4]]* $\equiv I$ **by** *blast*

AOT-theorem *en−eq:4[1]*:
‹$(x_1[F] \equiv y_1[G]) \equiv (\Box x_1[F] \equiv \Box y_1[G])$›
apply (*rule* $\equiv I$; *rule* $\rightarrow I$; *rule* $\equiv I$; *rule* $\rightarrow I$)
using *qml:2[axiom-inst, THEN* $\rightarrow E$] $\equiv E(1,2)$ *en−eq:2[1]* **by** *blast+*
AOT-theorem *en−eq:4[2]*:
‹$(x_1x_2[F] \equiv y_1y_2[G]) \equiv (\Box x_1x_2[F] \equiv \Box y_1y_2[G])$›
apply (*rule* $\equiv I$; *rule* $\rightarrow I$; *rule* $\equiv I$; *rule* $\rightarrow I$)
using *qml:2[axiom-inst, THEN* $\rightarrow E$] $\equiv E(1,2)$ *en−eq:2[2]* **by** *blast+*
AOT-theorem *en−eq:4[3]*:
‹$(x_1x_2x_3[F] \equiv y_1y_2y_3[G]) \equiv (\Box x_1x_2x_3[F] \equiv \Box y_1y_2y_3[G])$›
apply (*rule* $\equiv I$; *rule* $\rightarrow I$; *rule* $\equiv I$; *rule* $\rightarrow I$)
using *qml:2[axiom-inst, THEN* $\rightarrow E$] $\equiv E(1,2)$ *en−eq:2[3]* **by** *blast+*
AOT-theorem *en−eq:4[4]*:
‹$(x_1x_2x_3x_4[F] \equiv y_1y_2y_3y_4[G]) \equiv (\Box x_1x_2x_3x_4[F] \equiv \Box y_1y_2y_3y_4[G])$›
apply (*rule* $\equiv I$; *rule* $\rightarrow I$; *rule* $\equiv I$; *rule* $\rightarrow I$)
using *qml:2[axiom-inst, THEN* $\rightarrow E$] $\equiv E(1,2)$ *en−eq:2[4]* **by** *blast+*

AOT-theorem *en−eq:5[1]*:
‹$\Box(x_1[F] \equiv y_1[G]) \equiv (\Box x_1[F] \equiv \Box y_1[G])$›
apply (*rule* $\equiv I$; *rule* $\rightarrow I$)
using *en−eq:4[1][THEN* $\equiv E(1)$] *qml:2[axiom-inst, THEN* $\rightarrow E$]
apply *blast*
using *sc−eq−box−box:4[THEN* $\rightarrow E$, *THEN* $\rightarrow E$]
&*I[OF pre−en−eq:1[1][THEN RN], OF pre−en−eq:1[1][THEN RN]]*
by *blast*
AOT-theorem *en−eq:5[2]*:
‹$\Box(x_1x_2[F] \equiv y_1y_2[G]) \equiv (\Box x_1x_2[F] \equiv \Box y_1y_2[G])$›
apply (*rule* $\equiv I$; *rule* $\rightarrow I$)
using *en−eq:4[2][THEN* $\equiv E(1)$] *qml:2[axiom-inst, THEN* $\rightarrow E$]
apply *blast*
using *sc−eq−box−box:4[THEN* $\rightarrow E$, *THEN* $\rightarrow E$]
&*I[OF pre−en−eq:1[2][THEN RN], OF pre−en−eq:1[2][THEN RN]]*
by *blast*
AOT-theorem *en−eq:5[3]*:
‹$\Box(x_1x_2x_3[F] \equiv y_1y_2y_3[G]) \equiv (\Box x_1x_2x_3[F] \equiv \Box y_1y_2y_3[G])$›
apply (*rule* $\equiv I$; *rule* $\rightarrow I$)
using *en−eq:4[3][THEN* $\equiv E(1)$] *qml:2[axiom-inst, THEN* $\rightarrow E$]
apply *blast*
using *sc−eq−box−box:4[THEN* $\rightarrow E$, *THEN* $\rightarrow E$]
&*I[OF pre−en−eq:1[3][THEN RN], OF pre−en−eq:1[3][THEN RN]]*
by *blast*
AOT-theorem *en−eq:5[4]*:
‹$\Box(x_1x_2x_3x_4[F] \equiv y_1y_2y_3y_4[G]) \equiv (\Box x_1x_2x_3x_4[F] \equiv \Box y_1y_2y_3y_4[G])$›
apply (*rule* $\equiv I$; *rule* $\rightarrow I$)
using *en−eq:4[4][THEN* $\equiv E(1)$] *qml:2[axiom-inst, THEN* $\rightarrow E$]
apply *blast*
using *sc−eq−box−box:4[THEN* $\rightarrow E$, *THEN* $\rightarrow E$]
&*I[OF pre−en−eq:1[4][THEN RN], OF pre−en−eq:1[4][THEN RN]]*
by *blast*

AOT-theorem *en−eq:6[1]*:
‹$(x_1[F] \equiv y_1[G]) \equiv \Box(x_1[F] \equiv y_1[G])$›
using *en−eq:5[1][symmetric] en−eq:4[1]* $\equiv E(5)$ **by** *fast*
AOT-theorem *en−eq:6[2]*:

‹$(x_1x_2[F] \equiv y_1y_2[G]) \equiv \Box(x_1x_2[F] \equiv y_1y_2[G])$›
using *en−eq:5[2][symmetric] en−eq:4[2] ≡E(5)* **by** *fast*
AOT-theorem *en−eq:6[3]*:
‹$(x_1x_2x_3[F] \equiv y_1y_2y_3[G]) \equiv \Box(x_1x_2x_3[F] \equiv y_1y_2y_3[G])$›
using *en−eq:5[3][symmetric] en−eq:4[3] ≡E(5)* **by** *fast*
AOT-theorem *en−eq:6[4]*:
‹$(x_1x_2x_3x_4[F] \equiv y_1y_2y_3y_4[G]) \equiv \Box(x_1x_2x_3x_4[F] \equiv y_1y_2y_3y_4[G])$›
using *en−eq:5[4][symmetric] en−eq:4[4] ≡E(5)* **by** *fast*

AOT-theorem *en−eq:7[1]*: ‹$\neg x_1[F] \equiv \Box\neg x_1[F]$›
using *pre−en−eq:2[1] qml:2[axiom-inst] ≡I* **by** *blast*
AOT-theorem *en−eq:7[2]*: ‹$\neg x_1x_2[F] \equiv \Box\neg x_1x_2[F]$›
using *pre−en−eq:2[2] qml:2[axiom-inst] ≡I* **by** *blast*
AOT-theorem *en−eq:7[3]*: ‹$\neg x_1x_2x_3[F] \equiv \Box\neg x_1x_2x_3[F]$›
using *pre−en−eq:2[3] qml:2[axiom-inst] ≡I* **by** *blast*
AOT-theorem *en−eq:7[4]*: ‹$\neg x_1x_2x_3x_4[F] \equiv \Box\neg x_1x_2x_3x_4[F]$›
using *pre−en−eq:2[4] qml:2[axiom-inst] ≡I* **by** *blast*

AOT-theorem *en−eq:8[1]*: ‹$\Diamond\neg x_1[F] \equiv \neg x_1[F]$›
using *en−eq:2[1][THEN oth−class−taut:4:b[THEN ≡E(1)]]*
KBasic:11 ≡E(5)[symmetric] **by** *blast*
AOT-theorem *en−eq:8[2]*: ‹$\Diamond\neg x_1x_2[F] \equiv \neg x_1x_2[F]$›
using *en−eq:2[2][THEN oth−class−taut:4:b[THEN ≡E(1)]]*
KBasic:11 ≡E(5)[symmetric] **by** *blast*
AOT-theorem *en−eq:8[3]*: ‹$\Diamond\neg x_1x_2x_3[F] \equiv \neg x_1x_2x_3[F]$›
using *en−eq:2[3][THEN oth−class−taut:4:b[THEN ≡E(1)]]*
KBasic:11 ≡E(5)[symmetric] **by** *blast*
AOT-theorem *en−eq:8[4]*: ‹$\Diamond\neg x_1x_2x_3x_4[F] \equiv \neg x_1x_2x_3x_4[F]$›
using *en−eq:2[4][THEN oth−class−taut:4:b[THEN ≡E(1)]]*
KBasic:11 ≡E(5)[symmetric] **by** *blast*

AOT-theorem *en−eq:9[1]*: ‹$\Diamond\neg x_1[F] \equiv \Box\neg x_1[F]$›
using *en−eq:7[1] en−eq:8[1] ≡E(5)* **by** *blast*
AOT-theorem *en−eq:9[2]*: ‹$\Diamond\neg x_1x_2[F] \equiv \Box\neg x_1x_2[F]$›
using *en−eq:7[2] en−eq:8[2] ≡E(5)* **by** *blast*
AOT-theorem *en−eq:9[3]*: ‹$\Diamond\neg x_1x_2x_3[F] \equiv \Box\neg x_1x_2x_3[F]$›
using *en−eq:7[3] en−eq:8[3] ≡E(5)* **by** *blast*
AOT-theorem *en−eq:9[4]*: ‹$\Diamond\neg x_1x_2x_3x_4[F] \equiv \Box\neg x_1x_2x_3x_4[F]$›
using *en−eq:7[4] en−eq:8[4] ≡E(5)* **by** *blast*

AOT-theorem *en−eq:10[1]*: ‹$\boldsymbol{\mathcal{A}}x_1[F] \equiv x_1[F]$›
by (*metis Act−Sub:3 deduction−theorem ≡I ≡E(1)*
nec−imp−act en−eq:3[1] pre−en−eq:1[1])
AOT-theorem *en−eq:10[2]*: ‹$\boldsymbol{\mathcal{A}}x_1x_2[F] \equiv x_1x_2[F]$›
by (*metis Act−Sub:3 deduction−theorem ≡I ≡E(1)*
nec−imp−act en−eq:3[2] pre−en−eq:1[2])
AOT-theorem *en−eq:10[3]*: ‹$\boldsymbol{\mathcal{A}}x_1x_2x_3[F] \equiv x_1x_2x_3[F]$›
by (*metis Act−Sub:3 deduction−theorem ≡I ≡E(1)*
nec−imp−act en−eq:3[3] pre−en−eq:1[3])
AOT-theorem *en−eq:10[4]*: ‹$\boldsymbol{\mathcal{A}}x_1x_2x_3x_4[F] \equiv x_1x_2x_3x_4[F]$›
by (*metis Act−Sub:3 deduction−theorem ≡I ≡E(1)*
nec−imp−act en−eq:3[4] pre−en−eq:1[4])

AOT-theorem *oa−facts:1*: ‹$O!x \rightarrow \Box O!x$›
proof(*rule →I*)
AOT-modally-strict {
AOT-have ‹$[\lambda x\ \Diamond E!x]x \equiv \Diamond E!x$›
by (*rule lambda−predicates:2[axiom-inst, THEN →E]*) *cqt:2*
} note ϑ = *this*
AOT-assume ‹$O!x$›
AOT-hence ‹$[\lambda x\ \Diamond E!x]x$›
by (*rule* $=_{df}$*E(2)[OF AOT-ordinary, rotated 1]*) *cqt:2*
AOT-hence ‹$\Diamond E!x$› **using** ϑ*[THEN ≡E(1)]* **by** *blast*

```
  AOT-hence ‹□◇E!x› using qml:3[axiom-inst, THEN →E] by blast
  AOT-hence ‹□[λx ◇E!x]x›
    by (AOT-subst ‹[λx ◇E!x]x› ‹◇E!x›)
       (auto simp: ϑ)
  AOT-thus ‹□O!x›
    by (rule =df I(2)[OF AOT-ordinary, rotated 1]) cqt:2
qed

AOT-theorem oa−facts:2: ‹A!x → □A!x›
proof(rule →I)
  AOT-modally-strict {
    AOT-have ‹[λx ¬◇E!x]x ≡ ¬◇E!x›
      by (rule lambda−predicates:2[axiom-inst, THEN →E]) cqt:2
  } note ϑ = this
  AOT-assume ‹A!x›
  AOT-hence ‹[λx ¬◇E!x]x›
    by (rule =df E(2)[OF AOT-abstract, rotated 1]) cqt:2
  AOT-hence ‹¬◇E!x› using ϑ[THEN ≡E(1)] by blast
  AOT-hence ‹□¬E!x› using KBasic2:1[THEN ≡E(2)] by blast
  AOT-hence ‹□□¬E!x› using 4[THEN →E] by blast
  AOT-hence ‹□¬◇E!x›
    using KBasic2:1
    by (AOT-subst (reverse) ‹¬◇E!x› ‹□¬E!x›) blast
  AOT-hence ‹□[λx ¬◇E!x]x›
    by (AOT-subst ‹[λx ¬◇E!x]x› ‹¬◇E!x›)
       (auto simp: ϑ)
  AOT-thus ‹□A!x›
    by (rule =df I(2)[OF AOT-abstract, rotated 1]) cqt:2[lambda]
qed

AOT-theorem oa−facts:3: ‹◇O!x → O!x›
  using oa−facts:1 B◇ RM◇ Hypothetical Syllogism by blast
AOT-theorem oa−facts:4: ‹◇A!x → A!x›
  using oa−facts:2 B◇ RM◇ Hypothetical Syllogism by blast

AOT-theorem oa−facts:5: ‹◇O!x ≡ □O!x›
  by (meson Act−Sub:3 Hypothetical Syllogism ≡I nec−imp−act
          oa−facts:1 oa−facts:3)

AOT-theorem oa−facts:6: ‹◇A!x ≡ □A!x›
  by (meson Act−Sub:3 Hypothetical Syllogism ≡I nec−imp−act
          oa−facts:2 oa−facts:4)

AOT-theorem oa−facts:7: ‹O!x ≡ 𝓐O!x›
  by (meson Act−Sub:3 Hypothetical Syllogism ≡I nec−imp−act
          oa−facts:1 oa−facts:3)

AOT-theorem oa−facts:8: ‹A!x ≡ 𝓐A!x›
  by (meson Act−Sub:3 Hypothetical Syllogism ≡I nec−imp−act
          oa−facts:2 oa−facts:4)
```

8.10 The Theory of Relations

```
AOT-theorem beta−C−meta:
  ‹[λμ1...μn φ{μ1...μn, ν1...νn}]↓ →
   ([λμ1...μn φ{μ1...μn, ν1...νn}]ν1...νn ≡ φ{ν1...νn, ν1...νn})›
  using lambda−predicates:2[axiom-inst] by blast

AOT-theorem beta−C−cor:1:
  ‹(∀ν1...∀νn([λμ1...μn φ{μ1...μn, ν1...νn}]↓)) →
   ∀ν1...∀νn ([λμ1...μn φ{μ1...μn, ν1...νn}]ν1...νn ≡ φ{ν1...νn, ν1...νn})›
  apply (rule cqt−basic:14[where 'a='a, THEN →E])
  using beta−C−meta ∀I by fast
```

AOT-theorem *beta−C−cor:2*:
‹$[\lambda\mu_1...\mu_n\ \varphi\{\mu_1...\mu_n\}]\downarrow \rightarrow$
$\forall\nu_1...\forall\nu_n\ ([\lambda\mu_1...\mu_n\ \varphi\{\mu_1...\mu_n\}]\nu_1...\nu_n \equiv \varphi\{\nu_1...\nu_n\})$›
apply (*rule* $\rightarrow I$; *rule* $\forall I$)
using *beta−C−meta*[*THEN* $\rightarrow E$] **by** *fast*

theorem *beta−C−cor:3*:
assumes ‹$\bigwedge\nu_1\nu_n$. *AOT-instance-of-cqt-2* (φ (*AOT-term-of-var* $\nu_1\nu_n$))›
shows ‹$[v \models \forall\nu_1...\forall\nu_n\ ([\lambda\mu_1...\mu_n\ \varphi\{\nu_1...\nu_n,\ \mu_1...\mu_n\}]\nu_1...\nu_n \equiv$
$\varphi\{\nu_1...\nu_n,\ \nu_1...\nu_n\})]$›
using *cqt:2*[*lambda*][*axiom-inst*, *OF assms*]
beta−C−cor:1[*THEN* $\rightarrow E$] $\forall I$ **by** *fast*

AOT-theorem *betaC:1:a*: ‹$[\lambda\mu_1...\mu_n\ \varphi\{\mu_1...\mu_n\}]\kappa_1...\kappa_n \vdash_\Box \varphi\{\kappa_1...\kappa_n\}$›
proof −
AOT-modally-strict {
AOT-assume ‹$[\lambda\mu_1...\mu_n\ \varphi\{\mu_1...\mu_n\}]\kappa_1...\kappa_n$›
moreover AOT-have ‹$[\lambda\mu_1...\mu_n\ \varphi\{\mu_1...\mu_n\}]\downarrow$› **and** ‹$\kappa_1...\kappa_n\downarrow$›
using *calculation cqt:5:a*[*axiom-inst*, *THEN* $\rightarrow E$] &*E* **by** *blast+*
ultimately AOT-show ‹$\varphi\{\kappa_1...\kappa_n\}$›
using *beta−C−cor:2*[*THEN* $\rightarrow E$, *THEN* $\forall E(1)$, *THEN* $\equiv E(1)$] **by** *blast*
}
qed

AOT-theorem *betaC:1:b*: ‹$\neg\varphi\{\kappa_1...\kappa_n\} \vdash_\Box \neg[\lambda\mu_1...\mu_n\ \varphi\{\mu_1...\mu_n\}]\kappa_1...\kappa_n$›
using *betaC:1:a raa−cor:3* **by** *blast*

lemmas $\beta{\rightarrow}C$ = *betaC:1:a betaC:1:b*

AOT-theorem *betaC:2:a*:
‹$[\lambda\mu_1...\mu_n\ \varphi\{\mu_1...\mu_n\}]\downarrow,\ \kappa_1...\kappa_n\downarrow,\ \varphi\{\kappa_1...\kappa_n\} \vdash_\Box$
$[\lambda\mu_1...\mu_n\ \varphi\{\mu_1...\mu_n\}]\kappa_1...\kappa_n$›
proof −
AOT-modally-strict {
AOT-assume *1*: ‹$[\lambda\mu_1...\mu_n\ \varphi\{\mu_1...\mu_n\}]\downarrow$›
and *2*: ‹$\kappa_1...\kappa_n\downarrow$›
and *3*: ‹$\varphi\{\kappa_1...\kappa_n\}$›
AOT-hence ‹$[\lambda\mu_1...\mu_n\ \varphi\{\mu_1...\mu_n\}]\kappa_1...\kappa_n$›
using *beta−C−cor:2*[*THEN* $\rightarrow E$, *OF 1*, *THEN* $\forall E(1)$, *THEN* $\equiv E(2)$]
by *blast*
}
AOT-thus ‹$[\lambda\mu_1...\mu_n\ \varphi\{\mu_1...\mu_n\}]\downarrow,\ \kappa_1...\kappa_n\downarrow,\ \varphi\{\kappa_1...\kappa_n\} \vdash_\Box$
$[\lambda\mu_1...\mu_n\ \varphi\{\mu_1...\mu_n\}]\kappa_1...\kappa_n$›
by *blast*
qed

AOT-theorem *betaC:2:b*:
‹$[\lambda\mu_1...\mu_n\ \varphi\{\mu_1...\mu_n\}]\downarrow,\ \kappa_1...\kappa_n\downarrow,\ \neg[\lambda\mu_1...\mu_n\ \varphi\{\mu_1...\mu_n\}]\kappa_1...\kappa_n \vdash_\Box$
$\neg\varphi\{\kappa_1...\kappa_n\}$›
using *betaC:2:a raa−cor:3* **by** *blast*

lemmas $\beta{\leftarrow}C$ = *betaC:2:a betaC:2:b*

AOT-theorem *eta−conversion−lemma1:1*: ‹$\Pi\downarrow \rightarrow [\lambda x_1...x_n\ [\Pi]x_1...x_n] = \Pi$›
using *lambda−predicates:3*[*axiom-inst*] $\forall I$ $\forall E(1)$ $\rightarrow I$ **by** *fast*

AOT-theorem *eta−conversion−lemma1:2*: ‹$\Pi\downarrow \rightarrow [\lambda\nu_1...\nu_n\ [\Pi]\nu_1...\nu_n] = \Pi$›
using *eta−conversion−lemma1:1.*

Note: not explicitly part of PLM.

```
AOT-theorem id-sym:
  assumes ‹τ = τ'›
  shows ‹τ' = τ›
  using rule=E[where φ=λ τ' . «τ' = τ», rotated 1, OF assms]
        =I(1)[OF t=t-proper:1[THEN →E, OF assms]] by auto
declare id-sym[sym]
```

Note: not explicitly part of PLM.

```
AOT-theorem id-trans:
  assumes ‹τ = τ'› and ‹τ' = τ''›
  shows ‹τ = τ''›
  using rule=E assms by blast
declare id-trans[trans]

method ηC for Π :: ‹<'a::{AOT-Term-id-2,AOT-κs}>› =
  (match conclusion in [v ⊨ τ{Π} = τ'{Π}] for v τ τ' ⇒ ‹
   rule rule=E[rotated 1, OF eta−conversion−lemma1:2
    [THEN →E, of v «[Π]», symmetric]]›)

AOT-theorem sub−des−lam:1:
  ‹[λz1...zn  χ{z1...zn, ιx φ{x}}]↓ & ιx φ{x} = ιx ψ{x} →
   [λz1...zn χ{z1...zn, ιx φ{x}}] = [λz1...zn χ{z1...zn, ιx ψ{x}}]›
proof(rule →I)
  AOT-assume A: ‹[λz1...zn  χ{z1...zn, ιx φ{x}}]↓ & ιx φ{x} = ιx ψ{x}›
  AOT-show ‹[λz1...zn χ{z1...zn, ιx φ{x}}] = [λz1...zn χ{z1...zn, ιx ψ{x}}]›
    using rule=E[where φ=λ τ . «[λz1...zn χ{z1...zn, ιx φ{x}}] =
                               [λz1...zn χ{z1...zn, τ}]»,
              OF =I(1)[OF A[THEN &E(1)]], OF A[THEN &E(2)]]
    by blast
qed

AOT-theorem sub−des−lam:2:
  ‹ιx φ{x} = ιx ψ{x} → χ{ιx φ{x}} = χ{ιx ψ{x}}› for χ :: ‹κ ⇒ o›
  using rule=E[where φ=λ τ . «χ{ιx φ{x}} = χ{τ}»,
               OF =I(1)[OF log−prop−prop:2]] →I by blast

AOT-theorem prop−equiv: ‹F = G ≡ ∀ x (x[F] ≡ x[G])›
proof(rule ≡I; rule →I)
  AOT-assume ‹F = G›
  AOT-thus ‹∀ x (x[F] ≡ x[G])›
    by (rule rule=E[rotated]) (fact oth−class−taut:3:a[THEN GEN])
next
  AOT-assume ‹∀ x (x[F] ≡ x[G])›
  AOT-hence ‹x[F] ≡ x[G]› for x
    using ∀ E by blast
  AOT-hence ‹□(x[F] ≡ x[G])› for x
    using en−eq:6[1][THEN ≡E(1)] by blast
  AOT-hence ‹∀ x □(x[F] ≡ x[G])›
    by (rule GEN)
  AOT-hence ‹□∀ x (x[F] ≡ x[G])›
    using BF[THEN →E] by fast
  AOT-thus F = G
    using p−identity−thm2:1[THEN ≡E(2)] by blast
qed

AOT-theorem relations:1:
  assumes ‹INSTANCE-OF-CQT-2(φ)›
  shows ‹∃ F □∀ x1...∀ xn ([F]x1...xn ≡ φ{x1...xn})›
  apply (rule ∃ I(1)[where τ=«[λx1...xn φ{x1...xn}]»])
  using cqt:2[lambda][OF assms, axiom-inst]
        beta−C−cor:2[THEN →E, THEN RN] by blast+

AOT-theorem relations:2:
```

```
  assumes ‹INSTANCE-OF-CQT-2(φ)›
  shows ‹∃ F □∀ x ([F]x ≡ φ{x})›
  using relations:1 assms by blast

AOT-theorem block−paradox:1: ‹¬[λx ∃ G (x[G] & ¬[G]x)]↓›
proof(rule raa−cor:2)
  let ?K=«[λx ∃ G (x[G] & ¬[G]x)]»
  AOT-assume A: ‹«?K»↓›
  AOT-have ‹∃ x (A!x & ∀ F (x[F] ≡ F = «?K»))›
    using A−objects[axiom-inst] by fast
  then AOT-obtain a where ξ: ‹A!a & ∀ F (a[F] ≡ F = «?K»)›
    using ∃ E[rotated] by blast
  AOT-show ‹p & ¬p› for p
  proof (rule ∨E(1)[OF exc−mid]; rule →I)
    AOT-assume B: ‹[«?K»]a›
    AOT-hence ‹∃ G (a[G] & ¬[G]a)›
      using β→C A by blast
    then AOT-obtain P where ‹a[P] & ¬[P]a›
      using ∃ E[rotated] by blast
    moreover AOT-have ‹P = [«?K»]›
      using ξ[THEN &E(2), THEN ∀ E(2), THEN ≡E(1)]
            calculation[THEN &E(1)] by blast
    ultimately AOT-have ‹¬[«?K»]a›
      using rule=E &E(2) by fast
    AOT-thus ‹p & ¬p›
      using B RAA by blast
  next
    AOT-assume B: ‹¬[«?K»]a›
    AOT-hence ‹¬∃ G (a[G] & ¬[G]a)›
      using β←C cqt:2[const-var][of a, axiom-inst] A by blast
    AOT-hence C: ‹∀ G ¬(a[G] & ¬[G]a)›
      using cqt−further:4[THEN →E] by blast
    AOT-have ‹∀ G (a[G] → [G]a)›
      by (AOT-subst ‹a[G] → [G]a› ‹¬(a[G] & ¬[G]a)› for: G)
         (auto simp: oth−class−taut:1:a C)
    AOT-hence ‹a[«?K»] → [«?K»]a›
      using ∀ E A by blast
    moreover AOT-have ‹a[«?K»]›
      using ξ[THEN &E(2), THEN ∀ E(1), OF A, THEN ≡E(2)]
      using =I(1)[OF A] by blast
    ultimately AOT-show ‹p & ¬p›
      using B →E RAA by blast
  qed
qed

AOT-theorem block−paradox:2: ‹¬∃ F ∀ x([F]x ≡ ∃ G(x[G] & ¬[G]x))›
proof(rule RAA(2))
  AOT-assume ‹∃ F ∀ x ([F]x ≡ ∃ G (x[G] & ¬[G]x))›
  then AOT-obtain F where F-prop: ‹∀ x ([F]x ≡ ∃ G (x[G] & ¬[G]x))›
    using ∃ E[rotated] by blast
  AOT-have ‹∃ x (A!x & ∀ G (x[G] ≡ G = F))›
    using A−objects[axiom-inst] by fast
  then AOT-obtain a where ξ: ‹A!a & ∀ G (a[G] ≡ G = F)›
    using ∃ E[rotated] by blast
  AOT-show ‹¬∃ F ∀ x([F]x ≡ ∃ G(x[G] & ¬[G]x))›
  proof (rule ∨E(1)[OF exc−mid]; rule →I)
    AOT-assume B: ‹[F]a›
    AOT-hence ‹∃ G (a[G] & ¬[G]a)›
      using F-prop[THEN ∀ E(2), THEN ≡E(1)] by blast
    then AOT-obtain P where ‹a[P] & ¬[P]a›
      using ∃ E[rotated] by blast
    moreover AOT-have ‹P = F›
      using ξ[THEN &E(2), THEN ∀ E(2), THEN ≡E(1)]
```

```
        calculation[THEN &E(1)] by blast
    ultimately AOT-have ‹¬[F]a›
      using rule=E &E(2) by fast
    AOT-thus ‹¬∃ F ∀ x([F]x ≡ ∃ G(x[G] & ¬[G]x))›
      using B RAA by blast
  next
    AOT-assume B: ‹¬[F]a›
    AOT-hence ‹¬∃ G (a[G] & ¬[G]a)›
      using oth−class−taut:4:b[THEN ≡E(1),
            OF F-prop[THEN ∀ E(2)[of - - a]], THEN ≡E(1)]
      by simp
    AOT-hence C: ‹∀ G ¬(a[G] & ¬[G]a)›
      using cqt−further:4[THEN →E] by blast
    AOT-have ‹∀ G (a[G] → [G]a)›
      by (AOT-subst ‹a[G] → [G]a› ‹¬(a[G] & ¬[G]a)› for: G)
         (auto simp: oth−class−taut:1:a C)
    AOT-hence ‹a[F] → [F]a›
      using ∀ E by blast
    moreover AOT-have ‹a[F]›
      using ξ[THEN &E(2), THEN ∀ E(2), of F, THEN ≡E(2)]
      using =I(2) by blast
    ultimately AOT-show ‹¬∃ F ∀ x([F]x ≡ ∃ G(x[G] & ¬[G]x))›
      using B →E RAA by blast
  qed
qed(simp)

AOT-theorem block−paradox:3: ‹¬∀ y [λz z = y]↓›
proof(rule RAA(2))
  AOT-assume ϑ: ‹∀ y [λz z = y]↓›
  AOT-have ‹∃ x (A!x & ∀ F (x[F] ≡ ∃ y(F = [λz z = y] & ¬y[F])))›
    using A−objects[axiom-inst] by force
  then AOT-obtain a where
    a-prop: ‹A!a & ∀ F (a[F] ≡ ∃ y (F = [λz z = y] & ¬y[F]))›
    using ∃ E[rotated] by blast
  AOT-have ζ: ‹a[λz z = a] ≡ ∃ y ([λz z = a] = [λz z = y] & ¬y[λz z = a])›
    using ϑ[THEN ∀ E(2)] a-prop[THEN &E(2), THEN ∀ E(1)] by blast
  AOT-show ‹¬∀ y [λz z = y]↓›
  proof (rule ∨E(1)[OF exc−mid]; rule →I)
    AOT-assume A: ‹a[λz z = a]›
    AOT-hence ‹∃ y ([λz z = a] = [λz z = y] & ¬y[λz z = a])›
      using ζ[THEN ≡E(1)] by blast
    then AOT-obtain b where b-prop: ‹[λz z = a] = [λz z = b] & ¬b[λz z = a]›
      using ∃ E[rotated] by blast
    moreover AOT-have ‹a = a› by (rule =I)
    moreover AOT-have ‹[λz z = a]↓› using ϑ ∀ E by blast
    moreover AOT-have ‹a↓› using cqt:2[const-var][axiom-inst] .
    ultimately AOT-have ‹[λz z = a]a› using β←C by blast
    AOT-hence ‹[λz z = b]a› using rule=E b-prop[THEN &E(1)] by fast
    AOT-hence ‹a = b› using β→C by blast
    AOT-hence ‹b[λz z = a]› using A rule=E by fast
    AOT-thus ‹¬∀ y [λz z = y]↓› using b-prop[THEN &E(2)] RAA by blast
  next
    AOT-assume A: ‹¬a[λz z = a]›
    AOT-hence ‹¬∃ y ([λz z = a] = [λz z = y] & ¬y[λz z = a])›
      using ζ oth−class−taut:4:b[THEN ≡E(1), THEN ≡E(1)] by blast
    AOT-hence ‹∀ y ¬([λz z = a] = [λz z = y] & ¬y[λz z = a])›
      using cqt−further:4[THEN →E] by blast
    AOT-hence ‹¬([λz z = a] = [λz z = a] & ¬a[λz z = a])›
      using ∀ E by blast
    AOT-hence ‹[λz z = a] = [λz z = a] → a[λz z = a]›
      by (metis &I deduction−theorem raa−cor:4)
    AOT-hence ‹a[λz z = a]› using =I(1) ϑ[THEN ∀ E(2)] →E by blast
    AOT-thus ‹¬∀ y [λz z = y]↓› using A RAA by blast
```

qed
qed(*simp*)

AOT-theorem *block−paradox*:*4*: ‹¬∀ *y* ∃ *F* ∀ *x*([*F*]*x* ≡ *x* = *y*)›
proof(*rule RAA*(*2*))
AOT-assume *ϑ*: ‹∀ *y* ∃ *F* ∀ *x*([*F*]*x* ≡ *x* = *y*)›
AOT-have ‹∃ *x* (*A*!*x* & ∀ *F* (*x*[*F*] ≡ ∃ *z* (∀ *y*([*F*]*y* ≡ *y* = *z*) & ¬*z*[*F*])))›
using *A−objects*[*axiom-inst*] **by** *force*
then AOT-obtain *a* **where**
a-prop: ‹*A*!*a* & ∀ *F* (*a*[*F*] ≡ ∃ *z* (∀ *y*([*F*]*y* ≡ *y* = *z*) & ¬*z*[*F*]))›
using ∃ *E*[*rotated*] **by** *blast*
AOT-obtain *F* **where** *F-prop*: ‹∀ *x* ([*F*]*x* ≡ *x* = *a*)›
using *ϑ*[*THEN* ∀ *E*(*2*)] ∃ *E*[*rotated*] **by** *blast*
AOT-have *ζ*: ‹*a*[*F*] ≡ ∃ *z* (∀ *y* ([*F*]*y* ≡ *y* = *z*) & ¬*z*[*F*])›
using *a-prop*[*THEN* &*E*(*2*), *THEN* ∀ *E*(*2*)] **by** *blast*
AOT-show ‹¬∀ *y* ∃ *F* ∀ *x*([*F*]*x* ≡ *x* = *y*)›
proof (*rule* ∨*E*(*1*)[*OF exc−mid*]; *rule* →*I*)
AOT-assume *A*: ‹*a*[*F*]›
AOT-hence ‹∃ *z* (∀ *y* ([*F*]*y* ≡ *y* = *z*) & ¬*z*[*F*])›
using *ζ*[*THEN* ≡*E*(*1*)] **by** *blast*
then AOT-obtain *b* **where** *b-prop*: ‹∀ *y* ([*F*]*y* ≡ *y* = *b*) & ¬*b*[*F*]›
using ∃ *E*[*rotated*] **by** *blast*
moreover AOT-have ‹[*F*]*a*›
using *F-prop*[*THEN* ∀ *E*(*2*), *THEN* ≡*E*(*2*)] =*I*(*2*) **by** *blast*
ultimately AOT-have ‹*a* = *b*›
using ∀ *E*(*2*) ≡*E*(*1*) &*E* **by** *fast*
AOT-hence ‹*a* = *b*›
using *β*→*C* **by** *blast*
AOT-hence ‹*b*[*F*]›
using *A rule*=*E* **by** *fast*
AOT-thus ‹¬∀ *y* ∃ *F* ∀ *x*([*F*]*x* ≡ *x* = *y*)›
using *b-prop*[*THEN* &*E*(*2*)] *RAA* **by** *blast*
next
AOT-assume *A*: ‹¬*a*[*F*]›
AOT-hence ‹¬∃ *z* (∀ *y* ([*F*]*y* ≡ *y* = *z*) & ¬*z*[*F*])›
using *ζ oth−class−taut*:*4*:*b*[*THEN* ≡*E*(*1*), *THEN* ≡*E*(*1*)] **by** *blast*
AOT-hence ‹∀ *z* ¬(∀ *y* ([*F*]*y* ≡ *y* = *z*) & ¬*z*[*F*])›
using *cqt−further*:*4*[*THEN* →*E*] **by** *blast*
AOT-hence ‹¬(∀ *y* ([*F*]*y* ≡ *y* = *a*) & ¬*a*[*F*])›
using ∀ *E* **by** *blast*
AOT-hence ‹∀ *y* ([*F*]*y* ≡ *y* = *a*) → *a*[*F*]›
by (*metis* &*I deduction−theorem raa−cor*:*4*)
AOT-hence ‹*a*[*F*]› **using** *F-prop* →*E* **by** *blast*
AOT-thus ‹¬∀ *y* ∃ *F* ∀ *x*([*F*]*x* ≡ *x* = *y*)›
using *A RAA* **by** *blast*
qed
qed(*simp*)

AOT-theorem *block−paradox*:*5*: ‹¬∃ *F*∀ *x*∀ *y*([*F*]*xy* ≡ *y* = *x*)›
proof(*rule raa−cor*:*2*)
AOT-assume ‹∃ *F*∀ *x*∀ *y*([*F*]*xy* ≡ *y* = *x*)›
then AOT-obtain *F* **where** *F-prop*: ‹∀ *x*∀ *y*([*F*]*xy* ≡ *y* = *x*)›
using ∃ *E*[*rotated*] **by** *blast*
{
fix *x*
AOT-have *1*: ‹∀ *y*([*F*]*xy* ≡ *y* = *x*)›
using *F-prop* ∀ *E* **by** *blast*
AOT-have *2*: ‹[*λz* [*F*]*xz*]↓› **by** *cqt*:*2*
moreover AOT-have ‹∀ *y*([*λz* [*F*]*xz*]*y* ≡ *y* = *x*)›
proof(*rule* ∀ *I*)
fix *y*
AOT-have ‹[*λz* [*F*]*xz*]*y* ≡ [*F*]*xy*›
using *beta−C−meta*[*THEN* →*E*] *2* **by** *fast*

```
      also AOT-have ‹... ≡ y = x›
        using 1 ∀E by fast
      finally AOT-show ‹[λz [F]xz]y ≡ y = x›.
    qed
    ultimately AOT-have ‹∃F∀y([F]y ≡ y = x)›
      using ∃I by fast
  }
  AOT-hence ‹∀x∃F∀y([F]y ≡ y = x)›
    by (rule GEN)
  AOT-thus ‹∀x∃F∀y([F]y ≡ y = x) & ¬∀x∃F∀y([F]y ≡ y = x)›
    using &I block−paradox:4 by blast
qed

AOT-act-theorem block−paradox2:1:
  ‹∀x [G]x → ¬[λx [G]ιy (y = x & ∃H (x[H] & ¬[H]x))]↓›
proof(rule →I; rule raa−cor:2)
  AOT-assume antecedant: ‹∀x [G]x›
  AOT-have Lemma: ‹∀x ([G]ιy(y = x & ∃H (x[H] & ¬[H]x)) ≡ ∃H (x[H] & ¬[H]x))›
  proof(rule GEN)
    fix x
    AOT-have A: ‹[G]ιy (y = x & ∃H (x[H] & ¬[H]x)) ≡
                ∃!y (y = x & ∃H (x[H] & ¬[H]x))›
    proof(rule ≡I; rule →I)
      AOT-assume ‹[G]ιy (y = x & ∃H (x[H] & ¬[H]x))›
      AOT-hence ‹ιy (y = x & ∃H (x[H] & ¬[H]x))↓›
        using cqt:5:a[axiom-inst, THEN →E, THEN &E(2)] by blast
      AOT-thus ‹∃!y (y = x & ∃H (x[H] & ¬[H]x))›
        using !−exists:1[THEN ≡E(1)] by blast
    next
      AOT-assume A: ‹∃!y (y = x & ∃H (x[H] & ¬[H]x))›
      AOT-obtain a where a-1: ‹a = x & ∃H (x[H] & ¬[H]x)›
                  and a-2: ‹∀z (z = x & ∃H (x[H] & ¬[H]x) → z = a)›
        using uniqueness:1[THEN ≡df E, OF A] &E ∃E[rotated] by blast
      AOT-have a-3: ‹[G]a›
        using antecedant ∀E by blast
      AOT-show ‹[G]ιy (y = x & ∃H (x[H] & ¬[H]x))›
        apply (rule russell−axiom[exe,1].russell−axiom[THEN ≡E(2)])
        apply (rule ∃I(2))
        using a-1 a-2 a-3 &I by blast
    qed
    also AOT-have B: ‹... ≡ ∃H (x[H] & ¬[H]x)›
    proof (rule ≡I; rule →I)
      AOT-assume A: ‹∃!y (y = x & ∃H (x[H] & ¬[H]x))›
      AOT-obtain a where ‹a = x & ∃H (x[H] & ¬[H]x)›
        using uniqueness:1[THEN ≡df E, OF A] &E ∃E[rotated] by blast
      AOT-thus ‹∃H (x[H] & ¬[H]x)› using &E by blast
    next
      AOT-assume ‹∃H (x[H] & ¬[H]x)›
      AOT-hence ‹x = x & ∃H (x[H] & ¬[H]x)›
        using id−eq:1 &I by blast
      moreover AOT-have ‹∀z (z = x & ∃H (x[H] & ¬[H]x) → z = x)›
        by (simp add: Conjunction Simplification(1) universal−cor)
      ultimately AOT-show ‹∃!y (y = x & ∃H (x[H] & ¬[H]x))›
        using uniqueness:1[THEN ≡df I] &I ∃I(2) by fast
    qed
    finally AOT-show ‹([G]ιy(y = x & ∃H (x[H] & ¬[H]x)) ≡ ∃H (x[H] & ¬[H]x))› .
  qed

  AOT-assume A: ‹[λx [G]ιy (y = x & ∃H (x[H] & ¬[H]x))]↓›
  AOT-have ϑ: ‹∀x ([λx [G]ιy (y = x & ∃H (x[H] & ¬[H]x))]x ≡
                [G]ιy(y = x & ∃H (x[H] & ¬[H]x)))›
    using beta−C−meta[THEN →E, OF A] ∀I by fast
  AOT-have ‹∀x ([λx [G]ιy (y = x & ∃H (x[H] & ¬[H]x))]x ≡ ∃H (x[H] & ¬[H]x))›
```

```
    using ϑ Lemma cqt−basic:10[THEN →E] &I by fast
  AOT-hence ‹∃ F ∀ x ([F]x ≡ ∃ H (x[H] & ¬[H]x))›
    using ∃ I(1) A by fast
  AOT-thus ‹(∃ F ∀ x ([F]x ≡ ∃ H (x[H] & ¬[H]x))) &
          (¬∃ F ∀ x ([F]x ≡ ∃ H (x[H] & ¬[H]x)))›
    using block−paradox:2 &I by blast
qed
```

Note: Strengthens the above to a modally-strict theorem. Not explicitly part of PLM.

```
AOT-theorem block−paradox2:1[strict]:
  ‹∀ x 𝓐[G]x → ¬[λx [G]ιy (y = x & ∃ H (x[H] & ¬[H]x))]↓›
proof(rule →I; rule raa−cor:2)
  AOT-assume antecedant: ‹∀ x 𝓐[G]x›
  AOT-have Lemma: ‹𝓐∀ x ([G]ιy(y = x & ∃ H (x[H] & ¬[H]x)) ≡ ∃ H (x[H] & ¬[H]x))›
  proof(safe intro!: GEN Act−Basic:5[THEN ≡E(2)]
                    logic−actual−nec:3[axiom-inst, THEN ≡E(2)])
    fix x
    AOT-have A: ‹𝓐[G]ιy (y = x & ∃ H (x[H] & ¬[H]x)) ≡
                ∃ !y 𝓐(y = x & ∃ H (x[H] & ¬[H]x))›
    proof(rule ≡I; rule →I)
      AOT-assume ‹𝓐[G]ιy (y = x & ∃ H (x[H] & ¬[H]x))›
      moreover AOT-have ‹□([G]ιy (y = x & ∃ H (x[H] & ¬[H]x)) →
                               □ιy (y = x & ∃ H (x[H] & ¬[H]x))↓)›
      proof(rule RN; rule →I)
        AOT-modally-strict {
          AOT-assume ‹[G]ιy (y = x & ∃ H (x[H] & ¬[H]x))›
          AOT-hence ‹ιy (y = x & ∃ H (x[H] & ¬[H]x))↓›
            using cqt:5:a[axiom-inst, THEN →E, THEN &E(2)] by blast
          AOT-thus ‹□ιy (y = x & ∃ H (x[H] & ¬[H]x))↓›
            using exist−nec[THEN →E] by blast
        }
      qed
      ultimately AOT-have ‹𝓐□ιy (y = x & ∃ H (x[H] & ¬[H]x))↓›
        using act−cond[THEN →E, THEN →E] nec−imp−act[THEN →E] by blast
      AOT-hence ‹ιy (y = x & ∃ H (x[H] & ¬[H]x))↓›
        using Act−Sub:3 B◇ vdash−properties:10 by blast
      AOT-thus ‹∃ !y 𝓐(y = x & ∃ H (x[H] & ¬[H]x))›
        using actual−desc:1[THEN ≡E(1)] by blast
    next
      AOT-assume A: ‹∃ !y 𝓐(y = x & ∃ H (x[H] & ¬[H]x))›
      AOT-obtain a where a-1: ‹𝓐(a = x & ∃ H (x[H] & ¬[H]x))›
                   and a-2: ‹∀ z (𝓐(z = x & ∃ H (x[H] & ¬[H]x)) → z = a)›
        using uniqueness:1[THEN ≡df E, OF A] &E ∃ E[rotated] by blast
      AOT-have a-3: ‹𝓐[G]a›
        using antecedant ∀ E by blast
      moreover AOT-have ‹a = ιy(y = x & ∃ H (x[H] & ¬[H]x))›
        using nec−hintikka−scheme[THEN ≡E(2), OF &I] a-1 a-2 by auto
      ultimately AOT-show ‹𝓐[G]ιy (y = x & ∃ H (x[H] & ¬[H]x))›
        using rule=E by fast
    qed
    also AOT-have B: ‹... ≡ 𝓐∃ H (x[H] & ¬[H]x)›
    proof (rule ≡I; rule →I)
      AOT-assume A: ‹∃ !y 𝓐(y = x & ∃ H (x[H] & ¬[H]x))›
      AOT-obtain a where ‹𝓐(a = x & ∃ H (x[H] & ¬[H]x))›
        using uniqueness:1[THEN ≡df E, OF A] &E ∃ E[rotated] by blast
      AOT-thus ‹𝓐∃ H (x[H] & ¬[H]x)›
        using Act−Basic:2[THEN ≡E(1), THEN &E(2)] by blast
    next
      AOT-assume ‹𝓐∃ H (x[H] & ¬[H]x)›
      AOT-hence ‹𝓐x = x & 𝓐∃ H (x[H] & ¬[H]x)›
        using id−eq:1 &I RA[2] by blast
      AOT-hence ‹𝓐(x = x & ∃ H (x[H] & ¬[H]x))›
        using act−conj−act:3 Act−Basic:2 ≡E by blast
```

```
      moreover AOT-have ‹∀ z (𝓐(z = x & ∃ H (x[H] & ¬[H]x)) → z = x)›
      proof(safe intro!: GEN →I)
        fix z
        AOT-assume ‹𝓐(z = x & ∃ H (x[H] & ¬[H]x))›
        AOT-hence ‹𝓐(z = x)›
          using Act−Basic:2[THEN ≡E(1), THEN &E(1)] by blast
        AOT-thus ‹z = x›
           by (metis id−act:1 intro−elim:3:b)
      qed
      ultimately AOT-show ‹∃ !y 𝓐(y = x & ∃ H (x[H] & ¬[H]x))›
        using uniqueness:1[THEN ≡df I] &I ∃ I(2) by fast
    qed
    finally AOT-show ‹(𝓐[G]ιy(y = x & ∃ H (x[H] & ¬[H]x)) ≡ 𝓐∃ H (x[H] & ¬[H]x))›.
  qed

  AOT-assume A: ‹[λx [G]ιy (y = x & ∃ H (x[H] & ¬[H]x))]↓›
  AOT-hence ‹𝓐[λx [G]ιy (y = x & ∃ H (x[H] & ¬[H]x))]↓›
    using exist−nec →E nec−imp−act[THEN →E] by blast
  AOT-hence ‹𝓐([λx [G]ιy (y = x & ∃ H (x[H] & ¬[H]x))]↓ &
              ∀ x ([G]ιy(y = x & ∃ H (x[H] & ¬[H]x)) ≡ ∃ H (x[H] & ¬[H]x)))›
    using Lemma Act−Basic:2[THEN ≡E(2)] &I by blast
  moreover AOT-have ‹𝓐([λx [G]ιy (y = x & ∃ H (x[H] & ¬[H]x))]↓ &
              ∀ x ([G]ιy(y = x & ∃ H (x[H] & ¬[H]x)) ≡ ∃ H (x[H] & ¬[H]x)))
      → 𝓐∃ p (p & ¬p)›
  proof (rule logic−actual−nec:2[axiom-inst, THEN ≡E(1)];
         rule RA[2]; rule →I)
    AOT-modally-strict {
      AOT-assume 0: ‹[λx [G]ιy (y = x & ∃ H (x[H] & ¬[H]x))]↓ &
              ∀ x ([G]ιy(y = x & ∃ H (x[H] & ¬[H]x)) ≡ ∃ H (x[H] & ¬[H]x))›
      AOT-have ‹∃ F ∀ x ([F]x ≡ ∃ G (x[G] & ¬[G]x))›
      proof(rule ∃ I(1))
        AOT-show ‹∀ x ([λx [G]ιy (y = x & ∃ H (x[H] & ¬[H]x))]x ≡ ∃ H (x[H] & ¬[H]x))›
        proof(safe intro!: GEN ≡I →I β←C dest!: β→C)
          fix x
          AOT-assume ‹[G]ιy(y = x & ∃ H (x[H] & ¬[H]x))›
          AOT-thus ‹∃ H (x[H] & ¬[H]x)›
            using 0 &E ∀ E(2) ≡E(1) by blast
        next
          fix x
          AOT-assume ‹∃ H (x[H] & ¬[H]x)›
          AOT-thus ‹[G]ιy(y = x & ∃ H (x[H] & ¬[H]x))›
            using 0 &E ∀ E(2) ≡E(2) by blast
        qed(auto intro!: 0[THEN &E(1)] cqt:2)
      next
        AOT-show ‹[λx [G]ιy (y = x & ∃ H (x[H] & ¬[H]x))]↓›
          using 0 &E(1) by blast
      qed
      AOT-thus ‹∃ p (p & ¬p)›
        using block−paradox:2 reductio−aa:1 by blast
    }
  qed
  ultimately AOT-have ‹𝓐∃ p (p & ¬p)›
    using →E by blast
  AOT-hence ‹∃ p 𝓐(p & ¬p)›
    by (metis Act−Basic:10 intro−elim:3:a)
  then AOT-obtain p where ‹𝓐(p & ¬p)›
    using ∃ E[rotated] by blast
  moreover AOT-have ‹¬𝓐(p & ¬p)›
    using non−contradiction[THEN RA[2]]
    by (meson Act−Sub:1 ¬¬I intro−elim:3:d)
  ultimately AOT-show ‹p & ¬p› for p
    by (metis raa−cor:3)
qed
```

AOT-act-theorem *block−paradox2*:*2*:
‹∃ G ¬[λx [G]ιy (y = x & ∃ H (x[H] & ¬[H]x))]↓›
proof(*rule* ∃ *I*(*1*))
AOT-have *0*: ‹[λx ∀ p (p →p)]↓›
by *cqt*:*2*[*lambda*]
moreover AOT-have ‹∀ x [λx ∀ p (p →p)]x›
apply (*rule GEN*)
apply (*rule beta−C−cor*:*2*[*THEN* →*E*, *OF 0*, *THEN* ∀ *E*(*2*), *THEN* ≡*E*(*2*)])
using *if−p−then−p GEN* **by** *fast*
moreover AOT-have ‹∀ G (∀ x [G]x → ¬[λx [G]ιy (y = x & ∃ H (x[H] & ¬[H]x))]↓)›
using *block−paradox2*:*1* ∀ *I* **by** *fast*
ultimately AOT-show ‹¬[λx [λx ∀ p (p →p)]ιy (y = x & ∃ H (x[H] & ¬[H]x))]↓›
using ∀ *E*(*1*) →*E* **by** *blast*
qed(*cqt*:*2*[*lambda*])

AOT-theorem *propositions*: ‹∃ p □(p ≡ φ)›
proof(*rule* ∃ *I*(*1*))
AOT-show ‹□(φ ≡ φ)›
by (*simp add*: *RN oth−class−taut*:*3*:*a*)
next
AOT-show ‹φ↓›
by (*simp add*: *log−prop−prop*:*2*)
qed

AOT-theorem *pos−not−equiv−ne*:*1*:
‹(◇¬∀ x_1...∀ x_n ([F]x_1...x_n ≡ [G]x_1...x_n)) → F ≠ G›
proof (*rule* →*I*)
AOT-assume ‹◇¬∀ x_1...∀ x_n ([F]x_1...x_n ≡ [G]x_1...x_n)›
AOT-hence ‹¬□∀ x_1...∀ x_n ([F]x_1...x_n ≡ [G]x_1...x_n)›
using *KBasic*:*11*[*THEN* ≡*E*(*2*)] **by** *blast*
AOT-hence ‹¬(F = G)›
using *id−rel−nec−equiv*:*1 modus−tollens*:*1* **by** *blast*
AOT-thus ‹F ≠ G›
using =−*infix*[*THEN* $\equiv_{df}$ *I*] **by** *blast*
qed

AOT-theorem *pos−not−equiv−ne*:*2*: ‹(◇¬(φ{F} ≡ φ{G})) → F ≠ G›
proof (*rule* →*I*)
AOT-modally-strict {
AOT-have ‹¬(φ{F} ≡ φ{G}) → ¬(F = G)›
proof (*rule* →*I*; *rule raa−cor*:*2*)
AOT-assume *1*: ‹F = G›
AOT-hence ‹φ{F} → φ{G}›
using *l−identity*[*axiom-inst*, *THEN* →*E*] **by** *blast*
moreover {
AOT-have ‹G = F›
using *1 id-sym* **by** *blast*
AOT-hence ‹φ{G} → φ{F}›
using *l−identity*[*axiom-inst*, *THEN* →*E*] **by** *blast*
}
ultimately AOT-have ‹φ{F} ≡ φ{G}›
using ≡*I* **by** *blast*
moreover AOT-assume ‹¬(φ{F} ≡ φ{G})›
ultimately AOT-show ‹(φ{F} ≡ φ{G}) & ¬(φ{F} ≡ φ{G})›
using &*I* **by** *blast*
qed
}
AOT-hence ‹◇¬(φ{F} ≡ φ{G}) → ◇¬(F = G)›
using *RM*:*2*[*prem*] **by** *blast*
moreover AOT-assume ‹◇¬(φ{F} ≡ φ{G})›
ultimately AOT-have *0*: ‹◇¬(F = G)› **using** →*E* **by** *blast*
AOT-have ‹◇(F ≠ G)›

```
    by (AOT-subst ‹F ≠ G› ‹¬(F = G)›)
       (auto simp: =−infix ≡Df 0)
  AOT-thus ‹F ≠ G›
    using id−nec2:3[THEN →E] by blast
qed

AOT-theorem pos−not−equiv−ne:2[zero]: ‹(◇¬(φ{p} ≡ φ{q})) → p ≠ q›
proof (rule →I)
  AOT-modally-strict {
    AOT-have ‹¬(φ{p} ≡ φ{q}) → ¬(p = q)›
    proof (rule →I; rule raa−cor:2)
      AOT-assume 1: ‹p = q›
      AOT-hence ‹φ{p} → φ{q}›
        using l−identity[axiom-inst, THEN →E] by blast
      moreover {
        AOT-have ‹q = p›
          using 1 id-sym by blast
        AOT-hence ‹φ{q} → φ{p}›
          using l−identity[axiom-inst, THEN →E] by blast
      }
      ultimately AOT-have ‹φ{p} ≡ φ{q}›
        using ≡I by blast
      moreover AOT-assume ‹¬(φ{p} ≡ φ{q})›
      ultimately AOT-show ‹(φ{p} ≡ φ{q}) & ¬(φ{p} ≡ φ{q})›
        using &I by blast
    qed
  }
  AOT-hence ‹◇¬(φ{p} ≡ φ{q}) → ◇¬(p = q)›
    using RM:2[prem] by blast
  moreover AOT-assume ‹◇¬(φ{p} ≡ φ{q})›
  ultimately AOT-have 0: ‹◇¬(p = q)› using →E by blast
  AOT-have ‹◇(p ≠ q)›
    by (AOT-subst ‹p ≠ q› ‹¬(p = q)›)
       (auto simp: 0 =−infix ≡Df)
  AOT-thus ‹p ≠ q›
    using id−nec2:3[THEN →E] by blast
qed

AOT-theorem pos−not−equiv−ne:3:
  ‹(¬∀x1...∀xn ([F]x1...xn ≡ [G]x1...xn)) → F ≠ G›
  using →I pos−not−equiv−ne:1[THEN →E] T◇[THEN →E] by blast

AOT-theorem pos−not−equiv−ne:4: ‹(¬(φ{F} ≡ φ{G})) → F ≠ G›
  using →I pos−not−equiv−ne:2[THEN →E] T◇[THEN →E] by blast

AOT-theorem pos−not−equiv−ne:4[zero]: ‹(¬(φ{p} ≡ φ{q})) → p ≠ q›
  using →I pos−not−equiv−ne:2[zero][THEN →E]
        T◇[THEN →E] by blast

AOT-define relation-negation :: Π ⇒ Π (‹-⁻›)
  df−relation−negation: [F]⁻ =df [λx1...xn ¬[F]x1...xn]

nonterminal φneg
syntax :: φneg ⇒ τ (‹-›)
syntax :: φneg ⇒ φ (‹'(-')›)

AOT-define relation-negation-0 :: ‹φ ⇒ φneg› (‹'(-')⁻›)
  df−relation−negation[zero]: (p)⁻ =df [λ ¬p]

AOT-theorem rel−neg−T:1: ‹[λx1...xn ¬[Π]x1...xn]↓›
  by cqt:2[lambda]

AOT-theorem rel−neg−T:1[zero]: ‹[λ ¬φ]↓›
```

using *cqt:2[lambda0][axiom-inst]* **by** *blast*

AOT-theorem *rel−neg−T:2*: ‹$[\Pi]^- = [\lambda x_1...x_n\ \neg[\Pi]x_1...x_n]$›
using *=I(1)[OF rel−neg−T:1]*
by (*rule* $=_{df}$*I(1)[OF df−relation−negation, OF rel−neg−T:1]*)

AOT-theorem *rel−neg−T:2[zero]*: ‹$(\varphi)^- = [\lambda\ \neg\varphi]$›
using *=I(1)[OF rel−neg−T:1[zero]]*
by (*rule* $=_{df}$*I(1)[OF df−relation−negation[zero], OF rel−neg−T:1[zero]]*)

AOT-theorem *rel−neg−T:3*: ‹$[\Pi]^-\downarrow$›
using $=_{df}$*I(1)[OF df−relation−negation, OF rel−neg−T:1]*
rel−neg−T:1 **by** *blast*

AOT-theorem *rel−neg−T:3[zero]*: ‹$(\varphi)^-\downarrow$›
using *log−prop−prop:2* **by** *blast*

AOT-theorem *thm−relation−negation:1*: ‹$[F]^-x_1...x_n \equiv \neg[F]x_1...x_n$›
proof −
AOT-have ‹$[F]^-x_1...x_n \equiv [\lambda x_1...x_n\ \neg[F]x_1...x_n]x_1...x_n$›
using *rule=E[rotated, OF rel−neg−T:2]*
rule=E[rotated, OF rel−neg−T:2[THEN id-sym]]
→I ≡I **by** *fast*
also AOT-have ‹$... \equiv \neg[F]x_1...x_n$›
using *beta−C−meta[THEN →E, OF rel−neg−T:1]* **by** *fast*
finally show *?thesis*.
qed

AOT-theorem *thm−relation−negation:2*: ‹$\neg[F]^-x_1...x_n \equiv [F]x_1...x_n$›
apply (*AOT-subst* ‹$[F]x_1...x_n$› ‹$\neg\neg[F]x_1...x_n$›)
apply (*simp add: oth−class−taut:3:b*)
apply (*rule oth−class−taut:4:b[THEN ≡E(1)]*)
using *thm−relation−negation:1*.

AOT-theorem *thm−relation−negation:3*: ‹$((p)^-) \equiv \neg p$›
proof −
AOT-have ‹$(p)^- = [\lambda\ \neg p]$› **using** *rel−neg−T:2[zero]* **by** *blast*
AOT-hence ‹$((p)^-) \equiv [\lambda\ \neg p]$›
using *df−relation−negation[zero] log−prop−prop:2*
oth−class−taut:3:a rule−id−df:2:a **by** *blast*
also AOT-have ‹$[\lambda\ \neg p] \equiv \neg p$›
by (*simp add: propositions−lemma:2*)
finally show *?thesis*.
qed

AOT-theorem *thm−relation−negation:4*: ‹$(\neg((p)^-)) \equiv p$›
using *thm−relation−negation:3[THEN ≡E(1)]*
thm−relation−negation:3[THEN ≡E(2)]
≡I →I RAA **by** *metis*

AOT-theorem *thm−relation−negation:5*: ‹$[F] \neq [F]^-$›
proof −
AOT-have ‹$\neg([F] = [F]^-)$›
proof (*rule RAA(2)*)
AOT-show ‹$[F]x_1...x_n \rightarrow [F]x_1...x_n$› **for** x_1x_n
using *if−p−then−p*.
next
AOT-assume ‹$[F] = [F]^-$›
AOT-hence ‹$[F]^- = [F]$› **using** *id-sym* **by** *blast*
AOT-hence ‹$[F]x_1...x_n \equiv \neg[F]x_1...x_n$› **for** x_1x_n
using *rule=E thm−relation−negation:1* **by** *fast*
AOT-thus ‹$\neg([F]x_1...x_n \rightarrow [F]x_1...x_n)$› **for** x_1x_n
using *≡E RAA* **by** *metis*

qed
thus *?thesis*
using $\equiv_{df}I$ *=−infix* **by** *blast*
qed

AOT-theorem *thm−relation−negation:6*: ‹$p \neq (p)^-$›
proof −
AOT-have ‹$\neg(p = (p)^-)$›
proof (*rule RAA(2)*)
AOT-show ‹$p \rightarrow p$›
using *if−p−then−p*.
next
AOT-assume ‹$p = (p)^-$›
AOT-hence ‹$(p)^- = p$› **using** *id-sym* **by** *blast*
AOT-hence ‹$p \equiv \neg p$›
using *rule=E thm−relation−negation:3* **by** *fast*
AOT-thus ‹$\neg(p \rightarrow p)$›
using *≡E RAA* **by** *metis*
qed
thus *?thesis*
using $\equiv_{df}I$ *=−infix* **by** *blast*
qed

AOT-theorem *thm−relation−negation:7*: ‹$(p)^- = (\neg p)$›
apply (*rule df−relation−negation[zero][THEN* $=_{df}E(1)$])
using *cqt:2[lambda0][axiom-inst] rel−neg−T:2[zero]*
propositions−lemma:1 id-trans **by** *blast+*

AOT-theorem *thm−relation−negation:8*: ‹$p = q \rightarrow (\neg p) = (\neg q)$›
proof(*rule →I*)
AOT-assume ‹$p = q$›
moreover AOT-have ‹$(\neg p)\downarrow$› **using** *log−prop−prop:2*.
moreover AOT-have ‹$(\neg p) = (\neg p)$› **using** *calculation(2) =I* **by** *blast*
ultimately AOT-show ‹$(\neg p) = (\neg q)$›
using *rule=E* **by** *fast*
qed

AOT-theorem *thm−relation−negation:9*: ‹$p = q \rightarrow (p)^- = (q)^-$›
proof(*rule →I*)
AOT-assume ‹$p = q$›
AOT-hence ‹$(\neg p) = (\neg q)$› **using** *thm−relation−negation:8 →E* **by** *blast*
AOT-thus ‹$(p)^- = (q)^-$›
using *thm−relation−negation:7 id-sym id-trans* **by** *metis*
qed

AOT-define *Necessary* :: ‹$\Pi \Rightarrow \varphi$› (‹*Necessary'*(-')›)
contingent−properties:1:
‹$Necessary([F]) \equiv_{df} \Box\forall x_1...\forall x_n\ [F]x_1...x_n$›

AOT-define *Necessary0* :: ‹$\varphi \Rightarrow \varphi$› (‹*Necessary0'*(-')›)
contingent−properties:1[zero]:
‹$Necessary0(p) \equiv_{df} \Box p$›

AOT-define *Impossible* :: ‹$\Pi \Rightarrow \varphi$› (‹*Impossible'*(-')›)
contingent−properties:2:
‹$Impossible([F]) \equiv_{df} F\downarrow\ \&\ \Box\forall x_1...\forall x_n\ \neg[F]x_1...x_n$›

AOT-define *Impossible0* :: ‹$\varphi \Rightarrow \varphi$› (‹*Impossible0'*(-')›)
contingent−properties:2[zero]:
‹$Impossible0(p) \equiv_{df} \Box\neg p$›

AOT-define *NonContingent* :: ‹$\Pi \Rightarrow \varphi$› (‹*NonContingent'*(-')›)
contingent−properties:3:

```
‹NonContingent([F]) ≡df Necessary([F]) ∨ Impossible([F])›

AOT-define NonContingent0 :: ‹φ ⇒ φ› (‹NonContingent0'(-')›)
  contingent−properties:3[zero]:
  ‹NonContingent0(p) ≡df Necessary0(p) ∨ Impossible0(p)›

AOT-define Contingent :: ‹Π ⇒ φ› (‹Contingent'(-')›)
  contingent−properties:4:
  ‹Contingent([F]) ≡df F↓ & ¬(Necessary([F]) ∨ Impossible([F]))›

AOT-define Contingent0 :: ‹φ ⇒ φ› (‹Contingent0'(-')›)
  contingent−properties:4[zero]:
  ‹Contingent0(p) ≡df ¬(Necessary0(p) ∨ Impossible0(p))›


AOT-theorem thm−cont−prop:1: ‹NonContingent([F]) ≡ NonContingent([F]⁻)›
proof (rule ≡I; rule →I)
  AOT-assume ‹NonContingent([F])›
  AOT-hence ‹Necessary([F]) ∨ Impossible([F])›
    using ≡df E[OF contingent−properties:3] by blast
  moreover {
    AOT-assume ‹Necessary([F])›
    AOT-hence ‹□(∀x1...∀xn [F]x1...xn)›
      using ≡df E[OF contingent−properties:1] by blast
    moreover AOT-modally-strict {
      AOT-assume ‹∀x1...∀xn [F]x1...xn›
      AOT-hence ‹[F]x1...xn› for x1xn using ∀E by blast
      AOT-hence ‹¬[F]⁻x1...xn› for x1xn
        by (meson ≡E(6) oth−class−taut:3:a
                  thm−relation−negation:2 ≡E(1))
      AOT-hence ‹∀x1...∀xn ¬[F]⁻x1...xn› using ∀I by fast
    }
    ultimately AOT-have ‹□(∀x1...∀xn ¬[F]⁻x1...xn)›
      using RN[prem][where Γ={«∀x1...∀xn [F]x1...xn»}, simplified] by blast
    AOT-hence ‹Impossible([F]⁻)›
      using ≡Df[OF contingent−properties:2, THEN ≡S(1),
                OF rel−neg−T:3, THEN ≡E(2)]
      by blast
  }
  moreover {
    AOT-assume ‹Impossible([F])›
    AOT-hence ‹□(∀x1...∀xn ¬[F]x1...xn)›
      using ≡Df[OF contingent−properties:2, THEN ≡S(1),
                OF cqt:2[const-var][axiom-inst], THEN ≡E(1)]
      by blast
    moreover AOT-modally-strict {
      AOT-assume ‹∀x1...∀xn ¬[F]x1...xn›
      AOT-hence ‹¬[F]x1...xn› for x1xn using ∀E by blast
      AOT-hence ‹[F]⁻x1...xn› for x1xn
        by (meson ≡E(6) oth−class−taut:3:a
                  thm−relation−negation:1 ≡E(1))
      AOT-hence ‹∀x1...∀xn [F]⁻x1...xn› using ∀I by fast
    }
    ultimately AOT-have ‹□(∀x1...∀xn [F]⁻x1...xn)›
      using RN[prem][where Γ={«∀x1...∀xn ¬[F]x1...xn»}] by blast
    AOT-hence ‹Necessary([F]⁻)›
      using ≡df I[OF contingent−properties:1] by blast
  }
  ultimately AOT-have ‹Necessary([F]⁻) ∨ Impossible([F]⁻)›
    using ∨E(1) ∨I →I by metis
  AOT-thus ‹NonContingent([F]⁻)›
    using ≡df I[OF contingent−properties:3] by blast
next
```

```
  AOT-assume ‹NonContingent([F]⁻)›
  AOT-hence ‹Necessary([F]⁻) ∨ Impossible([F]⁻)›
    using ≡dfE[OF contingent−properties:3] by blast
  moreover {
    AOT-assume ‹Necessary([F]⁻)›
    AOT-hence ‹□(∀x1...∀xn [F]⁻x1...xn)›
      using ≡dfE[OF contingent−properties:1] by blast
    moreover AOT-modally-strict {
      AOT-assume ‹∀x1...∀xn [F]⁻x1...xn›
      AOT-hence ‹[F]⁻x1...xn› for x1xn using ∀E by blast
      AOT-hence ‹¬[F]x1...xn› for x1xn
        by (meson ≡E(6) oth−class−taut:3:a
                   thm−relation−negation:1 ≡E(2))
      AOT-hence ‹∀x1...∀xn ¬[F]x1...xn› using ∀I by fast
    }
    ultimately AOT-have ‹□∀x1...∀xn ¬[F]x1...xn›
      using RN[prem][where Γ={«∀x1...∀xn [F]⁻x1...xn»}] by blast
    AOT-hence ‹Impossible([F])›
      using ≡Df[OF contingent−properties:2, THEN ≡S(1),
                OF cqt:2[const-var][axiom-inst], THEN ≡E(2)]
      by blast
  }
  moreover {
    AOT-assume ‹Impossible([F]⁻)›
    AOT-hence ‹□(∀x1...∀xn ¬[F]⁻x1...xn)›
      using ≡Df[OF contingent−properties:2, THEN ≡S(1),
                OF rel−neg−T:3, THEN ≡E(1)]
      by blast
    moreover AOT-modally-strict {
      AOT-assume ‹∀x1...∀xn ¬[F]⁻x1...xn›
      AOT-hence ‹¬[F]⁻x1...xn› for x1xn using ∀E by blast
      AOT-hence ‹[F]x1...xn› for x1xn
        using thm−relation−negation:1[THEN
              oth−class−taut:4:b[THEN ≡E(1)], THEN ≡E(1)]
            useful−tautologies:1[THEN →E] by blast
      AOT-hence ‹∀x1...∀xn [F]x1...xn› using ∀I by fast
    }
    ultimately AOT-have ‹□(∀x1...∀xn [F]x1...xn)›
      using RN[prem][where Γ={«∀x1...∀xn ¬[F]⁻x1...xn»}] by blast
    AOT-hence ‹Necessary([F])›
      using ≡dfI[OF contingent−properties:1] by blast
  }
  ultimately AOT-have ‹Necessary([F]) ∨ Impossible([F])›
    using ∨E(1) ∨I →I by metis
  AOT-thus ‹NonContingent([F])›
    using ≡dfI[OF contingent−properties:3] by blast
qed

AOT-theorem thm−cont−prop:2: ‹Contingent([F]) ≡ ◇∃x [F]x & ◇∃x ¬[F]x›
proof −
  AOT-have ‹Contingent([F]) ≡ ¬(Necessary([F]) ∨ Impossible([F]))›
    using contingent−properties:4[THEN ≡Df, THEN ≡S(1),
                                 OF cqt:2[const-var][axiom-inst]]
    by blast
  also AOT-have ‹... ≡ ¬Necessary([F]) & ¬Impossible([F])›
    using oth−class−taut:5:d by fastforce
  also AOT-have ‹... ≡ ¬Impossible([F]) & ¬Necessary([F])›
    by (simp add: Commutativity of &)
  also AOT-have ‹... ≡ ◇∃x [F]x & ¬Necessary([F])›
  proof (rule oth−class−taut:4:e[THEN →E])
    AOT-have ‹¬Impossible([F]) ≡ ¬□¬ ∃x [F]x›
      apply (rule oth−class−taut:4:b[THEN ≡E(1)])
      apply (AOT-subst ‹∃x [F]x› ‹¬ ∀x ¬[F]x›)
```

```
      apply (simp add: conventions:4 ≡Df)
     apply (AOT-subst (reverse) ‹¬¬∀ x ¬[F]x› ‹∀ x ¬[F]x›)
      apply (simp add: oth−class−taut:3:b)
     using contingent−properties:2[THEN ≡Df, THEN ≡S(1),
                             OF cqt:2[const-var][axiom-inst]]
     by blast
   also AOT-have ‹... ≡ ◊∃ x [F]x›
     using conventions:5[THEN ≡Df, symmetric] by blast
   finally AOT-show ‹¬Impossible([F]) ≡ ◊∃ x [F]x› .
  qed
  also AOT-have ‹... ≡ ◊∃ x [F]x & ◊∃ x ¬[F]x›
  proof (rule oth−class−taut:4:f[THEN →E])
   AOT-have ‹¬Necessary([F]) ≡ ¬□¬∃ x ¬[F]x›
     apply (rule oth−class−taut:4:b[THEN ≡E(1)])
     apply (AOT-subst ‹∃ x ¬[F]x› ‹¬ ∀ x ¬¬[F]x›)
      apply (simp add: conventions:4 ≡Df)
     apply (AOT-subst (reverse) ‹¬¬[F]x› ‹[F]x› for: x)
      apply (simp add: oth−class−taut:3:b)
     apply (AOT-subst (reverse) ‹¬¬∀ x [F]x› ‹∀ x [F]x›)
     by (auto simp: oth−class−taut:3:b contingent−properties:1 ≡Df)
   also AOT-have ‹... ≡ ◊∃ x ¬[F]x›
     using conventions:5[THEN ≡Df, symmetric] by blast
   finally AOT-show ‹¬Necessary([F]) ≡ ◊∃ x ¬[F]x›.
  qed
  finally show ?thesis.
qed

AOT-theorem thm−cont−prop:3:
  ‹Contingent([F]) ≡ Contingent([F]⁻)› for F::‹<κ> AOT-var›
proof −
  {
   fix Π :: ‹<κ>›
   AOT-assume ‹Π↓›
   moreover AOT-have ‹∀ F (Contingent([F]) ≡ ◊∃ x [F]x & ◊∃ x ¬[F]x)›
     using thm−cont−prop:2 GEN by fast
   ultimately  AOT-have ‹Contingent([Π]) ≡ ◊∃ x [Π]x & ◊∃ x ¬[Π]x›
     using thm−cont−prop:2 ∀ E by fast
  } note 1 = this
  AOT-have ‹Contingent([F]) ≡ ◊∃ x [F]x & ◊∃ x ¬[F]x›
   using thm−cont−prop:2 by blast
  also AOT-have ‹... ≡ ◊∃ x ¬[F]x & ◊∃ x [F]x›
   by (simp add: Commutativity of &)
  also AOT-have ‹... ≡ ◊∃ x [F]⁻x & ◊∃ x [F]x›
   by (AOT-subst ‹[F]⁻x› ‹¬[F]x› for: x)
      (auto simp: thm−relation−negation:1 oth−class−taut:3:a)
  also AOT-have ‹... ≡ ◊∃ x [F]⁻x & ◊∃ x ¬[F]⁻x›
   by (AOT-subst (reverse) ‹[F]x› ‹¬[F]⁻x› for: x)
      (auto simp: thm−relation−negation:2 oth−class−taut:3:a)
  also AOT-have ‹... ≡ Contingent([F]⁻)›
   using 1[OF rel−neg−T:3, symmetric] by blast
  finally show ?thesis.
qed

AOT-define concrete-if-concrete :: ‹Π› (‹L›)
  L-def: ‹L =df [λx E!x → E!x]›

AOT-theorem thm−noncont−e−e:1: ‹Necessary(L)›
proof −
  AOT-modally-strict {
   fix x
   AOT-have ‹[λx E!x → E!x]↓› by cqt:2[lambda]
   moreover AOT-have ‹x↓› using cqt:2[const-var][axiom-inst] by blast
   moreover AOT-have ‹E!x → E!x› using if−p−then−p by blast
```

ultimately AOT-have ‹[λx E!x → E!x]x›
using *β←C* **by** *blast*
}
AOT-hence *0*: ‹□∀ x [λx E!x → E!x]x›
using *RN GEN* **by** *blast*
show *?thesis*
apply (*rule* $=_{df}$*I*(*2*)[*OF L-def*])
apply *cqt*:*2*[*lambda*]
by (*rule contingent−properties*:*1*[*THEN* $\equiv_{df}$*I*, *OF 0*])
qed

AOT-theorem *thm−noncont−e−e*:*2*: ‹*Impossible*([*L*]⁻)›
proof −
AOT-modally-strict {
fix *x*

AOT-have *0*: ‹∀ *F* (¬[*F*]⁻*x* ≡ [*F*]*x*)›
using *thm−relation−negation*:*2 GEN* **by** *fast*
AOT-have ‹¬[λx E!x → E!x]⁻x ≡ [λx E!x → E!x]x›
by (*rule 0*[*THEN* ∀ *E*(*1*)]) *cqt*:*2*[*lambda*]
moreover {
AOT-have ‹[λx E!x → E!x]↓› **by** *cqt*:*2*[*lambda*]
moreover AOT-have ‹*x*↓› **using** *cqt*:*2*[*const-var*][*axiom-inst*] **by** *blast*
moreover AOT-have ‹*E*!*x* → *E*!*x*› **using** *if−p−then−p* **by** *blast*
ultimately AOT-have ‹[λx E!x → E!x]x›
using *β←C* **by** *blast*
}
ultimately AOT-have ‹¬[λx E!x → E!x]⁻*x*›
using ≡*E* **by** *blast*
}
AOT-hence *0*: ‹□∀ x ¬[λx E!x → E!x]⁻x›
using *RN GEN* **by** *fast*
show *?thesis*
apply (*rule* $=_{df}$*I*(*2*)[*OF L-def*])
apply *cqt*:*2*[*lambda*]
apply (*rule contingent−properties*:*2*[*THEN* $\equiv_{df}$*I*]; *rule* &*I*)
using *rel−neg−T*:*3*
apply *blast*
using *0*
by *blast*
qed

AOT-theorem *thm−noncont−e−e*:*3*: ‹*NonContingent*(*L*)›
using *thm−noncont−e−e*:*1*
by (*rule contingent−properties*:*3*[*THEN* $\equiv_{df}$*I*, *OF* ∨*I*(*1*)])

AOT-theorem *thm−noncont−e−e*:*4*: ‹*NonContingent*([*L*]⁻)›
proof −
AOT-have *0*: ‹∀ *F* (*NonContingent*([*F*]) ≡ *NonContingent*([*F*]⁻))›
using *thm−cont−prop*:*1* ∀ *I* **by** *fast*
moreover AOT-have *1*: ‹*L*↓›
by (*rule* $=_{df}$*I*(*2*)[*OF L-def*]) *cqt*:*2*[*lambda*]+
AOT-show ‹*NonContingent*([*L*]⁻)›
using ∀ *E*(*1*)[*OF 0*, *OF 1*, *THEN* ≡*E*(*1*), *OF thm−noncont−e−e*:*3*] **by** *blast*
qed

AOT-theorem *thm−noncont−e−e*:*5*:
‹∃ *F* ∃ *G* (*F* ≠ «*G*::<κ>» & *NonContingent*([*F*]) & *NonContingent*([*G*]))›
proof (*rule* ∃ *I*)+
{
AOT-have ‹∀ *F* [*F*] ≠ [*F*]⁻›
using *thm−relation−negation*:*5 GEN* **by** *fast*
moreover AOT-have ‹*L*↓›

by (*rule* $=_{df}$*I*(*2*)[*OF L-def*]) *cqt*:*2*[*lambda*]+
ultimately AOT-have ‹$L \neq [L]^-$›
using ∀*E* **by** *blast*
}
AOT-thus ‹$L \neq [L]^-$ & *NonContingent*(L) & *NonContingent*($[L]^-$)›
using *thm−noncont−e−e*:*3* *thm−noncont−e−e*:*4* &*I* **by** *metis*
next
AOT-show ‹$[L]^-\downarrow$›
using *rel−neg−T*:*3* **by** *blast*
next
AOT-show ‹$L\downarrow$›
by (*rule* $=_{df}$*I*(*2*)[*OF L-def*]) *cqt*:*2*[*lambda*]+
qed

AOT-theorem *lem−cont−e*:*1*: ‹$\Diamond\exists x$ ($[F]x$ & $\Diamond\neg[F]x$) $\equiv$ $\Diamond\exists x$ ($\neg[F]x$ & $\Diamond[F]x$)›
proof −
AOT-have ‹$\Diamond\exists x$ ($[F]x$ & $\Diamond\neg[F]x$) $\equiv$ $\exists x$ $\Diamond$($[F]x$ & $\Diamond\neg[F]x$)›
using *BF*◇ *CBF*◇ ≡*I* **by** *blast*
also AOT-have ‹... $\equiv$ $\exists x$ ($\Diamond[F]x$ & $\Diamond\neg[F]x$)›
by (*AOT-subst* ‹$\Diamond$($[F]x$ & $\Diamond\neg[F]x$)› ‹$\Diamond[F]x$ & $\Diamond\neg[F]x$› **for**: *x*)
(*auto simp*: *S5Basic*:*11* *cqt−further*:*7*)
also AOT-have ‹... $\equiv$ $\exists x$ ($\Diamond\neg[F]x$ & $\Diamond[F]x$)›
by (*AOT-subst* ‹$\Diamond\neg[F]x$ & $\Diamond[F]x$› ‹$\Diamond[F]x$ & $\Diamond\neg[F]x$› **for**: *x*)
(*auto simp*: *Commutativity of* & *cqt−further*:*7*)
also AOT-have ‹... $\equiv$ $\exists x$ $\Diamond$($\neg[F]x$ & $\Diamond[F]x$)›
by (*AOT-subst* ‹$\Diamond$($\neg[F]x$ & $\Diamond[F]x$)› ‹$\Diamond\neg[F]x$ & $\Diamond[F]x$› **for**: *x*)
(*auto simp*: *S5Basic*:*11* *oth−class−taut*:*3*:*a*)
also AOT-have ‹... $\equiv$ $\Diamond\exists x$ ($\neg[F]x$ & $\Diamond[F]x$)›
using *BF*◇ *CBF*◇ ≡*I* **by** *fast*
finally show *?thesis*.
qed

AOT-theorem *lem−cont−e*:*2*:
‹$\Diamond\exists x$ ($[F]x$ & $\Diamond\neg[F]x$) $\equiv$ $\Diamond\exists x$ ($[F]^-x$ & $\Diamond\neg[F]^-x$)›
proof −
AOT-have ‹$\Diamond\exists x$ ($[F]x$ & $\Diamond\neg[F]x$) $\equiv$ $\Diamond\exists x$ ($\neg[F]x$ & $\Diamond[F]x$)›
using *lem−cont−e*:*1*.
also AOT-have ‹... $\equiv$ $\Diamond\exists x$ ($[F]^-x$ & $\Diamond\neg[F]^-x$)›
apply (*AOT-subst* ‹$\neg[F]^-x$› ‹$[F]x$› **for**: *x*)
apply (*simp add*: *thm−relation−negation*:*2*)
apply (*AOT-subst* ‹$[F]^-x$› ‹$\neg[F]x$› **for**: *x*)
apply (*simp add*: *thm−relation−negation*:*1*)
by (*simp add*: *oth−class−taut*:*3*:*a*)
finally show *?thesis*.
qed

AOT-theorem *thm−cont−e*:*1*: ‹$\Diamond\exists x$ ($E!x$ & $\Diamond\neg E!x$)›
proof (*rule CBF*◇[*THEN* →*E*])
AOT-have ‹$\exists x$ $\Diamond$($E!x$ & $\neg\mathcal{A}E!x$)›
using *qml*:*4*[*axiom-inst*] *BF*◇[*THEN* →*E*] **by** *blast*
then AOT-obtain *a* **where** ‹$\Diamond$($E!a$ & $\neg\mathcal{A}E!a$)›
using ∃*E*[*rotated*] **by** *blast*
AOT-hence ϑ: ‹$\Diamond E!a$ & $\Diamond\neg\mathcal{A}E!a$›
using *KBasic2*:*3*[*THEN* →*E*] **by** *blast*
AOT-have ξ: ‹$\Diamond E!a$ & $\Diamond\mathcal{A}\neg E!a$›
by (*AOT-subst* ‹$\mathcal{A}\neg E!a$› ‹$\neg\mathcal{A}E!a$›)
(*auto simp*: *logic−actual−nec*:*1*[*axiom-inst*] ϑ)
AOT-have ζ: ‹$\Diamond E!a$ & $\mathcal{A}\neg E!a$›
by (*AOT-subst* ‹$\mathcal{A}\neg E!a$› ‹$\Diamond\mathcal{A}\neg E!a$›)
(*auto simp add*: *Act−Sub*:*4* ξ)
AOT-hence ‹$\Diamond E!a$ & $\Diamond\neg E!a$›
using &*E* &*I* *Act−Sub*:*3*[*THEN* →*E*] **by** *blast*
AOT-hence ‹$\Diamond$($E!a$ & $\Diamond\neg E!a$)›

using *S5Basic:11[THEN ≡E(2)]* **by** *simp*
AOT-thus ‹∃ *x* ◇(*E*!*x* & ◇¬*E*!*x*)›
using ∃ *I(2)* **by** *fast*
qed

AOT-theorem *thm−cont−e:2*: ‹◇∃ *x* (¬*E*!*x* & ◇*E*!*x*)›
proof −
AOT-have ‹∀ *F* (◇∃ *x* ([*F*]*x* & ◇¬[*F*]*x*) ≡ ◇∃ *x* (¬[*F*]*x* & ◇[*F*]*x*))›
using *lem−cont−e:1 GEN* **by** *fast*
AOT-hence ‹(◇∃ *x* (*E*!*x* & ◇¬*E*!*x*) ≡ ◇∃ *x* (¬*E*!*x* & ◇*E*!*x*))›
using ∀ *E(2)* **by** *blast*
thus *?thesis* **using** *thm−cont−e:1 ≡E* **by** *blast*
qed

AOT-theorem *thm−cont−e:3*: ‹◇∃ *x E*!*x*›
proof (*rule CBF◇[THEN →E]*)
AOT-obtain *a* **where** ‹◇(*E*!*a* & ◇¬*E*!*a*)›
using ∃ *E[rotated, OF thm−cont−e:1[THEN BF◇[THEN →E]]]* **by** *blast*
AOT-hence ‹◇*E*!*a*›
using *KBasic2:3[THEN →E, THEN &E(1)]* **by** *blast*
AOT-thus ‹∃ *x* ◇*E*!*x*› **using** ∃ *I* **by** *fast*
qed

AOT-theorem *thm−cont−e:4*: ‹◇∃ *x* ¬*E*!*x*›
proof (*rule CBF◇[THEN →E]*)
AOT-obtain *a* **where** ‹◇(*E*!*a* & ◇¬*E*!*a*)›
using ∃ *E[rotated, OF thm−cont−e:1[THEN BF◇[THEN →E]]]* **by** *blast*
AOT-hence ‹◇◇¬*E*!*a*›
using *KBasic2:3[THEN →E, THEN &E(2)]* **by** *blast*
AOT-hence ‹◇¬*E*!*a*›
using *4◇[THEN →E]* **by** *blast*
AOT-thus ‹∃ *x* ◇¬*E*!*x*› **using** ∃ *I* **by** *fast*
qed

AOT-theorem *thm−cont−e:5*: ‹*Contingent*([*E*!])›
proof −
AOT-have ‹∀ *F* (*Contingent*([*F*]) ≡ ◇∃ *x* [*F*]*x* & ◇∃ *x* ¬[*F*]*x*)›
using *thm−cont−prop:2 GEN* **by** *fast*
AOT-hence ‹*Contingent*([*E*!]) ≡ ◇∃ *x E*!*x* & ◇∃ *x* ¬*E*!*x*›
using ∀ *E(2)* **by** *blast*
thus *?thesis*
using *thm−cont−e:3 thm−cont−e:4 ≡E(2) &I* **by** *blast*
qed

AOT-theorem *thm−cont−e:6*: ‹*Contingent*([*E*!]$^-$)›
proof −
AOT-have ‹∀ *F* (*Contingent*([«*F*::<κ>»]) ≡ *Contingent*([*F*]$^-$))›
using *thm−cont−prop:3 GEN* **by** *fast*
AOT-hence ‹*Contingent*([*E*!]) ≡ *Contingent*([*E*!]$^-$)›
using ∀ *E(2)* **by** *fast*
thus *?thesis* **using** *thm−cont−e:5 ≡E* **by** *blast*
qed

AOT-theorem *thm−cont−e:7*:
‹∃ *F*∃ *G* (*Contingent*([«*F*::<κ>»]) & *Contingent*([*G*]) & *F* ≠ *G*)›
proof (*rule* ∃ *I*)+
AOT-have ‹∀ *F* [«*F*::<κ>»] ≠ [*F*]$^-$›
using *thm−relation−negation:5 GEN* **by** *fast*
AOT-hence ‹[*E*!] ≠ [*E*!]$^-$›
using ∀ *E* **by** *fast*
AOT-thus ‹*Contingent*([*E*!]) & *Contingent*([*E*!]$^-$) & [*E*!] ≠ [*E*!]$^-$›
using *thm−cont−e:5 thm−cont−e:6 &I* **by** *metis*
next

```
  AOT-show ‹E!⁻↓›
    by (fact AOT)
qed(cqt:2)

AOT-theorem property−facts:1:
  ‹NonContingent([F]) → ¬∃ G (Contingent([G]) & G = F)›
proof (rule →I; rule raa−cor:2)
  AOT-assume ‹NonContingent([F])›
  AOT-hence 1: ‹Necessary([F]) ∨ Impossible([F])›
    using contingent−properties:3[THEN ≡df E] by blast
  AOT-assume ‹∃ G (Contingent([G]) & G = F)›
  then AOT-obtain G where ‹Contingent([G]) & G = F›
    using ∃ E[rotated] by blast
  AOT-hence ‹Contingent([F])› using rule=E &E by blast
  AOT-hence ‹¬(Necessary([F]) ∨ Impossible([F]))›
    using contingent−properties:4[THEN ≡Df, THEN ≡S(1),
            OF cqt:2[const-var][axiom-inst], THEN ≡E(1)] by blast
  AOT-thus ‹(Necessary([F]) ∨ Impossible([F])) &
            ¬(Necessary([F]) ∨ Impossible([F]))›
    using 1 &I by blast
qed

AOT-theorem property−facts:2:
  ‹Contingent([F]) → ¬∃ G (NonContingent([G]) & G = F)›
proof (rule →I; rule raa−cor:2)
  AOT-assume ‹Contingent([F])›
  AOT-hence 1: ‹¬(Necessary([F]) ∨ Impossible([F]))›
    using contingent−properties:4[THEN ≡Df, THEN ≡S(1),
            OF cqt:2[const-var][axiom-inst], THEN ≡E(1)] by blast
  AOT-assume ‹∃ G (NonContingent([G]) & G = F)›
  then AOT-obtain G where ‹NonContingent([G]) & G = F›
    using ∃ E[rotated] by blast
  AOT-hence ‹NonContingent([F])›
    using rule=E &E by blast
  AOT-hence ‹Necessary([F]) ∨ Impossible([F])›
    using contingent−properties:3[THEN ≡df E] by blast
  AOT-thus ‹(Necessary([F]) ∨ Impossible([F])) &
            ¬(Necessary([F]) ∨ Impossible([F]))›
    using 1 &I by blast
qed

AOT-theorem property−facts:3:
  ‹L ≠ [L]⁻ & L ≠ E! & L ≠ E!⁻ & [L]⁻ ≠ [E!]⁻ & E! ≠ [E!]⁻›
proof −
  AOT-have noneqI: ‹Π ≠ Π′› if ‹φ{Π}› and ‹¬φ{Π′}› for φ and Π Π′ :: ‹<κ>›
    apply (rule =−infix[THEN ≡df I]; rule raa−cor:2)
    using rule=E[where φ=φ and τ=Π and σ = Π′] that &I by blast
  AOT-have contingent-denotes: ‹Π↓› if ‹Contingent([Π])› for Π :: ‹<κ>›
    using that contingent−properties:4[THEN ≡df E, THEN &E(1)] by blast
  AOT-have not-noncontingent-if-contingent:
    ‹¬NonContingent([Π])› if ‹Contingent([Π])› for Π :: ‹<κ>›
  proof(rule RAA(2))
    AOT-show ‹¬(Necessary([Π]) ∨ Impossible([Π]))›
      using that contingent−properties:4[THEN ≡Df, THEN ≡S(1),
                  OF contingent-denotes[OF that], THEN ≡E(1)]
      by blast
  next
    AOT-assume ‹NonContingent([Π])›
    AOT-thus ‹Necessary([Π]) ∨ Impossible([Π])›
      using contingent−properties:3[THEN ≡df E] by blast
  qed

  show ?thesis
```

proof (*safe intro*!: &*I*)
  AOT-show ‹$L \neq [L]^-$›
    apply (*rule* $=_{df}I(2)$[*OF L-def*])
     apply *cqt*:*2*[*lambda*]
    apply (*rule* ∀ *E*(*1*)[**where** φ=λ Π . «Π ≠ [Π]$^-$»])
     apply (*rule GEN*) **apply** (*fact AOT*)
    by *cqt*:*2*[*lambda*]
next
  AOT-show ‹$L \neq E!$›
    apply (*rule noneqI*)
    using *thm−noncont−e−e*:*3*
        not-noncontingent-if-contingent[*OF thm−cont−e*:*5*]
    by *auto*
next
  AOT-show ‹$L \neq E!^-$›
    apply (*rule noneqI*)
    using *thm−noncont−e−e*:*3* **apply** *fast*
    apply (*rule not-noncontingent-if-contingent*)
    apply (*rule* ∀ *E*(*1*)[
        where φ=λ Π . «*Contingent*([Π]) ≡ *Contingent*([Π]$^-$)»,
        rotated, *OF contingent-denotes*, *THEN* ≡*E*(*1*), *rotated*])
    using *thm−cont−prop*:*3 GEN* **apply** *fast*
    using *thm−cont−e*:*5* **by** *fast*+
next
  AOT-show ‹$[L]^- \neq E!^-$›
    apply (*rule noneqI*)
    using *thm−noncont−e−e*:*4* **apply** *fast*
    apply (*rule not-noncontingent-if-contingent*)
    apply (*rule* ∀ *E*(*1*)[
        where φ=λ Π . «*Contingent*([Π]) ≡ *Contingent*([Π]$^-$)»,
        rotated, *OF contingent-denotes*, *THEN* ≡*E*(*1*), *rotated*])
    using *thm−cont−prop*:*3 GEN* **apply** *fast*
    using *thm−cont−e*:*5* **by** *fast*+
next
  AOT-show ‹$E! \neq E!^-$›
    apply (*rule* $=_{df}I(2)$[*OF L-def*])
     apply *cqt*:*2*[*lambda*]
    apply (*rule* ∀ *E*(*1*)[**where** φ=λ Π . «Π ≠ [Π]$^-$»])
     apply (*rule GEN*) **apply** (*fact AOT*)
    by *cqt*:*2*
qed
qed

AOT-theorem *thm−cont−propos*:*1*:
  ‹*NonContingent0*(*p*) ≡ *NonContingent0*(((p)$^-$))›
proof(*rule* ≡*I*; *rule* →*I*)
  AOT-assume ‹*NonContingent0*(*p*)›
  AOT-hence ‹*Necessary0*(*p*) ∨ *Impossible0*(*p*)›
    using *contingent−properties*:*3*[*zero*][*THEN* $\equiv_{df}E$] **by** *blast*
  moreover {
    AOT-assume ‹*Necessary0*(*p*)›
    AOT-hence *1*: ‹□*p*›
      using *contingent−properties*:*1*[*zero*][*THEN* $\equiv_{df}E$] **by** *blast*
    AOT-have ‹□¬((p)$^-$)›
      by (*AOT-subst* ‹¬((p)$^-$)› ‹*p*›)
        (*auto simp add*: *1 thm−relation−negation*:*4*)
    AOT-hence ‹*Impossible0*(((p)$^-$))›
      by (*rule contingent−properties*:*2*[*zero*][*THEN* $\equiv_{df}I$])
  }
  moreover {
    AOT-assume ‹*Impossible0*(*p*)›
    AOT-hence *1*: ‹□¬*p*›
      by (*rule contingent−properties*:*2*[*zero*][*THEN* $\equiv_{df}E$])

AOT-have ‹□$((p)^{-})$›
by (*AOT-subst* ‹$((p)^{-})$› ‹¬*p*›)
(*auto simp*: *1 thm−relation−negation*:*3*)
AOT-hence ‹*Necessary0*$(((p)^{-}))$›
by (*rule contingent−properties*:*1*[*zero*][*THEN* $\equiv_{df}$*I*])
}
ultimately AOT-have ‹*Necessary0*$(((p)^{-}))$ ∨ *Impossible0*$(((p)^{-}))$›
using ∨*E*(*1*) ∨*I* →*I* **by** *metis*
AOT-thus ‹*NonContingent0*$(((p)^{-}))$›
using *contingent−properties*:*3*[*zero*][*THEN* $\equiv_{df}$*I*] **by** *blast*
next
AOT-assume ‹*NonContingent0*$(((p)^{-}))$›
AOT-hence ‹*Necessary0*$(((p)^{-}))$ ∨ *Impossible0*$(((p)^{-}))$›
using *contingent−properties*:*3*[*zero*][*THEN* $\equiv_{df}$*E*] **by** *blast*
moreover {
AOT-assume ‹*Impossible0*$(((p)^{-}))$›
AOT-hence *1*: ‹□¬$((p)^{-})$›
by (*rule contingent−properties*:*2*[*zero*][*THEN* $\equiv_{df}$*E*])
AOT-have ‹□*p*›
by (*AOT-subst* (*reverse*) ‹*p*› ‹¬$((p)^{-})$›)
(*auto simp*: *1 thm−relation−negation*:*4*)
AOT-hence ‹*Necessary0*(*p*)›
using *contingent−properties*:*1*[*zero*][*THEN* $\equiv_{df}$*I*] **by** *blast*
}
moreover {
AOT-assume ‹*Necessary0*$(((p)^{-}))$›
AOT-hence *1*: ‹□$((p)^{-})$›
by (*rule contingent−properties*:*1*[*zero*][*THEN* $\equiv_{df}$*E*])
AOT-have ‹□¬*p*›
by (*AOT-subst* (*reverse*) ‹¬*p*› ‹$((p)^{-})$›)
(*auto simp*: *1 thm−relation−negation*:*3*)
AOT-hence ‹*Impossible0*(*p*)›
by (*rule contingent−properties*:*2*[*zero*][*THEN* $\equiv_{df}$*I*])
}
ultimately AOT-have ‹*Necessary0*(*p*) ∨ *Impossible0*(*p*)›
using ∨*E*(*1*) ∨*I* →*I* **by** *metis*
AOT-thus ‹*NonContingent0*(*p*)›
using *contingent−properties*:*3*[*zero*][*THEN* $\equiv_{df}$*I*] **by** *blast*
qed

AOT-theorem *thm−cont−propos*:*2*: ‹*Contingent0*(φ) ≡ ◊φ & ◊¬φ›
proof −
AOT-have ‹*Contingent0*(φ) ≡ ¬(*Necessary0*(φ) ∨ *Impossible0*(φ))›
using *contingent−properties*:*4*[*zero*][*THEN* ≡*Df*] **by** *simp*
also AOT-have ‹. . . ≡ ¬*Necessary0*(φ) & ¬*Impossible0*(φ)›
by (*fact AOT*)
also AOT-have ‹. . . ≡ ¬*Impossible0*(φ) & ¬*Necessary0*(φ)›
by (*fact AOT*)
also AOT-have ‹. . . ≡ ◊φ & ◊¬φ›
apply (*AOT-subst* ‹◊φ› ‹¬□¬φ›)
apply (*simp add*: *conventions*:*5* ≡*Df*)
apply (*AOT-subst* ‹*Impossible0*(φ)› ‹□¬φ›)
apply (*simp add*: *contingent−properties*:*2*[*zero*] ≡*Df*)
apply (*AOT-subst* (*reverse*) ‹◊¬φ› ‹¬□φ›)
apply (*simp add*: *KBasic*:*11*)
apply (*AOT-subst* ‹*Necessary0*(φ)› ‹□φ›)
apply (*simp add*: *contingent−properties*:*1*[*zero*] ≡*Df*)
by (*simp add*: *oth−class−taut*:*3*:*a*)
finally show *?thesis***.**
qed

AOT-theorem *thm−cont−propos*:*3*: ‹*Contingent0*(*p*) ≡ *Contingent0*$(((p)^{-}))$›
proof −

```
  AOT-have ‹Contingent0(p) ≡ ◇p & ◇¬p› using thm-cont-propos:2.
  also AOT-have ‹... ≡ ◇¬p & ◇p› by (fact AOT)
  also AOT-have ‹... ≡ ◇((p)⁻) & ◇p›
    by (AOT-subst ‹((p)⁻)› ‹¬p›)
       (auto simp: thm-relation-negation:3 oth-class-taut:3:a)
  also AOT-have ‹... ≡ ◇((p)⁻) & ◇¬((p)⁻)›
    by (AOT-subst ‹¬((p)⁻)› ‹p›)
       (auto simp: thm-relation-negation:4 oth-class-taut:3:a)
  also AOT-have ‹... ≡ Contingent0(((p)⁻))›
    using thm-cont-propos:2[symmetric] by blast
  finally show ?thesis.
qed

AOT-define noncontingent-prop :: ‹φ› (‹p_0›)
  p_0-def: (p_0) =_df (∀x (E!x → E!x))

AOT-theorem thm-noncont-propos:1: ‹Necessary0((p_0))›
proof(rule contingent-properties:1[zero][THEN ≡_df I])
  AOT-show ‹□(p_0)›
    apply (rule =_df I(2)[OF p_0-def])
    using log-prop-prop:2 apply simp
    using if-p-then-p RN GEN by fast
qed

AOT-theorem thm-noncont-propos:2: ‹Impossible0(((p_0)⁻))›
proof(rule contingent-properties:2[zero][THEN ≡_df I])
  AOT-show ‹□¬((p_0)⁻)›
    apply (AOT-subst ‹((p_0)⁻)› ‹¬p_0›)
    using thm-relation-negation:3 GEN ∀E(1)[rotated, OF log-prop-prop:2]
     apply fast
    apply (AOT-subst (reverse) ‹¬¬p_0› ‹p_0›)
     apply (simp add: oth-class-taut:3:b)
    apply (rule =_df I(2)[OF p_0-def])
    using log-prop-prop:2 apply simp
    using if-p-then-p RN GEN by fast
qed

AOT-theorem thm-noncont-propos:3: ‹NonContingent0((p_0))›
  apply(rule contingent-properties:3[zero][THEN ≡_df I])
  using thm-noncont-propos:1 ∨I by blast

AOT-theorem thm-noncont-propos:4: ‹NonContingent0(((p_0)⁻))›
  apply(rule contingent-properties:3[zero][THEN ≡_df I])
  using thm-noncont-propos:2 ∨I by blast

AOT-theorem thm-noncont-propos:5:
  ‹∃p∃q (NonContingent0((p)) & NonContingent0((q)) & p ≠ q)›
proof(rule ∃I)+
  AOT-have 0: ‹φ ≠ (φ)⁻› for φ
    using thm-relation-negation:6 ∀I
          ∀E(1)[rotated, OF log-prop-prop:2] by fast
  AOT-thus ‹NonContingent0((p_0)) & NonContingent0(((p_0)⁻)) & (p_0) ≠ (p_0)⁻›
    using thm-noncont-propos:3 thm-noncont-propos:4 &I by auto
qed(auto simp: log-prop-prop:2)

AOT-act-theorem no-cnac: ‹¬∃x(E!x & ¬𝓐E!x)›
proof(rule raa-cor:2)
  AOT-assume ‹∃x(E!x & ¬𝓐E!x)›
  then AOT-obtain a where a: ‹E!a & ¬𝓐E!a›
    using ∃E[rotated] by blast
  AOT-hence ‹𝓐¬E!a›
    using &E logic-actual-nec:1[axiom-inst, THEN ≡E(2)] by blast
  AOT-hence ‹¬E!a›
```

using *logic−actual*[*act-axiom-inst*, *THEN* →*E*] **by** *blast*
AOT-hence ‹$E!a$ & ¬$E!a$›
using *a* &*E* &*I* **by** *blast*
AOT-thus ‹*p* & ¬*p*› **for** *p* **using** *raa−cor*:*1* **by** *blast*
qed

AOT-theorem *pos−not−pna*:*1*: ‹¬$\mathcal{A}$∃ *x* ($E!x$ & ¬$\mathcal{A}E!x$)›
proof(*rule raa−cor*:*2*)
AOT-assume ‹$\mathcal{A}$∃ *x* ($E!x$ & ¬$\mathcal{A}E!x$)›
AOT-hence ‹∃ *x* $\mathcal{A}$($E!x$ & ¬$\mathcal{A}E!x$)›
using *Act−Basic*:*10*[*THEN* ≡*E*(*1*)] **by** *blast*
then AOT-obtain *a* **where** ‹$\mathcal{A}$($E!a$ & ¬$\mathcal{A}E!a$)›
using ∃ *E*[*rotated*] **by** *blast*
AOT-hence *1*: ‹$\mathcal{A}E!a$ & $\mathcal{A}$¬$\mathcal{A}E!a$›
using *Act−Basic*:*2*[*THEN* ≡*E*(*1*)] **by** *blast*
AOT-hence ‹¬$\mathcal{A}\mathcal{A}E!a$›
using &*E*(*2*) *logic−actual−nec*:*1*[*axiom-inst*, *THEN* ≡*E*(*1*)] **by** *blast*
AOT-hence ‹¬$\mathcal{A}E!a$›
using *logic−actual−nec*:*4*[*axiom-inst*, *THEN* ≡*E*(*1*)] *RAA* **by** *blast*
AOT-thus ‹*p* & ¬*p*› **for** *p* **using** *1*[*THEN* &*E*(*1*)] &*I* *raa−cor*:*1* **by** *blast*
qed

AOT-theorem *pos−not−pna*:*2*: ‹◇¬∃ *x*($E!x$ & ¬$\mathcal{A}E!x$)›
proof (*rule RAA*(*1*))
AOT-show ‹¬$\mathcal{A}$∃ *x* ($E!x$ & ¬$\mathcal{A}E!x$)›
using *pos−not−pna*:*1* **by** *blast*
next
AOT-assume ‹¬◇¬∃ *x* ($E!x$ & ¬$\mathcal{A}E!x$)›
AOT-hence ‹□∃ *x* ($E!x$ & ¬$\mathcal{A}E!x$)›
using *KBasic*:*12*[*THEN* ≡*E*(*2*)] **by** *blast*
AOT-thus ‹$\mathcal{A}$∃ *x* ($E!x$ & ¬$\mathcal{A}E!x$)›
using *nec−imp−act*[*THEN* →*E*] **by** *blast*
qed

AOT-theorem *pos−not−pna*:*3*: ‹∃ *x* (◇$E!x$ & ¬$\mathcal{A}E!x$)›
proof −
AOT-obtain *a* **where** ‹◇($E!a$ & ¬$\mathcal{A}E!a$)›
using *qml*:*4*[*axiom-inst*] *BF*◇[*THEN* →*E*] ∃ *E*[*rotated*] **by** *blast*
AOT-hence *ϑ*: ‹◇$E!a$› **and** *ξ*: ‹◇¬$\mathcal{A}E!a$›
using *KBasic2*:*3*[*THEN* →*E*] &*E* **by** *blast+*
AOT-have ‹¬□$\mathcal{A}E!a$›
using *ξ* *KBasic*:*11*[*THEN* ≡*E*(*2*)] **by** *blast*
AOT-hence ‹¬$\mathcal{A}E!a$›
using *Act−Basic*:*6*[*THEN* *oth−class−taut*:*4*:*b*[*THEN* ≡*E*(*1*)],
THEN ≡*E*(*2*)] **by** *blast*
AOT-hence ‹◇$E!a$ & ¬$\mathcal{A}E!a$› **using** *ϑ* &*I* **by** *blast*
thus *?thesis* **using** ∃ *I* **by** *fast*
qed

AOT-define *contingent-prop* :: *φ* (‹q_0›)
q_0-*def*: ‹(q_0) $=_{df}$ (∃ *x* ($E!x$ & ¬$\mathcal{A}E!x$))›

AOT-theorem q_0-*prop*: ‹◇q_0 & ◇¬q_0›
apply (*rule* $=_{df}I$(*2*)[*OF* q_0-*def*])
apply (*fact log−prop−prop*:*2*)
apply (*rule* &*I*)
apply (*fact qml*:*4*[*axiom-inst*])
by (*fact pos−not−pna*:*2*)

AOT-theorem *basic−prop*:*1*: ‹*Contingent0*((q_0))›
proof(*rule contingent−properties*:*4*[*zero*][*THEN* $\equiv_{df}I$])
AOT-have ‹¬*Necessary0*((q_0)) & ¬*Impossible0*((q_0))›
proof (*rule* &*I*;

```
        rule =_df I(2)[OF q_0-def];
        (rule log−prop−prop:2 | rule raa−cor:2))
    AOT-assume ‹Necessary0(∃ x (E!x & ¬𝓐E!x))›
    AOT-hence ‹□∃ x (E!x & ¬𝓐E!x)›
      using contingent−properties:1[zero][THEN ≡_df E] by blast
    AOT-hence ‹𝓐∃ x (E!x & ¬𝓐E!x)›
      using Act−Basic:8[THEN →E] qml:2[axiom-inst, THEN →E] by blast
    AOT-thus ‹𝓐∃ x (E!x & ¬𝓐E!x) & ¬𝓐∃ x (E!x & ¬𝓐E!x)›
      using pos−not−pna:1 &I by blast
  next
    AOT-assume ‹Impossible0(∃ x (E!x & ¬𝓐E!x))›
    AOT-hence ‹□¬(∃ x (E!x & ¬𝓐E!x))›
      using contingent−properties:2[zero][THEN ≡_df E] by blast
    AOT-hence ‹¬◇(∃ x (E!x & ¬𝓐E!x))›
      using KBasic2:1[THEN ≡E(1)] by blast
    AOT-thus ‹◇(∃ x (E!x & ¬𝓐E!x)) & ¬◇(∃ x (E!x & ¬𝓐E!x))›
      using qml:4[axiom-inst] &I by blast
  qed
  AOT-thus ‹¬(Necessary0((q_0)) ∨ Impossible0((q_0)))›
    using oth−class−taut:5:d ≡E(2) by blast
qed

AOT-theorem basic−prop:2: ‹∃ p Contingent0((p))›
  using ∃ I(1)[rotated, OF log−prop−prop:2] basic−prop:1 by blast

AOT-theorem basic−prop:3: ‹Contingent0(((q_0)^−))›
  apply (AOT-subst ‹((q_0)^−)› ‹¬q_0›)
   apply (insert thm−relation−negation:3 ∀ I
              ∀ E(1)[rotated, OF log−prop−prop:2]; fast)
  apply (rule contingent−properties:4[zero][THEN ≡_df I])
  apply (rule oth−class−taut:5:d[THEN ≡E(2)])
  apply (rule &I)
   apply (rule contingent−properties:1[zero][THEN df−rules−formulas[3],
              THEN useful−tautologies:5[THEN →E], THEN →E])
   apply (rule conventions:5[THEN ≡_df E])
   apply (rule =_df E(2)[OF q_0-def])
    apply (rule log−prop−prop:2)
   apply (rule q_0-prop[THEN &E(1)])
  apply (rule contingent−properties:2[zero][THEN df−rules−formulas[3],
             THEN useful−tautologies:5[THEN →E], THEN →E])
  apply (rule conventions:5[THEN ≡_df E])
  by (rule q_0-prop[THEN &E(2)])

AOT-theorem basic−prop:4:
  ‹∃ p∃ q (p ≠ q & Contingent0(p) & Contingent0(q))›
proof(rule ∃ I)+
  AOT-have 0: ‹φ ≠ (φ)^−› for φ
    using thm−relation−negation:6 ∀ I
        ∀ E(1)[rotated, OF log−prop−prop:2] by fast
  AOT-show ‹(q_0) ≠ (q_0)^− & Contingent0(q_0) & Contingent0(((q_0)^−))›
    using basic−prop:1 basic−prop:3 &I 0 by presburger
qed(auto simp: log−prop−prop:2)

AOT-theorem proposition−facts:1:
  ‹NonContingent0(p) → ¬∃ q (Contingent0(q) & q = p)›
proof(rule →I; rule raa−cor:2)
  AOT-assume ‹NonContingent0(p)›
  AOT-hence 1: ‹Necessary0(p) ∨ Impossible0(p)›
    using contingent−properties:3[zero][THEN ≡_df E] by blast
  AOT-assume ‹∃ q (Contingent0(q) & q = p)›
  then AOT-obtain q where ‹Contingent0(q) & q = p›
    using ∃ E[rotated] by blast
  AOT-hence ‹Contingent0(p)›
```

```
    using rule=E &E by fast
  AOT-thus ‹(Necessary0(p) ∨ Impossible0(p)) &
          ¬(Necessary0(p) ∨ Impossible0(p))›
    using contingent−properties:4[zero][THEN ≡_df E] 1 &I by blast
qed

AOT-theorem proposition−facts:2:
  ‹Contingent0(p) → ¬∃ q (NonContingent0(q) & q = p)›
proof(rule →I; rule raa−cor:2)
  AOT-assume ‹Contingent0(p)›
  AOT-hence 1: ‹¬(Necessary0(p) ∨ Impossible0(p))›
    using contingent−properties:4[zero][THEN ≡_df E] by blast
  AOT-assume ‹∃ q (NonContingent0(q) & q = p)›
  then AOT-obtain q where ‹NonContingent0(q) & q = p›
    using ∃ E[rotated] by blast
  AOT-hence ‹NonContingent0(p)›
    using rule=E &E by fast
  AOT-thus ‹(Necessary0(p) ∨ Impossible0(p)) &
          ¬(Necessary0(p) ∨ Impossible0(p))›
    using contingent−properties:3[zero][THEN ≡_df E] 1 &I by blast
qed

AOT-theorem proposition−facts:3:
  ‹(p_0) ≠ (p_0)⁻ & (p_0) ≠ (q_0) & (p_0) ≠ (q_0)⁻ & (p_0)⁻ ≠ (q_0)⁻ & (q_0) ≠ (q_0)⁻›
proof −
  {
    fix χ φ ψ
    AOT-assume ‹χ{φ}›
    moreover AOT-assume ‹¬χ{ψ}›
    ultimately AOT-have ‹¬(χ{φ} ≡ χ{ψ})›
      using RAA ≡E by metis
    moreover {
      AOT-have ‹∀ p∀ q ((¬(χ{p} ≡ χ{q})) → p ≠ q)›
        by (rule ∀ I; rule ∀ I; rule pos−not−equiv−ne:4[zero])
      AOT-hence ‹((¬(χ{φ} ≡ χ{ψ})) → φ ≠ ψ)›
        using ∀ E log−prop−prop:2 by blast
    }
    ultimately AOT-have ‹φ ≠ ψ›
      using →E by blast
  } note 0 = this
  AOT-have contingent-neg: ‹Contingent0(φ) ≡ Contingent0(((φ)⁻))› for φ
    using thm−cont−propos:3 ∀ I
        ∀ E(1)[rotated, OF log−prop−prop:2] by fast
  AOT-have not-noncontingent-if-contingent:
    ‹¬NonContingent0(φ)› if ‹Contingent0(φ)› for φ
    apply (rule contingent−properties:3[zero][THEN ≡Df,
                THEN oth−class−taut:4:b[THEN ≡E(1)], THEN ≡E(2)])
    using that contingent−properties:4[zero][THEN ≡_df E] by blast
  show ?thesis
    apply (rule &I)+
    using thm−relation−negation:6 ∀ I
        ∀ E(1)[rotated, OF log−prop−prop:2]
        apply fast
       apply (rule 0)
    using thm−noncont−propos:3 apply fast
       apply (rule not-noncontingent-if-contingent)
       apply (fact AOT)
      apply (rule 0)
    apply (rule thm−noncont−propos:3)
      apply (rule not-noncontingent-if-contingent)
      apply (rule contingent-neg[THEN ≡E(1)])
      apply (fact AOT)
     apply (rule 0)
```

apply (*rule thm−noncont−propos*:*4*)
apply (*rule not-noncontingent-if-contingent*)
apply (*rule contingent-neg*[*THEN* ≡*E*(*1*)])
apply (*fact AOT*)
using *thm−relation−negation*:*6* ∀ *I*
∀ *E*(*1*)[*rotated*, *OF log−prop−prop*:*2*] **by** *fast*
qed

AOT-define *ContingentlyTrue* :: ‹φ ⇒ φ› (‹*ContingentlyTrue*′(-′)›)
cont−tf:*1*: ‹*ContingentlyTrue*(*p*) $\equiv_{df}$ *p* & ◊¬*p*›

AOT-define *ContingentlyFalse* :: ‹φ ⇒ φ› (‹*ContingentlyFalse*′(-′)›)
cont−tf:*2*: ‹*ContingentlyFalse*(*p*) $\equiv_{df}$ ¬*p* & ◊*p*›

AOT-theorem *cont−true−cont*:*1*:
‹*ContingentlyTrue*((*p*)) → *Contingent0*((*p*))›
proof(*rule* →*I*)
AOT-assume ‹*ContingentlyTrue*((*p*))›
AOT-hence *1*: ‹*p*› **and** *2*: ‹◊¬*p*› **using** *cont−tf*:*1*[*THEN* $\equiv_{df}$*E*] &*E* **by** *blast*+
AOT-have ‹¬*Necessary0*((*p*))›
apply (*rule contingent−properties*:*1*[*zero*][*THEN* ≡*Df*,
THEN oth−class−taut:*4*:*b*[*THEN* ≡*E*(*1*)], *THEN* ≡*E*(*2*)])
using *2 KBasic*:*11*[*THEN* ≡*E*(*2*)] **by** *blast*
moreover AOT-have ‹¬*Impossible0*((*p*))›
apply (*rule contingent−properties*:*2*[*zero*][*THEN* ≡*Df*,
THEN oth−class−taut:*4*:*b*[*THEN* ≡*E*(*1*)], *THEN* ≡*E*(*2*)])
apply (*rule conventions*:*5*[*THEN* $\equiv_{df}$*E*])
using *T*◊[*THEN* →*E*, *OF 1*]**.**
ultimately AOT-have ‹¬(*Necessary0*((*p*)) ∨ *Impossible0*((*p*)))›
using *DeMorgan*(*2*)[*THEN* ≡*E*(*2*)] &*I* **by** *blast*
AOT-thus ‹*Contingent0*((*p*))›
using *contingent−properties*:*4*[*zero*][*THEN* $\equiv_{df}$*I*] **by** *blast*
qed

AOT-theorem *cont−true−cont*:*2*:
‹*ContingentlyFalse*((*p*)) → *Contingent0*((*p*))›
proof(*rule* →*I*)
AOT-assume ‹*ContingentlyFalse*((*p*))›
AOT-hence *1*: ‹¬*p*› **and** *2*: ‹◊*p*› **using** *cont−tf*:*2*[*THEN* $\equiv_{df}$*E*] &*E* **by** *blast*+
AOT-have ‹¬*Necessary0*((*p*))›
apply (*rule contingent−properties*:*1*[*zero*][*THEN* ≡*Df*,
THEN oth−class−taut:*4*:*b*[*THEN* ≡*E*(*1*)], *THEN* ≡*E*(*2*)])
using *KBasic*:*11*[*THEN* ≡*E*(*2*)] *T*◊[*THEN* →*E*, *OF 1*] **by** *blast*
moreover AOT-have ‹¬*Impossible0*((*p*))›
apply (*rule contingent−properties*:*2*[*zero*][*THEN* ≡*Df*,
THEN oth−class−taut:*4*:*b*[*THEN* ≡*E*(*1*)], *THEN* ≡*E*(*2*)])
apply (*rule conventions*:*5*[*THEN* $\equiv_{df}$*E*])
using *2***.**
ultimately AOT-have ‹¬(*Necessary0*((*p*)) ∨ *Impossible0*((*p*)))›
using *DeMorgan*(*2*)[*THEN* ≡*E*(*2*)] &*I* **by** *blast*
AOT-thus ‹*Contingent0*((*p*))›
using *contingent−properties*:*4*[*zero*][*THEN* $\equiv_{df}$*I*] **by** *blast*
qed

AOT-theorem *cont−true−cont*:*3*:
‹*ContingentlyTrue*((*p*)) ≡ *ContingentlyFalse*(((*p*)$^{-}$))›
proof(*rule* ≡*I*; *rule* →*I*)
AOT-assume ‹*ContingentlyTrue*((*p*))›
AOT-hence *0*: ‹*p* & ◊¬*p*› **using** *cont−tf*:*1*[*THEN* $\equiv_{df}$*E*] **by** *blast*
AOT-have *1*: ‹*ContingentlyFalse*(¬*p*)›
apply (*rule cont−tf*:*2*[*THEN* $\equiv_{df}$*I*])
apply (*AOT-subst* (*reverse*) ‹¬¬*p*› *p*)
by (*auto simp*: *oth−class−taut*:*3*:*b 0*)

```
  AOT-show ‹ContingentlyFalse(((p)⁻))›
    apply (AOT-subst ‹((p)⁻)› ‹¬p›)
    by (auto simp: thm−relation−negation:3 1)
next
  AOT-assume 1: ‹ContingentlyFalse(((p)⁻))›
  AOT-have ‹ContingentlyFalse(¬p)›
    by (AOT-subst (reverse) ‹¬p› ‹((p)⁻)›)
       (auto simp: thm−relation−negation:3 1)
  AOT-hence ‹¬¬p & ◊¬p› using cont−tf:2[THEN ≡dfE] by blast
  AOT-hence ‹p & ◊¬p›
    using &I &E useful−tautologies:1[THEN →E] by metis
  AOT-thus ‹ContingentlyTrue((p))›
    using cont−tf:1[THEN ≡dfI] by blast
qed

AOT-theorem cont−true−cont:4:
  ‹ContingentlyFalse((p)) ≡ ContingentlyTrue(((p)⁻))›
proof(rule ≡I; rule →I)
  AOT-assume ‹ContingentlyFalse(p)›
  AOT-hence 0: ‹¬p & ◊p›
    using cont−tf:2[THEN ≡dfE] by blast
  AOT-have ‹¬p & ◊¬¬p›
    by (AOT-subst (reverse) ‹¬¬p› p)
       (auto simp: oth−class−taut:3:b 0)
  AOT-hence 1: ‹ContingentlyTrue(¬p)›
    by (rule cont−tf:1[THEN ≡dfI])
  AOT-show ‹ContingentlyTrue(((p)⁻))›
    by (AOT-subst ‹((p)⁻)› ‹¬p›)
       (auto simp: thm−relation−negation:3 1)
next
  AOT-assume 1: ‹ContingentlyTrue(((p)⁻))›
  AOT-have ‹ContingentlyTrue(¬p)›
    by (AOT-subst (reverse) ‹¬p› ‹((p)⁻)›)
       (auto simp add: thm−relation−negation:3 1)
  AOT-hence 2: ‹¬p & ◊¬¬p› using cont−tf:1[THEN ≡dfE] by blast
  AOT-have ‹◊p›
    by (AOT-subst p ‹¬¬p›)
       (auto simp add: oth−class−taut:3:b 2[THEN &E(2)])
  AOT-hence ‹¬p & ◊p› using 2[THEN &E(1)] &I by blast
  AOT-thus ‹ContingentlyFalse(p)›
    by (rule cont−tf:2[THEN ≡dfI])
qed

AOT-theorem cont−true−cont:5:
  ‹(ContingentlyTrue((p)) & Necessary0((q))) → p ≠ q›
proof (rule →I; frule &E(1); drule &E(2); rule raa−cor:1)
  AOT-assume ‹ContingentlyTrue((p))›
  AOT-hence ‹◊¬p›
    using cont−tf:1[THEN ≡dfE] &E by blast
  AOT-hence 0: ‹¬□p› using KBasic:11[THEN ≡E(2)] by blast
  AOT-assume ‹Necessary0((q))›
  moreover AOT-assume ‹¬(p ≠ q)›
  AOT-hence ‹p = q›
    using =−infix[THEN ≡Df,
                  THEN oth−class−taut:4:b[THEN ≡E(1)],
                  THEN ≡E(1)]
          useful−tautologies:1[THEN →E] by blast
  ultimately AOT-have ‹Necessary0((p))› using rule=E id-sym by blast
  AOT-hence ‹□p›
    using contingent−properties:1[zero][THEN ≡dfE] by blast
  AOT-thus ‹□p & ¬□p› using 0 &I by blast
qed
```

```
AOT-theorem cont−true−cont:6:
  ‹(ContingentlyFalse((p)) & Impossible0((q))) → p ≠ q›
proof (rule →I; frule &E(1); drule &E(2); rule raa−cor:1)
  AOT-assume ‹ContingentlyFalse((p))›
  AOT-hence ‹◇p›
    using cont−tf:2[THEN ≡_df E] &E by blast
  AOT-hence 1: ‹¬□¬p›
    using conventions:5[THEN ≡_df E] by blast
  AOT-assume ‹Impossible0((q))›
  moreover AOT-assume ‹¬(p ≠ q)›
  AOT-hence ‹p = q›
    using =−infix[THEN ≡Df,
                  THEN oth−class−taut:4:b[THEN ≡E(1)],
                  THEN ≡E(1)]
          useful−tautologies:1[THEN →E] by blast
  ultimately AOT-have ‹Impossible0((p))› using rule=E id-sym by blast
  AOT-hence ‹□¬p›
    using contingent−properties:2[zero][THEN ≡_df E] by blast
  AOT-thus ‹□¬p & ¬□¬p› using 1 &I by blast
qed

AOT-act-theorem q0cf:1: ‹ContingentlyFalse(q_0)›
  apply (rule cont−tf:2[THEN ≡_df I])
  apply (rule =_df I(2)[OF q_0-def])
   apply (fact log−prop−prop:2)
  apply (rule &I)
   apply (fact no−cnac)
  by (fact qml:4[axiom-inst])

AOT-act-theorem q0cf:2: ‹ContingentlyTrue(((q_0)⁻))›
  apply (rule cont−tf:1[THEN ≡_df I])
  apply (rule =_df I(2)[OF q_0-def])
   apply (fact log−prop−prop:2)
  apply (rule &I)
   apply (rule thm−relation−negation:3
              [unvarify p, OF log−prop−prop:2, THEN ≡E(2)])
     apply (fact no−cnac)
  apply (rule rule=E[rotated,
                OF thm−relation−negation:7
                   [unvarify p, OF log−prop−prop:2, THEN id-sym]])
  apply (AOT-subst (reverse) ‹¬¬(∃x (E!x & ¬𝓐E!x))› ‹∃x (E!x & ¬𝓐E!x)›)
  by (auto simp: oth−class−taut:3:b qml:4[axiom-inst])

AOT-theorem cont−tf−thm:1: ‹∃p ContingentlyTrue((p))›
proof(rule ∨E(1)[OF exc−mid]; rule →I; rule ∃I)
  AOT-assume ‹q_0›
  AOT-hence ‹q_0 & ◇¬q_0› using q_0-prop[THEN &E(2)] &I by blast
  AOT-thus ‹ContingentlyTrue(q_0)›
    by (rule cont−tf:1[THEN ≡_df I])
next
  AOT-assume ‹¬q_0›
  AOT-hence ‹¬q_0 & ◇q_0› using q_0-prop[THEN &E(1)] &I by blast
  AOT-hence ‹ContingentlyFalse(q_0)›
    by (rule cont−tf:2[THEN ≡_df I])
  AOT-thus ‹ContingentlyTrue(((q_0)⁻))›
    by (rule cont−true−cont:4[unvarify p,
                OF log−prop−prop:2, THEN ≡E(1)])
qed(auto simp: log−prop−prop:2)


AOT-theorem cont−tf−thm:2: ‹∃p ContingentlyFalse((p))›
proof(rule ∨E(1)[OF exc−mid]; rule →I; rule ∃I)
  AOT-assume ‹q_0›
```

AOT-hence ‹q_0 & ◊¬q_0› **using** q_0*-prop*[*THEN* &*E*(*2*)] &*I* **by** *blast*
AOT-hence ‹*ContingentlyTrue*(q_0)›
by (*rule cont−tf*:*1*[*THEN* $\equiv_{df}$*I*])
AOT-thus ‹*ContingentlyFalse*(((q_0)$^-$))›
by (*rule cont−true−cont*:*3*[*unvarify p*,
OF log−prop−prop:*2*, *THEN* ≡*E*(*1*)])
next
AOT-assume ‹¬q_0›
AOT-hence ‹¬q_0 & ◊q_0› **using** q_0*-prop*[*THEN* &*E*(*1*)] &*I* **by** *blast*
AOT-thus ‹*ContingentlyFalse*(q_0)›
by (*rule cont−tf*:*2*[*THEN* $\equiv_{df}$*I*])
qed(*auto simp*: *log−prop−prop*:*2*)

AOT-theorem *property−facts1*:*1*: ‹∃ *F*∃ *x* ([*F*]*x* & ◊¬[*F*]*x*)›
proof −
fix *x*
AOT-obtain p_1 **where** ‹*ContingentlyTrue*((p_1))›
using *cont−tf−thm*:*1* ∃ *E*[*rotated*] **by** *blast*
AOT-hence *1*: ‹p_1 & ◊¬p_1› **using** *cont−tf*:*1*[*THEN* $\equiv_{df}$*E*] **by** *blast*
AOT-modally-strict {
AOT-have ‹*for arbitrary p*: $\vdash_\Box$ ([λ*z p*]*x* ≡ *p*)›
by (*rule beta−C−cor*:*3*[*THEN* ∀ *E*(*2*)]) *cqt-2-lambda-inst-prover*
AOT-hence ‹*for arbitrary p*: $\vdash_\Box$ □ ([λ*z p*]*x* ≡ *p*)›
by (*rule RN*)
AOT-hence ‹∀ *p* □([λ*z p*]*x* ≡ *p*)› **using** *GEN* **by** *fast*
AOT-hence ‹□([λ*z* p_1]*x* ≡ p_1)› **using** ∀ *E* **by** *fast*
} **note** *2* = *this*
AOT-hence ‹□([λ*z* p_1]*x* ≡ p_1)› **using** ∀ *E* **by** *blast*
AOT-hence ‹[λ*z* p_1]*x*›
using *1*[*THEN* &*E*(*1*)] *qml*:*2*[*axiom-inst*, *THEN* →*E*] ≡*E*(*2*) **by** *blast*
moreover AOT-have ‹◊¬[λ*z* p_1]*x*›
using *2*[*THEN qml*:*2*[*axiom-inst*, *THEN* →*E*]]
apply (*AOT-subst* ‹[λ*z* p_1]*x*› ‹p_1›)
using *1*[*THEN* &*E*(*2*)] **by** *blast*
ultimately AOT-have ‹[λ*z* p_1]*x* & ◊¬[λ*z* p_1]*x*› **using** &*I* **by** *blast*
AOT-hence ‹∃ *x* ([λ*z* p_1]*x* & ◊¬[λ*z* p_1]*x*)› **using** ∃ *I*(*2*) **by** *fast*
moreover AOT-have ‹[λ*z* p_1]↓› **by** *cqt*:*2*[*lambda*]
ultimately AOT-show ‹∃ *F*∃ *x* ([*F*]*x* & ◊¬[*F*]*x*)› **by** (*rule* ∃ *I*(*1*))
qed

AOT-theorem *property−facts1*:*2*: ‹∃ *F*∃ *x* (¬[*F*]*x* & ◊[*F*]*x*)›
proof −
fix *x*
AOT-obtain p_1 **where** ‹*ContingentlyFalse*((p_1))›
using *cont−tf−thm*:*2* ∃ *E*[*rotated*] **by** *blast*
AOT-hence *1*: ‹¬p_1 & ◊p_1› **using** *cont−tf*:*2*[*THEN* $\equiv_{df}$*E*] **by** *blast*
AOT-modally-strict {
AOT-have ‹*for arbitrary p*: $\vdash_\Box$ ([λ*z p*]*x* ≡ *p*)›
by (*rule beta−C−cor*:*3*[*THEN* ∀ *E*(*2*)]) *cqt-2-lambda-inst-prover*
AOT-hence ‹*for arbitrary p*: $\vdash_\Box$ (¬[λ*z p*]*x* ≡ ¬*p*)›
using *oth−class−taut*:*4*:*b* ≡*E* **by** *blast*
AOT-hence ‹*for arbitrary p*: $\vdash_\Box$ □(¬[λ*z p*]*x* ≡ ¬*p*)›
by (*rule RN*)
AOT-hence ‹∀ *p* □(¬[λ*z p*]*x* ≡ ¬*p*)› **using** *GEN* **by** *fast*
AOT-hence ‹□(¬[λ*z* p_1]*x* ≡ ¬p_1)› **using** ∀ *E* **by** *fast*
} **note** *2* = *this*
AOT-hence ‹□(¬[λ*z* p_1]*x* ≡ ¬p_1)› **using** ∀ *E* **by** *blast*
AOT-hence *3*: ‹¬[λ*z* p_1]*x*›
using *1*[*THEN* &*E*(*1*)] *qml*:*2*[*axiom-inst*, *THEN* →*E*] ≡*E*(*2*) **by** *blast*
AOT-modally-strict {
AOT-have ‹*for arbitrary p*: $\vdash_\Box$ ([λ*z p*]*x* ≡ *p*)›
by (*rule beta−C−cor*:*3*[*THEN* ∀ *E*(*2*)]) *cqt-2-lambda-inst-prover*
AOT-hence ‹*for arbitrary p*: $\vdash_\Box$ □([λ*z p*]*x* ≡ *p*)›

```
    by (rule RN)
    AOT-hence ‹∀ p □([λz p]x ≡ p)› using GEN by fast
    AOT-hence ‹□([λz p1]x ≡ p1)› using ∀ E by fast
  } note 4 = this
  AOT-have ‹◇[λz p1]x›
    using 4[THEN qml:2[axiom-inst, THEN →E]]
    apply (AOT-subst ‹[λz p1]x› ‹p1›)
    using 1[THEN &E(2)] by blast
  AOT-hence ‹¬[λz p1]x & ◇[λz p1]x› using 3 &I by blast
  AOT-hence ‹∃ x (¬[λz p1]x & ◇[λz p1]x)› using ∃ I(2) by fast
  moreover AOT-have ‹[λz p1]↓› by cqt:2[lambda]
  ultimately AOT-show ‹∃ F∃ x (¬[F]x & ◇[F]x)› by (rule ∃ I(1))
qed

context
begin

private AOT-lemma eqnotnec-123-Aux-ζ: ‹[L]x ≡ (E!x → E!x)›
    apply (rule =dfI(2)[OF L-def])
     apply cqt:2[lambda]
    apply (rule beta−C−meta[THEN →E])
  by cqt:2[lambda]

private AOT-lemma eqnotnec-123-Aux-ω: ‹[λz φ]x ≡ φ›
    by (rule beta−C−meta[THEN →E]) cqt:2[lambda]

private AOT-lemma eqnotnec-123-Aux-ϑ: ‹φ ≡ ∀ x([L]x ≡ [λz φ]x)›
proof(rule ≡I; rule →I; (rule ∀ I)?)
  fix x
  AOT-assume 1: ‹φ›
  AOT-have ‹[L]x ≡ (E!x → E!x)› using eqnotnec-123-Aux-ζ.
  also AOT-have ‹. . . ≡ φ›
    using if−p−then−p 1 ≡I →I by simp
  also AOT-have ‹. . . ≡ [λz φ]x›
    using Commutativity of ≡[THEN ≡E(1)] eqnotnec-123-Aux-ω by blast
  finally AOT-show ‹[L]x ≡ [λz φ]x›.
next
  fix x
  AOT-assume ‹∀ x([L]x ≡ [λz φ]x)›
  AOT-hence ‹[L]x ≡ [λz φ]x› using ∀ E by blast
  also AOT-have ‹. . . ≡ φ› using eqnotnec-123-Aux-ω.
  finally AOT-have ‹φ ≡ [L]x›
    using Commutativity of ≡[THEN ≡E(1)] by blast
  also AOT-have ‹. . . ≡ E!x → E!x› using eqnotnec-123-Aux-ζ.
  finally AOT-show ‹φ› using ≡E if−p−then−p by fast
qed
private lemmas eqnotnec-123-Aux-ξ =
  eqnotnec-123-Aux-ϑ[THEN oth−class−taut:4:b[THEN ≡E(1)],
    THEN conventions:3[THEN ≡Df, THEN ≡E(1), THEN &E(1)],
    THEN RM◇]
private lemmas eqnotnec-123-Aux-ξ' =
  eqnotnec-123-Aux-ϑ[
    THEN conventions:3[THEN ≡Df, THEN ≡E(1), THEN &E(1)],
    THEN RM◇]

AOT-theorem eqnotnec:1: ‹∃ F∃ G(∀ x([F]x ≡ [G]x) & ◇¬∀ x([F]x ≡ [G]x))›
proof−
  AOT-obtain p1 where ‹ContingentlyTrue(p1)›
    using cont−tf−thm:1 ∃ E[rotated] by blast
  AOT-hence ‹p1 & ◇¬p1› using cont−tf:1[THEN ≡dfE] by blast
  AOT-hence ‹∀ x ([L]x ≡ [λz p1]x) & ◇¬∀ x([L]x ≡ [λz p1]x)›
    apply − apply (rule &I)
    using &E eqnotnec-123-Aux-ϑ[THEN ≡E(1)]
```

```
        eqnotnec-123-Aux-ξ →E by fast+
  AOT-hence ‹∃ G (∀ x([L]x ≡ [G]x) & ◇¬∀ x([L]x ≡ [G]x))›
    by (rule ∃ I) cqt:2[lambda]
  AOT-thus ‹∃ F∃ G (∀ x([F]x ≡ [G]x) & ◇¬∀ x([F]x ≡ [G]x))›
    apply (rule ∃ I)
    by (rule =df I(2)[OF L-def]) cqt:2[lambda]+
qed

AOT-theorem eqnotnec:2: ‹∃ F∃ G(¬∀ x([F]x ≡ [G]x) & ◇∀ x([F]x ≡ [G]x))›
proof–
  AOT-obtain p1 where ‹ContingentlyFalse(p1)›
    using cont−tf−thm:2 ∃ E[rotated] by blast
  AOT-hence ‹¬p1 & ◇p1› using cont−tf:2[THEN ≡df E] by blast
  AOT-hence ‹¬∀ x ([L]x ≡ [λz p1]x) & ◇∀ x([L]x ≡ [λz p1]x)›
    apply − apply (rule &I)
    using eqnotnec-123-Aux-ϑ[THEN oth−class−taut:4:b[THEN ≡E(1)],
                           THEN ≡E(1)]
        &E eqnotnec-123-Aux-ξ' →E by fast+
  AOT-hence ‹∃ G (¬∀ x([L]x ≡ [G]x) & ◇∀ x([L]x ≡ [G]x))›
    by (rule ∃ I) cqt:2[lambda]
  AOT-thus ‹∃ F∃ G (¬∀ x([F]x ≡ [G]x) & ◇∀ x([F]x ≡ [G]x))›
    apply (rule ∃ I)
    by (rule =df I(2)[OF L-def]) cqt:2[lambda]+
qed

AOT-theorem eqnotnec:3: ‹∃ F∃ G(𝓐¬∀ x([F]x ≡ [G]x) & ◇∀ x([F]x ≡ [G]x))›
proof–
  AOT-have ‹¬𝓐q0›
    apply (rule =df I(2)[OF q0-def])
     apply (fact log−prop−prop:2)
    by (fact AOT)
  AOT-hence ‹𝓐¬q0›
    using logic−actual−nec:1[axiom-inst, THEN ≡E(2)] by blast
  AOT-hence ‹𝓐¬∀ x ([L]x ≡ [λz q0]x)›
    using eqnotnec-123-Aux-ϑ[THEN oth−class−taut:4:b[THEN ≡E(1)],
          THEN conventions:3[THEN ≡Df, THEN ≡E(1), THEN &E(1)],
          THEN RA[2], THEN act−cond[THEN →E], THEN →E] by blast
  moreover AOT-have ‹◇∀ x ([L]x ≡ [λz q0]x)›
    using eqnotnec-123-Aux-ξ'[THEN →E] q0-prop[THEN &E(1)] by blast
  ultimately AOT-have ‹𝓐¬∀ x ([L]x ≡ [λz q0]x) & ◇∀ x ([L]x ≡ [λz q0]x)›
    using &I by blast
  AOT-hence ‹∃ G (𝓐¬∀ x([L]x ≡ [G]x) & ◇∀ x([L]x ≡ [G]x))›
    by (rule ∃ I) cqt:2[lambda]
  AOT-thus ‹∃ F∃ G (𝓐¬∀ x([F]x ≡ [G]x) & ◇∀ x([F]x ≡ [G]x))›
    apply (rule ∃ I)
    by (rule =df I(2)[OF L-def]) cqt:2[lambda]+
qed

end

AOT-theorem eqnotnec:4: ‹∀ F∃ G(∀ x([F]x ≡ [G]x) & ◇¬∀ x([F]x ≡ [G]x))›
proof(rule GEN)
  fix F
  AOT-have Aux-A: ‹⊢□ ψ → ∀ x([F]x ≡ [λz [F]z & ψ]x)› for ψ
  proof(rule →I; rule GEN)
    AOT-modally-strict {
    fix x
    AOT-assume 0: ‹ψ›
    AOT-have ‹[λz [F]z & ψ]x ≡ [F]x & ψ›
      by (rule beta−C−meta[THEN →E]) cqt:2[lambda]
    also AOT-have ‹... ≡ [F]x›
      apply (rule ≡I; rule →I)
      using ∨E(3)[rotated, OF useful−tautologies:2[THEN →E], OF 0] &E
```

```
    apply blast
    using 0 &I by blast
  finally AOT-show ‹[F]x ≡ [λz [F]z & ψ]x›
    using Commutativity of ≡[THEN ≡E(1)] by blast
  }
qed

AOT-have Aux-B: ‹⊢□ ψ → ∀x([F]x ≡ [λz [F]z & ψ ∨ ¬ψ]x)› for ψ
proof (rule →I; rule GEN)
  AOT-modally-strict {
    fix x
    AOT-assume 0: ‹ψ›
    AOT-have ‹[λz ([F]z & ψ) ∨ ¬ψ]x ≡ (([F]x & ψ) ∨ ¬ψ)›
      by (rule beta−C−meta[THEN →E]) cqt:2[lambda]
    also AOT-have ‹... ≡ [F]x›
      apply (rule ≡I; rule →I)
      using ∨E(3)[rotated, OF useful−tautologies:2[THEN →E], OF 0]
            &E
       apply blast
      apply (rule ∨I(1)) using 0 &I by blast
    finally AOT-show ‹[F]x ≡ [λz ([F]z & ψ) ∨ ¬ψ]x›
      using Commutativity of ≡[THEN ≡E(1)] by blast
  }
qed

AOT-have Aux-C:
  ‹⊢□ ◇¬ψ → ◇¬∀z([λz [F]z & ψ]z ≡ [λz [F]z & ψ ∨ ¬ψ]z)› for ψ
proof(rule RM◇; rule →I; rule raa−cor:2)
AOT-modally-strict {
    AOT-assume 0: ‹¬ψ›
    AOT-assume ‹∀z ([λz [F]z & ψ]z ≡ [λz [F]z & ψ ∨ ¬ψ]z)›
    AOT-hence ‹[λz [F]z & ψ]z ≡ [λz [F]z & ψ ∨ ¬ψ]z› for z
      using ∀E by blast
    moreover AOT-have ‹[λz [F]z & ψ]z ≡ [F]z & ψ› for z
        by (rule beta−C−meta[THEN →E]) cqt:2[lambda]
    moreover AOT-have ‹[λz ([F]z & ψ) ∨ ¬ψ]z ≡ (([F]z & ψ) ∨ ¬ψ)› for z
      by (rule beta−C−meta[THEN →E]) cqt:2[lambda]
    ultimately AOT-have ‹[F]z & ψ ≡ (([F]z & ψ) ∨ ¬ψ)› for z
      using Commutativity of ≡[THEN ≡E(1)] ≡E(5) by meson
    moreover AOT-have ‹(([F]z & ψ) ∨ ¬ψ)› for z using 0 ∨I by blast
    ultimately AOT-have ‹ψ› using ≡E &E by metis
    AOT-thus ‹ψ & ¬ψ› using 0 &I by blast
  }
qed

AOT-have Aux-D: ‹□∀z ([F]z ≡ [λz [F]z & ψ]z) →
    (◇¬∀x ([λz [F]z & ψ]x ≡ [λz [F]z & ψ ∨ ¬ψ]x) ≡
     ◇¬∀x ([F]x ≡ [λz [F]z & ψ ∨ ¬ψ]x))› for ψ
proof (rule →I)
  AOT-assume A: ‹□∀z([F]z ≡ [λz [F]z & ψ]z)›
  AOT-show ‹◇¬∀x ([λz [F]z & ψ]x ≡ [λz [F]z & ψ ∨ ¬ψ]x) ≡
            ◇¬∀x ([F]x ≡ [λz [F]z & ψ ∨ ¬ψ]x)›
  proof(rule ≡I; rule KBasic:13[THEN →E];
        rule RN[prem][where Γ={«∀z([F]z ≡ [λz [F]z & ψ]z)»}, simplified];
        (rule useful−tautologies:5[THEN →E]; rule →I)?)
    AOT-modally-strict {
      AOT-assume ‹∀z ([F]z ≡ [λz [F]z & ψ]z)›
      AOT-hence 1: ‹[F]z ≡ [λz [F]z & ψ]z› for z
        using ∀E by blast
      AOT-assume ‹∀x ([F]x ≡ [λz [F]z & ψ ∨ ¬ψ]x)›
      AOT-hence 2: ‹[F]z ≡ [λz [F]z & ψ ∨ ¬ψ]z› for z
        using ∀E by blast
      AOT-have ‹[λz [F]z & ψ]z ≡ [λz [F]z & ψ ∨ ¬ψ]z› for z
```

```
      using ≡E 1 2 by meson
    AOT-thus ‹∀ x ([λz [F]z & ψ]x ≡ [λz [F]z & ψ ∨ ¬ψ]x)›
      by (rule GEN)
  }
  next
    AOT-modally-strict {
      AOT-assume ‹∀ z ([F]z ≡ [λz [F]z & ψ]z)›
      AOT-hence 1: ‹[F]z ≡ [λz [F]z & ψ]z› for z
        using ∀ E by blast
      AOT-assume ‹∀ x ([λz [F]z & ψ]x ≡ [λz [F]z & ψ ∨ ¬ψ]x)›
      AOT-hence 2: ‹[λz [F]z & ψ]z ≡ [λz [F]z & ψ ∨ ¬ψ]z› for z
        using ∀ E by blast
      AOT-have ‹[F]z ≡ [λz [F]z & ψ ∨ ¬ψ]z› for z
        using 1 2 ≡E by meson
      AOT-thus ‹ ∀ x ([F]x ≡ [λz [F]z & ψ ∨ ¬ψ]x)›
        by (rule GEN)
    }
  qed(auto simp: A)
qed

AOT-obtain p1 where p1-prop: ‹p1 & ◇¬p1›
  using cont−tf−thm:1 ∃ E[rotated]
        cont−tf:1[THEN ≡df E] by blast
{
  AOT-assume 1: ‹□∀ x([F]x ≡ [λz [F]z & p1]x)›
  AOT-have 2: ‹∀ x([F]x ≡ [λz [F]z & p1 ∨ ¬p1]x)›
    using Aux-B[THEN →E, OF p1-prop[THEN &E(1)]].
  AOT-have ‹◇¬∀ x([λz [F]z & p1]x ≡ [λz [F]z & p1 ∨ ¬p1]x)›
    using Aux-C[THEN →E, OF p1-prop[THEN &E(2)]].
  AOT-hence 3: ‹◇¬∀ x([F]x ≡ [λz [F]z & p1 ∨ ¬p1]x)›
    using Aux-D[THEN →E, OF 1, THEN ≡E(1)] by blast
  AOT-hence ‹∀ x([F]x ≡ [λz [F]z & p1 ∨ ¬p1]x) &
           ◇¬∀ x([F]x ≡ [λz [F]z & p1 ∨ ¬p1]x)›
    using 2 &I by blast
  AOT-hence ‹∃ G (∀ x ([F]x ≡ [G]x) & ◇¬∀ x([F]x ≡ [G]x))›
    by (rule ∃ I(1)) cqt:2[lambda]
}
moreover {
  AOT-assume 2: ‹¬□∀ x([F]x ≡ [λz [F]z & p1]x)›
  AOT-hence ‹◇¬∀ x([F]x ≡ [λz [F]z & p1]x)›
    using KBasic:11[THEN ≡E(1)] by blast
  AOT-hence ‹∀ x ([F]x ≡ [λz [F]z & p1]x) & ◇¬∀ x([F]x ≡ [λz [F]z & p1]x)›
    using Aux-A[THEN →E, OF p1-prop[THEN &E(1)]] &I by blast
  AOT-hence ‹∃ G (∀ x ([F]x ≡ [G]x) & ◇¬∀ x([F]x ≡ [G]x))›
    by (rule ∃ I(1)) cqt:2[lambda]
}
ultimately AOT-show ‹∃ G (∀ x ([F]x ≡ [G]x) & ◇¬∀ x([F]x ≡ [G]x))›
  using ∨E(1)[OF exc−mid] →I by blast
qed

AOT-theorem eqnotnec:5: ‹∀ F∃ G(¬∀ x([F]x ≡ [G]x) & ◇∀ x([F]x ≡ [G]x))›
proof(rule GEN)
  fix F
  AOT-have Aux-A: ‹⊢□ ◇ψ → ◇∀ x([F]x ≡ [λz [F]z & ψ]x)› for ψ
  proof(rule RM◇; rule →I; rule GEN)
    AOT-modally-strict {
    fix x
    AOT-assume 0: ‹ψ›
    AOT-have ‹[λz [F]z & ψ]x ≡ [F]x & ψ›
      by (rule beta−C−meta[THEN →E]) cqt:2[lambda]
    also AOT-have ‹... ≡ [F]x›
      apply (rule ≡I; rule →I)
      using ∨E(3)[rotated, OF useful−tautologies:2[THEN →E], OF 0] &E
```

  apply *blast*
  using *0* &*I* **by** *blast*
 finally AOT-show ‹$[F]x \equiv [\lambda z\ [F]z\ \&\ \psi]x$›
  using *Commutativity of* ≡[*THEN* ≡*E*(*1*)] **by** *blast*
 }
qed

AOT-have *Aux-B*: ‹$\vdash_\square \Diamond\psi \rightarrow \Diamond\forall x([F]x \equiv [\lambda z\ [F]z\ \&\ \psi \vee \neg\psi]x)$› **for** ψ
proof (*rule RM*◇; *rule* →*I*; *rule GEN*)
 AOT-modally-strict {
  fix x
  AOT-assume *0*: ‹ψ›
  AOT-have ‹$[\lambda z\ ([F]z\ \&\ \psi) \vee \neg\psi]x \equiv (([F]x\ \&\ \psi) \vee \neg\psi)$›
   by (*rule beta*−*C*−*meta*[*THEN* →*E*]) *cqt*:*2*[*lambda*]
  also AOT-have ‹$\ldots \equiv [F]x$›
   apply (*rule* ≡*I*; *rule* →*I*)
   using ∨*E*(*3*)[*rotated*, *OF useful*−*tautologies*:*2*[*THEN* →*E*], *OF 0*] &*E*
    apply *blast*
   apply (*rule* ∨*I*(*1*)) **using** *0* &*I* **by** *blast*
  finally AOT-show ‹$[F]x \equiv [\lambda z\ ([F]z\ \&\ \psi) \vee \neg\psi]x$›
   using *Commutativity of* ≡[*THEN* ≡*E*(*1*)] **by** *blast*
 }
qed

AOT-have *Aux-C*: ‹$\vdash_\square \neg\psi \rightarrow \neg\forall z([\lambda z\ [F]z\ \&\ \psi]z \equiv [\lambda z\ [F]z\ \&\ \psi \vee \neg\psi]z)$› **for** ψ
proof(*rule* →*I*; *rule raa*−*cor*:*2*)
AOT-modally-strict {
  AOT-assume *0*: ‹$\neg\psi$›
  AOT-assume ‹$\forall z\ ([\lambda z\ [F]z\ \&\ \psi]z \equiv [\lambda z\ [F]z\ \&\ \psi \vee \neg\psi]z)$›
  AOT-hence ‹$[\lambda z\ [F]z\ \&\ \psi]z \equiv [\lambda z\ [F]z\ \&\ \psi \vee \neg\psi]z$› **for** z
   using ∀*E* **by** *blast*
  moreover AOT-have ‹$[\lambda z\ [F]z\ \&\ \psi]z \equiv [F]z\ \&\ \psi$› **for** z
    by (*rule beta*−*C*−*meta*[*THEN* →*E*]) *cqt*:*2*[*lambda*]
  moreover AOT-have ‹$[\lambda z\ ([F]z\ \&\ \psi) \vee \neg\psi]z \equiv (([F]z\ \&\ \psi) \vee \neg\psi)$› **for** z
   by (*rule beta*−*C*−*meta*[*THEN* →*E*]) *cqt*:*2*[*lambda*]
  ultimately AOT-have ‹$[F]z\ \&\ \psi \equiv (([F]z\ \&\ \psi) \vee \neg\psi)$› **for** z
   using *Commutativity of* ≡[*THEN* ≡*E*(*1*)] ≡*E*(*5*) **by** *meson*
  moreover AOT-have ‹$(([F]z\ \&\ \psi) \vee \neg\psi)$› **for** z
   using *0* ∨*I* **by** *blast*
  ultimately AOT-have ‹ψ› **using** ≡*E* &*E* **by** *metis*
  AOT-thus ‹$\psi\ \&\ \neg\psi$› **using** *0* &*I* **by** *blast*
 }
qed

AOT-have *Aux-D*: ‹$\forall z\ ([F]z \equiv [\lambda z\ [F]z\ \&\ \psi]z) \rightarrow$
 $(\neg\forall x\ ([\lambda z\ [F]z\ \&\ \psi]x \equiv [\lambda z\ [F]z\ \&\ \psi \vee \neg\psi]x) \equiv$
  $\neg\forall x\ ([F]x \equiv [\lambda z\ [F]z\ \&\ \psi \vee \neg\psi]x))$› **for** ψ
proof (*rule* →*I*; *rule* ≡*I*;
    (*rule useful*−*tautologies*:*5*[*THEN* →*E*]; *rule* →*I*)*?*)
 AOT-modally-strict {
  AOT-assume ‹$\forall z\ ([F]z \equiv [\lambda z\ [F]z\ \&\ \psi]z)$›
  AOT-hence *1*: ‹$[F]z \equiv [\lambda z\ [F]z\ \&\ \psi]z$› **for** z
   using ∀*E* **by** *blast*
  AOT-assume ‹$\forall x\ ([F]x \equiv [\lambda z\ [F]z\ \&\ \psi \vee \neg\psi]x)$›
  AOT-hence *2*: ‹$[F]z \equiv [\lambda z\ [F]z\ \&\ \psi \vee \neg\psi]z$› **for** z
   using ∀*E* **by** *blast*
  AOT-have ‹$[\lambda z\ [F]z\ \&\ \psi]z \equiv [\lambda z\ [F]z\ \&\ \psi \vee \neg\psi]z$› **for** z
   using ≡*E 1 2* **by** *meson*
  AOT-thus ‹$\forall x\ ([\lambda z\ [F]z\ \&\ \psi]x \equiv [\lambda z\ [F]z\ \&\ \psi \vee \neg\psi]x)$›
   by (*rule GEN*)
 }
next
 AOT-modally-strict {

```
      AOT-assume ‹∀ z ([F]z ≡ [λz [F]z & ψ]z)›
      AOT-hence 1: ‹[F]z ≡ [λz [F]z & ψ]z› for z
        using ∀ E by blast
      AOT-assume ‹∀ x ([λz [F]z & ψ]x ≡ [λz [F]z & ψ ∨ ¬ψ]x)›
      AOT-hence 2: ‹[λz [F]z & ψ]z ≡ [λz [F]z & ψ ∨ ¬ψ]z› for z
        using ∀ E by blast
      AOT-have ‹[F]z ≡ [λz [F]z & ψ ∨ ¬ψ]z› for z
        using 1 2 ≡E by meson
      AOT-thus ‹ ∀ x ([F]x ≡ [λz [F]z & ψ ∨ ¬ψ]x)›
        by (rule GEN)
    }
  qed

  AOT-obtain p₁ where p₁-prop: ‹¬p₁ & ◊p₁›
    using cont−tf−thm:2 ∃ E[rotated] cont−tf:2[THEN ≡df E] by blast
  {
    AOT-assume 1: ‹∀ x([F]x ≡ [λz [F]z & p₁]x)›
    AOT-have 2: ‹◊∀ x([F]x ≡ [λz [F]z & p₁ ∨ ¬p₁]x)›
      using Aux-B[THEN →E, OF p₁-prop[THEN &E(2)]].
    AOT-have ‹¬∀ x([λz [F]z & p₁]x ≡ [λz [F]z & p₁ ∨ ¬p₁]x)›
      using Aux-C[THEN →E, OF p₁-prop[THEN &E(1)]].
    AOT-hence 3: ‹¬∀ x([F]x ≡ [λz [F]z & p₁ ∨ ¬p₁]x)›
      using Aux-D[THEN →E, OF 1, THEN ≡E(1)] by blast
    AOT-hence ‹¬∀ x([F]x ≡ [λz [F]z & p₁ ∨ ¬p₁]x) &
              ◊∀ x([F]x ≡ [λz [F]z & p₁ ∨ ¬p₁]x)›
      using 2 &I by blast
    AOT-hence ‹∃ G (¬∀ x ([F]x ≡ [G]x) & ◊∀ x([F]x ≡ [G]x))›
      by (rule ∃ I(1)) cqt:2[lambda]
  }
  moreover {
    AOT-assume 2: ‹¬∀ x([F]x ≡ [λz [F]z & p₁]x)›
    AOT-hence ‹¬∀ x([F]x ≡ [λz [F]z & p₁]x)›
      using KBasic:11[THEN ≡E(1)] by blast
    AOT-hence ‹¬∀ x ([F]x ≡ [λz [F]z & p₁]x) &
              ◊∀ x([F]x ≡ [λz [F]z & p₁]x)›
      using Aux-A[THEN →E, OF p₁-prop[THEN &E(2)]] &I by blast
    AOT-hence ‹∃ G (¬∀ x ([F]x ≡ [G]x) & ◊∀ x([F]x ≡ [G]x))›
      by (rule ∃ I(1)) cqt:2[lambda]
  }
  ultimately AOT-show ‹∃ G (¬∀ x ([F]x ≡ [G]x) & ◊∀ x([F]x ≡ [G]x))›
    using ∨E(1)[OF exc−mid] →I by blast
qed

AOT-theorem eqnotnec:6: ‹∀ F∃ G(𝒜¬∀ x([F]x ≡ [G]x) & ◊∀ x([F]x ≡ [G]x))›
proof(rule GEN)
  fix F
  AOT-have Aux-A: ‹⊢□ ◊ψ → ◊∀ x([F]x ≡ [λz [F]z & ψ]x)› for ψ
  proof(rule RM◊; rule →I; rule GEN)
    AOT-modally-strict {
    fix x
    AOT-assume 0: ‹ψ›
    AOT-have ‹[λz [F]z & ψ]x ≡ [F]x & ψ›
      by (rule beta−C−meta[THEN →E]) cqt:2[lambda]
    also AOT-have ‹... ≡ [F]x›
      apply (rule ≡I; rule →I)
      using ∨E(3)[rotated, OF useful−tautologies:2[THEN →E], OF 0]
            &E
       apply blast
      using 0 &I by blast
    finally AOT-show ‹[F]x ≡ [λz [F]z & ψ]x›
      using Commutativity of ≡[THEN ≡E(1)] by blast
    }
  qed
```

```
AOT-have Aux-B: ‹⊢□ ◇ψ → ◇∀x([F]x ≡ [λz [F]z & ψ ∨ ¬ψ]x)› for ψ
proof (rule RM◇; rule →I; rule GEN)
  AOT-modally-strict {
    fix x
    AOT-assume 0: ‹ψ›
    AOT-have ‹[λz ([F]z & ψ) ∨ ¬ψ]x ≡ (([F]x & ψ) ∨ ¬ψ)›
      by (rule beta−C−meta[THEN →E]) cqt:2[lambda]
    also AOT-have ‹... ≡ [F]x›
      apply (rule ≡I; rule →I)
      using ∨E(3)[rotated, OF useful−tautologies:2[THEN →E], OF 0] &E
       apply blast
      apply (rule ∨I(1)) using 0 &I by blast
    finally AOT-show ‹[F]x ≡ [λz ([F]z & ψ) ∨ ¬ψ]x›
      using Commutativity of ≡[THEN ≡E(1)] by blast
  }
qed

AOT-have Aux-C:
  ‹⊢□ 𝓐¬ψ → 𝓐¬∀z([λz [F]z & ψ]z ≡ [λz [F]z & ψ ∨ ¬ψ]z)› for ψ
proof(rule act−cond[THEN →E]; rule RA[2]; rule →I; rule raa−cor:2)
AOT-modally-strict {
    AOT-assume 0: ‹¬ψ›
    AOT-assume ‹∀z ([λz [F]z & ψ]z ≡ [λz [F]z & ψ ∨ ¬ψ]z)›
    AOT-hence ‹[λz [F]z & ψ]z ≡ [λz [F]z & ψ ∨ ¬ψ]z› for z
      using ∀E by blast
    moreover AOT-have ‹[λz [F]z & ψ]z ≡ [F]z & ψ› for z
        by (rule beta−C−meta[THEN →E]) cqt:2[lambda]
    moreover AOT-have ‹[λz ([F]z & ψ) ∨ ¬ψ]z ≡ (([F]z & ψ) ∨ ¬ψ)› for z
      by (rule beta−C−meta[THEN →E]) cqt:2[lambda]
    ultimately AOT-have ‹[F]z & ψ ≡ (([F]z & ψ) ∨ ¬ψ)› for z
      using Commutativity of ≡[THEN ≡E(1)] ≡E(5) by meson
    moreover AOT-have ‹(([F]z & ψ) ∨ ¬ψ)› for z
      using 0 ∨I by blast
    ultimately AOT-have ‹ψ› using ≡E &E by metis
    AOT-thus ‹ψ & ¬ψ› using 0 &I by blast
  }
qed

AOT-have ‹□(∀z ([F]z ≡ [λz [F]z & ψ]z) →
  (¬∀x ([λz [F]z & ψ]x ≡ [λz [F]z & ψ ∨ ¬ψ]x) ≡
   ¬∀x ([F]x ≡ [λz [F]z & ψ ∨ ¬ψ]x)))› for ψ
proof (rule RN; rule →I)
  AOT-modally-strict {
      AOT-assume ‹∀z ([F]z ≡ [λz [F]z & ψ]z)›
      AOT-thus ‹¬∀x ([λz [F]z & ψ]x ≡ [λz [F]z & ψ ∨ ¬ψ]x) ≡
                ¬∀x ([F]x ≡ [λz [F]z & ψ ∨ ¬ψ]x)›
        apply −
      proof(rule ≡I; (rule useful−tautologies:5[THEN →E]; rule →I)?)
        AOT-assume ‹∀z ([F]z ≡ [λz [F]z & ψ]z)›
        AOT-hence 1: ‹[F]z ≡ [λz [F]z & ψ]z› for z
          using ∀E by blast
        AOT-assume ‹∀x ([F]x ≡ [λz [F]z & ψ ∨ ¬ψ]x)›
        AOT-hence 2: ‹[F]z ≡ [λz [F]z & ψ ∨ ¬ψ]z› for z
          using ∀E by blast
        AOT-have ‹[λz [F]z & ψ]z ≡ [λz [F]z & ψ ∨ ¬ψ]z› for z
          using ≡E 1 2 by meson
        AOT-thus ‹∀x ([λz [F]z & ψ]x ≡ [λz [F]z & ψ ∨ ¬ψ]x)›
          by (rule GEN)
      next
          AOT-assume ‹∀z ([F]z ≡ [λz [F]z & ψ]z)›
          AOT-hence 1: ‹[F]z ≡ [λz [F]z & ψ]z› for z
            using ∀E by blast
```

```
          AOT-assume ‹∀ x ([λz [F]z & ψ]x ≡ [λz [F]z & ψ ∨ ¬ψ]x)›
          AOT-hence 2: ‹[λz [F]z & ψ]z ≡ [λz [F]z & ψ ∨ ¬ψ]z› for z
            using ∀ E by blast
          AOT-have ‹[F]z ≡ [λz [F]z & ψ ∨ ¬ψ]z› for z
            using 1 2 ≡E by meson
          AOT-thus ‹ ∀ x ([F]x ≡ [λz [F]z & ψ ∨ ¬ψ]x)›
            by (rule GEN)
        qed
    }
  qed
  AOT-hence ‹𝓐(∀ z ([F]z ≡ [λz [F]z & ψ]z) →
    (¬∀ x ([λz [F]z & ψ]x ≡ [λz [F]z & ψ ∨ ¬ψ]x) ≡
     ¬∀ x ([F]x ≡ [λz [F]z & ψ ∨ ¬ψ]x)))› for ψ
    using nec−imp−act[THEN →E] by blast
  AOT-hence ‹𝓐∀ z ([F]z ≡ [λz [F]z & ψ]z) →
    𝓐(¬∀ x ([λz [F]z & ψ]x ≡ [λz [F]z & ψ ∨ ¬ψ]x) ≡
    ¬∀ x ([F]x ≡ [λz [F]z & ψ ∨ ¬ψ]x))› for ψ
    using act−cond[THEN →E] by blast
  AOT-hence Aux-D: ‹𝓐∀ z ([F]z ≡ [λz [F]z & ψ]z) →
    (𝓐¬∀ x ([λz [F]z & ψ]x ≡ [λz [F]z & ψ ∨ ¬ψ]x) ≡
     𝓐¬∀ x ([F]x ≡ [λz [F]z & ψ ∨ ¬ψ]x))› for ψ
    by (auto intro!: →I Act−Basic:5[THEN ≡E(1)] dest!: →E)

  AOT-have ‹¬𝓐q_0›
    apply (rule =_df I(2)[OF q_0-def])
     apply (fact log−prop−prop:2)
    by (fact AOT)
  AOT-hence q_0-prop-1: ‹𝓐¬q_0›
    using logic−actual−nec:1[axiom-inst, THEN ≡E(2)] by blast
  {
    AOT-assume 1: ‹𝓐∀ x([F]x ≡ [λz [F]z & q_0]x)›
    AOT-have 2: ‹◇∀ x([F]x ≡ [λz [F]z & q_0 ∨ ¬q_0]x)›
      using Aux-B[THEN →E, OF q_0-prop[THEN &E(1)]].
    AOT-have ‹𝓐¬∀ x([λz [F]z & q_0]x ≡ [λz [F]z & q_0 ∨ ¬q_0]x)›
      using Aux-C[THEN →E, OF q_0-prop-1].
    AOT-hence 3: ‹𝓐¬∀ x([F]x ≡ [λz [F]z & q_0 ∨ ¬q_0]x)›
      using Aux-D[THEN →E, OF 1, THEN ≡E(1)] by blast
    AOT-hence ‹𝓐¬∀ x([F]x ≡ [λz [F]z & q_0 ∨ ¬q_0]x) &
               ◇∀ x([F]x ≡ [λz [F]z & q_0 ∨ ¬q_0]x)›
      using 2 &I by blast
    AOT-hence ‹∃ G (𝓐¬∀ x ([F]x ≡ [G]x) & ◇∀ x([F]x ≡ [G]x))›
      by (rule ∃ I(1)) cqt:2[lambda]
  }
  moreover {
    AOT-assume 2: ‹¬𝓐∀ x([F]x ≡ [λz [F]z & q_0]x)›
    AOT-hence ‹𝓐¬∀ x([F]x ≡ [λz [F]z & q_0]x)›
      using logic−actual−nec:1[axiom-inst, THEN ≡E(2)] by blast
    AOT-hence ‹𝓐¬∀ x ([F]x ≡ [λz [F]z & q_0]x) & ◇∀ x([F]x ≡ [λz [F]z & q_0]x)›
      using Aux-A[THEN →E, OF q_0-prop[THEN &E(1)]] &I by blast
    AOT-hence ‹∃ G (𝓐¬∀ x ([F]x ≡ [G]x) & ◇∀ x([F]x ≡ [G]x))›
      by (rule ∃ I(1)) cqt:2[lambda]
  }
  ultimately AOT-show ‹∃ G (𝓐¬∀ x ([F]x ≡ [G]x) & ◇∀ x([F]x ≡ [G]x))›
    using ∨E(1)[OF exc−mid] →I by blast
qed

AOT-theorem oa−contingent:1: ‹O! ≠ A!›
proof(rule ≡_df I[OF =−infix]; rule raa−cor:2)
  fix x
  AOT-assume 1: ‹O! = A!›
  AOT-hence ‹[λx ◇E!x] = A!›
    by (rule =_df E(2)[OF AOT-ordinary, rotated]) cqt:2[lambda]
  AOT-hence ‹[λx ◇E!x] = [λx ¬◇E!x]›
```

```
    by (rule =dfE(2)[OF AOT-abstract, rotated]) cqt:2[lambda]
  moreover AOT-have ‹[λx ◇E!x]x ≡ ◇E!x›
    by (rule beta−C−meta[THEN →E]) cqt:2[lambda]
  ultimately AOT-have ‹[λx ¬◇E!x]x ≡ ◇E!x›
    using rule=E by fast
  moreover AOT-have ‹[λx ¬◇E!x]x ≡ ¬◇E!x›
    by (rule beta−C−meta[THEN →E]) cqt:2[lambda]
  ultimately AOT-have ‹◇E!x ≡ ¬◇E!x›
    using ≡E(6) Commutativity of ≡[THEN ≡E(1)] by blast
  AOT-thus (◇E!x ≡ ¬◇E!x) & ¬(◇E!x ≡ ¬◇E!x)
    using oth−class−taut:3:c &I by blast
qed

AOT-theorem oa−contingent:2: ‹O!x ≡ ¬A!x›
proof −
  AOT-have ‹O!x ≡ [λx ◇E!x]x›
    apply (rule ≡I; rule →I)
     apply (rule =dfE(2)[OF AOT-ordinary])
      apply cqt:2[lambda]
     apply argo
    apply (rule  =dfI(2)[OF AOT-ordinary])
     apply cqt:2[lambda]
    by argo
  also AOT-have ‹. . . ≡ ◇E!x›
    by (rule beta−C−meta[THEN →E]) cqt:2[lambda]
  also AOT-have ‹. . . ≡ ¬¬◇E!x›
    using oth−class−taut:3:b.
  also AOT-have ‹. . . ≡ ¬[λx ¬◇E!x]x›
    by (rule beta−C−meta[THEN →E,
            THEN oth−class−taut:4:b[THEN ≡E(1)], symmetric])
       cqt:2
  also AOT-have ‹. . . ≡ ¬A!x›
    apply (rule ≡I; rule →I)
     apply (rule =dfI(2)[OF AOT-abstract])
      apply cqt:2[lambda]
     apply argo
    apply (rule =dfE(2)[OF AOT-abstract])
     apply cqt:2[lambda]
    by argo
  finally show ?thesis.
qed

AOT-theorem oa−contingent:3: ‹A!x ≡ ¬O!x›
  by (AOT-subst ‹A!x› ‹¬¬A!x›)
     (auto simp add: oth−class−taut:3:b oa−contingent:2[THEN
          oth−class−taut:4:b[THEN ≡E(1)], symmetric])

AOT-theorem oa−contingent:4: ‹Contingent(O!)›
proof (rule thm−cont−prop:2[unvarify F, OF oa−exist:1, THEN ≡E(2)];
       rule &I)
  AOT-have ‹◇∃x E!x› using thm−cont−e:3 .
  AOT-hence ‹∃x ◇E!x› using BF◇[THEN →E] by blast
  then AOT-obtain a where ‹◇E!a› using ∃E[rotated] by blast
  AOT-hence ‹[λx ◇E!x]a›
    by (rule beta−C−meta[THEN →E, THEN ≡E(2), rotated]) cqt:2
  AOT-hence ‹O!a›
    by (rule =dfI(2)[OF AOT-ordinary, rotated]) cqt:2
  AOT-hence ‹∃x O!x› using ∃I by blast
  AOT-thus ‹◇∃x O!x› using T◇[THEN →E] by blast
next
  AOT-obtain a where ‹A!a›
    using A−objects[axiom-inst] ∃E[rotated] &E by blast
  AOT-hence ‹¬O!a› using oa−contingent:3[THEN ≡E(1)] by blast
```

```
  AOT-hence ‹∃ x ¬O!x› using ∃ I by fast
  AOT-thus ‹◇∃ x ¬O!x› using T◇[THEN →E] by blast
qed

AOT-theorem oa−contingent:5: ‹Contingent(A!)›
proof (rule thm−cont−prop:2[unvarify F, OF oa−exist:2, THEN ≡E(2)];
      rule &I)
  AOT-obtain a where ‹A!a›
    using A−objects[axiom-inst] ∃ E[rotated] &E by blast
  AOT-hence ‹∃ x A!x› using ∃ I by fast
  AOT-thus ‹◇∃ x A!x› using T◇[THEN →E] by blast
next
  AOT-have ‹◇∃ x E!x› using thm−cont−e:3 .
  AOT-hence ‹∃ x ◇E!x› using BF◇[THEN →E] by blast
  then AOT-obtain a where ‹◇E!a› using ∃ E[rotated] by blast
  AOT-hence ‹[λx ◇E!x]a›
    by (rule beta−C−meta[THEN →E, THEN ≡E(2), rotated]) cqt:2[lambda]
  AOT-hence ‹O!a›
    by (rule =dfI(2)[OF AOT-ordinary, rotated]) cqt:2[lambda]
  AOT-hence ‹¬A!a› using oa−contingent:2[THEN ≡E(1)] by blast
  AOT-hence ‹∃ x ¬A!x› using ∃ I by fast
  AOT-thus ‹◇∃ x ¬A!x› using T◇[THEN →E] by blast
qed

AOT-theorem oa−contingent:7: ‹O!⁻x ≡ ¬A!⁻x›
proof −
  AOT-have ‹O!x ≡ ¬A!x›
    using oa−contingent:2 by blast
  also AOT-have ‹. . . ≡ A!⁻x›
    using thm−relation−negation:1[symmetric, unvarify F, OF oa−exist:2].
  finally AOT-have 1: ‹O!x ≡ A!⁻x›.

  AOT-have ‹A!x ≡ ¬O!x›
    using oa−contingent:3 by blast
  also AOT-have ‹. . . ≡ O!⁻x›
    using thm−relation−negation:1[symmetric, unvarify F, OF oa−exist:1].
  finally AOT-have 2: ‹A!x ≡ O!⁻x›.

  AOT-show ‹O!⁻x ≡ ¬A!⁻x›
    using 1[THEN oth−class−taut:4:b[THEN ≡E(1)]]
        oa−contingent:3[of - x] 2[symmetric]
        ≡E(5) by blast
qed

AOT-theorem oa−contingent:6: ‹O!⁻ ≠ A!⁻›
proof (rule =−infix[THEN ≡dfI]; rule raa−cor:2)
  AOT-assume 1: ‹O!⁻ = A!⁻›
  fix x
  AOT-have ‹A!⁻x ≡ O!⁻x›
    apply (rule rule=E[rotated, OF 1])
    by (fact oth−class−taut:3:a)
  AOT-hence ‹A!⁻x ≡ ¬A!⁻x›
    using oa−contingent:7 ≡E by fast
  AOT-thus ‹(A!⁻x ≡ ¬A!⁻x) & ¬(A!⁻x ≡ ¬A!⁻x)›
    using oth−class−taut:3:c &I by blast
qed

AOT-theorem oa−contingent:8: ‹Contingent(O!⁻)›
  using thm−cont−prop:3[unvarify F, OF oa−exist:1, THEN ≡E(1),
        OF oa−contingent:4].

AOT-theorem oa−contingent:9: ‹Contingent(A!⁻)›
  using thm−cont−prop:3[unvarify F, OF oa−exist:2, THEN ≡E(1),
```

OF oa−contingent:5].

AOT-define *WeaklyContingent* :: ‹Π ⇒ φ› (‹*WeaklyContingent*′(-′)›)
df−cont−nec:
‹*WeaklyContingent*([F]) $\equiv_{df}$ *Contingent*([F]) & ∀ x (◊[F]x → □[F]x)›

AOT-theorem *cont−nec−fact1:1*:
‹*WeaklyContingent*([F]) ≡ *WeaklyContingent*([F]⁻)›
proof −
AOT-have ‹*WeaklyContingent*([F]) ≡ *Contingent*([F]) & ∀ x (◊[F]x → □[F]x)›
using *df−cont−nec*[*THEN* ≡*Df*] **by** *blast*
also AOT-have ‹... ≡ *Contingent*([F]⁻) & ∀ x (◊[F]x → □[F]x)›
apply (*rule oth−class−taut:8:f*[*THEN* ≡*E*(*2*)]; *rule* →*I*)
using *thm−cont−prop:3*.
also AOT-have ‹... ≡ *Contingent*([F]⁻) & ∀ x (◊[F]⁻x → □[F]⁻x)›
proof (*rule oth−class−taut:8:e*[*THEN* ≡*E*(*2*)];
rule →*I*; *rule* ≡*I*; *rule* →*I*; *rule GEN*; *rule* →*I*)
fix *x*
AOT-assume *0*: ‹∀ x (◊[F]x → □[F]x)›
AOT-assume *1*: ‹◊[F]⁻x›
AOT-have ‹◊¬[F]x›
by (*AOT-subst* (*reverse*) ‹¬[F]x› ‹[F]⁻x›)
(*auto simp add*: *thm−relation−negation:1 1*)
AOT-hence *2*: ‹¬□[F]x›
using *KBasic:11*[*THEN* ≡*E*(*2*)] **by** *blast*
AOT-show ‹□[F]⁻x›
proof (*rule raa−cor:1*)
AOT-assume *3*: ‹¬□[F]⁻x›
AOT-have ‹¬□¬[F]x›
by (*AOT-subst* (*reverse*) ‹¬[F]x› ‹[F]⁻x›)
(*auto simp add*: *thm−relation−negation:1 3*)
AOT-hence ‹◊[F]x›
using *conventions:5*[*THEN* $\equiv_{df}$*I*] **by** *simp*
AOT-hence ‹□[F]x› **using** *0* ∀ *E* →*E* **by** *fast*
AOT-thus ‹□[F]x & ¬□[F]x› **using** &*I 2* **by** *blast*
qed
next
fix *x*
AOT-assume *0*: ‹∀ x (◊[F]⁻x → □[F]⁻x)›
AOT-assume *1*: ‹◊[F]x›
AOT-have ‹◊¬[F]⁻x›
by (*AOT-subst* ‹¬[F]⁻x› ‹[F]x›)
(*auto simp*: *thm−relation−negation:2 1*)
AOT-hence *2*: ‹¬□[F]⁻x›
using *KBasic:11*[*THEN* ≡*E*(*2*)] **by** *blast*
AOT-show ‹□[F]x›
proof (*rule raa−cor:1*)
AOT-assume *3*: ‹¬□[F]x›
AOT-have ‹¬□¬[F]⁻x›
by (*AOT-subst* ‹¬[F]⁻x› ‹[F]x›)
(*auto simp add*: *thm−relation−negation:2 3*)
AOT-hence ‹◊[F]⁻x›
using *conventions:5*[*THEN* $\equiv_{df}$*I*] **by** *simp*
AOT-hence ‹□[F]⁻x› **using** *0* ∀ *E* →*E* **by** *fast*
AOT-thus ‹□[F]⁻x & ¬□[F]⁻x› **using** &*I 2* **by** *blast*
qed
qed
also AOT-have ‹... ≡ *WeaklyContingent*([F]⁻)›
using *df−cont−nec*[*THEN* ≡*Df*, *symmetric*] **by** *blast*
finally show *?thesis*.
qed

AOT-theorem *cont−nec−fact1:2*:

```
  ‹(WeaklyContingent([F]) & ¬WeaklyContingent([G])) → F ≠ G›
proof (rule →I; rule =−infix[THEN ≡df I]; rule raa−cor:2)
  AOT-assume 1: ‹WeaklyContingent([F]) & ¬WeaklyContingent([G])›
  AOT-hence ‹WeaklyContingent([F])› using &E by blast
  moreover AOT-assume ‹F = G›
  ultimately AOT-have ‹WeaklyContingent([G])›
    using rule=E by blast
  AOT-thus ‹WeaklyContingent([G]) & ¬WeaklyContingent([G])›
    using 1 &I &E by blast
qed

AOT-theorem cont−nec−fact2:1: ‹WeaklyContingent(O!)›
proof (rule df−cont−nec[THEN ≡df I]; rule &I)
  AOT-show ‹Contingent(O!)›
    using oa−contingent:4.
next
  AOT-show ‹∀x (◇[O!]x → □[O!]x)›
    apply (rule GEN; rule →I)
    using oa−facts:5[THEN ≡E(1)] by blast
qed


AOT-theorem cont−nec−fact2:2: ‹WeaklyContingent(A!)›
proof (rule df−cont−nec[THEN ≡df I]; rule &I)
  AOT-show ‹Contingent(A!)›
    using oa−contingent:5.
next
  AOT-show ‹∀x (◇[A!]x → □[A!]x)›
    apply (rule GEN; rule →I)
    using oa−facts:6[THEN ≡E(1)] by blast
qed

AOT-theorem cont−nec−fact2:3: ‹¬WeaklyContingent(E!)›
proof (rule df−cont−nec[THEN ≡Df,
                        THEN oth−class−taut:4:b[THEN ≡E(1)],
                        THEN ≡E(2)];
      rule DeMorgan(1)[THEN ≡E(2)]; rule ∨I(2); rule raa−cor:2)
  AOT-have ‹◇∃x (E!x & ¬𝓐E!x)› using qml:4[axiom-inst].
  AOT-hence ‹∃x ◇(E!x & ¬𝓐E!x)› using BF◇[THEN →E] by blast
  then AOT-obtain a where ‹◇(E!a & ¬𝓐E!a)› using ∃E[rotated] by blast
  AOT-hence 1: ‹◇E!a & ◇¬𝓐E!a› using KBasic2:3[THEN →E] by simp
  moreover AOT-assume ‹∀x (◇[E!]x → □[E!]x)›
  ultimately AOT-have ‹□E!a› using &E ∀E →E by fast
  AOT-hence ‹𝓐E!a› using nec−imp−act[THEN →E] by blast
  AOT-hence ‹□𝓐E!a› using qml−act:1[axiom-inst, THEN →E] by blast
  moreover AOT-have ‹¬□𝓐E!a›
    using KBasic:11[THEN ≡E(2)] 1[THEN &E(2)] by meson
  ultimately AOT-have ‹□𝓐E!a & ¬□𝓐E!a› using &I by blast
  AOT-thus ‹p & ¬p› for p using raa−cor:1 by blast
qed

AOT-theorem cont−nec−fact2:4: ‹¬WeaklyContingent(L)›
  apply (rule df−cont−nec[THEN ≡Df,
                          THEN oth−class−taut:4:b[THEN ≡E(1)],
                          THEN ≡E(2)];
      rule DeMorgan(1)[THEN ≡E(2)]; rule ∨I(1))
  apply (rule contingent−properties:4
              [THEN ≡Df,
               THEN oth−class−taut:4:b[THEN ≡E(1)],
               THEN ≡E(2)])
  apply (rule DeMorgan(1)[THEN ≡E(2)];
         rule ∨I(2);
         rule useful−tautologies:2[THEN →E])
```

```
  using thm−noncont−e−e:3[THEN contingent−properties:3[THEN ≡df E]].

AOT-theorem cont−nec−fact2:5: ‹O! ≠ E! & O! ≠ E!⁻ & O! ≠ L & O! ≠ L⁻›
proof −
  AOT-have 1: ‹L↓›
    by (rule =df I(2)[OF L-def]) cqt:2[lambda]+
  {
    fix φ and Π Π' :: ‹<κ>›
    AOT-have A: ‹¬(φ{Π'} ≡ φ{Π})› if ‹φ{Π}› and ‹¬φ{Π'}›
    proof (rule raa−cor:2)
      AOT-assume ‹φ{Π'} ≡ φ{Π}›
      AOT-hence ‹φ{Π'}› using that(1) ≡E by blast
      AOT-thus ‹φ{Π'} & ¬φ{Π'}› using that(2) &I by blast
    qed
    AOT-have ‹Π' ≠ Π› if ‹Π↓› and ‹Π'↓› and ‹φ{Π}› and ‹¬φ{Π'}›
      using pos−not−equiv−ne:4[unvarify F G, THEN →E,
                                OF that(1,2), OF A[OF that(3, 4)]].
  } note 0 = this
  show ?thesis
    apply(safe intro!: &I; rule 0)
    apply cqt:2
    using oa−exist:1 apply blast
    using cont−nec−fact2:3 apply fast
    apply (rule useful−tautologies:2[THEN →E])
    using cont−nec−fact2:1 apply fast
    using rel−neg−T:3 apply fast
    using oa−exist:1 apply blast
    using cont−nec−fact1:1[THEN oth−class−taut:4:b[THEN ≡E(1)],
          THEN ≡E(1), rotated, OF cont−nec−fact2:3] apply fast
    apply (rule useful−tautologies:2[THEN →E])
    using cont−nec−fact2:1 apply blast
    apply (rule =df I(2)[OF L-def]; cqt:2[lambda])
    using oa−exist:1 apply fast
    using cont−nec−fact2:4 apply fast
    apply (rule useful−tautologies:2[THEN →E])
    using cont−nec−fact2:1 apply fast
    using rel−neg−T:3 apply fast
    using oa−exist:1 apply fast
     apply (rule cont−nec−fact1:1[unvarify F,
                 THEN oth−class−taut:4:b[THEN ≡E(1)],
                 THEN ≡E(1), rotated, OF cont−nec−fact2:4])
    apply (rule =df I(2)[OF L-def]; cqt:2[lambda])
    apply (rule useful−tautologies:2[THEN →E])
    using cont−nec−fact2:1 by blast
qed

AOT-theorem cont−nec−fact2:6: ‹A! ≠ E! & A! ≠ E!⁻ & A! ≠ L & A! ≠ L⁻›
proof −
  AOT-have 1: ‹L↓›
    by (rule =df I(2)[OF L-def]) cqt:2[lambda]+
  {
    fix φ and Π Π' :: ‹<κ>›
    AOT-have A: ‹¬(φ{Π'} ≡ φ{Π})› if ‹φ{Π}› and ‹¬φ{Π'}›
    proof (rule raa−cor:2)
      AOT-assume ‹φ{Π'} ≡ φ{Π}›
      AOT-hence ‹φ{Π'}› using that(1) ≡E by blast
      AOT-thus ‹φ{Π'} & ¬φ{Π'}› using that(2) &I by blast
    qed
    AOT-have ‹Π' ≠ Π› if ‹Π↓› and ‹Π'↓› and ‹φ{Π}› and ‹¬φ{Π'}›
      using pos−not−equiv−ne:4[unvarify F G, THEN →E,
            OF that(1,2), OF A[OF that(3, 4)]].
  } note 0 = this
  show ?thesis
```

apply(*safe intro*!: &*I*; *rule 0*)
apply *cqt*:*2*
using *oa−exist*:*2* **apply** *blast*
using *cont−nec−fact2*:*3* **apply** *fast*
apply (*rule useful−tautologies*:*2*[*THEN* →*E*])
using *cont−nec−fact2*:*2* **apply** *fast*
using *rel−neg−T*:*3* **apply** *fast*
using *oa−exist*:*2* **apply** *blast*
using *cont−nec−fact1*:*1*[*THEN oth−class−taut*:*4*:*b*[*THEN* ≡*E*(*1*)],
THEN ≡*E*(*1*), *rotated*, *OF cont−nec−fact2*:*3*] **apply** *fast*
apply (*rule useful−tautologies*:*2*[*THEN* →*E*])
using *cont−nec−fact2*:*2* **apply** *blast*
apply (*rule* $=_{df}$*I*(*2*)[*OF L-def*]; *cqt*:*2*[*lambda*])
using *oa−exist*:*2* **apply** *fast*
using *cont−nec−fact2*:*4* **apply** *fast*
apply (*rule useful−tautologies*:*2*[*THEN* →*E*])
using *cont−nec−fact2*:*2* **apply** *fast*
using *rel−neg−T*:*3* **apply** *fast*
using *oa−exist*:*2* **apply** *fast*
apply (*rule cont−nec−fact1*:*1*[*unvarify F*,
THEN oth−class−taut:*4*:*b*[*THEN* ≡*E*(*1*)],
THEN ≡*E*(*1*), *rotated*, *OF cont−nec−fact2*:*4*])
apply (*rule* $=_{df}$*I*(*2*)[*OF L-def*]; *cqt*:*2*[*lambda*])
apply (*rule useful−tautologies*:*2*[*THEN* →*E*])
using *cont−nec−fact2*:*2* **by** *blast*
qed

AOT-define *necessary-or-contingently-false* :: ‹φ ⇒ φ› (‹**Δ**-› [*49*] *54*)
‹$\mathbf{\Delta}p \equiv_{df} \Box p \vee (\neg\mathcal{A}p\ \&\ \Diamond p)$›

AOT-theorem *sixteen*:
shows ‹$\exists F_1\exists F_2\exists F_3\exists F_4\exists F_5\exists F_6\exists F_7\exists F_8\exists F_9\exists F_{10}\exists F_{11}\exists F_{12}\exists F_{13}\exists F_{14}\exists F_{15}\exists F_{16}$ (
«F_1::<κ>» $\neq F_2$ & $F_1 \neq F_3$ & $F_1 \neq F_4$ & $F_1 \neq F_5$ & $F_1 \neq F_6$ & $F_1 \neq F_7$ &
$F_1 \neq F_8$ & $F_1 \neq F_9$ & $F_1 \neq F_{10}$ & $F_1 \neq F_{11}$ & $F_1 \neq F_{12}$ & $F_1 \neq F_{13}$ &
$F_1 \neq F_{14}$ & $F_1 \neq F_{15}$ & $F_1 \neq F_{16}$ &
$F_2 \neq F_3$ & $F_2 \neq F_4$ & $F_2 \neq F_5$ & $F_2 \neq F_6$ & $F_2 \neq F_7$ & $F_2 \neq F_8$ &
$F_2 \neq F_9$ & $F_2 \neq F_{10}$ & $F_2 \neq F_{11}$ & $F_2 \neq F_{12}$ & $F_2 \neq F_{13}$ & $F_2 \neq F_{14}$ &
$F_2 \neq F_{15}$ & $F_2 \neq F_{16}$ &
$F_3 \neq F_4$ & $F_3 \neq F_5$ & $F_3 \neq F_6$ & $F_3 \neq F_7$ & $F_3 \neq F_8$ & $F_3 \neq F_9$ & $F_3 \neq F_{10}$ &
$F_3 \neq F_{11}$ & $F_3 \neq F_{12}$ & $F_3 \neq F_{13}$ & $F_3 \neq F_{14}$ & $F_3 \neq F_{15}$ & $F_3 \neq F_{16}$ &
$F_4 \neq F_5$ & $F_4 \neq F_6$ & $F_4 \neq F_7$ & $F_4 \neq F_8$ & $F_4 \neq F_9$ & $F_4 \neq F_{10}$ & $F_4 \neq F_{11}$ &
$F_4 \neq F_{12}$ & $F_4 \neq F_{13}$ & $F_4 \neq F_{14}$ & $F_4 \neq F_{15}$ & $F_4 \neq F_{16}$ &
$F_5 \neq F_6$ & $F_5 \neq F_7$ & $F_5 \neq F_8$ & $F_5 \neq F_9$ & $F_5 \neq F_{10}$ & $F_5 \neq F_{11}$ & $F_5 \neq F_{12}$ &
$F_5 \neq F_{13}$ & $F_5 \neq F_{14}$ & $F_5 \neq F_{15}$ & $F_5 \neq F_{16}$ &
$F_6 \neq F_7$ & $F_6 \neq F_8$ & $F_6 \neq F_9$ & $F_6 \neq F_{10}$ & $F_6 \neq F_{11}$ & $F_6 \neq F_{12}$ & $F_6 \neq F_{13}$ &
$F_6 \neq F_{14}$ & $F_6 \neq F_{15}$ & $F_6 \neq F_{16}$ &
$F_7 \neq F_8$ & $F_7 \neq F_9$ & $F_7 \neq F_{10}$ & $F_7 \neq F_{11}$ & $F_7 \neq F_{12}$ & $F_7 \neq F_{13}$ & $F_7 \neq F_{14}$ &
$F_7 \neq F_{15}$ & $F_7 \neq F_{16}$ &
$F_8 \neq F_9$ & $F_8 \neq F_{10}$ & $F_8 \neq F_{11}$ & $F_8 \neq F_{12}$ & $F_8 \neq F_{13}$ & $F_8 \neq F_{14}$ & $F_8 \neq F_{15}$ &
$F_8 \neq F_{16}$ &
$F_9 \neq F_{10}$ & $F_9 \neq F_{11}$ & $F_9 \neq F_{12}$ & $F_9 \neq F_{13}$ & $F_9 \neq F_{14}$ & $F_9 \neq F_{15}$ & $F_9 \neq F_{16}$ &
$F_{10} \neq F_{11}$ & $F_{10} \neq F_{12}$ & $F_{10} \neq F_{13}$ & $F_{10} \neq F_{14}$ & $F_{10} \neq F_{15}$ & $F_{10} \neq F_{16}$ &
$F_{11} \neq F_{12}$ & $F_{11} \neq F_{13}$ & $F_{11} \neq F_{14}$ & $F_{11} \neq F_{15}$ & $F_{11} \neq F_{16}$ &
$F_{12} \neq F_{13}$ & $F_{12} \neq F_{14}$ & $F_{12} \neq F_{15}$ & $F_{12} \neq F_{16}$ &
$F_{13} \neq F_{14}$ & $F_{13} \neq F_{15}$ & $F_{13} \neq F_{16}$ &
$F_{14} \neq F_{15}$ & $F_{14} \neq F_{16}$ &
$F_{15} \neq F_{16}$)›
proof −
AOT-have *Delta-pos*: ‹$\mathbf{\Delta}\varphi \rightarrow \Diamond\varphi$› **for** φ
proof(*rule* →*I*)
AOT-assume ‹$\mathbf{\Delta}\varphi$›
AOT-hence ‹$\Box\varphi \vee (\neg\mathcal{A}\varphi\ \&\ \Diamond\varphi)$›
using $\equiv_{df}$*E*[*OF necessary-or-contingently-false*] **by** *blast*

```
  moreover {
    AOT-assume ‹□φ›
    AOT-hence ‹◇φ›
      by (metis B◇ T◇ vdash−properties:10)
  }
  moreover {
    AOT-assume ‹¬𝓐φ & ◇φ›
    AOT-hence ‹◇φ›
      using &E by blast
  }
  ultimately AOT-show ‹◇φ›
    by (metis ∨E(2) raa−cor:1)
qed

AOT-have act-and-not-nec-not-delta: ‹¬Δφ› if ‹𝓐φ› and ‹¬□φ› for φ
  using ≡df E &E(1) ∨E(2) necessary-or-contingently-false
        raa−cor:3 that(1,2) by blast
AOT-have act-and-pos-not-not-delta: ‹¬Δφ› if ‹𝓐φ› and ‹◇¬φ› for φ
  using KBasic:11 act-and-not-nec-not-delta ≡E(2) that(1,2) by blast
AOT-have impossible-delta: ‹¬Δφ› if ‹¬◇φ› for φ
  using Delta-pos modus−tollens:1 that by blast
AOT-have not-act-and-pos-delta: ‹Δφ› if ‹¬𝓐φ› and ‹◇φ› for φ
  by (meson ≡df I &I ∨I(2) necessary-or-contingently-false that(1,2))
AOT-have nec-delta: ‹Δφ› if ‹□φ› for φ
  using ≡df I ∨I(1) necessary-or-contingently-false that by blast

AOT-obtain a where a-prop: ‹A!a›
  using A−objects[axiom-inst] ∃E[rotated] &E by blast
AOT-obtain b where b-prop: ‹◇[E!]b & ¬𝓐[E!]b›
  using pos−not−pna:3 using ∃E[rotated] by blast

AOT-have b-ord: ‹[O!]b›
proof(rule =df I(2)[OF AOT-ordinary])
  AOT-show ‹[λx ◇[E!]x]↓› by cqt:2[lambda]
next
  AOT-show ‹[λx ◇[E!]x]b›
  proof (rule β←C(1); (cqt:2[lambda])?)
    AOT-show ‹b↓› by (rule cqt:2[const-var][axiom-inst])
    AOT-show ‹◇[E!]b› by (fact b-prop[THEN &E(1)])
  qed
qed

AOT-have nec-not-L-neg: ‹□¬[L⁻]x› for x
  using thm−noncont−e−e:2 contingent−properties:2[THEN ≡df E] &E
        CBF[THEN →E] ∀E by blast
AOT-have nec-L: ‹□[L]x› for x
  using thm−noncont−e−e:1 contingent−properties:1[THEN ≡df E]
    CBF[THEN →E] ∀E by blast

AOT-have act-ord-b: ‹𝓐[O!]b›
  using b-ord ≡E(1) oa−facts:7 by blast
AOT-have delta-ord-b: ‹Δ[O!]b›
  by (meson ≡df I b-ord ∨I(1) necessary-or-contingently-false
            oa−facts:1 →E)
AOT-have not-act-ord-a: ‹¬𝓐[O!]a›
  by (meson a-prop ≡E(1) ≡E(3) oa−contingent:3 oa−facts:7)
AOT-have not-delta-ord-a: ‹¬Δ[O!]a›
  by (metis Delta-pos ≡E(4) not-act-ord-a oa−facts:3 oa−facts:7
            reductio−aa:1 →E)

AOT-have not-act-abs-b: ‹¬𝓐[A!]b›
  by (meson b-ord ≡E(1) ≡E(3) oa−contingent:2 oa−facts:8)
AOT-have not-delta-abs-b: ‹¬Δ[A!]b›
```

proof(*rule raa−cor*:*2*)
  AOT-assume ‹$\mathbf{\Delta}[A!]b$›
  AOT-hence ‹$\Diamond[A!]b$›
    by (*metis Delta-pos vdash−properties*:*10*)
  AOT-thus ‹$[A!]b$ & $\neg[A!]b$›
    by (*metis b-ord* &*I* ≡*E*(*1*) *oa−contingent*:*2*
        oa−facts:*4* →*E*)
qed
AOT-have *act-abs-a*: ‹$\mathcal{A}[A!]a$›
  using *a-prop* ≡*E*(*1*) *oa−facts*:*8* **by** *blast*
AOT-have *delta-abs-a*: ‹$\mathbf{\Delta}[A!]a$›
  by (*metis* $\equiv_{df}I$ *a-prop oa−facts*:*2* →*E* ∨*I*(*1*)
        necessary-or-contingently-false)

AOT-have *not-act-concrete-b*: ‹$\neg\mathcal{A}[E!]b$›
  using *b-prop* &*E*(*2*) **by** *blast*
AOT-have *delta-concrete-b*: ‹$\mathbf{\Delta}[E!]b$›
proof (*rule* $\equiv_{df}I$[*OF necessary-or-contingently-false*];
    rule ∨*I*(*2*); *rule* &*I*)
  AOT-show ‹$\neg\mathcal{A}[E!]b$› **using** *b-prop* &*E*(*2*) **by** *blast*
next
  AOT-show ‹$\Diamond[E!]b$› **using** *b-prop* &*E*(*1*) **by** *blast*
qed
AOT-have *not-act-concrete-a*: ‹$\neg\mathcal{A}[E!]a$›
proof (*rule raa−cor*:*2*)
  AOT-assume ‹$\mathcal{A}[E!]a$›
  AOT-hence *1*: ‹$\Diamond[E!]a$› **by** (*metis Act−Sub*:*3* →*E*)
  AOT-have ‹$[A!]a$› **by** (*simp add*: *a-prop*)
  AOT-hence ‹$[\lambda x\ \neg\Diamond[E!]x]a$›
    by (*rule* $=_{df}E(2)$[*OF AOT-abstract*, *rotated*]) *cqt*:*2*
  AOT-hence ‹$\neg\Diamond[E!]a$› **using** β→*C*(*1*) **by** *blast*
  AOT-thus ‹$\Diamond[E!]a$ & $\neg\Diamond[E!]a$› **using** *1* &*I* **by** *blast*
qed
AOT-have *not-delta-concrete-a*: ‹$\neg\mathbf{\Delta}[E!]a$›
proof (*rule raa−cor*:*2*)
  AOT-assume ‹$\mathbf{\Delta}[E!]a$›
  AOT-hence *1*: ‹$\Diamond[E!]a$› **by** (*metis Delta-pos vdash−properties*:*10*)
  AOT-have ‹$[A!]a$› **by** (*simp add*: *a-prop*)
  AOT-hence ‹$[\lambda x\ \neg\Diamond[E!]x]a$›
    by (*rule* $=_{df}E(2)$[*OF AOT-abstract*, *rotated*]) *cqt*:*2*[*lambda*]
  AOT-hence ‹$\neg\Diamond[E!]a$› **using** β→*C*(*1*) **by** *blast*
  AOT-thus ‹$\Diamond[E!]a$ & $\neg\Diamond[E!]a$› **using** *1* &*I* **by** *blast*
qed

AOT-have *not-act-q-zero*: ‹$\neg\mathcal{A}q_0$›
  by (*meson log−prop−prop*:*2 pos−not−pna*:*1*
        q_0*-def reductio−aa*:*1 rule−id−df*:*2*:*a*[*zero*])
AOT-have *delta-q-zero*: ‹$\mathbf{\Delta}q_0$›
proof(*rule* $\equiv_{df}I$[*OF necessary-or-contingently-false*];
    rule ∨*I*(*2*); *rule* &*I*)
  AOT-show ‹$\neg\mathcal{A}q_0$› **using** *not-act-q-zero*.
  AOT-show ‹$\Diamond q_0$› **by** (*meson* &*E*(*1*) q_0*-prop*)
qed
AOT-have *act-not-q-zero*: ‹$\mathcal{A}\neg q_0$›
  using *Act−Basic*:*1* ∨*E*(*2*) *not-act-q-zero* **by** *blast*
AOT-have *not-delta-not-q-zero*: ‹$\neg\mathbf{\Delta}\neg q_0$›
  using $\equiv_{df}E$ *conventions*:*5 Act−Basic*:*1 act-and-not-nec-not-delta*
    &*E*(*1*) ∨*E*(*2*) *not-act-q-zero* q_0*-prop* **by** *blast*

AOT-have ‹$[L^-]\downarrow$› **by** (*simp add*: *rel−neg−T*:*3*)
moreover AOT-have ‹$\neg\mathcal{A}[L^-]b$ & $\neg\mathbf{\Delta}[L^-]b$ & $\neg\mathcal{A}[L^-]a$ & $\neg\mathbf{\Delta}[L^-]a$›
proof (*safe intro*!: &*I*)
  AOT-show ‹$\neg\mathcal{A}[L^-]b$›

by (*meson* ≡*E*(*1*) *logic−actual−nec*:*1*[*axiom-inst*] *nec−imp−act*
nec-not-L-neg →*E*)
AOT-show ‹¬**Δ**[L^-]*b*›
by (*meson Delta-pos KBasic2*:*1* ≡*E*(*1*)
modus−tollens:*1 nec-not-L-neg*)
AOT-show ‹¬**𝓐**[L^-]*a*›
by (*meson* ≡*E*(*1*) *logic−actual−nec*:*1*[*axiom-inst*]
nec−imp−act nec-not-L-neg →*E*)
AOT-show ‹¬**Δ**[L^-]*a*›
using *Delta-pos KBasic2*:*1* ≡*E*(*1*) *modus−tollens*:*1*
nec-not-L-neg **by** *blast*
qed
ultimately AOT-obtain F_0 **where** ‹¬**𝓐**[F_0]*b* & ¬**Δ**[F_0]*b* & ¬**𝓐**[F_0]*a* & ¬**Δ**[F_0]*a*›
using ∃*I*(*1*)[*rotated*, *THEN* ∃*E*[*rotated*]] **by** *fastforce*
AOT-hence ‹¬**𝓐**[F_0]*b*› **and** ‹¬**Δ**[F_0]*b*› **and** ‹¬**𝓐**[F_0]*a*› **and** ‹¬**Δ**[F_0]*a*›
using &*E* **by** *blast*+
note *props* = *this*

let *?Π* = «[λ*y* [*A*!]*y* & q_0]»
AOT-modally-strict {
AOT-have ‹[«*?Π*»]↓› **by** *cqt*:*2*[*lambda*]
} note *1* = *this*
moreover AOT-have ‹¬**𝓐**[«*?Π*»]*b* & ¬**Δ**[«*?Π*»]*b* & ¬**𝓐**[«*?Π*»]*a* & **Δ**[«*?Π*»]*a*›
proof (*safe intro*!: &*I*; *AOT-subst* ‹[λ*y A*!*y* & q_0]*x*› ‹*A*!*x* & q_0› **for**: *x*)
AOT-show ‹¬**𝓐**([*A*!]*b* & q_0)›
using *Act−Basic*:*2* &*E*(*1*) ≡*E*(*1*) *not-act-abs-b raa−cor*:*3* **by** *blast*
next AOT-show ‹¬**Δ**([*A*!]*b* & q_0)›
by (*metis Delta-pos KBasic2*:*3* &*E*(*1*) ≡*E*(*4*) *not-act-abs-b*
oa−facts:*4 oa−facts*:*8 raa−cor*:*3* →*E*)
next AOT-show ‹¬**𝓐**([*A*!]*a* & q_0)›
using *Act−Basic*:*2* &*E*(*2*) ≡*E*(*1*) *not-act-q-zero*
raa−cor:*3* **by** *blast*
next AOT-show ‹**Δ**([*A*!]*a* & q_0)›
proof (*rule not-act-and-pos-delta*)
AOT-show ‹¬**𝓐**([*A*!]*a* & q_0)›
using *Act−Basic*:*2* &*E*(*2*) ≡*E*(*4*) *not-act-q-zero*
raa−cor:*3* **by** *blast*
next AOT-show ‹◇([*A*!]*a* & q_0)›
by (*metis* &*I* →*E Delta-pos KBasic*:*16* &*E*(*1*) *delta-abs-a*
≡*E*(*1*) *oa−facts*:*6* q_0*-prop*)
qed
qed(*auto simp*: *beta−C−meta*[*THEN* →*E*, *OF 1*])
ultimately AOT-obtain F_1 **where** ‹¬**𝓐**[F_1]*b* & ¬**Δ**[F_1]*b* & ¬**𝓐**[F_1]*a* & **Δ**[F_1]*a*›
using ∃*I*(*1*)[*rotated*, *THEN* ∃*E*[*rotated*]] **by** *fastforce*
AOT-hence ‹¬**𝓐**[F_1]*b*› **and** ‹¬**Δ**[F_1]*b*› **and** ‹¬**𝓐**[F_1]*a*› **and** ‹**Δ**[F_1]*a*›
using &*E* **by** *blast*+
note *props* = *props this*

let *?Π* = «[λ*y* [*A*!]*y* & ¬q_0]»
AOT-modally-strict {
AOT-have ‹[«*?Π*»]↓› **by** *cqt*:*2*[*lambda*]
} note *1* = *this*
moreover AOT-have ‹¬**𝓐**[«*?Π*»]*b* & ¬**Δ**[«*?Π*»]*b* & **𝓐**[«*?Π*»]*a* & ¬**Δ**[«*?Π*»]*a*›
proof (*safe intro*!: &*I*; *AOT-subst* ‹[λ*y A*!*y* & ¬q_0]*x*› ‹*A*!*x* & ¬q_0› **for**: *x*)
AOT-show ‹¬**𝓐**([*A*!]*b* & ¬q_0)›
using *Act−Basic*:*2* &*E*(*1*) ≡*E*(*1*) *not-act-abs-b raa−cor*:*3* **by** *blast*
next AOT-show ‹¬**Δ**([*A*!]*b* & ¬q_0)›
by (*meson RM*◇ *Delta-pos Conjunction Simplification*(*1*) ≡*E*(*4*)
modus−tollens:*1 not-act-abs-b oa−facts*:*4 oa−facts*:*8*)
next AOT-show ‹**𝓐**([*A*!]*a* & ¬q_0)›
by (*metis Act−Basic*:*1 Act−Basic*:*2 act-abs-a* &*I* ∨*E*(*2*)
≡*E*(*3*) *not-act-q-zero raa−cor*:*3*)
next AOT-show ‹¬**Δ**([*A*!]*a* & ¬q_0)›

proof (*rule act-and-not-nec-not-delta*)
  AOT-show ‹$\mathcal{A}([A!]a \mathbin{\&} \neg q_0)$›
    by (*metis Act−Basic:1 Act−Basic:2 act-abs-a &I ∨E(2)*
      ≡E(3) not-act-q-zero raa−cor:3)
  next
  AOT-show ‹$\neg\Box([A!]a \mathbin{\&} \neg q_0)$›
    by (*metis KBasic2:1 KBasic:3 &E(1) &E(2) ≡E(4)*
      q_0-prop raa−cor:3)
  qed
qed(*auto simp: beta−C−meta[THEN →E, OF 1]*)
ultimately AOT-obtain F_2 **where** ‹$\neg\mathcal{A}[F_2]b \mathbin{\&} \neg\mathbf{\Delta}[F_2]b \mathbin{\&} \mathcal{A}[F_2]a \mathbin{\&} \neg\mathbf{\Delta}[F_2]a$›
  using *∃I(1)[rotated, THEN ∃E[rotated]]* **by** *fastforce*
AOT-hence ‹$\neg\mathcal{A}[F_2]b$› **and** ‹$\neg\mathbf{\Delta}[F_2]b$› **and** ‹$\mathcal{A}[F_2]a$› **and** ‹$\neg\mathbf{\Delta}[F_2]a$›
  using *&E* **by** *blast+*
note *props = props this*

AOT-have *abstract-prop*: ‹$\neg\mathcal{A}[A!]b \mathbin{\&} \neg\mathbf{\Delta}[A!]b \mathbin{\&} \mathcal{A}[A!]a \mathbin{\&} \mathbf{\Delta}[A!]a$›
  using *act-abs-a &I delta-abs-a not-act-abs-b not-delta-abs-b*
  by *presburger*
then AOT-obtain F_3 **where** ‹$\neg\mathcal{A}[F_3]b \mathbin{\&} \neg\mathbf{\Delta}[F_3]b \mathbin{\&} \mathcal{A}[F_3]a \mathbin{\&} \mathbf{\Delta}[F_3]a$›
  using *∃I(1)[rotated, THEN ∃E[rotated]] oa−exist:2* **by** *fastforce*
AOT-hence ‹$\neg\mathcal{A}[F_3]b$› **and** ‹$\neg\mathbf{\Delta}[F_3]b$› **and** ‹$\mathcal{A}[F_3]a$› **and** ‹$\mathbf{\Delta}[F_3]a$›
  using *&E* **by** *blast+*
note *props = props this*

AOT-have ‹$\neg\mathcal{A}[E!]b \mathbin{\&} \mathbf{\Delta}[E!]b \mathbin{\&} \neg\mathcal{A}[E!]a \mathbin{\&} \neg\mathbf{\Delta}[E!]a$›
  by (*meson &I delta-concrete-b not-act-concrete-a*
    not-act-concrete-b not-delta-concrete-a)
then AOT-obtain F_4 **where** ‹$\neg\mathcal{A}[F_4]b \mathbin{\&} \mathbf{\Delta}[F_4]b \mathbin{\&} \neg\mathcal{A}[F_4]a \mathbin{\&} \neg\mathbf{\Delta}[F_4]a$›
  using *∃I(1)[rotated, THEN ∃E[rotated]]*
  by *fastforce*
AOT-hence ‹$\neg\mathcal{A}[F_4]b$› **and** ‹$\mathbf{\Delta}[F_4]b$› **and** ‹$\neg\mathcal{A}[F_4]a$› **and** ‹$\neg\mathbf{\Delta}[F_4]a$›
  using *&E* **by** *blast+*
note *props = props this*

AOT-modally-strict {
  AOT-have ‹$[\lambda y\ q_0]\downarrow$› **by** *cqt:2[lambda]*
} note *1 = this*
moreover AOT-have ‹$\neg\mathcal{A}[\lambda y\ q_0]b \mathbin{\&} \mathbf{\Delta}[\lambda y\ q_0]b \mathbin{\&} \neg\mathcal{A}[\lambda y\ q_0]a \mathbin{\&} \mathbf{\Delta}[\lambda y\ q_0]a$›
  by (*safe intro!: &I; AOT-subst* ‹$[\lambda y\ q_0]b$› ‹q_0› **for**: *b*)
    (*auto simp: not-act-q-zero delta-q-zero beta−C−meta[THEN →E, OF 1]*)
ultimately AOT-obtain F_5 **where** ‹$\neg\mathcal{A}[F_5]b \mathbin{\&} \mathbf{\Delta}[F_5]b \mathbin{\&} \neg\mathcal{A}[F_5]a \mathbin{\&} \mathbf{\Delta}[F_5]a$›
  using *∃I(1)[rotated, THEN ∃E[rotated]]*
  by *fastforce*
AOT-hence ‹$\neg\mathcal{A}[F_5]b$› **and** ‹$\mathbf{\Delta}[F_5]b$› **and** ‹$\neg\mathcal{A}[F_5]a$› **and** ‹$\mathbf{\Delta}[F_5]a$›
  using *&E* **by** *blast+*
note *props = props this*

let *?Π* = «$[\lambda y\ [E!]y \vee ([A!]y \mathbin{\&} \neg q_0)]$»
AOT-modally-strict {
  AOT-have ‹[«?Π»]↓› **by** *cqt:2[lambda]*
} note *1 = this*
moreover AOT-have ‹¬𝓐[«?Π»]b & 𝚫[«?Π»]b & 𝓐[«?Π»]a & ¬𝚫[«?Π»]a›
proof(*safe intro!: &I;*
    AOT-subst ‹$[\lambda y\ E!y \vee (A!y \mathbin{\&} \neg q_0)]x$› ‹$E!x \vee (A!x \mathbin{\&} \neg q_0)$› **for**: *x*)
  AOT-have ‹$\mathcal{A}\neg([A!]b \mathbin{\&} \neg q_0)$›
    by (*metis Act−Basic:1 Act−Basic:2 abstract-prop &E(1) ∨E(2)*
      ≡E(1) raa−cor:3)
  moreover AOT-have ‹$\neg\mathcal{A}[E!]b$›
    using *b-prop &E(2)* **by** *blast*
  ultimately AOT-have *2*: ‹$\mathcal{A}(\neg[E!]b \mathbin{\&} \neg([A!]b \mathbin{\&} \neg q_0))$›
    by (*metis Act−Basic:2 Act−Sub:1 &I ≡E(3) raa−cor:1*)
  AOT-have ‹$\mathcal{A}\neg([E!]b \vee ([A!]b \mathbin{\&} \neg q_0))$›

```
    by (AOT-subst ‹¬([E!]b ∨ ([A!]b & ¬q₀))› ‹¬[E!]b & ¬([A!]b & ¬q₀)›)
       (auto simp: oth−class−taut:5:d 2)
  AOT-thus ‹¬𝓐([E!]b ∨ ([A!]b & ¬q₀))›
    by (metis ¬¬I Act−Sub:1 ≡E(4))
next
  AOT-show ‹Δ([E!]b ∨ ([A!]b & ¬q₀))›
  proof (rule not-act-and-pos-delta)
    AOT-show ‹¬𝓐([E!]b ∨ ([A!]b & ¬q₀))›
      by (metis Act−Basic:2 Act−Basic:9 ∨E(2) raa−cor:3
                Conjunction Simplification(1) ≡E(4)
                modus−tollens:1 not-act-abs-b not-act-concrete-b)
  next
    AOT-show ‹◇([E!]b ∨ ([A!]b & ¬q₀))›
      using KBasic2:2 b-prop &E(1) ∨I(1) ≡E(3) raa−cor:3 by blast
  qed
next AOT-show ‹𝓐([E!]a ∨ ([A!]a & ¬q₀))›
    by (metis Act−Basic:1 Act−Basic:2 Act−Basic:9 act-abs-a &I
            ∨I(2) ∨E(2) ≡E(3) not-act-q-zero raa−cor:1)
next AOT-show ‹¬Δ([E!]a ∨ ([A!]a & ¬q₀))›
  proof (rule act-and-not-nec-not-delta)
    AOT-show ‹𝓐([E!]a ∨ ([A!]a & ¬q₀))›
      by (metis Act−Basic:1 Act−Basic:2 Act−Basic:9 act-abs-a &I
                ∨I(2) ∨E(2) ≡E(3) not-act-q-zero raa−cor:1)
  next
    AOT-have ‹□¬[E!]a›
      by (metis ≡dfI conventions:5 &I ∨I(2)
                necessary-or-contingently-false
                not-act-concrete-a not-delta-concrete-a raa−cor:3)
    moreover AOT-have ‹◇¬([A!]a & ¬q₀)›
      by (metis KBasic2:1 KBasic:11 KBasic:3
                &E(1,2) ≡E(1) q₀-prop raa−cor:3)
    ultimately AOT-have ‹◇(¬[E!]a & ¬([A!]a & ¬q₀))›
      by (metis KBasic:16 &I vdash−properties:10)
    AOT-hence ‹◇¬([E!]a ∨ ([A!]a & ¬q₀))›
      by (metis RE◇ ≡E(2) oth−class−taut:5:d)
    AOT-thus ‹¬□([E!]a ∨ ([A!]a & ¬q₀))›
      by (metis KBasic:12 ≡E(1) raa−cor:3)
  qed
qed(auto simp: beta−C−meta[THEN →E, OF 1])
ultimately AOT-obtain F₆ where ‹¬𝓐[F₆]b & Δ[F₆]b & 𝓐[F₆]a & ¬Δ[F₆]a›
  using ∃I(1)[rotated, THEN ∃E[rotated]] by fastforce
AOT-hence ‹¬𝓐[F₆]b› and ‹Δ[F₆]b› and ‹𝓐[F₆]a› and ‹¬Δ[F₆]a›
  using &E by blast+
note props = props this

let ?Π = «[λy [A!]y ∨ [E!]y]»
AOT-modally-strict {
  AOT-have ‹[«?Π»]↓› by cqt:2[lambda]
} note 1 = this
moreover AOT-have ‹¬𝓐[«?Π»]b & Δ[«?Π»]b & 𝓐[«?Π»]a & Δ[«?Π»]a›
proof(safe intro!: &I; AOT-subst ‹[λy A!y ∨ E!y]x› ‹A!x ∨ E!x› for: x)
  AOT-show ‹¬𝓐([A!]b ∨ [E!]b)›
    using Act−Basic:9 ∨E(2) ≡E(4) not-act-abs-b
          not-act-concrete-b raa−cor:3 by blast
next AOT-show ‹Δ([A!]b ∨ [E!]b)›
  proof (rule not-act-and-pos-delta)
    AOT-show ‹¬𝓐([A!]b ∨ [E!]b)›
      using Act−Basic:9 ∨E(2) ≡E(4) not-act-abs-b
            not-act-concrete-b raa−cor:3 by blast
  next AOT-show ‹◇([A!]b ∨ [E!]b)›
      using KBasic2:2 b-prop &E(1) ∨I(2) ≡E(2) by blast
  qed
next AOT-show ‹𝓐([A!]a ∨ [E!]a)›
```

```
    by (meson Act−Basic:9 act-abs-a ∨I(1) ≡E(2))
  next AOT-show ‹𝚫([A!]a ∨ [E!]a) ›
    proof (rule nec-delta)
      AOT-show ‹□([A!]a ∨ [E!]a)›
        by (metis KBasic:15 act-abs-a act-and-not-nec-not-delta
                  Disjunction Addition(1) delta-abs-a raa−cor:3 →E)
    qed
  qed(auto simp: beta−C−meta[THEN →E, OF 1])
  ultimately AOT-obtain F7 where ‹¬𝓐[F7]b & 𝚫[F7]b & 𝓐[F7]a & 𝚫[F7]a›
    using ∃ I(1)[rotated, THEN ∃ E[rotated]] by fastforce
  AOT-hence ‹¬𝓐[F7]b› and ‹𝚫[F7]b› and ‹𝓐[F7]a› and ‹𝚫[F7]a›
    using &E by blast+
  note props = props this

  let ?Π = «[λy [O!]y & ¬[E!]y]»
  AOT-modally-strict {
    AOT-have ‹[«?Π»]↓› by cqt:2[lambda]
  } note 1 = this
  moreover AOT-have ‹𝓐[«?Π»]b & ¬𝚫[«?Π»]b & ¬𝓐[«?Π»]a & ¬𝚫[«?Π»]a›
  proof(safe intro!: &I; AOT-subst ‹[λy O!y & ¬E!y]x› ‹O!x & ¬E!x› for: x)
    AOT-show ‹𝓐([O!]b & ¬[E!]b)›
      by (metis Act−Basic:1 Act−Basic:2 act-ord-b &I ∨E(2)
                ≡E(3) not-act-concrete-b raa−cor:3)
  next AOT-show ‹¬𝚫([O!]b & ¬[E!]b)›
      by (metis (no-types, opaque-lifting) conventions:5 Act−Sub:1 RM:1
                act-and-not-nec-not-delta act−conj−act:3
                act-ord-b b-prop &I &E(1) Conjunction Simplification(2)
                df−rules−formulas[3]
                ≡E(3) raa−cor:1 →E)
  next AOT-show ‹¬𝓐([O!]a & ¬[E!]a)›
      using Act−Basic:2 &E(1) ≡E(1) not-act-ord-a raa−cor:3 by blast
  next AOT-have ‹¬◊([O!]a & ¬[E!]a)›
      by (metis KBasic2:3 &E(1) ≡E(4) not-act-ord-a oa−facts:3
                oa−facts:7 raa−cor:3 vdash−properties:10)
    AOT-thus ‹¬𝚫([O!]a & ¬[E!]a)›
      by (rule impossible-delta)
  qed(auto simp: beta−C−meta[THEN →E, OF 1])
  ultimately AOT-obtain F8 where ‹𝓐[F8]b & ¬𝚫[F8]b & ¬𝓐[F8]a & ¬𝚫[F8]a›
    using ∃ I(1)[rotated, THEN ∃ E[rotated]] by fastforce
  AOT-hence ‹𝓐[F8]b› and ‹¬𝚫[F8]b› and ‹¬𝓐[F8]a› and ‹¬𝚫[F8]a›
    using &E by blast+
  note props = props this

  let ?Π = «[λy ¬[E!]y & ([O!]y ∨ q0)]»
  AOT-modally-strict {
    AOT-have ‹[«?Π»]↓› by cqt:2[lambda]
  } note 1 = this
  moreover AOT-have ‹𝓐[«?Π»]b & ¬𝚫[«?Π»]b & ¬𝓐[«?Π»]a & 𝚫[«?Π»]a›
  proof(safe intro!: &I;
        AOT-subst ‹[λy ¬E!y & (O!y ∨ q0)]x› ‹¬E!x & (O!x ∨ q0)› for: x)
    AOT-show ‹𝓐(¬[E!]b & ([O!]b ∨ q0))›
      by (metis Act−Basic:1 Act−Basic:2 Act−Basic:9 act-ord-b &I
                ∨I(1) ∨E(2) ≡E(3) not-act-concrete-b raa−cor:1)
  next AOT-show ‹¬𝚫(¬[E!]b & ([O!]b ∨ q0))›
    proof (rule act-and-pos-not-not-delta)
      AOT-show ‹𝓐(¬[E!]b & ([O!]b ∨ q0))›
        by (metis Act−Basic:1 Act−Basic:2 Act−Basic:9 act-ord-b &I
                  ∨I(1) ∨E(2) ≡E(3) not-act-concrete-b raa−cor:1)
    next
      AOT-show ‹◊¬(¬[E!]b & ([O!]b ∨ q0))›
      proof (AOT-subst ‹¬(¬[E!]b & ([O!]b ∨ q0))› ‹[E!]b ∨ ¬([O!]b ∨ q0)›)
        AOT-modally-strict {
          AOT-show ‹¬(¬[E!]b & ([O!]b ∨ q0)) ≡ [E!]b ∨ ¬([O!]b ∨ q0)›
```

```
          by (metis &I &E(1,2) ∨I(1,2) ∨E(2)
                  →I ≡I reductio−aa:1)
        }
      next
        AOT-show ‹◇([E!]b ∨ ¬([O!]b ∨ q0))›
          using KBasic2:2 b-prop &E(1) ∨I(1) ≡E(3)
                raa−cor:3 by blast
      qed
    qed
  next
    AOT-show ‹¬𝓐(¬[E!]a & ([O!]a ∨ q0))›
      using Act−Basic:2 Act−Basic:9 &E(2) ∨E(3) ≡E(1)
            not-act-ord-a not-act-q-zero reductio−aa:2 by blast
  next
    AOT-show ‹Δ(¬[E!]a & ([O!]a ∨ q0))›
    proof (rule not-act-and-pos-delta)
      AOT-show ‹¬𝓐(¬[E!]a & ([O!]a ∨ q0))›
        by (metis Act−Basic:2 Act−Basic:9 &E(2) ∨E(3) ≡E(1)
                  not-act-ord-a not-act-q-zero reductio−aa:2)
    next
      AOT-have ‹□¬[E!]a›
        using KBasic2:1 ≡E(2) not-act-and-pos-delta not-act-concrete-a
              not-delta-concrete-a raa−cor:5 by blast
      moreover AOT-have ‹◇([O!]a ∨ q0)›
        by (metis KBasic2:2 &E(1) ∨I(2) ≡E(3) q0-prop raa−cor:3)
      ultimately AOT-show ‹◇(¬[E!]a & ([O!]a ∨ q0))›
        by (metis KBasic:16 &I vdash−properties:10)
    qed
  qed(auto simp: beta−C−meta[THEN →E, OF 1])
ultimately AOT-obtain F9 where ‹𝓐[F9]b & ¬Δ[F9]b & ¬𝓐[F9]a & Δ[F9]a›
  using ∃I(1)[rotated, THEN ∃E[rotated]] by fastforce
AOT-hence ‹𝓐[F9]b› and ‹¬Δ[F9]b› and ‹¬𝓐[F9]a› and ‹Δ[F9]a›
  using &E by blast+
note props = props this

AOT-modally-strict {
  AOT-have ‹[λy ¬q0]↓› by cqt:2[lambda]
} note 1 = this
moreover AOT-have ‹𝓐[λy ¬q0]b & ¬Δ[λy ¬q0]b & 𝓐[λy ¬q0]a & ¬Δ[λy ¬q0]a›
  by (safe intro!: &I; AOT-subst ‹[λy ¬q0]x› ‹¬q0› for: x)
     (auto simp: act-not-q-zero not-delta-not-q-zero
                 beta−C−meta[THEN →E, OF 1])
ultimately AOT-obtain F10 where ‹𝓐[F10]b & ¬Δ[F10]b & 𝓐[F10]a & ¬Δ[F10]a›
  using ∃I(1)[rotated, THEN ∃E[rotated]] by fastforce
AOT-hence ‹𝓐[F10]b› and ‹¬Δ[F10]b› and ‹𝓐[F10]a› and ‹¬Δ[F10]a›
  using &E by blast+
note props = props this

AOT-modally-strict {
  AOT-have ‹[λy ¬[E!]y]↓› by cqt:2[lambda]
} note 1 = this
moreover AOT-have ‹𝓐[λy ¬[E!]y]b & ¬Δ[λy ¬[E!]y]b &
                   𝓐[λy ¬[E!]y]a & Δ[λy ¬[E!]y]a›
proof (safe intro!: &I; AOT-subst ‹[λy ¬[E!]y]x› ‹¬[E!]x› for: x)
  AOT-show ‹𝓐¬[E!]b›
    using Act−Basic:1 ∨E(2) not-act-concrete-b by blast
next AOT-show ‹¬Δ¬[E!]b›
    using ≡dfE conventions:5 Act−Basic:1 act-and-not-nec-not-delta
          b-prop &E(1) ∨E(2) not-act-concrete-b by blast
next AOT-show ‹𝓐¬[E!]a›
    using Act−Basic:1 ∨E(2) not-act-concrete-a by blast
next AOT-show ‹Δ¬[E!]a›
    using KBasic2:1 ≡E(2) nec-delta not-act-and-pos-delta
```

not-act-concrete-a not-delta-concrete-a reductio−aa:1
by *blast*
qed(*auto simp*: *beta−C−meta*[*THEN* →*E*, *OF 1*])
ultimately AOT-obtain F_{11} **where** ‹$\boldsymbol{\mathcal{A}}[F_{11}]b$ & $\neg\boldsymbol{\Delta}[F_{11}]b$ & $\boldsymbol{\mathcal{A}}[F_{11}]a$ & $\boldsymbol{\Delta}[F_{11}]a$›
using ∃*I*(*1*)[*rotated*, *THEN* ∃*E*[*rotated*]] **by** *fastforce*
AOT-hence ‹$\boldsymbol{\mathcal{A}}[F_{11}]b$› **and** ‹$\neg\boldsymbol{\Delta}[F_{11}]b$› **and** ‹$\boldsymbol{\mathcal{A}}[F_{11}]a$› **and** ‹$\boldsymbol{\Delta}[F_{11}]a$›
using &*E* **by** *blast+*
note *props* = *props this*

AOT-have ‹$\boldsymbol{\mathcal{A}}[O!]b$ & $\boldsymbol{\Delta}[O!]b$ & $\neg\boldsymbol{\mathcal{A}}[O!]a$ & $\neg\boldsymbol{\Delta}[O!]a$›
by (*simp add*: *act-ord-b* &*I delta-ord-b not-act-ord-a not-delta-ord-a*)
then AOT-obtain F_{12} **where** ‹$\boldsymbol{\mathcal{A}}[F_{12}]b$ & $\boldsymbol{\Delta}[F_{12}]b$ & $\neg\boldsymbol{\mathcal{A}}[F_{12}]a$ & $\neg\boldsymbol{\Delta}[F_{12}]a$›
using *oa−exist*:*1* ∃*I*(*1*)[*rotated*, *THEN* ∃*E*[*rotated*]] **by** *fastforce*
AOT-hence ‹$\boldsymbol{\mathcal{A}}[F_{12}]b$› **and** ‹$\boldsymbol{\Delta}[F_{12}]b$› **and** ‹$\neg\boldsymbol{\mathcal{A}}[F_{12}]a$› **and** ‹$\neg\boldsymbol{\Delta}[F_{12}]a$›
using &*E* **by** *blast+*
note *props* = *props this*

let *?*Π = «$[\lambda y\ [O!]y \vee q_0]$»
AOT-modally-strict {
AOT-have ‹[«*?*Π»]↓› **by** *cqt*:*2*[*lambda*]
} note *1* = *this*
moreover AOT-have ‹$\boldsymbol{\mathcal{A}}$[«*?*Π»]b & $\boldsymbol{\Delta}$[«*?*Π»]b & $\neg\boldsymbol{\mathcal{A}}$[«*?*Π»]$a$ & $\boldsymbol{\Delta}$[«*?*Π»]a›
proof (*safe intro*!: &*I*; *AOT-subst* ‹$[\lambda y\ O!y \vee q_0]x$› ‹$O!x \vee q_0$› **for**: *x*)
AOT-show ‹$\boldsymbol{\mathcal{A}}([O!]b \vee q_0)$›
by (*meson Act−Basic*:*9 act-ord-b* ∨*I*(*1*) ≡*E*(*2*))
next AOT-show ‹$\boldsymbol{\Delta}([O!]b \vee q_0)$›
by (*meson KBasic*:*15 b-ord* ∨*I*(*1*) *nec-delta oa−facts*:*1* →*E*)
next AOT-show ‹$\neg\boldsymbol{\mathcal{A}}([O!]a \vee q_0)$›
using *Act−Basic*:*9* ∨*E*(*2*) ≡*E*(*4*) *not-act-ord-a*
not-act-q-zero raa−cor:*3* **by** *blast*
next AOT-show ‹$\boldsymbol{\Delta}([O!]a \vee q_0)$›
proof (*rule not-act-and-pos-delta*)
AOT-show ‹$\neg\boldsymbol{\mathcal{A}}([O!]a \vee q_0)$›
using *Act−Basic*:*9* ∨*E*(*2*) ≡*E*(*4*) *not-act-ord-a*
not-act-q-zero raa−cor:*3* **by** *blast*
next AOT-show ‹$\Diamond([O!]a \vee q_0)$›
using *KBasic2*:*2* &*E*(*1*) ∨*I*(*2*) ≡*E*(*2*) q_0*-prop* **by** *blast*
qed
qed(*auto simp*: *beta−C−meta*[*THEN* →*E*, *OF 1*])
ultimately AOT-obtain F_{13} **where** ‹$\boldsymbol{\mathcal{A}}[F_{13}]b$ & $\boldsymbol{\Delta}[F_{13}]b$ & $\neg\boldsymbol{\mathcal{A}}[F_{13}]a$ & $\boldsymbol{\Delta}[F_{13}]a$›
using ∃*I*(*1*)[*rotated*, *THEN* ∃*E*[*rotated*]] **by** *fastforce*
AOT-hence ‹$\boldsymbol{\mathcal{A}}[F_{13}]b$› **and** ‹$\boldsymbol{\Delta}[F_{13}]b$› **and** ‹$\neg\boldsymbol{\mathcal{A}}[F_{13}]a$› **and** ‹$\boldsymbol{\Delta}[F_{13}]a$›
using &*E* **by** *blast+*
note *props* = *props this*

let *?*Π = «$[\lambda y\ [O!]y \vee \neg q_0]$»
AOT-modally-strict {
AOT-have ‹[«*?*Π»]↓› **by** *cqt*:*2*[*lambda*]
} note *1* = *this*
moreover AOT-have ‹$\boldsymbol{\mathcal{A}}$[«*?*Π»]b & $\boldsymbol{\Delta}$[«*?*Π»]b & $\boldsymbol{\mathcal{A}}$[«*?*Π»]a & $\neg\boldsymbol{\Delta}$[«*?*Π»]a›
proof (*safe intro*!: &*I*; *AOT-subst* ‹$[\lambda y\ O!y \vee \neg q_0]x$› ‹$O!x \vee \neg q_0$› **for**: *x*)
AOT-show ‹$\boldsymbol{\mathcal{A}}([O!]b \vee \neg q_0)$›
by (*meson Act−Basic*:*9 act-not-q-zero* ∨*I*(*2*) ≡*E*(*2*))
next AOT-show ‹$\boldsymbol{\Delta}([O!]b \vee \neg q_0)$›
by (*meson KBasic*:*15 b-ord* ∨*I*(*1*) *nec-delta oa−facts*:*1* →*E*)
next AOT-show ‹$\boldsymbol{\mathcal{A}}([O!]a \vee \neg q_0)$›
by (*meson Act−Basic*:*9 act-not-q-zero* ∨*I*(*2*) ≡*E*(*2*))
next AOT-show ‹$\neg\boldsymbol{\Delta}([O!]a \vee \neg q_0)$›
proof(*rule act-and-pos-not-not-delta*)
AOT-show ‹$\boldsymbol{\mathcal{A}}([O!]a \vee \neg q_0)$›
by (*meson Act−Basic*:*9 act-not-q-zero* ∨*I*(*2*) ≡*E*(*2*))
next
AOT-have ‹$\Box\neg[O!]a$›

```
        using KBasic2:1 ≡E(2) not-act-and-pos-delta
              not-act-ord-a not-delta-ord-a raa−cor:6 by blast
      moreover AOT-have ‹◇q0›
        by (meson &E(1) q0-prop)
      ultimately AOT-have 2: ‹◇(¬[O!]a & q0)›
         by (metis KBasic:16 &I vdash−properties:10)
      AOT-show ‹◇¬([O!]a ∨ ¬q0)›
      proof (AOT-subst (reverse) ‹¬([O!]a ∨ ¬q0)› ‹¬[O!]a & q0›)
        AOT-modally-strict {
          AOT-show ‹¬[O!]a & q0 ≡ ¬([O!]a ∨ ¬q0)›
            by (metis &I &E(1) &E(2) ∨I(1) ∨I(2)
                      ∨E(3) deduction−theorem ≡I raa−cor:3)
        }
      next
        AOT-show ‹◇(¬[O!]a & q0)›
          using 2 by blast
      qed
    qed
  qed(auto simp: beta−C−meta[THEN →E, OF 1])
  ultimately AOT-obtain F14 where ‹𝓐[F14]b & Δ[F14]b & 𝓐[F14]a & ¬Δ[F14]a›
    using ∃I(1)[rotated, THEN ∃E[rotated]] by fastforce
  AOT-hence ‹𝓐[F14]b› and ‹Δ[F14]b› and ‹𝓐[F14]a› and ‹¬Δ[F14]a›
    using &E by blast+
  note props = props this

  AOT-have ‹[L]↓›
    by (rule =df I(2)[OF L-def]) cqt:2[lambda]+
  moreover AOT-have ‹𝓐[L]b & Δ[L]b & 𝓐[L]a & Δ[L]a›
  proof (safe intro!: &I)
    AOT-show ‹𝓐[L]b›
      by (meson nec-L nec−imp−act vdash−properties:10)
    next AOT-show ‹Δ[L]b› using nec-L nec-delta by blast
    next AOT-show ‹𝓐[L]a› by (meson nec-L nec−imp−act →E)
    next AOT-show ‹Δ[L]a› using nec-L nec-delta by blast
  qed
  ultimately AOT-obtain F15 where ‹𝓐[F15]b & Δ[F15]b & 𝓐[F15]a & Δ[F15]a›
    using ∃I(1)[rotated, THEN ∃E[rotated]] by fastforce
  AOT-hence ‹𝓐[F15]b› and ‹Δ[F15]b› and ‹𝓐[F15]a› and ‹Δ[F15]a›
    using &E by blast+
  note props = props this

  show ?thesis
    by (rule ∃I(2)[where β=F0]; rule ∃I(2)[where β=F1];
        rule ∃I(2)[where β=F2]; rule ∃I(2)[where β=F3];
        rule ∃I(2)[where β=F4]; rule ∃I(2)[where β=F5];
        rule ∃I(2)[where β=F6]; rule ∃I(2)[where β=F7];
        rule ∃I(2)[where β=F8]; rule ∃I(2)[where β=F9];
        rule ∃I(2)[where β=F10]; rule ∃I(2)[where β=F11];
        rule ∃I(2)[where β=F12]; rule ∃I(2)[where β=F13];
        rule ∃I(2)[where β=F14]; rule ∃I(2)[where β=F15];
        safe intro!: &I)
       (match conclusion in [?v ⊨ [F] ≠ [G]] for F G ⇒ ‹
        match props in A: [?v ⊨ ¬φ{F}] for φ ⇒ ‹
        match (φ) in λa . ?p ⇒ ‹fail› | λa . a ⇒ ‹fail› | - ⇒ ‹
        match props in B: [?v ⊨ φ{G}] ⇒ ‹
        fact pos−not−equiv−ne:4[where F=F and G=G and φ=φ, THEN →E,
                                OF oth−class−taut:4:h[THEN ≡E(2)],
                                OF Disjunction Addition(2)[THEN →E],
                                OF &I, OF A, OF B]››››)+
qed
```

8.11 The Theory of Objects

```
AOT-theorem o−objects−exist:1: ‹□∃ x O!x›
proof(rule RN)
  AOT-modally-strict {
    AOT-obtain a where ‹◇(E!a & ¬𝓐[E!]a)›
      using ∃ E[rotated, OF qml:4[axiom-inst, THEN BF◇[THEN →E]]]
      by blast
    AOT-hence 1: ‹◇E!a› by (metis KBasic2:3 &E(1) →E)
    AOT-have ‹[λx ◇[E!]x]a›
    proof (rule β←C(1); cqt:2[lambda]?)
      AOT-show ‹a↓› using cqt:2[const-var][axiom-inst] by blast
    next
      AOT-show ‹◇E!a› by (fact 1)
    qed
    AOT-hence ‹O!a› by (rule =dfI(2)[OF AOT-ordinary, rotated]) cqt:2
    AOT-thus ‹∃ x [O!]x› by (rule ∃ I)
  }
qed

AOT-theorem o−objects−exist:2: ‹□∃ x A!x›
proof (rule RN)
  AOT-modally-strict {
    AOT-obtain a where ‹[A!]a›
      using A−objects[axiom-inst] ∃ E[rotated] &E by blast
    AOT-thus ‹∃ x A!x› using ∃ I by blast
  }
qed

AOT-theorem o−objects−exist:3: ‹□¬∀ x O!x›
  by (rule RN)
     (metis (no-types, opaque-lifting) ∃ E cqt−orig:1[const-var]
       ≡E(4) modus−tollens:1 o−objects−exist:2 oa−contingent:2
       qml:2[axiom-inst] reductio−aa:2)

AOT-theorem o−objects−exist:4: ‹□¬∀ x A!x›
  by (rule RN)
     (metis (mono-tags, opaque-lifting) ∃ E cqt−orig:1[const-var]
       ≡E(1) modus−tollens:1 o−objects−exist:1 oa−contingent:2
       qml:2[axiom-inst] →E)

AOT-theorem o−objects−exist:5: ‹□¬∀ x E!x›
proof (rule RN; rule raa−cor:2)
  AOT-modally-strict {
    AOT-assume ‹∀ x E!x›
    moreover AOT-obtain a where abs: ‹A!a›
      using o−objects−exist:2[THEN qml:2[axiom-inst, THEN →E]]
            ∃ E[rotated] by blast
    ultimately AOT-have ‹E!a› using ∀ E by blast
    AOT-hence 1: ‹◇E!a› by (metis T◇ →E)
    AOT-have ‹[λy ◇E!y]a›
    proof (rule β←C(1); cqt:2[lambda]?)
      AOT-show ‹a↓› using cqt:2[const-var][axiom-inst].
    next
      AOT-show ‹◇E!a› by (fact 1)
    qed
    AOT-hence ‹O!a›
      by (rule =dfI(2)[OF AOT-ordinary, rotated]) cqt:2[lambda]
    AOT-hence ‹¬A!a› by (metis ≡E(1) oa−contingent:2)
    AOT-thus ‹p & ¬p› for p using abs by (metis raa−cor:3)
  }
qed
```

AOT-theorem *partition*: ‹¬∃ x (O!x & A!x)›
proof(*rule raa−cor*:*2*)
  AOT-assume ‹∃ x (O!x & A!x)›
  then AOT-obtain *a* **where** ‹O!a & A!a›
    using ∃ *E*[*rotated*] **by** *blast*
  AOT-thus ‹p & ¬p› **for** *p*
    by (*metis* &*E*(*1*) *Conjunction Simplification*(*2*) ≡*E*(*1*)
          modus−tollens:*1* *oa−contingent*:*2* *raa−cor*:*3*)
qed

AOT-define *eq-E* :: ‹Π› (‹′($=_E$′)›)
  =*E*: ‹($=_E$) $=_{df}$ [λxy O!x & O!y & □∀ F ([F]x ≡ [F]y)]›

syntax *-AOT-eq-E-infix* :: ‹τ ⇒ τ ⇒ φ› (**infixl** ‹$=_E$› *50*)
translations
  -AOT-eq-E-infix κ κ′ == *CONST AOT-exe* (*CONST eq-E*) (*CONST Pair* κ κ′)
print-translation‹
AOT-syntax-print-translations
[(***const-syntax***‹*AOT-exe*›, *fn ctxt* => *fn* [
  Const (***const-name***‹*eq-E*›, -),
  Const (***const-syntax***‹*Pair*›, -) $ *lhs* $ *rhs*
] => *Const* (***syntax-const***‹*-AOT-eq-E-infix*›, *dummyT*) $ *lhs* $ *rhs*)]›

Note: Not explicitly mentioned as theorem in PLM.

AOT-theorem =*E*[*denotes*]: ‹[($=_E$)]↓›
  by (*rule* $=_{df}$*I*(*2*)[*OF* =*E*]) *cqt*:*2*[*lambda*]+

AOT-theorem =*E−simple*:*1*: ‹x $=_E$ y ≡ (O!x & O!y & □∀ F ([F]x ≡ [F]y))›
proof −
  AOT-have *1*: ‹[λxy [O!]x & [O!]y & □∀ F ([F]x ≡ [F]y)]↓› **by** *cqt*:*2*
  show *?thesis*
    apply (*rule* $=_{df}$*I*(*2*)[*OF* =*E*]; *cqt*:*2*[*lambda*]*?*)
    using *beta−C−meta*[*THEN* →*E*, *OF 1*, *unvarify* $\nu_1\nu_n$, *of* (-,-),
                    OF tuple-denotes[*THEN* $\equiv_{df}$*I*], *OF* &*I*,
                    OF cqt:*2*[*const-var*][*axiom-inst*],
                    OF cqt:*2*[*const-var*][*axiom-inst*]]
    by *fast*
qed

AOT-theorem =*E−simple*:*2*: ‹x $=_E$ y → x = y›
proof (*rule* →*I*)
  AOT-assume ‹x $=_E$ y›
  AOT-hence ‹O!x & O!y & □∀ F ([F]x ≡ [F]y)›
    using =*E−simple*:*1*[*THEN* ≡*E*(*1*)] **by** *blast*
  AOT-thus ‹x = y›
    using $\equiv_{df}$*I*[*OF identity*:*1*] ∨*I* **by** *blast*
qed

AOT-theorem *id−nec3*:*1*: ‹x $=_E$ y ≡ □(x $=_E$ y)›
proof (*rule* ≡*I*; *rule* →*I*)
  AOT-assume ‹x $=_E$ y›
  AOT-hence ‹O!x & O!y & □∀ F ([F]x ≡ [F]y)›
    using =*E−simple*:*1* ≡*E* **by** *blast*
  AOT-hence ‹□O!x & □O!y & □□∀ F ([F]x ≡ [F]y)›
    by (*metis S5Basic*:*6* &*I* &*E*(*1*) &*E*(*2*) ≡*E*(*4*)
          oa−facts:*1* *raa−cor*:*3* *vdash−properties*:*10*)
  AOT-hence ‹□(O!x & O!y & □∀ F ([F]x ≡ [F]y))›
    by (*metis* &*E*(*1*) &*E*(*2*) ≡*E*(*2*) *KBasic*:*3* &*I*)
  AOT-thus ‹□(x $=_E$ y)›
    using =*E−simple*:*1*
    by (*AOT-subst* ‹x $=_E$ y› ‹O!x & O!y & □∀ F ([F]x ≡ [F]y)›) *auto*
next
  AOT-assume ‹□(x $=_E$ y)›

AOT-thus ‹$x =_E y$› **using** *qml:2[axiom-inst, THEN →E]* **by** *blast*
qed

AOT-theorem *id−nec3:2*: ‹$\Diamond(x =_E y) \equiv x =_E y$›
by (*meson RE◇ S5Basic:2 id−nec3:1 ≡E(1,5) Commutativity of ≡*)

AOT-theorem *id−nec3:3*: ‹$\Diamond(x =_E y) \equiv \Box(x =_E y)$›
by (*meson id−nec3:1 id−nec3:2 ≡E(5)*)

syntax *-AOT-non-eq-E* :: ‹Π› (‹'(≠$_E$')›)
translations
(Π) (≠$_E$) == (Π) (=$_E$)$^-$
syntax *-AOT-non-eq-E-infix* :: ‹τ ⇒ τ ⇒ φ› (**infixl** ‹≠$_E$› *50*)
translations
-AOT-non-eq-E-infix κ κ' ==
CONST AOT-exe (*CONST relation-negation* (*CONST eq-E*)) (*CONST Pair* κ κ')
print-translation‹
AOT-syntax-print-translations
[(***const-syntax***‹*AOT-exe*›, *fn ctxt* => *fn* [
Const (***const-syntax***‹*relation-negation*›, -) $ *Const* (***const-name***‹*eq-E*›, -),
Const (***const-syntax***‹*Pair*›, -) $ *lhs* $ *rhs*
] => *Const* (***syntax-const***‹*-AOT-non-eq-E-infix*›, *dummyT*) $ *lhs* $ *rhs*)]›
AOT-theorem *thm−neg=E*: ‹$x \neq_E y \equiv \neg(x =_E y)$›
proof −
AOT-have ϑ: ‹$[\lambda x_1...x_2\ \neg(=_E)x_1...x_2]\downarrow$› **by** *cqt:2*
AOT-have ‹$x \neq_E y \equiv [\lambda x_1...x_2\ \neg(=_E)x_1...x_2]xy$›
by (*rule* $=_{df}I(1)$[*OF df−relation−negation, OF* ϑ])
(*meson oth−class−taut:3:a*)
also AOT-have ‹... ≡ ¬(=$_E$)*xy*›
by (*safe intro*!: *beta−C−meta*[*THEN →E, unvarify* $\nu_1\nu_n$] *cqt:2*
tuple-denotes[*THEN* $\equiv_{df}I$] &*I*)
finally show *?thesis*.
qed

AOT-theorem *id−nec4:1*: ‹$x \neq_E y \equiv \Box(x \neq_E y)$›
proof −
AOT-have ‹$x \neq_E y \equiv \neg(x =_E y)$› **using** *thm−neg=E*.
also AOT-have ‹... ≡ ¬◇($x =_E y$)›
by (*meson id−nec3:2 ≡E(1) Commutativity of ≡ oth−class−taut:4:b*)
also AOT-have ‹... ≡ □¬($x =_E y$)›
by (*meson KBasic2:1 ≡E(2) Commutativity of ≡*)
also AOT-have ‹... ≡ □($x \neq_E y$)›
by (*AOT-subst* (*reverse*) ‹¬($x =_E y$)› ‹$x \neq_E y$›)
(*auto simp*: *thm−neg=E oth−class−taut:3:a*)
finally show *?thesis*.
qed

AOT-theorem *id−nec4:2*: ‹$\Diamond(x \neq_E y) \equiv (x \neq_E y)$›
by (*meson RE◇ S5Basic:2 id−nec4:1 ≡E(2,5) Commutativity of ≡*)

AOT-theorem *id−nec4:3*: ‹$\Diamond(x \neq_E y) \equiv \Box(x \neq_E y)$›
by (*meson id−nec4:1 id−nec4:2 ≡E(5)*)

AOT-theorem *id−act2:1*: ‹$x =_E y \equiv \mathcal{A}x =_E y$›
by (*meson Act−Basic:5 Act−Sub:2 RA[2] id−nec3:2 ≡E(1,6)*)
AOT-theorem *id−act2:2*: ‹$x \neq_E y \equiv \mathcal{A}x \neq_E y$›
by (*meson Act−Basic:5 Act−Sub:2 RA[2] id−nec4:2 ≡E(1,6)*)

AOT-theorem *ord=Eequiv:1*: ‹$O!x \to x =_E x$›
proof (*rule →I*)
AOT-assume *1*: ‹*O*!*x*›
AOT-show ‹$x =_E x$›
apply (*rule* $=_{df}I(2)$[*OF =E*]) **apply** *cqt:2*[*lambda*]

apply (*rule* $\beta{\leftarrow}C(1)$)
apply *cqt:2[lambda]*
apply (*simp add*: *&I cqt:2[const-var][axiom-inst] prod-denotesI*)
by (*simp add*: *1 RN &I oth−class−taut:3:a universal−cor*)
qed

AOT-theorem *ord=Eequiv:2*: ‹$x =_E y \rightarrow y =_E x$›
proof(*rule CP*)
AOT-assume *1*: ‹$x =_E y$›
AOT-hence *2*: ‹$x = y$› **by** (*metis =E−simple:2 vdash−properties:10*)
AOT-have ‹$O!x$› **using** *1* **by** (*meson &E(1) =E−simple:1 ≡E(1)*)
AOT-hence ‹$x =_E x$› **using** *ord=Eequiv:1 →E* **by** *blast*
AOT-thus ‹$y =_E x$› **using** *rule=E[rotated, OF 2]* **by** *fast*
qed

AOT-theorem *ord=Eequiv:3*: ‹$(x =_E y \ \& \ y =_E z) \rightarrow x =_E z$›
proof (*rule CP*)
AOT-assume *1*: ‹$x =_E y \ \& \ y =_E z$›
AOT-hence ‹$x = y \ \& \ y = z$›
by (*metis &I &E(1) &E(2) =E−simple:2 vdash−properties:6*)
AOT-hence ‹$x = z$› **by** (*metis id−eq:3 vdash−properties:6*)
moreover AOT-have ‹$x =_E x$›
using *1[THEN &E(1)] &E(1) =E−simple:1 ≡E(1)*
ord=Eequiv:1 →E **by** *blast*
ultimately AOT-show ‹$x =_E z$›
using *rule=E* **by** *fast*
qed

AOT-theorem *ord−=E=:1*: ‹$(O!x \vee O!y) \rightarrow \Box(x = y \equiv x =_E y)$›
proof(*rule CP*)
AOT-assume ‹$O!x \vee O!y$›
moreover {
AOT-assume ‹$O!x$›
AOT-hence ‹$\Box O!x$› **by** (*metis oa−facts:1 vdash−properties:10*)
moreover {
AOT-modally-strict {
AOT-have ‹$O!x \rightarrow (x = y \equiv x =_E y)$›
proof (*rule →I; rule ≡I; rule →I*)
AOT-assume ‹$O!x$›
AOT-hence ‹$x =_E x$› **by** (*metis ord=Eequiv:1 →E*)
moreover AOT-assume ‹$x = y$›
ultimately AOT-show ‹$x =_E y$› **using** *rule=E* **by** *fast*
next
AOT-assume ‹$x =_E y$›
AOT-thus ‹$x = y$› **by** (*metis =E−simple:2 →E*)
qed
}
AOT-hence ‹$\Box O!x \rightarrow \Box(x = y \equiv x =_E y)$› **by** (*metis RM:1*)
}
ultimately AOT-have ‹$\Box(x = y \equiv x =_E y)$› **using** *→E* **by** *blast*
}
moreover {
AOT-assume ‹$O!y$›
AOT-hence ‹$\Box O!y$› **by** (*metis oa−facts:1 vdash−properties:10*)
moreover {
AOT-modally-strict {
AOT-have ‹$O!y \rightarrow (x = y \equiv x =_E y)$›
proof (*rule →I; rule ≡I; rule →I*)
AOT-assume ‹$O!y$›
AOT-hence ‹$y =_E y$› **by** (*metis ord=Eequiv:1 →E*)
moreover AOT-assume ‹$x = y$›
ultimately AOT-show ‹$x =_E y$› **using** *rule=E id-sym* **by** *fast*
next

AOT-assume ‹$x =_E y$›
AOT-thus ‹$x = y$› **by** (*metis =E−simple:2 →E*)
qed
}
AOT-hence ‹$\Box O!y \to \Box(x = y \equiv x =_E y)$› **by** (*metis RM:1*)
}
ultimately AOT-have ‹$\Box(x = y \equiv x =_E y)$› **using** $\to E$ **by** *blast*
}
ultimately AOT-show ‹$\Box(x = y \equiv x =_E y)$› **by** (*metis* $\vee E(3)$ *raa−cor:1*)
qed

AOT-theorem *ord−=E=:2*: ‹$O!y \to [\lambda x\ x = y]\downarrow$›
proof (*rule* $\to I$; *rule safe−ext[axiom-inst, THEN* $\to E$]; *rule* &*I*)
AOT-show ‹$[\lambda x\ x =_E y]\downarrow$› **by** *cqt:2[lambda]*
next
AOT-assume ‹$O!y$›
AOT-hence *1*: ‹$\Box(x = y \equiv x =_E y)$› **for** *x*
using *ord−=E=:1* $\to E$ $\vee I$ **by** *blast*
AOT-have ‹$\Box(x =_E y \equiv x = y)$› **for** *x*
by (*AOT-subst* ‹$x =_E y \equiv x = y$› ‹$x = y \equiv x =_E y$›)
(*auto simp add*: *Commutativity of* $\equiv$ *1*)
AOT-hence ‹$\forall x\ \Box(x =_E y \equiv x = y)$› **by** (*rule GEN*)
AOT-thus ‹$\Box\forall x\ (x =_E y \equiv x = y)$› **by** (*rule BF[THEN* $\to E$])
qed

AOT-theorem *ord−=E=:3*: ‹$[\lambda xy\ O!x\ \&\ O!y\ \&\ x = y]\downarrow$›
proof (*rule safe−ext[2][axiom-inst, THEN* $\to E$]; *rule* &*I*)
AOT-show ‹$[\lambda xy\ O!x\ \&\ O!y\ \&\ x =_E y]\downarrow$› **by** *cqt:2[lambda]*
next
AOT-show ‹$\Box\forall x\forall y\ ([O!]x\ \&\ [O!]y\ \&\ x =_E y \equiv [O!]x\ \&\ [O!]y\ \&\ x = y)$›
proof (*rule RN*; *rule GEN*; *rule GEN*; *rule* $\equiv I$; *rule* $\to I$)
AOT-modally-strict {
AOT-show ‹$[O!]x\ \&\ [O!]y\ \&\ x = y$› **if** ‹$[O!]x\ \&\ [O!]y\ \&\ x =_E y$› **for** *x y*
by (*metis* &*I* &*E(1) Conjunction Simplification(2) =E−simple:2*
modus−tollens:1 raa−cor:1 that)
}
next
AOT-modally-strict {
AOT-show ‹$[O!]x\ \&\ [O!]y\ \&\ x =_E y$› **if** ‹$[O!]x\ \&\ [O!]y\ \&\ x = y$› **for** *x y*
apply(*safe intro*!: &*I*)
apply (*metis that[THEN* &*E(1), THEN* &*E(1)]*)
apply (*metis that[THEN* &*E(1), THEN* &*E(2)]*)
using *rule=E[rotated, OF that[THEN* &*E(2)]]*
ord=Eequiv:1[THEN $\to E$*, OF that[THEN* &*E(1), THEN* &*E(1)]]*
by *fast*
}
qed
qed

AOT-theorem *ind−nec*: ‹$\forall F\ ([F]x \equiv [F]y) \to \Box\forall F\ ([F]x \equiv [F]y)$›
proof(*rule* $\to I$)
AOT-assume ‹$\forall F\ ([F]x \equiv [F]y)$›
moreover AOT-have ‹$[\lambda x\ \Box\forall F\ ([F]x \equiv [F]y)]\downarrow$› **by** *cqt:2[lambda]*
ultimately AOT-have ‹$[\lambda x\ \Box\forall F\ ([F]x \equiv [F]y)]x \equiv [\lambda x\ \Box\forall F\ ([F]x \equiv [F]y)]y$›
using $\forall E$ **by** *blast*
moreover AOT-have ‹$[\lambda x\ \Box\forall F\ ([F]x \equiv [F]y)]y$›
apply (*rule* $\beta\leftarrow C(1)$)
apply *cqt:2[lambda]*
apply (*fact cqt:2[const-var][axiom-inst]*)
by (*simp add*: *RN GEN oth−class−taut:3:a*)
ultimately AOT-have ‹$[\lambda x\ \Box\forall F\ ([F]x \equiv [F]y)]x$› **using** $\equiv E$ **by** *blast*
AOT-thus ‹$\Box\forall F\ ([F]x \equiv [F]y)$›

using $\beta{\rightarrow}C(1)$ **by** *blast*
qed

AOT-theorem *ord=E:1*: ‹$(O!x \;\&\; O!y) \rightarrow (\forall F\ ([F]x \equiv [F]y) \rightarrow x =_E y)$›
proof (*rule* →*I*; *rule* →*I*)
AOT-assume ‹$\forall F\ ([F]x \equiv [F]y)$›
AOT-hence ‹$\Box\forall F\ ([F]x \equiv [F]y)$›
using *ind−nec*[*THEN* →*E*] **by** *blast*
moreover AOT-assume ‹$O!x \;\&\; O!y$›
ultimately AOT-have ‹$O!x \;\&\; O!y \;\&\; \Box\forall F\ ([F]x \equiv [F]y)$›
using &*I* **by** *blast*
AOT-thus ‹$x =_E y$› **using** =*E−simple*:*1*[*THEN* ≡*E*(*2*)] **by** *blast*
qed

AOT-theorem *ord=E:2*: ‹$(O!x \;\&\; O!y) \rightarrow (\forall F\ ([F]x \equiv [F]y) \rightarrow x = y)$›
proof (*rule* →*I*; *rule* →*I*)
AOT-assume ‹$O!x \;\&\; O!y$›
moreover AOT-assume ‹$\forall F\ ([F]x \equiv [F]y)$›
ultimately AOT-have ‹$x =_E y$›
using *ord=E:1* →*E* **by** *blast*
AOT-thus ‹$x = y$› **using** =*E−simple*:*2*[*THEN* →*E*] **by** *blast*
qed

AOT-theorem *ord=E2:1*:
‹$(O!x \;\&\; O!y) \rightarrow (x \neq y \equiv [\lambda z\ z =_E x] \neq [\lambda z\ z =_E y])$›
proof (*rule* →*I*; *rule* ≡*I*; *rule* →*I*;
rule $\equiv_{df}$*I*[*OF* =*−infix*]; *rule raa−cor*:*2*)
AOT-assume *0*: ‹$O!x \;\&\; O!y$›
AOT-assume ‹$x \neq y$›
AOT-hence *1*: ‹$\neg(x = y)$› **using** $\equiv_{df}$*E*[*OF* =*−infix*] **by** *blast*
AOT-assume ‹$[\lambda z\ z =_E x] = [\lambda z\ z =_E y]$›
moreover AOT-have ‹$[\lambda z\ z =_E x]x$›
apply (*rule* $\beta{\leftarrow}C(1)$)
apply *cqt*:*2*[*lambda*]
apply (*fact cqt*:*2*[*const-var*][*axiom-inst*])
using *ord=Eequiv:1*[*THEN* →*E*, *OF 0*[*THEN* &*E*(*1*)]]**.**
ultimately AOT-have ‹$[\lambda z\ z =_E y]x$› **using** *rule=E* **by** *fast*
AOT-hence ‹$x =_E y$› **using** $\beta{\rightarrow}C(1)$ **by** *blast*
AOT-hence ‹$x = y$› **by** (*metis* =*E−simple*:*2 vdash−properties*:*6*)
AOT-thus ‹$x = y \;\&\; \neg(x = y)$› **using** *1* &*I* **by** *blast*
next
AOT-assume ‹$[\lambda z\ z =_E x] \neq [\lambda z\ z =_E y]$›
AOT-hence *0*: ‹$\neg([\lambda z\ z =_E x] = [\lambda z\ z =_E y])$›
using $\equiv_{df}$*E*[*OF* =*−infix*] **by** *blast*
AOT-have ‹$[\lambda z\ z =_E x]\downarrow$› **by** *cqt*:*2*[*lambda*]
AOT-hence ‹$[\lambda z\ z =_E x] = [\lambda z\ z =_E x]$›
by (*metis rule=I*:*1*)
moreover AOT-assume ‹$x = y$›
ultimately AOT-have ‹$[\lambda z\ z =_E x] = [\lambda z\ z =_E y]$›
using *rule=E* **by** *fast*
AOT-thus ‹$[\lambda z\ z =_E x] = [\lambda z\ z =_E y] \;\&\; \neg([\lambda z\ z =_E x] = [\lambda z\ z =_E y])$›
using *0* &*I* **by** *blast*
qed

AOT-theorem *ord=E2:2*:
‹$(O!x \;\&\; O!y) \rightarrow (x \neq y \equiv [\lambda z\ z = x] \neq [\lambda z\ z = y])$›
proof (*rule* →*I*; *rule* ≡*I*; *rule* →*I*;
rule $\equiv_{df}$*I*[*OF* =*−infix*]; *rule raa−cor*:*2*)
AOT-assume *0*: ‹$O!x \;\&\; O!y$›
AOT-assume ‹$x \neq y$›
AOT-hence *1*: ‹$\neg(x = y)$› **using** $\equiv_{df}$*E*[*OF* =*−infix*] **by** *blast*
AOT-assume ‹$[\lambda z\ z = x] = [\lambda z\ z = y]$›
moreover AOT-have ‹$[\lambda z\ z = x]x$›

apply (*rule* $\beta\leftarrow C(1)$)
apply (*fact ord*$-$*=E=*:*2*[*THEN* $\rightarrow$*E*, *OF 0*[*THEN* &*E*(*1*)]])
apply (*fact cqt*:*2*[*const-var*][*axiom-inst*])
by (*simp add*: *id*$-$*eq*:*1*)
ultimately AOT-have ‹[λz $z = y$]x› **using** *rule=E* **by** *fast*
AOT-hence ‹$x = y$› **using** $\beta\rightarrow C(1)$ **by** *blast*
AOT-thus ‹$x = y$ & $\neg(x = y)$› **using** *1* &*I* **by** *blast*
next
AOT-assume *0*: ‹$O!x$ & $O!y$›
AOT-assume ‹[λz $z = x$] $\neq$ [λz $z = y$]›
AOT-hence *1*: ‹$\neg$([λz $z = x$] = [λz $z = y$])›
using $\equiv_{df}E$[*OF* =$-$*infix*] **by** *blast*
AOT-have ‹[λz $z = x$]$\downarrow$›
by (*fact ord*$-$*=E=*:*2*[*THEN* $\rightarrow$*E*, *OF 0*[*THEN* &*E*(*1*)]])
AOT-hence ‹[λz $z = x$] = [λz $z = x$]›
by (*metis rule=I*:*1*)
moreover AOT-assume ‹$x = y$›
ultimately AOT-have ‹[λz $z = x$] = [λz $z = y$]›
using *rule=E* **by** *fast*
AOT-thus ‹[λz $z = x$] = [λz $z = y$] & $\neg$([λz $z = x$] = [λz $z = y$])›
using *1* &*I* **by** *blast*
qed

AOT-theorem *ordnecfail*: ‹$O!x \rightarrow \Box\neg\exists F\ x[F]$›
by (*meson RM*:*1* $\rightarrow$*I nocoder*[*axiom-inst*] *oa*$-$*facts*:*1* $\rightarrow$*E*)

AOT-theorem *ab*$-$*obey*:*1*: ‹$(A!x\ \&\ A!y) \rightarrow (\forall F\ (x[F] \equiv y[F]) \rightarrow x = y)$›
proof (*rule* $\rightarrow$*I*; *rule* $\rightarrow$*I*)
AOT-assume *1*: ‹$A!x$ & $A!y$›
AOT-assume ‹$\forall F\ (x[F] \equiv y[F])$›
AOT-hence ‹$x[F] \equiv y[F]$› **for** F **using** $\forall E$ **by** *blast*
AOT-hence ‹$\Box(x[F] \equiv y[F])$› **for** F **by** (*metis en*$-$*eq*:*6*[*1*] $\equiv$*E*(*1*))
AOT-hence ‹$\forall F\ \Box(x[F] \equiv y[F])$› **by** (*rule GEN*)
AOT-hence ‹$\Box\forall F\ (x[F] \equiv y[F])$› **by** (*rule BF*[*THEN* $\rightarrow$*E*])
AOT-thus ‹$x = y$›
using $\equiv_{df}I$[*OF identity*:*1*, *OF* $\vee I(2)$] *1* &*I* **by** *blast*
qed

AOT-theorem *ab*$-$*obey*:*2*:
‹$(\exists F\ (x[F]\ \&\ \neg y[F]) \vee \exists F\ (y[F]\ \&\ \neg x[F])) \rightarrow x \neq y$›
proof (*rule* $\rightarrow$*I*; *rule* $\equiv_{df}I$[*OF* =$-$*infix*]; *rule raa*$-$*cor*:*2*)
AOT-assume *1*: ‹$x = y$›
AOT-assume ‹$\exists F\ (x[F]\ \&\ \neg y[F]) \vee \exists F\ (y[F]\ \&\ \neg x[F])$›
moreover {
AOT-assume ‹$\exists F\ (x[F]\ \&\ \neg y[F])$›
then AOT-obtain F **where** ‹$x[F]\ \&\ \neg y[F]$›
using $\exists E$[*rotated*] **by** *blast*
moreover AOT-have ‹$y[F]$›
using *calculation*[*THEN* &*E*(*1*)] *1 rule=E* **by** *fast*
ultimately AOT-have ‹$p\ \&\ \neg p$› **for** p
by (*metis Conjunction Simplification*(*2*) *modus*$-$*tollens*:*2 raa*$-$*cor*:*3*)
}
moreover {
AOT-assume ‹$\exists F\ (y[F]\ \&\ \neg x[F])$›
then AOT-obtain F **where** ‹$y[F]\ \&\ \neg x[F]$›
using $\exists E$[*rotated*] **by** *blast*
moreover AOT-have ‹$\neg y[F]$›
using *calculation*[*THEN* &*E*(*2*)] *1 rule=E* **by** *fast*
ultimately AOT-have ‹$p\ \&\ \neg p$› **for** p
by (*metis Conjunction Simplification*(*1*) *modus*$-$*tollens*:*1 raa*$-$*cor*:*3*)
}
ultimately AOT-show ‹$p\ \&\ \neg p$› **for** p
by (*metis* $\vee E(3)$ *raa*$-$*cor*:*1*)

qed

AOT-theorem *encoders−are−abstract*: ‹$\exists F\ x[F] \rightarrow A!x$›
by (*meson deduction−theorem* $\equiv E(2)$ *modus−tollens*:*2 nocoder oa−contingent*:*3 vdash−properties*:*1*[*2*])

AOT-theorem *denote=*:*1*: ‹$\forall H \exists x\ x[H]$›
by (*rule GEN*; *rule existence*:*2*[*1*][*THEN* $\equiv_{df}E$]; *cqt*:*2*)

AOT-theorem *denote=*:*2*: ‹$\forall G \exists x_1 ... \exists x_n\ x_1 ... x_n[H]$›
by (*rule GEN*; *rule existence*:*2*[*THEN* $\equiv_{df}E$]; *cqt*:*2*)

AOT-theorem *denote=*:*2*[*2*]: ‹$\forall G \exists x_1 \exists x_2\ x_1x_2[H]$›
by (*rule GEN*; *rule existence*:*2*[*2*][*THEN* $\equiv_{df}E$]; *cqt*:*2*)

AOT-theorem *denote=*:*2*[*3*]: ‹$\forall G \exists x_1 \exists x_2 \exists x_3\ x_1x_2x_3[H]$›
by (*rule GEN*; *rule existence*:*2*[*3*][*THEN* $\equiv_{df}E$]; *cqt*:*2*)

AOT-theorem *denote=*:*2*[*4*]: ‹$\forall G \exists x_1 \exists x_2 \exists x_3 \exists x_4\ x_1x_2x_3x_4[H]$›
by (*rule GEN*; *rule existence*:*2*[*4*][*THEN* $\equiv_{df}E$]; *cqt*:*2*)

AOT-theorem *denote=*:*3*: ‹$\exists x\ x[\Pi] \equiv \exists H\ (H = \Pi)$›
using *existence*:*2*[*1*] *free−thms*:*1* $\equiv E(2,5)$
Commutativity of $\equiv$ $\equiv Df$ **by** *blast*

AOT-theorem *denote=*:*4*: ‹$(\exists x_1 ... \exists x_n\ x_1 ... x_n[\Pi]) \equiv \exists H\ (H = \Pi)$›
using *existence*:*2 free−thms*:*1* $\equiv E(6)$ $\equiv Df$ **by** *blast*

AOT-theorem *denote=*:*4*[*2*]: ‹$(\exists x_1 \exists x_2\ x_1x_2[\Pi]) \equiv \exists H\ (H = \Pi)$›
using *existence*:*2*[*2*] *free−thms*:*1* $\equiv E(6)$ $\equiv Df$ **by** *blast*

AOT-theorem *denote=*:*4*[*3*]: ‹$(\exists x_1 \exists x_2 \exists x_3\ x_1x_2x_3[\Pi]) \equiv \exists H\ (H = \Pi)$›
using *existence*:*2*[*3*] *free−thms*:*1* $\equiv E(6)$ $\equiv Df$ **by** *blast*

AOT-theorem *denote=*:*4*[*4*]: ‹$(\exists x_1 \exists x_2 \exists x_3 \exists x_4\ x_1x_2x_3x_4[\Pi]) \equiv \exists H\ (H = \Pi)$›
using *existence*:*2*[*4*] *free−thms*:*1* $\equiv E(6)$ $\equiv Df$ **by** *blast*

AOT-theorem *A−objects*!: ‹$\exists !x\ (A!x\ \&\ \forall F\ (x[F] \equiv \varphi\{F\}))$›
proof (*rule uniqueness*:*1*[*THEN* $\equiv_{df}I$])
AOT-obtain *a* **where** *a-prop*: ‹$A!a\ \&\ \forall F\ (a[F] \equiv \varphi\{F\})$›
using *A−objects*[*axiom-inst*] $\exists E$[*rotated*] **by** *blast*
AOT-have ‹$(A!\beta\ \&\ \forall F\ (\beta[F] \equiv \varphi\{F\})) \rightarrow \beta = a$› **for** β
proof (*rule* $\rightarrow I$)
AOT-assume β*-prop*: ‹$[A!]\beta\ \&\ \forall F\ (\beta[F] \equiv \varphi\{F\})$›
AOT-hence ‹$\beta[F] \equiv \varphi\{F\}$› **for** F
using $\forall E$ $\& E$ **by** *blast*
AOT-hence ‹$\beta[F] \equiv a[F]$› **for** F
using *a-prop*[*THEN* $\& E(2)$] $\forall E$ $\equiv E(2,5)$
Commutativity of $\equiv$ **by** *fast*
AOT-hence ‹$\forall F\ (\beta[F] \equiv a[F])$› **by** (*rule GEN*)
AOT-thus ‹$\beta = a$›
using *ab−obey*:*1*[*THEN* $\rightarrow E$,
OF $\& I$[*OF* β*-prop*[*THEN* $\& E(1)$], *OF a-prop*[*THEN* $\& E(1)$]],
THEN $\rightarrow E$] **by** *blast*
qed
AOT-hence ‹$\forall \beta\ ((A!\beta\ \&\ \forall F\ (\beta[F] \equiv \varphi\{F\})) \rightarrow \beta = a)$› **by** (*rule GEN*)
AOT-thus ‹$\exists \alpha\ ([A!]\alpha\ \&\ \forall F\ (\alpha[F] \equiv \varphi\{F\})\ \&$
$\forall \beta\ ([A!]\beta\ \&\ \forall F\ (\beta[F] \equiv \varphi\{F\}) \rightarrow \beta = \alpha))$›
using $\exists I$ **using** *a-prop* $\& I$ **by** *fast*
qed

AOT-theorem *obj−oth*:*1*: ‹$\exists !x\ (A!x\ \&\ \forall F\ (x[F] \equiv [F]y))$›
using *A−objects*! **by** *fast*

AOT-theorem *obj−oth:2*: ‹∃!x ($A!x$ & ∀ F ($x[F]$ ≡ $[F]y$ & $[F]z$))›
  using *A−objects*! **by** *fast*

AOT-theorem *obj−oth:3*: ‹∃!x ($A!x$ & ∀ F ($x[F]$ ≡ $[F]y$ ∨ $[F]z$))›
  using *A−objects*! **by** *fast*

AOT-theorem *obj−oth:4*: ‹∃!x ($A!x$ & ∀ F ($x[F]$ ≡ □$[F]y$))›
  using *A−objects*! **by** *fast*

AOT-theorem *obj−oth:5*: ‹∃!x ($A!x$ & ∀ F ($x[F]$ ≡ F = G))›
  using *A−objects*! **by** *fast*

AOT-theorem *obj−oth:6*: ‹∃!x ($A!x$ & ∀ F ($x[F]$ ≡ □∀ y($[G]y$ → $[F]y$)))›
  using *A−objects*! **by** *fast*

AOT-theorem *A−descriptions*: ‹ɩx ($A!x$ & ∀ F ($x[F]$ ≡ $\varphi\{F\}$))↓›
  by (*rule A−Exists:2*[*THEN* ≡*E*(*2*)]; *rule RA*[*2*]; *rule A−objects*!)

AOT-act-theorem *thm−can−terms2*:
  ‹y = ɩx($A!x$ & ∀ F ($x[F]$ ≡ $\varphi\{F\}$)) → ($A!y$ & ∀ F ($y[F]$ ≡ $\varphi\{F\}$))›
  using *y−in:2* **by** *blast*

AOT-theorem *can−ab2*: ‹y = ɩx($A!x$ & ∀ F ($x[F]$ ≡ $\varphi\{F\}$)) → $A!y$›
proof(*rule* →*I*)
  AOT-assume ‹y = ɩx($A!x$ & ∀ F ($x[F]$ ≡ $\varphi\{F\}$))›
  AOT-hence ‹𝓐($A!y$ & ∀ F ($y[F]$ ≡ $\varphi\{F\}$))›
    using *actual−desc:2*[*THEN* →*E*] **by** *blast*
  AOT-hence ‹𝓐$A!y$› **by** (*metis Act−Basic:2* &*E*(*1*) ≡*E*(*1*))
  AOT-thus ‹$A!y$› **by** (*metis* ≡*E*(*2*) *oa−facts:8*)
qed

AOT-act-theorem *desc−encode:1*: ‹ɩx($A!x$ & ∀ F ($x[F]$ ≡ $\varphi\{F\}$))[F] ≡ $\varphi\{F\}$›
proof −
  AOT-have ‹ɩx($A!x$ & ∀ F ($x[F]$ ≡ $\varphi\{F\}$))↓›
    by (*simp add*: *A−descriptions*)
  AOT-hence ‹$A!$ɩx($A!x$ & ∀ F ($x[F]$ ≡ $\varphi\{F\}$)) &
      ∀ F(ɩx($A!x$ & ∀ F ($x[F]$ ≡ $\varphi\{F\}$))[F] ≡ $\varphi\{F\}$)›
    using *y−in:3*[*THEN* →*E*] **by** *blast*
  AOT-thus ‹ɩx($A!x$ & ∀ F ($x[F]$ ≡ $\varphi\{F\}$))[F] ≡ $\varphi\{F\}$›
    using &*E* ∀ *E* **by** *blast*
qed

AOT-act-theorem *desc−encode:2*: ‹ɩx($A!x$ & ∀ F ($x[F]$ ≡ $\varphi\{F\}$))[G] ≡ $\varphi\{G\}$›
  using *desc−encode:1*.

AOT-theorem *desc−nec−encode:1*:
  ‹ɩx ($A!x$ & ∀ F ($x[F]$ ≡ $\varphi\{F\}$))[F] ≡ 𝓐$\varphi\{F\}$›
proof −
  AOT-have *0*: ‹ɩx($A!x$ & ∀ F ($x[F]$ ≡ $\varphi\{F\}$))↓›
    by (*simp add*: *A−descriptions*)
  AOT-hence ‹𝓐($A!$ɩx($A!x$ & ∀ F ($x[F]$ ≡ $\varphi\{F\}$)) &
      ∀ F(ɩx($A!x$ & ∀ F ($x[F]$ ≡ $\varphi\{F\}$))[F] ≡ $\varphi\{F\}$))›
    using *actual−desc:4*[*THEN* →*E*] **by** *blast*
  AOT-hence ‹𝓐∀ F (ɩx($A!x$ & ∀ F ($x[F]$ ≡ $\varphi\{F\}$))[F] ≡ $\varphi\{F\}$)›
    using *Act−Basic:2* &*E*(*2*) ≡*E*(*1*) **by** *blast*
  AOT-hence ‹∀ F 𝓐(ɩx($A!x$ & ∀ F ($x[F]$ ≡ $\varphi\{F\}$))[F] ≡ $\varphi\{F\}$)›
    using ≡*E*(*1*) *logic−actual−nec:3 vdash−properties:1*[*2*] **by** *blast*
  AOT-hence ‹𝓐(ɩx($A!x$ & ∀ F ($x[F]$ ≡ $\varphi\{F\}$))[F] ≡ $\varphi\{F\}$)›
    using ∀ *E* **by** *blast*
  AOT-hence ‹𝓐ɩx($A!x$ & ∀ F ($x[F]$ ≡ $\varphi\{F\}$))[F] ≡ 𝓐$\varphi\{F\}$›
    using *Act−Basic:5* ≡*E*(*1*) **by** *blast*
  AOT-thus ‹ɩx($A!x$ & ∀ F ($x[F]$ ≡ $\varphi\{F\}$))[F] ≡ 𝓐$\varphi\{F\}$›

```
    using en−eq:10[1][unvarify x₁, OF 0] ≡E(6) by blast
qed

AOT-theorem desc−nec−encode:2:
  ‹ιx (A!x & ∀F (x[F] ≡ φ{F}))[G] ≡ 𝓐φ{G}›
  using desc−nec−encode:1.

AOT-theorem Box−desc−encode:1: ‹□φ{G} → ιx(A!x & ∀F (x[F] ≡ φ{G}))[G]›
  by (rule →I; rule desc−nec−encode:2[THEN ≡E(2)])
     (meson nec−imp−act vdash−properties:10)

AOT-theorem Box−desc−encode:2:
‹□φ{G} → □(ιx(A!x & ∀F (x[F] ≡ φ{G}))[G] ≡ φ{G})›
proof(rule CP)
  AOT-assume ‹□φ{G}›
  AOT-hence ‹□□φ{G}› by (metis S5Basic:6 ≡E(1))
  moreover AOT-have ‹□□φ{G} → □(ιx(A!x & ∀F (x[F] ≡ φ{G}))[G] ≡ φ{G})›
  proof (rule RM; rule →I)
    AOT-modally-strict {
      AOT-assume 1: ‹□φ{G}›
      AOT-hence ‹ιx(A!x & ∀F (x[F] ≡ φ{G}))[G]›
        using Box−desc−encode:1 →E by blast
      moreover AOT-have ‹φ{G}›
        using 1 by (meson qml:2[axiom-inst] →E)
      ultimately AOT-show ‹ιx(A!x & ∀F (x[F] ≡ φ{G}))[G] ≡ φ{G}›
        using →I ≡I by simp
    }
  qed
  ultimately AOT-show ‹□(ιx(A!x & ∀F (x[F] ≡ φ{G}))[G] ≡ φ{G})›
    using →E by blast
qed

definition rigid-condition where
  ‹rigid-condition φ ≡ ∀v . [v ⊨ ∀α (φ{α} → □φ{α})]›
syntax rigid-condition :: ‹id-position ⇒ AOT-prop› (‹RIGID′-CONDITION′(-′)›)

AOT-theorem strict−can:1[E]:
  assumes ‹RIGID-CONDITION(φ)›
  shows ‹∀α (φ{α} → □φ{α})›
  using assms[unfolded rigid-condition-def] by auto

AOT-theorem strict−can:1[I]:
  assumes ‹⊢□ ∀α (φ{α} → □φ{α})›
  shows ‹RIGID-CONDITION(φ)›
  using assms rigid-condition-def by auto

AOT-theorem box−phi−a:1:
  assumes ‹RIGID-CONDITION(φ)›
  shows ‹(A!x & ∀F (x[F] ≡ φ{F})) → □(A!x & ∀F (x[F] ≡ φ{F}))›
proof (rule →I)
  AOT-assume a: ‹A!x & ∀F (x[F] ≡ φ{F})›
  AOT-hence b: ‹□A!x›
    by (metis Conjunction Simplification(1) oa−facts:2 →E)
  AOT-have ‹x[F] ≡ φ{F}› for F
    using a[THEN &E(2)] ∀E by blast
  moreover AOT-have ‹□(x[F] → □x[F])› for F
    by (meson pre−en−eq:1[1] RN)
  moreover AOT-have ‹□(φ{F} → □φ{F})› for F
    using RN strict−can:1[E][OF assms] ∀E by blast
  ultimately AOT-have ‹□(x[F] ≡ φ{F})› for F
    using sc−eq−box−box:5 qml:2[axiom-inst, THEN →E] →E &I by metis
  AOT-hence ‹∀F □(x[F] ≡ φ{F})› by (rule GEN)
  AOT-hence ‹□∀F (x[F] ≡ φ{F})› by (rule BF[THEN →E])
```

```
  AOT-thus ‹□([A!]x & ∀ F (x[F] ≡ φ{F}))›
    using b KBasic:3 ≡S(1) ≡E(2) by blast
qed

AOT-theorem box−phi−a:2:
  assumes ‹RIGID-CONDITION(φ)›
  shows ‹y = ιx(A!x & ∀ F (x[F] ≡ φ{F})) → (A!y & ∀ F (y[F] ≡ φ{F}))›
proof(rule →I)
  AOT-assume ‹y = ιx(A!x & ∀ F (x[F] ≡ φ{F}))›
  AOT-hence ‹𝓐(A!y & ∀ F (y[F] ≡ φ{F}))›
    using actual−desc:2[THEN →E] by fast
  AOT-hence abs: ‹𝓐A!y› and ‹𝓐∀ F (y[F] ≡ φ{F})›
    using Act−Basic:2 &E ≡E(1) by blast+
  AOT-hence ‹∀ F 𝓐(y[F] ≡ φ{F})›
    by (metis ≡E(1) logic−actual−nec:3 vdash−properties:1[2])
  AOT-hence ‹𝓐(y[F] ≡ φ{F})› for F
    using ∀ E by blast
  AOT-hence ‹𝓐y[F] ≡ 𝓐φ{F}› for F
    by (metis Act−Basic:5 ≡E(1))
  AOT-hence ‹y[F] ≡ φ{F}› for F
    using sc−eq−fur:2[THEN →E,
            OF strict−can:1[E][OF assms,
                THEN ∀ E(2)[where β=F], THEN RN]]
    by (metis en−eq:10[1] ≡E(6))
  AOT-hence ‹∀ F (y[F] ≡ φ{F})› by (rule GEN)
  AOT-thus ‹[A!]y & ∀ F (y[F] ≡ φ{F})›
    using abs &I ≡E(2) oa−facts:8 by blast
qed

AOT-theorem box−phi−a:3:
  assumes ‹RIGID-CONDITION(φ)›
  shows ‹ιx(A!x & ∀ F (x[F] ≡ φ{F}))[F] ≡ φ{F}›
  using desc−nec−encode:2
    sc−eq−fur:2[THEN →E,
        OF strict−can:1[E][OF assms,
          THEN ∀ E(2)[where β=F], THEN RN]]
    ≡E(5) by blast

AOT-define Null :: ‹τ ⇒ φ› (‹Null′(-′)›)
  df−null−uni:1: ‹Null(x) ≡df A!x & ¬∃ F x[F]›

AOT-define Universal :: ‹τ ⇒ φ› (‹Universal′(-′)›)
  df−null−uni:2: ‹Universal(x) ≡df A!x & ∀ F x[F]›

AOT-theorem null−uni−uniq:1: ‹∃ !x Null(x)›
proof (rule uniqueness:1[THEN ≡df I])
  AOT-obtain a where a-prop: ‹A!a & ∀ F (a[F] ≡ ¬(F = F))›
    using A−objects[axiom-inst] ∃ E[rotated] by fast
  AOT-have a-null: ‹¬a[F]› for F
  proof (rule raa−cor:2)
    AOT-assume ‹a[F]›
    AOT-hence ‹¬(F = F)› using a-prop[THEN &E(2)] ∀ E ≡E by blast
    AOT-hence ‹F = F & ¬(F = F)› by (metis id−eq:1 raa−cor:3)
    AOT-thus ‹p & ¬p› for p  by (metis raa−cor:1)
  qed
  AOT-have ‹Null(a) & ∀ β (Null(β) → β = a)›
  proof (rule &I)
    AOT-have ‹¬∃ F a[F]›
      using a-null by (metis instantiation reductio−aa:1)
    AOT-thus ‹Null(a)›
      using df−null−uni:1[THEN ≡df I] a-prop[THEN &E(1)] &I by metis
  next
    AOT-show ‹∀ β (Null(β) → β = a)›
```

```
    proof (rule GEN; rule →I)
      fix β
      AOT-assume a: ‹Null(β)›
      AOT-hence ‹¬∃ F β[F]›
        using df−null−uni:1[THEN ≡df E] &E by blast
      AOT-hence β-null: ‹¬β[F]› for F
        by (metis existential:2[const-var] reductio−aa:1)
      AOT-have ‹∀ F (β[F] ≡ a[F])›
        apply (rule GEN; rule ≡I; rule CP)
        using raa−cor:3 β-null a-null by blast+
      moreover AOT-have ‹A!β›
        using a df−null−uni:1[THEN ≡df E] &E by blast
      ultimately AOT-show ‹β = a›
        using a-prop[THEN &E(1)] ab−obey:1[THEN →E, THEN →E]
              &I by blast
    qed
  qed
  AOT-thus ‹∃ α (Null(α) & ∀ β (Null(β) → β = α))›
    using ∃ I(2) by fast
qed

AOT-theorem null−uni−uniq:2: ‹∃ !x Universal(x)›
proof (rule uniqueness:1[THEN ≡df I])
  AOT-obtain a where a-prop: ‹A!a & ∀ F (a[F] ≡ F = F)›
    using A−objects[axiom-inst] ∃ E[rotated] by fast
  AOT-hence aF: ‹a[F]› for F using &E ∀ E ≡E id−eq:1 by fast
  AOT-hence ‹Universal(a)›
    using df−null−uni:2[THEN ≡df I] &I a-prop[THEN &E(1)] GEN by blast
  moreover AOT-have ‹∀ β (Universal(β) → β = a)›
  proof (rule GEN; rule →I)
    fix β
    AOT-assume ‹Universal(β)›
    AOT-hence abs-β: ‹A!β› and ‹β[F]› for F
      using df−null−uni:2[THEN ≡df E] &E ∀ E by blast+
    AOT-hence ‹β[F] ≡ a[F]› for F
      using aF by (metis deduction−theorem ≡I)
    AOT-hence ‹∀ F (β[F] ≡ a[F])› by (rule GEN)
    AOT-thus ‹β = a›
      using a-prop[THEN &E(1)] ab−obey:1[THEN →E, THEN →E]
            &I abs-β by blast
  qed
  ultimately AOT-show ‹∃ α (Universal(α) & ∀ β (Universal(β) → β = α))›
    using &I ∃ I by fast
qed

AOT-theorem null−uni−uniq:3: ‹ιx Null(x)↓›
  using A−Exists:2 RA[2] ≡E(2) null−uni−uniq:1 by blast

AOT-theorem null−uni−uniq:4: ‹ιx Universal(x)↓›
  using A−Exists:2 RA[2] ≡E(2) null−uni−uniq:2 by blast

AOT-define Null-object :: ‹κs› (‹a∅›)
  df−null−uni−terms:1: ‹a∅ =df ιx Null(x)›

AOT-define Universal-object :: ‹κs› (‹aV›)
  df−null−uni−terms:2: ‹aV =df ιx Universal(x)›

AOT-theorem null−uni−facts:1: ‹Null(x) → □Null(x)›
proof (rule →I)
  AOT-assume ‹Null(x)›
  AOT-hence x-abs: ‹A!x› and x-null: ‹¬∃ F x[F]›
    using df−null−uni:1[THEN ≡df E] &E by blast+
  AOT-have ‹¬x[F]› for F using x-null
```

```
    using existential:2[const-var] reductio−aa:1
    by metis
  AOT-hence ‹□¬x[F]› for F by (metis en−eq:7[1] ≡E(1))
  AOT-hence ‹∀F □¬x[F]› by (rule GEN)
  AOT-hence ‹□∀F ¬x[F]› by (rule BF[THEN →E])
  moreover AOT-have ‹□∀F ¬x[F] → □¬∃F x[F]›
    apply (rule RM)
    by (metis (full-types) instantiation cqt:2[const-var][axiom-inst]
                        →I reductio−aa:1 rule−ui:1)
  ultimately AOT-have ‹□¬∃F x[F]›
    by (metis →E)
  moreover AOT-have ‹□A!x› using x-abs
    using oa−facts:2 vdash−properties:10 by blast
  ultimately AOT-have r: ‹□(A!x & ¬∃F x[F])›
    by (metis KBasic:3 &I ≡E(3) raa−cor:3)
  AOT-show ‹□Null(x)›
    by (AOT-subst ‹Null(x)› ‹A!x & ¬∃F x[F]›)
       (auto simp: df−null−uni:1 ≡Df r)
qed

AOT-theorem null−uni−facts:2: ‹Universal(x) → □Universal(x)›
proof (rule →I)
  AOT-assume ‹Universal(x)›
  AOT-hence x-abs: ‹A!x› and x-univ: ‹∀F x[F]›
    using df−null−uni:2[THEN ≡_{df}E] &E by blast+
  AOT-have ‹x[F]› for F using x-univ ∀E by blast
  AOT-hence ‹□x[F]› for F by (metis en−eq:2[1] ≡E(1))
  AOT-hence ‹∀F □x[F]› by (rule GEN)
  AOT-hence ‹□∀F x[F]› by (rule BF[THEN →E])
  moreover AOT-have ‹□A!x› using x-abs
    using oa−facts:2 vdash−properties:10 by blast
  ultimately AOT-have r: ‹□(A!x & ∀F x[F])›
    by (metis KBasic:3 &I ≡E(3) raa−cor:3)
  AOT-show ‹□Universal(x)›
    by (AOT-subst ‹Universal(x)› ‹A!x & ∀F x[F]›)
       (auto simp add: df−null−uni:2 ≡Df r)
qed

AOT-theorem null−uni−facts:3: ‹Null(a_∅)›
  apply (rule =_{df}I(2)[OF df−null−uni−terms:1])
   apply (simp add: null−uni−uniq:3)
  using actual−desc:4[THEN →E, OF null−uni−uniq:3]
    sc−eq−fur:2[THEN →E,
        OF null−uni−facts:1[unvarify x, THEN RN, OF null−uni−uniq:3],
        THEN ≡E(1)]
  by blast

AOT-theorem null−uni−facts:4: ‹Universal(a_V)›
  apply (rule =_{df}I(2)[OF df−null−uni−terms:2])
   apply (simp add: null−uni−uniq:4)
  using actual−desc:4[THEN →E, OF null−uni−uniq:4]
    sc−eq−fur:2[THEN →E,
        OF null−uni−facts:2[unvarify x, THEN RN, OF null−uni−uniq:4],
        THEN ≡E(1)]
  by blast

AOT-theorem null−uni−facts:5: ‹a_∅ ≠ a_V›
proof (rule =_{df}I(2)[OF df−null−uni−terms:1, OF null−uni−uniq:3];
    rule =_{df}I(2)[OF df−null−uni−terms:2, OF null−uni−uniq:4];
    rule ≡_{df}I[OF =−infix];
    rule raa−cor:2)
  AOT-obtain x where nullx: ‹Null(x)›
    by (metis instantiation df−null−uni−terms:1 existential:1
```

null−uni−facts:*3* *null−uni−uniq*:*3* *rule−id−df*:*2*:*b*[*zero*])
AOT-hence *act-null*: ‹$\mathcal{A}$*Null*(*x*)›
by (*metis nec−imp−act null−uni−facts*:*1* →*E*)
AOT-assume ‹ιx *Null*(*x*) = ιx *Universal*(*x*)›
AOT-hence ‹$\mathcal{A}$∀ *x*(*Null*(*x*) ≡ *Universal*(*x*))›
using *actual−desc*:*5*[*THEN* →*E*] **by** *blast*
AOT-hence ‹∀ *x* $\mathcal{A}$(*Null*(*x*) ≡ *Universal*(*x*))›
by (*metis* ≡*E*(*1*) *logic−actual−nec*:*3* *vdash−properties*:*1*[*2*])
AOT-hence ‹$\mathcal{A}$*Null*(*x*) ≡ $\mathcal{A}$*Universal*(*x*)›
using *Act−Basic*:*5* ≡*E*(*1*) *rule−ui*:*3* **by** *blast*
AOT-hence ‹$\mathcal{A}$*Universal*(*x*)› **using** *act-null* ≡*E* **by** *blast*
AOT-hence ‹*Universal*(*x*)›
by (*metis RN* ≡*E*(*1*) *null−uni−facts*:*2* *sc−eq−fur*:*2* →*E*)
AOT-hence ‹∀ *F x*[*F*]› **using** $\equiv_{df}$*E*[*OF df−null−uni*:*2*] &*E* **by** *metis*
moreover AOT-have ‹¬∃ *F x*[*F*]›
using *nullx* $\equiv_{df}$*E*[*OF df−null−uni*:*1*] &*E* **by** *metis*
ultimately AOT-show ‹*p* & ¬*p*› **for** *p*
by (*metis cqt−further*:*1* *raa−cor*:*3* →*E*)
qed

AOT-theorem *null−uni−facts*:*6*: ‹$a_\emptyset$ = ιx(*A*!*x* & ∀ *F* (*x*[*F*] ≡ *F* ≠ *F*))›
proof (*rule ab−obey*:*1*[*unvarify x y*, *THEN* →*E*, *THEN* →*E*])
AOT-show ‹ιx([*A*!]*x* & ∀ *F* (*x*[*F*] ≡ *F* ≠ *F*))↓›
by (*simp add*: *A−descriptions*)
next
AOT-show ‹$a_\emptyset$↓›
by (*rule* $=_{df}$*I*(*2*)[*OF df−null−uni−terms*:*1*, *OF null−uni−uniq*:*3*])
(*simp add*: *null−uni−uniq*:*3*)
next
AOT-have ‹ιx([*A*!]*x* & ∀ *F* (*x*[*F*] ≡ *F* ≠ *F*))↓›
by (*simp add*: *A−descriptions*)
AOT-hence *1*: ‹ιx([*A*!]*x* & ∀ *F* (*x*[*F*] ≡ *F* ≠ *F*)) = ιx([*A*!]*x* & ∀ *F* (*x*[*F*] ≡ *F* ≠ *F*))›
using *rule*=*I*:*1* **by** *blast*
AOT-show ‹[*A*!]$a_\emptyset$ & [*A*!]ιx([*A*!]*x* & ∀ *F* (*x*[*F*] ≡ *F* ≠ *F*))›
apply (*rule* $=_{df}$*I*(*2*)[*OF df−null−uni−terms*:*1*, *OF null−uni−uniq*:*3*];
rule &*I*)
apply (*meson* $\equiv_{df}$*E Conjunction Simplification*(*1*)
df−null−uni:*1* *df−null−uni−terms*:*1* *null−uni−facts*:*3*
null−uni−uniq:*3* *rule−id−df*:*2*:*a*[*zero*] →*E*)
using *can−ab2*[*unvarify y*, *OF A−descriptions*, *THEN* →*E*, *OF 1*]**.**
next
AOT-show ‹∀ *F* ($a_\emptyset$[*F*] ≡ ιx([*A*!]*x* & ∀ *F* (*x*[*F*] ≡ *F* ≠ *F*))[*F*])›
proof (*rule GEN*)
fix *F*
AOT-have ‹¬$a_\emptyset$[*F*]›
by (*rule* $=_{df}$*I*(*2*)[*OF df−null−uni−terms*:*1*, *OF null−uni−uniq*:*3*])
(*metis* (*no-types*, *lifting*) $\equiv_{df}$*E* &*E*(*2*) ∨*I*(*2*) ∨*E*(*3*) ∃ *I*(*2*)
df−null−uni:*1* *df−null−uni−terms*:*1* *null−uni−facts*:*3*
raa−cor:*2* *rule−id−df*:*2*:*a*[*zero*]
russell−axiom[*enc*,*1*].*ψ-denotes-asm*)
moreover AOT-have ‹¬ιx([*A*!]*x* & ∀ *F* (*x*[*F*] ≡ *F* ≠ *F*))[*F*]›
proof(*rule raa−cor*:*2*)
AOT-assume *0*: ‹ιx([*A*!]*x* & ∀ *F* (*x*[*F*] ≡ *F* ≠ *F*))[*F*]›
AOT-hence ‹$\mathcal{A}$(*F* ≠ *F*)›
using *desc−nec−encode*:*2*[*THEN* ≡*E*(*1*), *OF 0*] **by** *blast*
moreover AOT-have ‹¬$\mathcal{A}$(*F* ≠ *F*)›
using $\equiv_{df}$*E id−act*:*2* *id−eq*:*1* ≡*E*(*2*)
=−*infix raa−cor*:*3* **by** *blast*
ultimately AOT-show ‹$\mathcal{A}$(*F* ≠ *F*) & ¬$\mathcal{A}$(*F* ≠ *F*)› **by** (*rule* &*I*)
qed
ultimately AOT-show ‹$a_\emptyset$[*F*] ≡ ιx([*A*!]*x* & ∀ *F* (*x*[*F*] ≡ *F* ≠ *F*))[*F*]›
using *deduction−theorem* ≡*I* *raa−cor*:*4* **by** *blast*
qed

```
qed

AOT-theorem null−uni−facts:7: ‹a_V = ιx(A!x & ∀F (x[F] ≡ F = F))›
proof (rule ab−obey:1[unvarify x y, THEN →E, THEN →E])
  AOT-show ‹ιx([A!]x & ∀F (x[F] ≡ F = F))↓›
    by (simp add: A−descriptions)
next
  AOT-show ‹a_V↓›
    by (rule =_df I(2)[OF df−null−uni−terms:2, OF null−uni−uniq:4])
       (simp add: null−uni−uniq:4)
next
  AOT-have ‹ιx([A!]x & ∀F (x[F] ≡ F = F))↓›
    by (simp add: A−descriptions)
  AOT-hence 1: ‹ιx([A!]x & ∀F (x[F] ≡ F = F)) = ιx([A!]x & ∀F (x[F] ≡ F = F))›
    using rule=I:1 by blast
  AOT-show ‹[A!]a_V & [A!]ιx([A!]x & ∀F (x[F] ≡ F = F))›
    apply (rule =_df I(2)[OF df−null−uni−terms:2, OF null−uni−uniq:4];
           rule &I)
     apply (meson ≡_df E Conjunction Simplification(1) df−null−uni:2
                  df−null−uni−terms:2 null−uni−facts:4 null−uni−uniq:4
                  rule−id−df:2:a[zero] →E)
    using can−ab2[unvarify y, OF A−descriptions, THEN →E, OF 1].
next
  AOT-show ‹∀F (a_V[F] ≡ ιx([A!]x & ∀F (x[F] ≡ F = F))[F])›
  proof (rule GEN)
    fix F
    AOT-have ‹a_V[F]›
      apply (rule =_df I(2)[OF df−null−uni−terms:2, OF null−uni−uniq:4])
      using ≡_df E &E(2) df−null−uni:2 df−null−uni−terms:2
            null−uni−facts:4 null−uni−uniq:4 rule−id−df:2:a[zero]
            rule−ui:3 by blast
    moreover AOT-have ‹ιx([A!]x & ∀F (x[F] ≡ F = F))[F]›
      using RA[2] desc−nec−encode:2 id−eq:1 ≡E(2) by fastforce
    ultimately AOT-show ‹a_V[F] ≡ ιx([A!]x & ∀F (x[F] ≡ F = F))[F]›
      using deduction−theorem ≡I by simp
  qed
qed

AOT-theorem aclassical:1:
‹∀R∃x∃y(A!x & A!y & x ≠ y & [λz [R]zx] = [λz [R]zy])›
proof(rule GEN)
  fix R
  AOT-obtain a where a-prop:
    ‹A!a & ∀F (a[F] ≡ ∃y(A!y & F = [λz [R]zy] & ¬y[F]))›
    using A−objects[axiom-inst] ∃E[rotated] by fast
  AOT-have a-enc: ‹a[λz [R]za]›
  proof (rule raa−cor:1)
    AOT-assume 0: ‹¬a[λz [R]za]›
    AOT-hence ‹¬∃y(A!y & [λz [R]za] = [λz [R]zy] & ¬y[λz [R]za])›
      by (rule a-prop[THEN &E(2), THEN ∀E(1)[where τ=«[λz [R]za]»],
                THEN oth−class−taut:4:b[THEN ≡E(1)],
                THEN ≡E(1), rotated])
         cqt:2[lambda]
    AOT-hence ‹∀y ¬(A!y & [λz [R]za] = [λz [R]zy] & ¬y[λz [R]za])›
      using cqt−further:4 vdash−properties:10 by blast
    AOT-hence ‹¬(A!a & [λz [R]za] = [λz [R]za] & ¬a[λz [R]za])›
      using ∀E by blast
    AOT-hence ‹(A!a & [λz [R]za] = [λz [R]za]) → a[λz [R]za]›
      by (metis &I deduction−theorem raa−cor:3)
    moreover AOT-have ‹[λz [R]za] = [λz [R]za]›
      by (rule =I) cqt:2[lambda]
    ultimately AOT-have ‹a[λz [R]za]›
      using a-prop[THEN &E(1)] →E &I by blast
```

```
    AOT-thus ‹a[λz [R]za] & ¬a[λz [R]za]›
      using 0 &I by blast
  qed
  AOT-hence ‹∃ y(A!y & [λz [R]za] = [λz [R]zy] & ¬y[λz [R]za])›
    by (rule a-prop[THEN &E(2), THEN ∀ E(1), THEN ≡E(1), rotated])
       cqt:2
  then AOT-obtain b where b-prop:
    ‹A!b & [λz [R]za] = [λz [R]zb] & ¬b[λz [R]za]›
    using ∃ E[rotated] by blast
  AOT-have ‹a ≠ b›
    apply (rule ≡df I[OF =−infix])
    using a-enc b-prop[THEN &E(2)]
    using ¬¬I rule=E id-sym ≡E(4) oth−class−taut:3:a
          raa−cor:3 reductio−aa:1 by fast
  AOT-hence ‹A!a & A!b & a ≠ b & [λz [R]za] = [λz [R]zb]›
    using b-prop &E a-prop &I by meson
  AOT-hence ‹∃ y (A!a & A!y & a ≠ y & [λz [R]za] = [λz [R]zy])› by (rule ∃ I)
  AOT-thus ‹∃ x∃ y (A!x & A!y & x ≠ y & [λz [R]zx] = [λz [R]zy])› by (rule ∃ I)
qed

AOT-theorem aclassical:2:
  ‹∀ R∃ x∃ y(A!x & A!y & x ≠ y & [λz [R]xz] = [λz [R]yz])›
proof(rule GEN)
  fix R
  AOT-obtain a where a-prop:
    ‹A!a & ∀ F (a[F] ≡ ∃ y(A!y & F = [λz [R]yz] & ¬y[F]))›
    using A−objects[axiom-inst] ∃ E[rotated] by fast
  AOT-have a-enc: ‹a[λz [R]az]›
  proof (rule raa−cor:1)
    AOT-assume 0: ‹¬a[λz [R]az]›
    AOT-hence ‹¬∃ y(A!y & [λz [R]az] = [λz [R]yz] & ¬y[λz [R]az])›
      by (rule a-prop[THEN &E(2), THEN ∀ E(1)[where τ=«[λz [R]az]»],
                THEN oth−class−taut:4:b[THEN ≡E(1)],
                THEN ≡E(1), rotated])
         cqt:2[lambda]
    AOT-hence ‹∀ y ¬(A!y & [λz [R]az] = [λz [R]yz] & ¬y[λz [R]az])›
      using cqt−further:4 vdash−properties:10 by blast
    AOT-hence ‹¬(A!a & [λz [R]az] = [λz [R]az] & ¬a[λz [R]az])›
      using ∀ E by blast
    AOT-hence ‹(A!a & [λz [R]az] = [λz [R]az]) → a[λz [R]az]›
      by (metis &I deduction−theorem raa−cor:3)
    moreover AOT-have ‹[λz [R]az] = [λz [R]az]›
      by (rule =I) cqt:2[lambda]
    ultimately AOT-have ‹a[λz [R]az]›
      using a-prop[THEN &E(1)] →E &I by blast
    AOT-thus ‹a[λz [R]az] & ¬a[λz [R]az]›
      using 0 &I by blast
  qed
  AOT-hence ‹∃ y(A!y & [λz [R]az] = [λz [R]yz] & ¬y[λz [R]az])›
    by (rule a-prop[THEN &E(2), THEN ∀ E(1), THEN ≡E(1), rotated])
       cqt:2
  then AOT-obtain b where b-prop:
    ‹A!b & [λz [R]az] = [λz [R]bz] & ¬b[λz [R]az]›
    using ∃ E[rotated] by blast
  AOT-have ‹a ≠ b›
    apply (rule ≡df I[OF =−infix])
    using a-enc b-prop[THEN &E(2)]
    using ¬¬I rule=E id-sym ≡E(4) oth−class−taut:3:a
          raa−cor:3 reductio−aa:1 by fast
  AOT-hence ‹A!a & A!b & a ≠ b & [λz [R]az] = [λz [R]bz]›
    using b-prop &E a-prop &I by meson
  AOT-hence ‹∃ y (A!a & A!y & a ≠ y & [λz [R]az] = [λz [R]yz])› by (rule ∃ I)
  AOT-thus ‹∃ x∃ y (A!x & A!y & x ≠ y & [λz [R]xz] = [λz [R]yz])› by (rule ∃ I)
```

```
qed

AOT-theorem aclassical:3:
  ‹∀F∃x∃y(A!x & A!y & x ≠ y & [λ [F]x] = [λ [F]y])›
proof(rule GEN)
  fix R
  AOT-obtain a where a-prop:
    ‹A!a & ∀F (a[F] ≡ ∃y(A!y & F = [λz [R]y] & ¬y[F]))›
    using A−objects[axiom-inst] ∃E[rotated] by fast
  AOT-have den: ‹[λz [R]a]↓› by cqt:2[lambda]
  AOT-have a-enc: ‹a[λz [R]a]›
  proof (rule raa−cor:1)
    AOT-assume 0: ‹¬a[λz [R]a]›
    AOT-hence ‹¬∃y(A!y & [λz [R]a] = [λz [R]y] & ¬y[λz [R]a])›
      by (safe intro!: a-prop[THEN &E(2), THEN ∀E(1)[where τ=‹«[λz [R]a]»›],
                THEN oth−class−taut:4:b[THEN ≡E(1)],
                THEN ≡E(1), rotated] cqt:2)
    AOT-hence ‹∀y ¬(A!y & [λz [R]a] = [λz [R]y] & ¬y[λz [R]a])›
      using cqt−further:4 →E by blast
    AOT-hence ‹¬(A!a & [λz [R]a] = [λz [R]a] & ¬a[λz [R]a])› using ∀E by blast
    AOT-hence ‹(A!a & [λz [R]a] = [λz [R]a]) → a[λz [R]a]›
      by (metis &I deduction−theorem raa−cor:3)
    AOT-hence ‹a[λz [R]a]›
      using a-prop[THEN &E(1)] →E &I
      by (metis rule=I:1 den)
    AOT-thus ‹a[λz [R]a] & ¬a[λz [R]a]›  by (metis 0 raa−cor:3)
  qed
  AOT-hence ‹∃y(A!y & [λz [R]a] = [λz [R]y] & ¬y[λz [R]a])›
    by (rule a-prop[THEN &E(2), THEN ∀E(1), OF den, THEN ≡E(1), rotated])
  then AOT-obtain b where b-prop: ‹A!b & [λz [R]a] = [λz [R]b] & ¬b[λz [R]a]›
    using ∃E[rotated] by blast
  AOT-have 1: ‹a ≠ b›
    apply (rule ≡dfI[OF =−infix])
    using a-enc b-prop[THEN &E(2)]
    using ¬¬I rule=E id-sym ≡E(4) oth−class−taut:3:a
          raa−cor:3 reductio−aa:1 by fast
  AOT-have a: ‹[λ [R]a] = ([R]a)›
    apply (rule lambda−predicates:3[zero][axiom-inst, unvarify p])
    by (meson log−prop−prop:2)
  AOT-have b: ‹[λ [R]b] = ([R]b)›
    apply (rule lambda−predicates:3[zero][axiom-inst, unvarify p])
    by (meson log−prop−prop:2)
  AOT-have ‹[λ [R]a] = [λ [R]b]›
    apply (rule rule=E[rotated, OF a[THEN id-sym]])
    apply (rule rule=E[rotated, OF b[THEN id-sym]])
    apply (rule identity:4[THEN ≡dfI, OF &I, rotated])
    using b-prop &E apply blast
    apply (safe intro!: &I)
    by (simp add: log−prop−prop:2)+
  AOT-hence ‹A!a & A!b & a ≠ b & [λ [R]a] = [λ [R]b]›
    using 1 a-prop[THEN &E(1)] b-prop[THEN &E(1), THEN &E(1)]
          &I by auto
  AOT-hence ‹∃y (A!a & A!y & a ≠ y & [λ [R]a] = [λ [R]y])› by (rule ∃I)
  AOT-thus ‹∃x∃y (A!x & A!y & x ≠ y & [λ [R]x] = [λ [R]y])› by (rule ∃I)
qed

AOT-theorem aclassical2: ‹∃x∃y (A!x & A!y & x ≠ y & ∀F ([F]x ≡ [F]y))›
proof −
  AOT-have ‹∃x ∃y ([A!]x & [A!]y & x ≠ y &
              [λz [λxy ∀F ([F]x ≡ [F]y)]zx] =
              [λz [λxy ∀F ([F]x ≡ [F]y)]zy])›
    by (rule aclassical:1[THEN ∀E(1)[where τ=«[λxy ∀F ([F]x ≡ [F]y)]»]])
       cqt:2
```

```
  then AOT-obtain x where ‹∃ y ([A!]x & [A!]y & x ≠ y &
              [λz [λxy ∀ F ([F]x ≡ [F]y)]zx] =
              [λz [λxy ∀ F ([F]x ≡ [F]y)]zy])›
    using ∃ E[rotated] by blast
  then AOT-obtain y where 0: ‹([A!]x & [A!]y & x ≠ y &
              [λz [λxy ∀ F ([F]x ≡ [F]y)]zx] =
              [λz [λxy ∀ F ([F]x ≡ [F]y)]zy])›
    using ∃ E[rotated] by blast
  AOT-have ‹[λz [λxy ∀ F ([F]x ≡ [F]y)]zx]x›
    by (auto intro!: β←C(1) cqt:2
            simp: &I ex:1:a prod-denotesI rule−ui:3
                 oth−class−taut:3:a universal−cor)
  AOT-hence ‹[λz [λxy ∀ F ([F]x ≡ [F]y)]zy]x›
    by (rule rule=E[rotated, OF 0[THEN &E(2)]])
  AOT-hence ‹[λxy ∀ F ([F]x ≡ [F]y)]xy›
    by (rule β→C(1))
  AOT-hence ‹∀ F ([F]x ≡ [F]y)›
    using β→C(1) old.prod.case by fast
  AOT-hence ‹[A!]x & [A!]y & x ≠ y & ∀ F ([F]x ≡ [F]y)›
    using 0 &E &I by blast
  AOT-hence ‹∃ y ([A!]x & [A!]y & x ≠ y & ∀ F ([F]x ≡ [F]y))› by (rule ∃ I)
  AOT-thus ‹∃ x∃ y ([A!]x & [A!]y & x ≠ y & ∀ F ([F]x ≡ [F]y))› by (rule ∃ I(2))
qed

AOT-theorem kirchner−thm:1:
  ‹[λx φ{x}]↓ ≡ □∀ x∀ y(∀ F([F]x ≡ [F]y) → (φ{x} ≡ φ{y}))›
proof(rule ≡I; rule →I)
  AOT-assume ‹[λx φ{x}]↓›
  AOT-hence ‹□[λx φ{x}]↓› by (metis exist−nec vdash−properties:10)
  moreover AOT-have ‹□[λx φ{x}]↓ → □∀ x∀ y(∀ F([F]x ≡ [F]y) → (φ{x} ≡ φ{y}))›
  proof (rule RM:1; rule →I; rule GEN; rule GEN; rule →I)
    AOT-modally-strict {
      fix x y
      AOT-assume 0: ‹[λx φ{x}]↓›
      moreover AOT-assume ‹∀ F([F]x ≡ [F]y)›
      ultimately AOT-have ‹[λx φ{x}]x ≡ [λx φ{x}]y›
        using ∀ E by blast
      AOT-thus ‹(φ{x} ≡ φ{y})›
        using beta−C−meta[THEN →E, OF 0] ≡E(6) by meson
    }
  qed
  ultimately AOT-show ‹□∀ x∀ y(∀ F([F]x ≡ [F]y) → (φ{x} ≡ φ{y}))›
    using →E by blast
next
  AOT-have ‹□∀ x∀ y(∀ F([F]x ≡ [F]y) → (φ{x} ≡ φ{y})) →
          □∀ y(∃ x(∀ F([F]x ≡ [F]y) & φ{x}) ≡ φ{y})›
  proof(rule RM:1; rule →I; rule GEN)
    AOT-modally-strict {
      AOT-assume ‹∀ x∀ y(∀ F([F]x ≡ [F]y) → (φ{x} ≡ φ{y}))›
      AOT-hence indisc: ‹φ{x} ≡ φ{y}› if ‹∀ F([F]x ≡ [F]y)› for x y
        using ∀ E(2) →E that by blast
      AOT-show ‹(∃ x(∀ F([F]x ≡ [F]y) & φ{x}) ≡ φ{y})› for y
      proof (rule raa−cor:1)
        AOT-assume ‹¬(∃ x(∀ F([F]x ≡ [F]y) & φ{x}) ≡ φ{y})›
        AOT-hence ‹(∃ x(∀ F([F]x ≡ [F]y) & φ{x}) & ¬φ{y}) ∨
                (¬(∃ x(∀ F([F]x ≡ [F]y) & φ{x})) & φ{y})›
          using ≡E(1) oth−class−taut:4:h by blast
        moreover {
          AOT-assume 0: ‹∃ x(∀ F([F]x ≡ [F]y) & φ{x}) & ¬φ{y}›
          AOT-obtain a where ‹∀ F([F]a ≡ [F]y) & φ{a}›
            using ∃ E[rotated, OF 0[THEN &E(1)]]  by blast
          AOT-hence ‹φ{y}›
            using indisc[THEN ≡E(1)] &E by blast
```

```
        AOT-hence ‹p & ¬p› for p
          using 0[THEN &E(2)] &I raa−cor:3 by blast
      }
      moreover {
        AOT-assume 0: ‹(¬(∃x(∀F([F]x ≡ [F]y) & φ{x})) & φ{y})›
        AOT-hence ‹∀x ¬(∀F([F]x ≡ [F]y) & φ{x})›
          using &E(1) cqt−further:4 →E by blast
        AOT-hence ‹¬(∀F([F]y ≡ [F]y) & φ{y})›
          using ∀E by blast
        AOT-hence ‹¬∀F([F]y ≡ [F]y) ∨ ¬φ{y}›
          using ≡E(1) oth−class−taut:5:c by blast
        moreover AOT-have ‹∀F([F]y ≡ [F]y)›
          by (simp add: oth−class−taut:3:a universal−cor)
        ultimately AOT-have ‹¬φ{y}› by (metis ¬¬I ∨E(2))
        AOT-hence ‹p & ¬p› for p
          using 0[THEN &E(2)] &I raa−cor:3 by blast
      }
      ultimately AOT-show ‹p & ¬p› for p
        using ∨E(3) raa−cor:1 by blast
    qed
  }
 qed
 moreover AOT-assume ‹□∀x∀y(∀F([F]x ≡ [F]y) → (φ{x} ≡ φ{y}))›
 ultimately AOT-have ‹□∀y(∃x(∀F([F]x ≡ [F]y) & φ{x}) ≡ φ{y})›
   using →E by blast
 AOT-thus ‹[λx φ{x}]↓›
   by (rule safe−ext[axiom-inst, THEN →E, OF &I, rotated]) cqt:2
qed

AOT-theorem kirchner−thm:2:
 ‹[λx1...xn φ{x1...xn}]↓ ≡ □∀x1...∀xn∀y1...∀yn
   (∀F([F]x1...xn ≡ [F]y1...yn) → (φ{x1...xn} ≡ φ{y1...yn}))›
proof(rule ≡I; rule →I)
 AOT-assume ‹[λx1...xn φ{x1...xn}]↓›
 AOT-hence ‹□[λx1...xn φ{x1...xn}]↓› by (metis exist−nec →E)
 moreover AOT-have ‹□[λx1...xn φ{x1...xn}]↓ → □∀x1...∀xn∀y1...∀yn
   (∀F([F]x1...xn ≡ [F]y1...yn) → (φ{x1...xn} ≡ φ{y1...yn}))›
 proof (rule RM:1; rule →I; rule GEN; rule GEN; rule →I)
   AOT-modally-strict {
     fix x1xn y1yn :: ‹'a AOT-var›
     AOT-assume 0: ‹[λx1...xn φ{x1...xn}]↓›
     moreover AOT-assume ‹∀F([F]x1...xn ≡ [F]y1...yn)›
     ultimately AOT-have ‹[λx1...xn φ{x1...xn}]x1...xn ≡
                        [λx1...xn φ{x1...xn}]y1...yn›
       using ∀E by blast
     AOT-thus ‹(φ{x1...xn} ≡ φ{y1...yn})›
       using beta−C−meta[THEN →E, OF 0] ≡E(6) by meson
   }
 qed
 ultimately AOT-show ‹□∀x1...∀xn∀y1...∀yn(
   ∀F([F]x1...xn ≡ [F]y1...yn) → (φ{x1...xn} ≡ φ{y1...yn})
 )›
   using →E by blast
next
 AOT-have ‹
   □(∀x1...∀xn∀y1...∀yn
     (∀F([F]x1...xn ≡ [F]y1...yn) → (φ{x1...xn} ≡ φ{y1...yn})))
   → □∀y1...∀yn
       ((∃x1...∃xn(∀F([F]x1...xn ≡ [F]y1...yn) & φ{x1...xn})) ≡
        φ{y1...yn})›
 proof(rule RM:1; rule →I; rule GEN)
   AOT-modally-strict {
     AOT-assume ‹∀x1...∀xn∀y1...∀yn
```

$(\forall F([F]x_1...x_n \equiv [F]y_1...y_n) \rightarrow (\varphi\{x_1...x_n\} \equiv \varphi\{y_1...y_n\}))$›
AOT-hence *indisc*: ‹$\varphi\{x_1...x_n\} \equiv \varphi\{y_1...y_n\}$›
if ‹$\forall F([F]x_1...x_n \equiv [F]y_1...y_n)$› **for** $x_1 x_n$ $y_1 y_n$
using $\forall E(2) \rightarrow E$ *that* **by** *blast*
AOT-show ‹$(\exists x_1...\exists x_n(\forall F([F]x_1...x_n \equiv [F]y_1...y_n) \& \varphi\{x_1...x_n\})) \equiv$
$\varphi\{y_1...y_n\}$› **for** $y_1 y_n$
proof (*rule raa−cor:1*)
AOT-assume ‹$\neg((\exists x_1...\exists x_n(\forall F([F]x_1...x_n \equiv [F]y_1...y_n) \& \varphi\{x_1...x_n\})) \equiv$
$\varphi\{y_1...y_n\})$›
AOT-hence ‹$((\exists x_1...\exists x_n(\forall F([F]x_1...x_n \equiv [F]y_1...y_n)$
$\& \varphi\{x_1...x_n\}))$
$\& \neg\varphi\{y_1...y_n\}) \vee$
$(\neg(\exists x_1...\exists x_n(\forall F([F]x_1...x_n \equiv [F]y_1...y_n) \& \varphi\{x_1...x_n\}))$
$\& \varphi\{y_1...y_n\})$›
using $\equiv E(1)$ *oth−class−taut:4:h* **by** *blast*
moreover {
AOT-assume *0*: ‹$(\exists x_1...\exists x_n(\forall F([F]x_1...x_n \equiv [F]y_1...y_n) \& \varphi\{x_1...x_n\}))$
$\& \neg\varphi\{y_1...y_n\}$›
AOT-obtain $a_1 a_n$ **where** ‹$\forall F([F]a_1...a_n \equiv [F]y_1...y_n) \& \varphi\{a_1...a_n\}$›
using $\exists E$[*rotated*, *OF 0*[*THEN* $\&E(1)$]] **by** *blast*
AOT-hence ‹$\varphi\{y_1...y_n\}$›
using *indisc*[*THEN* $\equiv E(1)$] $\&E$ **by** *blast*
AOT-hence ‹$p \& \neg p$› **for** p
using *0*[*THEN* $\&E(2)$] $\&I$ *raa−cor:3* **by** *blast*
}
moreover {
AOT-assume *0*: ‹$\neg(\exists x_1...\exists x_n(\forall F([F]x_1...x_n \equiv [F]y_1...y_n) \& \varphi\{x_1...x_n\}))$
$\& \varphi\{y_1...y_n\}$›
AOT-hence ‹$\forall x_1...\forall x_n\ \neg(\forall F([F]x_1...x_n \equiv [F]y_1...y_n) \& \varphi\{x_1...x_n\})$›
using $\&E(1)$ *cqt−further:4* $\rightarrow E$ **by** *blast*
AOT-hence ‹$\neg(\forall F([F]y_1...y_n \equiv [F]y_1...y_n) \& \varphi\{y_1...y_n\})$›
using $\forall E$ **by** *blast*
AOT-hence ‹$\neg\forall F([F]y_1...y_n \equiv [F]y_1...y_n) \vee \neg\varphi\{y_1...y_n\}$›
using $\equiv E(1)$ *oth−class−taut:5:c* **by** *blast*
moreover AOT-have ‹$\forall F([F]y_1...y_n \equiv [F]y_1...y_n)$›
by (*simp add: oth−class−taut:3:a universal−cor*)
ultimately AOT-have ‹$\neg\varphi\{y_1...y_n\}$›
by (*metis* $\neg\neg I \vee E(2)$)
AOT-hence ‹$p \& \neg p$› **for** p
using *0*[*THEN* $\&E(2)$] $\&I$ *raa−cor:3* **by** *blast*
}
ultimately AOT-show ‹$p \& \neg p$› **for** p
using $\vee E(3)$ *raa−cor:1* **by** *blast*
qed
}
qed
moreover AOT-assume ‹$\Box\forall x_1...\forall x_n\forall y_1...\forall y_n$
$(\forall F([F]x_1...x_n \equiv [F]y_1...y_n) \rightarrow (\varphi\{x_1...x_n\} \equiv \varphi\{y_1...y_n\}))$›
ultimately AOT-have ‹$\Box\forall y_1...\forall y_n$
$((\exists x_1...\exists x_n(\forall F([F]x_1...x_n \equiv [F]y_1...y_n) \& \varphi\{x_1...x_n\})) \equiv$
$\varphi\{y_1...y_n\})$›
using $\rightarrow E$ **by** *blast*
AOT-thus ‹$[\lambda x_1...x_n\ \varphi\{x_1...x_n\}]\downarrow$›
by (*rule safe−ext*[*axiom-inst*, *THEN* $\rightarrow E$, *OF* $\&I$, *rotated*]) *cqt:2*
qed

AOT-theorem *kirchner−thm−cor:1*:
‹$[\lambda x\ \varphi\{x\}]\downarrow \rightarrow \forall x\forall y(\forall F([F]x \equiv [F]y) \rightarrow \Box(\varphi\{x\} \equiv \varphi\{y\}))$›
proof(*rule* $\rightarrow I$; *rule GEN*; *rule GEN*; *rule* $\rightarrow I$)
fix x y
AOT-assume ‹$[\lambda x\ \varphi\{x\}]\downarrow$›
AOT-hence ‹$\Box\forall x\forall y\ (\forall F\ ([F]x \equiv [F]y) \rightarrow (\varphi\{x\} \equiv \varphi\{y\}))$›
by (*rule kirchner−thm:1*[*THEN* $\equiv E(1)$])

AOT-hence ‹$\forall x \Box \forall y\ (\forall F\ ([F]x \equiv [F]y) \rightarrow (\varphi\{x\} \equiv \varphi\{y\}))$›
  using *CBF*[*THEN* →*E*] **by** *blast*
  AOT-hence ‹$\Box \forall y\ (\forall F\ ([F]x \equiv [F]y) \rightarrow (\varphi\{x\} \equiv \varphi\{y\}))$›
  using ∀ *E* **by** *blast*
  AOT-hence ‹$\forall y\ \Box(\forall F\ ([F]x \equiv [F]y) \rightarrow (\varphi\{x\} \equiv \varphi\{y\}))$›
  using *CBF*[*THEN* →*E*] **by** *blast*
  AOT-hence ‹$\Box(\forall F\ ([F]x \equiv [F]y) \rightarrow (\varphi\{x\} \equiv \varphi\{y\}))$›
  using ∀ *E* **by** *blast*
  AOT-hence ‹$\Box \forall F\ ([F]x \equiv [F]y) \rightarrow \Box(\varphi\{x\} \equiv \varphi\{y\})$›
  using *qml*:*1*[*axiom-inst*] *vdash−properties*:*6* **by** *blast*
  moreover AOT-assume ‹$\forall F([F]x \equiv [F]y)$›
  ultimately AOT-show ‹$\Box(\varphi\{x\} \equiv \varphi\{y\})$› **using** →*E* *ind−nec* **by** *blast*
qed

AOT-theorem *kirchner−thm−cor*:*2*:
‹$[\lambda x_1...x_n\ \varphi\{x_1...x_n\}]\downarrow \rightarrow \forall x_1...\forall x_n \forall y_1...\forall y_n$
  $(\forall F([F]x_1...x_n \equiv [F]y_1...y_n) \rightarrow \Box(\varphi\{x_1...x_n\} \equiv \varphi\{y_1...y_n\}))$›
proof(*rule* →*I*; *rule GEN*; *rule GEN*; *rule* →*I*)
  fix $x_1x_n\ y_1y_n$
  AOT-assume ‹$[\lambda x_1...x_n\ \varphi\{x_1...x_n\}]\downarrow$›
  AOT-hence *0*: ‹$\Box \forall x_1...\forall x_n \forall y_1...\forall y_n$
  $(\forall F\ ([F]x_1...x_n \equiv [F]y_1...y_n) \rightarrow (\varphi\{x_1...x_n\} \equiv \varphi\{y_1...y_n\}))$›
  by (*rule kirchner−thm*:*2*[*THEN* ≡*E*(*1*)])
  AOT-have ‹$\forall x_1...\forall x_n \forall y_1...\forall y_n$
  $\Box(\forall F\ ([F]x_1...x_n \equiv [F]y_1...y_n) \rightarrow (\varphi\{x_1...x_n\} \equiv \varphi\{y_1...y_n\}))$›
  proof(*rule GEN*; *rule GEN*)
    fix $x_1x_n\ y_1y_n$
    AOT-show ‹$\Box(\forall F\ ([F]x_1...x_n \equiv [F]y_1...y_n) \rightarrow (\varphi\{x_1...x_n\} \equiv \varphi\{y_1...y_n\}))$›
      apply (*rule RM*:*1*[*THEN* →*E*, *rotated*, *OF 0*]; *rule* →*I*)
      using ∀ *E* **by** *blast*
  qed
  AOT-hence ‹$\forall y_1...\forall y_n\ \Box(\forall F\ ([F]x_1...x_n \equiv [F]y_1...y_n) \rightarrow$
  $(\varphi\{x_1...x_n\} \equiv \varphi\{y_1...y_n\}))$›
  using ∀ *E* **by** *blast*
  AOT-hence ‹$\Box(\forall F\ ([F]x_1...x_n \equiv [F]y_1...y_n) \rightarrow (\varphi\{x_1...x_n\} \equiv \varphi\{y_1...y_n\}))$›
  using ∀ *E* **by** *blast*
  AOT-hence ‹$\Box(\forall F\ ([F]x_1...x_n \equiv [F]y_1...y_n) \rightarrow (\varphi\{x_1...x_n\} \equiv \varphi\{y_1...y_n\}))$›
  using ∀ *E* **by** *blast*
  AOT-hence *0*: ‹$\Box \forall F\ ([F]x_1...x_n \equiv [F]y_1...y_n) \rightarrow \Box(\varphi\{x_1...x_n\} \equiv \varphi\{y_1...y_n\})$›
  using *qml*:*1*[*axiom-inst*] *vdash−properties*:*6* **by** *blast*
  moreover AOT-assume ‹$\forall F([F]x_1...x_n \equiv [F]y_1...y_n)$›
  moreover AOT-have ‹$[\lambda x_1...x_n\ \Box \forall F\ ([F]x_1...x_n \equiv [F]y_1...y_n)]\downarrow$› **by** *cqt*:*2*
  ultimately AOT-have ‹$[\lambda x_1...x_n\ \Box \forall F\ ([F]x_1...x_n \equiv [F]y_1...y_n)]x_1...x_n \equiv$
  $[\lambda x_1...x_n\ \Box \forall F\ ([F]x_1...x_n \equiv [F]y_1...y_n)]y_1...y_n$›
  using ∀ *E* **by** *blast*
  moreover AOT-have ‹$[\lambda x_1...x_n\ \Box \forall F\ ([F]x_1...x_n \equiv [F]y_1...y_n)]y_1...y_n$›
  apply (*rule* β←*C*(*1*))
    apply *cqt*:*2*[*lambda*]
   apply (*fact cqt*:*2*[*const-var*][*axiom-inst*])
  by (*simp add*: *RN GEN oth−class−taut*:*3*:*a*)
  ultimately AOT-have ‹$[\lambda x_1...x_n\ \Box \forall F\ ([F]x_1...x_n \equiv [F]y_1...y_n)]x_1...x_n$›
  using ≡*E*(*2*) **by** *blast*
  AOT-hence ‹$\Box \forall F\ ([F]x_1...x_n \equiv [F]y_1...y_n)$›
  using β→*C*(*1*) **by** *blast*
  AOT-thus ‹$\Box(\varphi\{x_1...x_n\} \equiv \varphi\{y_1...y_n\})$› **using** →*E* *0* **by** *blast*
qed

8.12 Propositional Properties

AOT-define *propositional* :: ‹$\Pi \Rightarrow \varphi$› (‹*Propositional*′(-′)›)
  prop−prop1: ‹$Propositional([F]) \equiv_{df} \exists p(F = [\lambda y\ p])$›

AOT-theorem *prop−prop2*:*1*: ‹$\forall p\ [\lambda y\ p]\downarrow$›

```
  by (rule GEN) cqt:2[lambda]

AOT-theorem prop−prop2:2: ‹[λν φ]↓›
  by cqt:2[lambda]

AOT-theorem prop−prop2:3: ‹F = [λy p] → □∀x([F]x ≡ p)›
proof (rule →I)
  AOT-assume 0: ‹F = [λy p]›
  AOT-show ‹□∀x([F]x ≡ p)›
    by (rule rule=E[rotated, OF 0[symmetric]];
          rule RN; rule GEN; rule beta−C−meta[THEN →E])
        cqt:2[lambda]
qed

AOT-theorem prop−prop2:4: ‹Propositional([F]) → □Propositional([F])›
proof(rule →I)
  AOT-assume ‹Propositional([F])›
  AOT-hence ‹∃p(F = [λy p])›
    using ≡dfE[OF prop−prop1] by blast
  then AOT-obtain p where ‹F = [λy p]›
    using ∃E[rotated] by blast
  AOT-hence ‹□(F = [λy p])›
    using id−nec:2 modus−tollens:1 raa−cor:3 by blast
  AOT-hence ‹∃p □(F = [λy p])›
    using ∃I by fast
  AOT-hence 0: ‹□∃p (F = [λy p])›
    by (metis Buridan vdash−properties:10)
  AOT-thus ‹□Propositional([F])›
    using prop−prop1[THEN ≡Df]
    by (AOT-subst ‹Propositional([F])› ‹∃p (F = [λy p])›) auto
qed

AOT-define indicriminate :: ‹Π ⇒ φ› (‹Indiscriminate′(-′)›)
  prop−indis: ‹Indiscriminate([F]) ≡df F↓ & □(∃x [F]x → ∀x [F]x)›

AOT-theorem prop−in−thm: ‹Propositional([Π]) → Indiscriminate([Π])›
proof(rule →I)
  AOT-assume ‹Propositional([Π])›
  AOT-hence ‹∃p Π = [λy p]› using ≡dfE[OF prop−prop1] by blast
  then AOT-obtain p where Π-def: ‹Π = [λy p]› using ∃E[rotated] by blast
  AOT-show ‹Indiscriminate([Π])›
  proof (rule ≡dfI[OF prop−indis]; rule &I)
    AOT-show ‹Π↓›
      using Π-def by (meson t=t−proper:1 vdash−properties:6)
  next
    AOT-show ‹□(∃x [Π]x → ∀x [Π]x)›
    proof (rule rule=E[rotated, OF Π-def[symmetric]];
            rule RN; rule →I; rule GEN)
      AOT-modally-strict {
        AOT-assume ‹∃x [λy p]x›
        then AOT-obtain a where ‹[λy p]a› using ∃E[rotated] by blast
        AOT-hence 0: ‹p› by (metis β→C(1))
        AOT-show ‹[λy p]x› for x
          apply (rule β←C(1))
            apply cqt:2[lambda]
           apply (fact cqt:2[const-var][axiom-inst])
          by (fact 0)
      }
    qed
  qed
qed

AOT-theorem prop−in−f:1: ‹Necessary([F]) → Indiscriminate([F])›
```

```
proof (rule →I)
  AOT-assume ‹Necessary([F])›
  AOT-hence 0: ‹□∀ x1...∀ xn [F]x1...xn›
    using ≡df E[OF contingent−properties:1] by blast
  AOT-show ‹Indiscriminate([F])›
    by (rule ≡df I[OF prop−indis])
       (metis 0 KBasic:1 &I ex:1:a rule−ui:2[const-var] →E)
qed

AOT-theorem prop−in−f:2: ‹Impossible([F]) → Indiscriminate([F])›
proof (rule →I)
  AOT-modally-strict {
    AOT-have ‹∀ x ¬[F]x → (∃ x [F]x → ∀ x [F]x)›
      by (metis ∃ E cqt−orig:3 Hypothetical Syllogism →I raa−cor:3)
  }
  AOT-hence 0: ‹□∀ x ¬[F]x → □(∃ x [F]x → ∀ x [F]x)›
    by (rule RM:1)
  AOT-assume ‹Impossible([F])›
  AOT-hence ‹□∀ x ¬[F]x›
    using ≡df E[OF contingent−properties:2] &E by blast
  AOT-hence 1: ‹□(∃ x [F]x → ∀ x [F]x)›
    using 0 →E by blast
  AOT-show ‹Indiscriminate([F])›
    by (rule ≡df I[OF prop−indis]; rule &I)
       (simp add: ex:1:a rule−ui:2[const-var] 1)+
qed

AOT-theorem prop−in−f:3:a: ‹¬Indiscriminate([E!])›
proof(rule raa−cor:2)
  AOT-assume ‹Indiscriminate([E!])›
  AOT-hence 0: ‹□(∃ x [E!]x → ∀ x [E!]x)›
    using ≡df E[OF prop−indis] &E by blast
  AOT-hence ‹◊∃ x [E!]x → ◊∀ x [E!]x›
    using KBasic:13 vdash−properties:10 by blast
  moreover AOT-have ‹◊∃ x [E!]x›
    by (simp add: thm−cont−e:3)
  ultimately AOT-have ‹◊∀ x [E!]x›
    by (metis vdash−properties:6)
  AOT-thus ‹p & ¬p› for p
    by (metis ≡df E conventions:5 o−objects−exist:5 reductio−aa:1)
qed

AOT-theorem prop−in−f:3:b: ‹¬Indiscriminate([E!]⁻)›
proof (rule rule=E[rotated, OF rel−neg−T:2[symmetric]];
       rule raa−cor:2)
  AOT-assume ‹Indiscriminate([λx ¬[E!]x])›
  AOT-hence 0: ‹□(∃ x [λx ¬[E!]x]x → ∀ x [λx ¬[E!]x]x)›
    using ≡df E[OF prop−indis] &E by blast
  AOT-hence ‹□∃ x [λx ¬[E!]x]x → □∀ x [λx ¬[E!]x]x›
    using →E qml:1 vdash−properties:1[2] by blast
  moreover AOT-have ‹□∃ x [λx ¬[E!]x]x›
    apply (AOT-subst ‹[λx ¬E!x]x› ‹¬E!x› for: x)
    apply (rule beta−C−meta[THEN →E])
     apply cqt:2
    by (metis (full-types) B◊ RN T◊ cqt−further:2
                          o−objects−exist:5 →E)
  ultimately AOT-have 1: ‹□∀ x [λx ¬[E!]x]x›
    by (metis vdash−properties:6)
  AOT-hence ‹□∀ x ¬[E!]x›
    by (AOT-subst (reverse) ‹¬[E!]x›  ‹[λx ¬[E!]x]x› for: x)
       (auto intro!: cqt:2 beta−C−meta[THEN →E])
  AOT-hence ‹∀ x □¬[E!]x› by (metis CBF vdash−properties:10)
  moreover AOT-obtain a where abs-a: ‹O!a›
```

using $\exists E$ *o−objects−exist:1 qml:2[axiom-inst]* $\rightarrow E$ **by** *blast*
ultimately AOT-have ‹□¬[E!]a› **using** $\forall E$ **by** *blast*
AOT-hence *2*: ‹¬◇[E!]a› **by** (*metis* $\equiv_{df}E$ *conventions:5 reductio−aa:1*)
AOT-have ‹A!a›
apply (*rule* $=_{df}I(2)$[*OF AOT-abstract*])
apply *cqt:2[lambda]*
apply (*rule* $\beta\leftarrow C(1)$)
apply *cqt:2[lambda]*
using *cqt:2[const-var][axiom-inst]* **apply** *blast*
by (*fact 2*)
AOT-thus ‹p & ¬p› **for** *p* **using** *abs-a*
by (*metis* $\equiv E(1)$ *oa−contingent:2 reductio−aa:1*)
qed

AOT-theorem *prop−in−f:3:c*: ‹¬*Indiscriminate*(O!)›
proof(*rule raa−cor:2*)
AOT-assume ‹*Indiscriminate*(O!)›
AOT-hence *0*: ‹□(∃x O!x → ∀x O!x)›
using $\equiv_{df}E$[*OF prop−indis*] &E **by** *blast*
AOT-hence ‹□∃x O!x → □∀x O!x›
using *qml:1[axiom-inst] vdash−properties:6* **by** *blast*
moreover AOT-have ‹□∃x O!x›
using *o−objects−exist:1* **by** *blast*
ultimately AOT-have ‹□∀x O!x›
by (*metis vdash−properties:6*)
AOT-thus ‹p & ¬p› **for** *p*
by (*metis o−objects−exist:3 qml:2[axiom-inst] raa−cor:3* $\rightarrow E$)
qed

AOT-theorem *prop−in−f:3:d*: ‹¬*Indiscriminate*(A!)›
proof(*rule raa−cor:2*)
AOT-assume ‹*Indiscriminate*(A!)›
AOT-hence *0*: ‹□(∃x A!x → ∀x A!x)›
using $\equiv_{df}E$[*OF prop−indis*] &E **by** *blast*
AOT-hence ‹□∃x A!x → □∀x A!x›
using *qml:1[axiom-inst] vdash−properties:6* **by** *blast*
moreover AOT-have ‹□∃x A!x›
using *o−objects−exist:2* **by** *blast*
ultimately AOT-have ‹□∀x A!x›
by (*metis vdash−properties:6*)
AOT-thus ‹p & ¬p› **for** *p*
by (*metis o−objects−exist:4 qml:2[axiom-inst] raa−cor:3* $\rightarrow E$)
qed

AOT-theorem *prop−in−f:4:a*: ‹¬*Propositional*(E!)›
using *modus−tollens:1 prop−in−f:3:a prop−in−thm* **by** *blast*

AOT-theorem *prop−in−f:4:b*: ‹¬*Propositional*($E!^-$)›
using *modus−tollens:1 prop−in−f:3:b prop−in−thm* **by** *blast*

AOT-theorem *prop−in−f:4:c*: ‹¬*Propositional*(O!)›
using *modus−tollens:1 prop−in−f:3:c prop−in−thm* **by** *blast*

AOT-theorem *prop−in−f:4:d*: ‹¬*Propositional*(A!)›
using *modus−tollens:1 prop−in−f:3:d prop−in−thm* **by** *blast*

AOT-theorem *prop−prop−nec:1*: ‹◇∃p (F = [λy p]) → ∃p(F = [λy p])›
proof(*rule* $\rightarrow I$)
AOT-assume ‹◇∃p (F = [λy p])›
AOT-hence ‹∃p ◇(F = [λy p])›
by (*metis BF◇* $\rightarrow E$)
then AOT-obtain *p* **where** ‹◇(F = [λy p])›
using $\exists E$[*rotated*] **by** *blast*

AOT-hence ‹$F = [\lambda y\ p]$›
by (*metis derived−S5−rules:2 emptyE id−nec:2* $\rightarrow E$)
AOT-thus ‹$\exists p(F = [\lambda y\ p])$› **by** (*rule* $\exists I$)
qed

AOT-theorem *prop−prop−nec:2*: ‹$\forall p\ (F \neq [\lambda y\ p]) \rightarrow \Box\forall p(F \neq [\lambda y\ p])$›
proof(*rule* $\rightarrow I$)
AOT-assume ‹$\forall p\ (F \neq [\lambda y\ p])$›
AOT-hence ‹$(F \neq [\lambda y\ p])$› **for** *p*
using $\forall E$ **by** *blast*
AOT-hence ‹$\Box(F \neq [\lambda y\ p])$› **for** *p*
by (*rule id−nec2:2*[*unvarify* β, *THEN* $\rightarrow E$, *rotated*]) *cqt:2*
AOT-hence ‹$\forall p\ \Box(F \neq [\lambda y\ p])$› **by** (*rule GEN*)
AOT-thus ‹$\Box\forall p\ (F \neq [\lambda y\ p])$› **using** *BF*[*THEN* $\rightarrow E$] **by** *fast*
qed

AOT-theorem *prop−prop−nec:3*: ‹$\exists p\ (F = [\lambda y\ p]) \rightarrow \Box\exists p(F = [\lambda y\ p])$›
proof(*rule* $\rightarrow I$)
AOT-assume ‹$\exists p\ (F = [\lambda y\ p])$›
then AOT-obtain *p* **where** ‹$(F = [\lambda y\ p])$› **using** $\exists E$[*rotated*] **by** *blast*
AOT-hence ‹$\Box(F = [\lambda y\ p])$› **by** (*metis id−nec:2* $\rightarrow E$)
AOT-hence ‹$\exists p\Box(F = [\lambda y\ p])$› **by** (*rule* $\exists I$)
AOT-thus ‹$\Box\exists p(F = [\lambda y\ p])$› **by** (*metis Buridan* $\rightarrow E$)
qed

AOT-theorem *prop−prop−nec:4*: ‹$\Diamond\forall p\ (F \neq [\lambda y\ p]) \rightarrow \forall p(F \neq [\lambda y\ p])$›
proof(*rule* $\rightarrow I$)
AOT-assume ‹$\Diamond\forall p\ (F \neq [\lambda y\ p])$›
AOT-hence ‹$\forall p\ \Diamond(F \neq [\lambda y\ p])$› **by** (*metis Buridan*$\Diamond$ $\rightarrow E$)
AOT-hence ‹$\Diamond(F \neq [\lambda y\ p])$› **for** *p*
using $\forall E$ **by** *blast*
AOT-hence ‹$F \neq [\lambda y\ p]$› **for** *p*
by (*rule id−nec2:3*[*unvarify* β, *THEN* $\rightarrow E$, *rotated*]) *cqt:2*
AOT-thus ‹$\forall p\ (F \neq [\lambda y\ p])$› **by** (*rule GEN*)
qed

AOT-theorem *enc−prop−nec:1*:
‹$\Diamond\forall F\ (x[F] \rightarrow \exists p(F = [\lambda y\ p])) \rightarrow \forall F(x[F] \rightarrow \exists p\ (F = [\lambda y\ p]))$›
proof(*rule* $\rightarrow I$; *rule GEN*; *rule* $\rightarrow I$)
fix *F*
AOT-assume ‹$\Diamond\forall F\ (x[F] \rightarrow \exists p(F = [\lambda y\ p]))$›
AOT-hence ‹$\forall F\ \Diamond(x[F] \rightarrow \exists p(F = [\lambda y\ p]))$›
using *Buridan*$\Diamond$ *vdash−properties:10* **by** *blast*
AOT-hence *0*: ‹$\Diamond(x[F] \rightarrow \exists p(F = [\lambda y\ p]))$› **using** $\forall E$ **by** *blast*
AOT-assume ‹$x[F]$›
AOT-hence ‹$\Box x[F]$› **by** (*metis en−eq:2*[*1*] $\equiv E(1)$)
AOT-hence ‹$\Diamond\exists p(F = [\lambda y\ p])$›
using *0* **by** (*metis KBasic2:4* $\equiv E(1)$ *vdash−properties:10*)
AOT-thus ‹$\exists p(F = [\lambda y\ p])$›
using *prop−prop−nec:1*[*THEN* $\rightarrow E$] **by** *blast*
qed

AOT-theorem *enc−prop−nec:2*:
‹$\forall F\ (x[F] \rightarrow \exists p(F = [\lambda y\ p])) \rightarrow \Box\forall F(x[F] \rightarrow \exists p\ (F = [\lambda y\ p]))$›
using *derived−S5−rules:1*[**where** Γ={}, *simplified*, *OF enc−prop−nec:1*]
by *blast*

9 Basic Logical Objects

AOT-define *TruthValueOf* :: ‹$\tau \Rightarrow \varphi \Rightarrow \varphi$› (‹*TruthValueOf*′(-,-′)›)
tv−p: ‹*TruthValueOf(x,p)* $\equiv_{df}$ $A!x$ & $\forall F\ (x[F] \equiv \exists q((q \equiv p)$ & $F = [\lambda y\ q]))$›

AOT-theorem *p−has−!tv:1*: ‹∃ *x TruthValueOf*(*x*,*p*)›
using *tv−p*[*THEN* ≡*Df*]
by (*AOT-subst* ‹*TruthValueOf*(*x*,*p*)›
‹*A*!*x* & ∀ *F* (*x*[*F*] ≡ ∃ *q*((*q* ≡ *p*) & *F* = [λ*y q*]))› **for**: *x*)
(*simp add*: *A−objects*[*axiom-inst*])

AOT-theorem *p−has−!tv:2*: ‹∃!*x TruthValueOf*(*x*,*p*)›
using *tv−p*[*THEN* ≡*Df*]
by (*AOT-subst* ‹*TruthValueOf*(*x*,*p*)›
‹*A*!*x* & ∀ *F* (*x*[*F*] ≡ ∃ *q*((*q* ≡ *p*) & *F* = [λ*y q*]))› **for**: *x*)
(*simp add*: *A−objects*!)

AOT-theorem *uni−tv*: ‹ι*x TruthValueOf*(*x*,*p*)↓›
using *A−Exists:2 RA*[*2*] ≡*E*(*2*) *p−has−!tv:2* **by** *blast*

AOT-define *TheTruthValueOf* :: ‹φ ⇒ κ_s› (‹∘-› [*100*] *100*)
the−tv−p: ‹∘*p* $=_{df}$ ι*x TruthValueOf*(*x*,*p*)›

AOT-define *PropEnc* :: ‹τ ⇒ φ ⇒ φ› (**infixl** ‹**Σ**› *40*)
prop−enc: ‹*x***Σ***p* $\equiv_{df}$ *x*↓ & *x*[λ*y p*]›

AOT-theorem *tv−id:1*: ‹∘*p* = ι*x* (*A*!*x* & ∀ *F* (*x*[*F*] ≡ ∃ *q*((*q* ≡ *p*) & *F* = [λ*y q*])))›
proof −
AOT-have ‹□∀ *x*(*TruthValueOf*(*x*,*p*) ≡ *A*!*x* & ∀ *F* (*x*[*F*] ≡ ∃ *q*((*q* ≡ *p*) & *F* = [λ*y q*])))›
by (*rule RN*; *rule GEN*; *rule tv−p*[*THEN* ≡*Df*])
AOT-hence ‹ι*x TruthValueOf*(*x*,*p*) = ι*x* (*A*!*x* & ∀ *F* (*x*[*F*] ≡ ∃ *q*((*q* ≡ *p*) & *F* = [λ*y q*])))›
using *equiv−desc−eq:3*[*THEN* →*E*, *OF* &*I*, *OF uni−tv*] **by** *simp*
thus *?thesis*
using $=_{df}$*I*(*1*)[*OF the−tv−p*, *OF uni−tv*] **by** *fast*
qed

AOT-theorem *tv−id:2*: ‹∘*p***Σ***p*›
proof −
AOT-modally-strict {
AOT-have ‹(*p* ≡ *p*) & [λ*y p*] = [λ*y p*]›
by (*auto simp*: *prop−prop2:2 rule*=*I:1 intro*!: ≡*I* →*I* &*I*)
AOT-hence ‹∃ *q* ((*q* ≡ *p*) & [λ*y p*] = [λ*y q*])›
using ∃ *I* **by** *fast*
}
AOT-hence ‹𝓐∃ *q* ((*q* ≡ *p*) & [λ*y p*] = [λ*y q*])›
using *RA*[*2*] **by** *blast*
AOT-hence ‹ι*x*(*A*!*x* & ∀ *F* (*x*[*F*] ≡ ∃ *q* ((*q* ≡ *p*) & *F* = [λ*y q*])))[λ*y p*]›
by (*safe intro*!: *desc−nec−encode:1*[*unvarify F*, *THEN* ≡*E*(*2*)] *cqt:2*)
AOT-hence ‹ι*x*(*A*!*x* & ∀ *F* (*x*[*F*] ≡ ∃ *q* ((*q* ≡ *p*) & *F* = [λ*y q*])))**Σ***p*›
by (*safe intro*!: *prop−enc*[*THEN* $\equiv_{df}$*I*] &*I A−descriptions*)
AOT-thus ‹∘*p***Σ***p*›
by (*rule rule*=*E*[*rotated*, *OF tv−id:1*[*symmetric*]])
qed

AOT-theorem *TV−lem1:1*:
‹*p* ≡ ∀ *F*(∃ *q* (*q* & *F* = [λ*y q*]) ≡ ∃ *q*((*q* ≡ *p*) & *F* = [λ*y q*]))›
proof(*safe intro*!: ≡*I* →*I GEN*)
fix *F*
AOT-assume ‹∃ *q* (*q* & *F* = [λ*y q*])›
then AOT-obtain *q* **where** ‹*q* & *F* = [λ*y q*]› **using** ∃ *E*[*rotated*] **by** *blast*
moreover AOT-assume *p*
ultimately AOT-have ‹(*q* ≡ *p*) & *F* = [λ*y q*]›
by (*metis* &*I* &*E*(*1*) &*E*(*2*) *deduction−theorem* ≡*I*)
AOT-thus ‹∃ *q* ((*q* ≡ *p*) & *F* = [λ*y q*])› **by** (*rule* ∃ *I*)

```
next
  fix F
  AOT-assume ‹∃ q ((q ≡ p) & F = [λy q])›
  then AOT-obtain q where ‹(q ≡ p) & F = [λy q]› using ∃ E[rotated] by blast
  moreover AOT-assume p
  ultimately AOT-have ‹q & F = [λy q]›
    by (metis &I &E(1) &E(2) ≡E(2))
  AOT-thus ‹∃ q (q & F = [λy q])› by (rule ∃ I)
next
  AOT-assume ‹∀ F (∃ q (q & F = [λy q]) ≡ ∃ q ((q ≡ p) & F = [λy q]))›
  AOT-hence ‹∃ q (q & [λy p] = [λy q]) ≡ ∃ q ((q ≡ p) & [λy p] = [λy q])›
    using ∀ E(1)[rotated, OF prop−prop2:2] by blast
  moreover AOT-have ‹∃ q ((q ≡ p) & [λy p] = [λy q])›
    by (rule ∃ I(2)[where β=p])
       (simp add: rule=I:1 &I oth−class−taut:3:a prop−prop2:2)
  ultimately AOT-have ‹∃ q (q & [λy p] = [λy q])› using ≡E(2) by blast
  then AOT-obtain q where ‹q & [λy p] = [λy q]› using ∃ E[rotated] by blast
  AOT-thus ‹p›
    using rule=E &E(1) &E(2) id-sym ≡E(2) p−identity−thm2:3 by fast
qed

AOT-theorem TV−lem1:2:
  ‹¬p ≡ ∀ F(∃ q (¬q & F = [λy q]) ≡ ∃ q((q ≡ p) & F = [λy q]))›
proof(safe intro!: ≡I →I GEN)
  fix F
  AOT-assume ‹∃ q (¬q & F = [λy q])›
  then AOT-obtain q where ‹¬q & F = [λy q]› using ∃ E[rotated] by blast
  moreover AOT-assume ‹¬p›
  ultimately AOT-have ‹(q ≡ p) & F = [λy q]›
    by (metis &I &E(1) &E(2) deduction−theorem ≡I raa−cor:3)
  AOT-thus ‹∃ q ((q ≡ p) & F = [λy q])› by (rule ∃ I)
next
  fix F
  AOT-assume ‹∃ q ((q ≡ p) & F = [λy q])›
  then AOT-obtain q where ‹(q ≡ p) & F = [λy q]› using ∃ E[rotated] by blast
  moreover AOT-assume ‹¬p›
  ultimately AOT-have ‹¬q & F = [λy q]›
    by (metis &I &E(1) &E(2) ≡E(1) raa−cor:3)
  AOT-thus ‹∃ q (¬q & F = [λy q])› by (rule ∃ I)
next
  AOT-assume ‹∀ F (∃ q (¬q & F = [λy q]) ≡ ∃ q ((q ≡ p) & F = [λy q]))›
  AOT-hence ‹∃ q (¬q & [λy p] = [λy q]) ≡ ∃ q ((q ≡ p) & [λy p] = [λy q])›
    using ∀ E(1)[rotated, OF prop−prop2:2] by blast
  moreover AOT-have ‹∃ q ((q ≡ p) & [λy p] = [λy q])›
    by (rule ∃ I(2)[where β=p])
       (simp add: rule=I:1 &I oth−class−taut:3:a prop−prop2:2)
  ultimately AOT-have ‹∃ q (¬q & [λy p] = [λy q])› using ≡E(2) by blast
  then AOT-obtain q where ‹¬q & [λy p] = [λy q]› using ∃ E[rotated] by blast
  AOT-thus ‹¬p›
    using rule=E &E(1) &E(2) id-sym ≡E(2) p−identity−thm2:3 by fast
qed


AOT-define TruthValue :: ‹τ ⇒ φ› (‹TruthValue'(-')›)
  T−value: ‹TruthValue(x) ≡df ∃ p (TruthValueOf(x,p))›



AOT-act-theorem T−lem:1: ‹TruthValueOf(∘p, p)›
proof −
  AOT-have ϑ: ‹∘p = ιx TruthValueOf(x, p)›
    using rule−id−df:1 the−tv−p uni−tv by blast
  moreover AOT-have ‹∘p↓›
```

```
    using t=t−proper:1 calculation vdash−properties:10 by blast
  ultimately show ?thesis by (metis rule=E id-sym vdash−properties:10 y−in:3)
qed

AOT-act-theorem T−lem:2: ‹∀ F (◦p[F] ≡ ∃ q((q ≡ p) & F = [λy q]))›
  using T−lem:1[THEN tv−p[THEN ≡df E], THEN &E(2)].

AOT-act-theorem T−lem:3: ‹◦pΣr ≡ (r ≡ p)›
proof −
  AOT-have ϑ: ‹◦p[λy r] ≡ ∃ q ((q ≡ p) & [λy r] = [λy q])›
    using T−lem:2[THEN ∀ E(1), OF prop−prop2:2].
  show ?thesis
  proof(rule ≡I; rule →I)
    AOT-assume ‹◦pΣr›
    AOT-hence ‹◦p[λy r]› by (metis ≡df E &E(2) prop−enc)
    AOT-hence ‹∃ q ((q ≡ p) & [λy r] = [λy q])› using ϑ ≡E(1) by blast
    then AOT-obtain q where ‹(q ≡ p) & [λy r] = [λy q]› using ∃ E[rotated] by blast
    moreover AOT-have ‹r = q› using calculation
      using &E(2) ≡E(2) p−identity−thm2:3 by blast
    ultimately AOT-show ‹r ≡ p›
      by (metis rule=E &E(1) ≡E(6) oth−class−taut:3:a)
  next
    AOT-assume ‹r ≡ p›
    moreover AOT-have ‹[λy r] = [λy r]›
      by (simp add: rule=I:1 prop−prop2:2)
    ultimately AOT-have ‹(r ≡ p) & [λy r] = [λy r]› using &I by blast
    AOT-hence ‹∃ q ((q ≡ p) & [λy r] = [λy q])› by (rule ∃ I(2)[where β=r])
    AOT-hence ‹◦p[λy r]› using ϑ ≡E(2) by blast
    AOT-thus ‹◦pΣr›
      by (metis ≡df I &I prop−enc russell−axiom[enc,1].ψ-denotes-asm)
  qed
qed

AOT-act-theorem T−lem:4: ‹TruthValueOf(x, p) ≡ x = ◦p›
proof −
  AOT-have ‹∀ x (x = ιx TruthValueOf(x, p) ≡ ∀ z (TruthValueOf(z, p) ≡ z = x))›
    by (simp add: fund−cont−desc GEN)
  moreover AOT-have ‹◦p↓›
    using ≡df E tv−id:2 &E(1) prop−enc by blast
  ultimately AOT-have
    ‹(◦p = ιx TruthValueOf(x, p)) ≡ ∀ z (TruthValueOf(z, p) ≡ z = ◦p)›
    using ∀ E(1) by blast
  AOT-hence ‹∀ z (TruthValueOf(z, p) ≡ z = ◦p)›
    using ≡E(1) rule−id−df:1 the−tv−p uni−tv by blast
  AOT-thus ‹TruthValueOf(x, p) ≡ x = ◦p› using ∀ E(2) by blast
qed


AOT-theorem TV−lem2:1:
  ‹(A!x & ∀ F (x[F] ≡ ∃ q (q & F = [λy q]))) → TruthValue(x)›
proof(safe intro!: →I T−value[THEN ≡df I] tv−p[THEN ≡df I]
                   ∃ I(1)[rotated, OF log−prop−prop:2])
  AOT-assume ‹[A!]x & ∀ F (x[F] ≡ ∃ q (q & F = [λy q]))›
  AOT-thus ‹[A!]x & ∀ F (x[F] ≡ ∃ q ((q ≡ (∀ p (p → p))) & F = [λy q]))›
    apply (AOT-subst ‹∃ q ((q ≡ (∀ p (p → p))) & F = [λy q])›
                     ‹∃ q (q & F = [λy q])› for: F :: ‹<κ>›)
     apply (AOT-subst ‹q ≡ ∀ p (p →p)› ‹q› for: q)
    apply (metis (no-types, lifting) →I ≡I ≡E(2) GEN)
    by (auto simp: cqt−further:7)
qed
```

AOT-theorem *TV−lem2:2*:
‹(*A*!*x* & ∀ *F* (*x*[*F*] ≡ ∃ *q* (¬*q* & *F* = [λ*y q*]))) → *TruthValue*(*x*)›
proof(*safe intro*!: →*I T−value*[*THEN* ≡$_{df}$*I*] *tv−p*[*THEN* ≡$_{df}$*I*]
∃ *I*(*1*)[*rotated*, *OF log−prop−prop*:*2*])
AOT-assume ‹[*A*!]*x* & ∀ *F* (*x*[*F*] ≡ ∃ *q* (¬*q* & *F* = [λ*y q*]))›
AOT-thus ‹[*A*!]*x* & ∀ *F* (*x*[*F*] ≡ ∃ *q* ((*q* ≡ (∃ *p* (*p* & ¬*p*))) & *F* = [λ*y q*]))›
apply (*AOT-subst* ‹∃ *q* ((*q* ≡ (∃ *p* (*p* & ¬*p*))) & *F* = [λ*y q*])›
‹∃ *q* (¬*q* & *F* = [λ*y q*])› **for**: *F* :: ‹<κ>›)
apply (*AOT-subst* ‹*q* ≡ ∃ *p* (*p* & ¬*p*)› ‹¬*q*› **for**: *q*)
apply (*metis* (*no-types*, *lifting*)
→*I* ∃ *E* ≡*E*(*1*) ≡*I raa−cor*:*1 raa−cor*:*3*)
by (*auto simp add*: *cqt−further*:*7*)
qed

AOT-define *TheTrue* :: κ$_s$ (‹⊤›)
the−true:*1*: ‹⊤ =$_{df}$ ɩ*x* (*A*!*x* & ∀ *F* (*x*[*F*] ≡ ∃ *p*(*p* & *F* = [λ*y p*])))›
AOT-define *TheFalse* :: κ$_s$ (‹⊥›)
the−true:*2*: ‹⊥ =$_{df}$ ɩ*x* (*A*!*x* & ∀ *F* (*x*[*F*] ≡ ∃ *p*(¬*p* & *F* = [λ*y p*])))›

AOT-theorem *the−true*:*3*: ‹⊤ ≠ ⊥›
proof(*safe intro*!: *ab−obey*:*2*[*unvarify x y*, *THEN* →*E*, *rotated 2*, *OF* ∨*I*(*1*)]
∃ *I*(*1*)[**where** τ=‹«[λ*x* ∀ *q*(*q* → *q*)]»›] &*I prop−prop2*:*2*)
AOT-have *false-def*: ‹⊥ = ɩ*x* (*A*!*x* & ∀ *F* (*x*[*F*] ≡ ∃ *p*(¬*p* & *F* = [λ*y p*])))›
by (*simp add*: *A−descriptions rule−id−df*:*1*[*zero*] *the−true*:*2*)
moreover AOT-show *false-den*: ‹⊥↓›
by (*meson* →*E t*=*t−proper*:*1 A−descriptions*
rule−id−df:*1*[*zero*] *the−true*:*2*)
ultimately AOT-have *false-prop*: ‹𝓐(*A*!⊥ & ∀ *F* (⊥[*F*] ≡ ∃ *p*(¬*p* & *F* = [λ*y p*])))›
using *nec−hintikka−scheme*[*unvarify x*, *THEN* ≡*E*(*1*), *THEN* &*E*(*1*)] **by** *blast*
AOT-hence ‹𝓐∀ *F* (⊥[*F*] ≡ ∃ *p*(¬*p* & *F* = [λ*y p*]))›
using *Act−Basic*:*2* &*E*(*2*) ≡*E*(*1*) **by** *blast*
AOT-hence ‹∀ *F* 𝓐(⊥[*F*] ≡ ∃ *p*(¬*p* & *F* = [λ*y p*]))›
using ≡*E*(*1*) *logic−actual−nec*:*3*[*axiom-inst*] **by** *blast*
AOT-hence *false-enc-cond*:
‹𝓐(⊥[λ*x* ∀ *q*(*q* → *q*)] ≡ ∃ *p*(¬*p* & [λ*x* ∀ *q*(*q* → *q*)] = [λ*y p*]))›
using ∀ *E*(*1*)[*rotated*, *OF prop−prop2*:*2*] **by** *blast*

AOT-have *true-def*: ‹⊤ = ɩ*x* (*A*!*x* & ∀ *F* (*x*[*F*] ≡ ∃ *p*(*p* & *F* = [λ*y p*])))›
by (*simp add*: *A−descriptions rule−id−df*:*1*[*zero*] *the−true*:*1*)
moreover AOT-show *true-den*: ‹⊤↓›
by (*meson t*=*t−proper*:*1 A−descriptions rule−id−df*:*1*[*zero*] *the−true*:*1* →*E*)
ultimately AOT-have *true-prop*: ‹𝓐(*A*!⊤ & ∀ *F* (⊤[*F*] ≡ ∃ *p*(*p* & *F* = [λ*y p*])))›
using *nec−hintikka−scheme*[*unvarify x*, *THEN* ≡*E*(*1*), *THEN* &*E*(*1*)] **by** *blast*
AOT-hence ‹𝓐∀ *F* (⊤[*F*] ≡ ∃ *p*(*p* & *F* = [λ*y p*]))›
using *Act−Basic*:*2* &*E*(*2*) ≡*E*(*1*) **by** *blast*
AOT-hence ‹∀ *F* 𝓐(⊤[*F*] ≡ ∃ *p*(*p* & *F* = [λ*y p*]))›
using ≡*E*(*1*) *logic−actual−nec*:*3*[*axiom-inst*] **by** *blast*
AOT-hence ‹𝓐(⊤[λ*x* ∀ *q*(*q* → *q*)] ≡ ∃ *p*(*p* & [λ*x* ∀ *q*(*q* → *q*)] = [λ*y p*]))›
using ∀ *E*(*1*)[*rotated*, *OF prop−prop2*:*2*] **by** *blast*
moreover AOT-have ‹𝓐∃ *p*(*p* & [λ*x* ∀ *q*(*q* → *q*)] = [λ*y p*])›
by (*safe intro*!: *nec−imp−act*[*THEN* →*E*] *RN* ∃ *I*(*1*)[**where** τ=«∀ *q*(*q* → *q*)»] &*I*
GEN →*I log−prop−prop*:*2 rule*=*I*:*1 prop−prop2*:*2*)
ultimately AOT-have ‹𝓐(⊤[λ*x* ∀ *q*(*q* → *q*)])›
using *Act−Basic*:*5* ≡*E*(*1*,*2*) **by** *blast*
AOT-thus ‹⊤[λ*x* ∀ *q*(*q* → *q*)]›
using *en−eq*:*10*[*1*][*unvarify* x_1 *F*, *THEN* ≡*E*(*1*)] *true-den prop−prop2*:*2* **by** *blast*

AOT-show ‹¬⊥[λ*x* ∀ *q*(*q* → *q*)]›
proof(*rule raa−cor*:*2*)
AOT-assume ‹⊥[λ*x* ∀ *q*(*q* → *q*)]›
AOT-hence ‹𝓐⊥[λ*x* ∀ *q*(*q* → *q*)]›
using *en−eq*:*10*[*1*][*unvarify* x_1 *F*, *THEN* ≡*E*(*2*)]

false-den prop−prop2:*2* **by** *blast*
AOT-hence ‹𝓐∃ *p*(¬*p* & [λ*x* ∀ *q*(*q* → *q*)] = [λ*y* *p*])›
using *false-enc-cond Act−Basic*:*5* ≡*E*(*1*) **by** *blast*
AOT-hence ‹∃ *p* 𝓐(¬*p* & [λ*x* ∀ *q*(*q* → *q*)] = [λ*y* *p*])›
using *Act−Basic*:*10* ≡*E*(*1*) **by** *blast*
then AOT-obtain *p* **where** *p-prop*: ‹𝓐(¬*p* & [λ*x* ∀ *q*(*q* → *q*)] = [λ*y* *p*])›
using ∃ *E*[*rotated*] **by** *blast*
AOT-hence ‹𝓐[λ*x* ∀ *q*(*q* → *q*)] = [λ*y* *p*]›
by (*metis Act−Basic*:*2* &*E*(*2*) ≡*E*(*1*))
AOT-hence ‹[λ*x* ∀ *q*(*q* → *q*)] = [λ*y* *p*]›
using *id−act*:*1*[*unvarify* α β, *THEN* ≡*E*(*2*)] *prop−prop2*:*2* **by** *blast*
AOT-hence ‹(∀ *q*(*q* → *q*)) = *p*›
using *p−identity−thm2*:*3*[*unvarify p*, *THEN* ≡*E*(*2*)]
log−prop−prop:*2* **by** *blast*
moreover AOT-have ‹𝓐¬*p*› **using** *p-prop*
using *Act−Basic*:*2* &*E*(*1*) ≡*E*(*1*) **by** *blast*
ultimately AOT-have ‹𝓐¬∀ *q*(*q* → *q*)›
by (*metis Act−Sub*:*1* ≡*E*(*1*,*2*) *raa−cor*:*3* *rule*=*E*)
moreover AOT-have ‹¬𝓐¬∀ *q*(*q* → *q*)›
by (*meson Act−Sub*:*1* *RA*[*2*] *if−p−then−p* ≡*E*(*1*) *universal−cor*)
ultimately AOT-show ‹𝓐¬∀ *q*(*q* → *q*) & ¬𝓐¬∀ *q*(*q* → *q*)›
using &*I* **by** *blast*
qed
qed

AOT-act-theorem *T−T−value*:*1*: ‹*TruthValue*(⊤)›
proof −
AOT-have *true-def*: ‹⊤ = ι*x* (*A*!*x* & ∀ *F* (*x*[*F*] ≡ ∃ *p*(*p* & *F* = [λ*y* *p*])))›
by (*simp add*: *A−descriptions rule−id−df*:*1*[*zero*] *the−true*:*1*)
AOT-hence *true-den*: ‹⊤↓›
using *t*=*t−proper*:*1* *vdash−properties*:*6* **by** *blast*
AOT-show ‹*TruthValue*(⊤)›
using *y−in*:*2*[*unvarify z*, *OF true-den*, *THEN* →*E*, *OF true-def*]
TV−lem2:*1*[*unvarify x*, *OF true-den*, *THEN* →*E*] **by** *blast*
qed

AOT-act-theorem *T−T−value*:*2*: ‹*TruthValue*(⊥)›
proof −
AOT-have *false-def*: ‹⊥ = ι*x* (*A*!*x* & ∀ *F* (*x*[*F*] ≡ ∃ *p*(¬*p* & *F* = [λ*y* *p*])))›
by (*simp add*: *A−descriptions rule−id−df*:*1*[*zero*] *the−true*:*2*)
AOT-hence *false-den*: ‹⊥↓›
using *t*=*t−proper*:*1* *vdash−properties*:*6* **by** *blast*
AOT-show ‹*TruthValue*(⊥)›
using *y−in*:*2*[*unvarify z*, *OF false-den*, *THEN* →*E*, *OF false-def*]
TV−lem2:*2*[*unvarify x*, *OF false-den*, *THEN* →*E*] **by** *blast*
qed

AOT-theorem *two−T*: ‹∃ *x*∃ *y*(*TruthValue*(*x*) & *TruthValue*(*y*) & *x* ≠ *y* &
∀ *z* (*TruthValue*(*z*) → *z* = *x* ∨ *z* = *y*))›
proof −
AOT-obtain *a* **where** *a-prop*: ‹*A*!*a* & ∀ *F* (*a*[*F*] ≡ ∃ *p* (*p* & *F* = [λ*y* *p*]))›
using *A−objects*[*axiom-inst*] ∃ *E*[*rotated*] **by** *fast*
AOT-obtain *b* **where** *b-prop*: ‹*A*!*b* & ∀ *F* (*b*[*F*] ≡ ∃ *p* (¬*p* & *F* = [λ*y* *p*]))›
using *A−objects*[*axiom-inst*] ∃ *E*[*rotated*] **by** *fast*
AOT-obtain *p* **where** *p*: *p*
by (*metis log−prop−prop*:*2* *raa−cor*:*3* *rule−ui*:*1* *universal−cor*)
show *?thesis*
proof(*rule* ∃ *I*(*2*)[**where** β=*a*]; *rule* ∃ *I*(*2*)[**where** β=*b*];
safe intro!: &*I GEN* →*I*)
AOT-show ‹*TruthValue*(*a*)›
using *TV−lem2*:*1* *a-prop vdash−properties*:*10* **by** *blast*
next
AOT-show ‹*TruthValue*(*b*)›

```
    using TV−lem2:2 b-prop vdash−properties:10 by blast
next
  AOT-show ‹a ≠ b›
  proof(rule ab−obey:2[THEN →E, OF ∨I(1)])
    AOT-show ‹∃F (a[F] & ¬b[F])›
    proof(rule ∃I(1)[where τ=«[λy p]»]; rule &I prop−prop2:2)
      AOT-show ‹a[λy p]›
        by(safe intro!: ∃I(2)[where β=p] &I p rule=I:1[OF prop−prop2:2]
            a-prop[THEN &E(2), THEN ∀E(1), THEN ≡E(2), OF prop−prop2:2])
    next
      AOT-show ‹¬b[λy p]›
      proof (rule raa−cor:2)
        AOT-assume ‹b[λy p]›
        AOT-hence ‹∃q (¬q & [λy p] = [λy q])›
          using ∀E(1)[rotated, OF prop−prop2:2, THEN ≡E(1)]
                b-prop[THEN &E(2)] by fast
        then AOT-obtain q where ‹¬q & [λy p] = [λy q]›
          using ∃E[rotated] by blast
        AOT-hence ‹¬p›
          by (metis rule=E &E(1) &E(2) deduction−theorem ≡I
                  ≡E(2) p−identity−thm2:3 raa−cor:3)
        AOT-thus ‹p & ¬p› using p &I by blast
      qed
    qed
  qed
next
  fix z
  AOT-assume ‹TruthValue(z)›
  AOT-hence ‹∃p (TruthValueOf(z, p))›
    by (metis ≡df E T−value)
  then AOT-obtain p where ‹TruthValueOf(z, p)› using ∃E[rotated] by blast
  AOT-hence z-prop: ‹A!z & ∀F (z[F] ≡ ∃q ((q ≡ p) & F = [λy q]))›
    using ≡df E tv−p by blast
  {
    AOT-assume p: ‹p›
    AOT-have ‹z = a›
    proof(rule ab−obey:1[THEN →E, THEN →E, OF &I,
                         OF z-prop[THEN &E(1)], OF a-prop[THEN &E(1)]];
          rule GEN)
      fix G
      AOT-have ‹z[G] ≡ ∃q ((q ≡ p) & G = [λy q])›
        using z-prop[THEN &E(2)] ∀E(2) by blast
      also AOT-have ‹∃q ((q ≡ p) & G = [λy q]) ≡ ∃q (q & G = [λy q])›
        using TV−lem1:1[THEN ≡E(1), OF p, THEN ∀E(2)[where β=G], symmetric].
      also AOT-have ‹... ≡ a[G]›
        using a-prop[THEN &E(2), THEN ∀E(2)[where β=G], symmetric].
      finally AOT-show ‹z[G] ≡ a[G]›.
    qed
    AOT-hence ‹z = a ∨ z = b› by (rule ∨I)
  }
  moreover {
    AOT-assume notp: ‹¬p›
    AOT-have ‹z = b›
    proof(rule ab−obey:1[THEN →E, THEN →E, OF &I,
                         OF z-prop[THEN &E(1)], OF b-prop[THEN &E(1)]];
          rule GEN)
      fix G
      AOT-have ‹z[G] ≡ ∃q ((q ≡ p) & G = [λy q])›
        using z-prop[THEN &E(2)] ∀E(2) by blast
      also AOT-have ‹∃q ((q ≡ p) & G = [λy q]) ≡ ∃q (¬q & G = [λy q])›
        using TV−lem1:2[THEN ≡E(1), OF notp, THEN ∀E(2), symmetric].
      also AOT-have ‹... ≡ b[G]›
        using b-prop[THEN &E(2), THEN ∀E(2), symmetric].
```

finally AOT-show ‹$z[G] \equiv b[G]$›.
qed
AOT-hence ‹$z = a \vee z = b$› **by** (*rule* $\vee I$)
}
ultimately AOT-show ‹$z = a \vee z = b$›
by (*metis reductio−aa*:*1*)
qed
qed

AOT-act-theorem *valueof−facts*:*1*: ‹$TruthValueOf(x, p) \rightarrow (p \equiv x = \top)$›
proof(*safe intro*!: $\rightarrow I$ *dest*!: *tv−p*[*THEN* $\equiv_{df} E$])
AOT-assume ϑ: ‹$[A!]x \;\&\; \forall F\ (x[F] \equiv \exists q\ ((q \equiv p) \;\&\; F = [\lambda y\ q]))$›
AOT-have *a*: ‹$A!\top$›
using $\exists E$ *T−T−value*:*1* *T−value* &*E*(*1*) $\equiv_{df} E$ *tv−p* **by** *blast*
AOT-have *true-def*: ‹$\top = \iota x\ (A!x \;\&\; \forall F\ (x[F] \equiv \exists p(p \;\&\; F = [\lambda y\ p])))$›
by (*simp add*: *A−descriptions rule−id−df*:*1*[*zero*] *the−true*:*1*)
AOT-hence *true-den*: ‹$\top\downarrow$›
using *t*=*t−proper*:*1* *vdash−properties*:*6* **by** *blast*
AOT-have *b*: ‹$\forall F\ (\top[F] \equiv \exists q\ (q \;\&\; F = [\lambda y\ q]))$›
using *y−in*:*2*[*unvarify z*, *OF true-den*, *THEN* $\rightarrow E$, *OF true-def*] &*E* **by** *blast*
AOT-show ‹$p \equiv x = \top$›
proof(*safe intro*!: $\equiv I \rightarrow I$)
AOT-assume *p*
AOT-hence ‹$\forall F\ (\exists q\ (q \;\&\; F = [\lambda y\ q]) \equiv \exists q\ ((q \equiv p) \;\&\; F = [\lambda y\ q]))$›
using *TV−lem1*:*1*[*THEN* $\equiv E$(*1*)] **by** *blast*
AOT-hence ‹$\forall F(\top[F] \equiv \exists q\ ((q \equiv p) \;\&\; F = [\lambda y\ q]))$›
using *b cqt−basic*:*10*[*THEN* $\rightarrow E$, *OF* &*I*, *OF b*] **by** *fast*
AOT-hence *c*: ‹$\forall F(\exists q((q \equiv p) \;\&\; F = [\lambda y\ q]) \equiv \top[F])$›
using *cqt−basic*:*11*[*THEN* $\equiv E$(*1*)] **by** *fast*
AOT-hence ‹$\forall F\ (x[F] \equiv \top[F])$›
using *cqt−basic*:*10*[*THEN* $\rightarrow E$, *OF* &*I*, *OF* ϑ[*THEN* &*E*(*2*)]] **by** *fast*
AOT-thus ‹$x = \top$›
by (*rule ab−obey*:*1*[*unvarify y*, *OF true-den*, *THEN* $\rightarrow E$, *THEN* $\rightarrow E$,
OF &*I*, *OF* ϑ[*THEN* &*E*(*1*)], *OF a*])
next
AOT-assume ‹$x = \top$›
AOT-hence *d*: ‹$\forall F\ (\top[F] \equiv \exists q\ ((q \equiv p) \;\&\; F = [\lambda y\ q]))$›
using *rule*=*E* ϑ[*THEN* &*E*(*2*)] **by** *fast*
AOT-have ‹$\forall F\ (\exists q\ (q \;\&\; F = [\lambda y\ q]) \equiv \exists q\ ((q \equiv p) \;\&\; F = [\lambda y\ q]))$›
using *cqt−basic*:*10*[*THEN* $\rightarrow E$, *OF* &*I*,
OF b[*THEN cqt−basic*:*11*[*THEN* $\equiv E$(*1*)]], *OF d*].
AOT-thus *p* **using** *TV−lem1*:*1*[*THEN* $\equiv E$(*2*)] **by** *blast*
qed
qed

AOT-act-theorem *valueof−facts*:*2*: ‹$TruthValueOf(x, p) \rightarrow (\neg p \equiv x = \bot)$›
proof(*safe intro*!: $\rightarrow I$ *dest*!: *tv−p*[*THEN* $\equiv_{df} E$])
AOT-assume ϑ: ‹$[A!]x \;\&\; \forall F\ (x[F] \equiv \exists q\ ((q \equiv p) \;\&\; F = [\lambda y\ q]))$›
AOT-have *a*: ‹$A!\bot$›
using $\exists E$ *T−T−value*:*2* *T−value* &*E*(*1*) $\equiv_{df} E$ *tv−p* **by** *blast*
AOT-have *false-def*: ‹$\bot = \iota x\ (A!x \;\&\; \forall F\ (x[F] \equiv \exists p(\neg p \;\&\; F = [\lambda y\ p])))$›
by (*simp add*: *A−descriptions rule−id−df*:*1*[*zero*] *the−true*:*2*)
AOT-hence *false-den*: ‹$\bot\downarrow$›
using *t*=*t−proper*:*1* *vdash−properties*:*6* **by** *blast*
AOT-have *b*: ‹$\forall F\ (\bot[F] \equiv \exists q\ (\neg q \;\&\; F = [\lambda y\ q]))$›
using *y−in*:*2*[*unvarify z*, *OF false-den*, *THEN* $\rightarrow E$, *OF false-def*] &*E* **by** *blast*
AOT-show ‹$\neg p \equiv x = \bot$›
proof(*safe intro*!: $\equiv I \rightarrow I$)
AOT-assume ‹$\neg p$›
AOT-hence ‹$\forall F\ (\exists q\ (\neg q \;\&\; F = [\lambda y\ q]) \equiv \exists q\ ((q \equiv p) \;\&\; F = [\lambda y\ q]))$›
using *TV−lem1*:*2*[*THEN* $\equiv E$(*1*)] **by** *blast*
AOT-hence ‹$\forall F(\bot[F] \equiv \exists q\ ((q \equiv p) \;\&\; F = [\lambda y\ q]))$›
using *b cqt−basic*:*10*[*THEN* $\rightarrow E$, *OF* &*I*, *OF b*] **by** *fast*

```
    AOT-hence c: ‹∀F(∃q((q ≡ p) & F = [λy q]) ≡ ⊥[F])›
      using cqt−basic:11[THEN ≡E(1)] by fast
    AOT-hence ‹∀F (x[F] ≡ ⊥[F])›
      using cqt−basic:10[THEN →E, OF &I, OF ϑ[THEN &E(2)]] by fast
    AOT-thus ‹x = ⊥›
      by (rule ab−obey:1[unvarify y, OF false-den, THEN →E, THEN →E,
                        OF &I, OF ϑ[THEN &E(1)], OF a])
  next
    AOT-assume ‹x = ⊥›
    AOT-hence d: ‹∀F (⊥[F] ≡ ∃q ((q ≡ p) & F = [λy q]))›
      using rule=E ϑ[THEN &E(2)] by fast
    AOT-have ‹∀F (∃q (¬q & F = [λy q]) ≡ ∃q ((q ≡ p) & F = [λy q]))›
      using cqt−basic:10[THEN →E, OF &I,
              OF b[THEN cqt−basic:11[THEN ≡E(1)]], OF d].
    AOT-thus ‹¬p› using TV−lem1:2[THEN ≡E(2)] by blast
  qed
qed

AOT-act-theorem q−True:1: ‹p ≡ (◦p = ⊤)›
  apply (rule valueof−facts:1[unvarify x, THEN →E, rotated, OF T−lem:1])
  using ≡df E tv−id:2 &E(1) prop−enc by blast

AOT-act-theorem q−True:2: ‹¬p ≡ (◦p = ⊥)›
  apply (rule valueof−facts:2[unvarify x, THEN →E, rotated, OF T−lem:1])
  using ≡df E tv−id:2 &E(1) prop−enc by blast

AOT-act-theorem q−True:3: ‹p ≡ ⊤Σp›
proof(safe intro!: ≡I →I)
  AOT-assume p
  AOT-hence ‹◦p = ⊤› by (metis ≡E(1) q−True:1)
  moreover AOT-have ‹◦pΣp›
    by (simp add: tv−id:2)
  ultimately AOT-show ‹⊤Σp›
    using rule=E T−lem:4 by fast
next
  AOT-have true-def: ‹⊤ = ιx (A!x & ∀F (x[F] ≡ ∃p(p & F = [λy p])))›
    by (simp add: A−descriptions rule−id−df:1[zero] the−true:1)
  AOT-hence true-den: ‹⊤↓›
    using t=t−proper:1 vdash−properties:6 by blast
  AOT-have b: ‹∀F (⊤[F] ≡ ∃q (q & F = [λy q]))›
    using y−in:2[unvarify z, OF true-den, THEN →E, OF true-def] &E by blast

  AOT-assume ‹⊤Σp›
  AOT-hence ‹⊤[λy p]› by (metis ≡df E &E(2) prop−enc)
  AOT-hence ‹∃q (q & [λy p] = [λy q])›
    using b[THEN ∀E(1), OF prop−prop2:2, THEN ≡E(1)] by blast
  then AOT-obtain q where ‹q & [λy p] = [λy q]› using ∃E[rotated] by blast
  AOT-thus ‹p›
    using rule=E &E(1) &E(2) id-sym ≡E(2) p−identity−thm2:3 by fast
qed


AOT-act-theorem q−True:5: ‹¬p ≡ ⊥Σp›
proof(safe intro!: ≡I →I)
  AOT-assume ‹¬p›
  AOT-hence ‹◦p = ⊥› by (metis ≡E(1) q−True:2)
  moreover AOT-have ‹◦pΣp›
    by (simp add: tv−id:2)
  ultimately AOT-show ‹⊥Σp›
    using rule=E T−lem:4 by fast
next
  AOT-have false-def: ‹⊥ = ιx (A!x & ∀F (x[F] ≡ ∃p(¬p & F = [λy p])))›
    by (simp add: A−descriptions rule−id−df:1[zero] the−true:2)
```

```
  AOT-hence false-den: ‹⊥↓›
    using t=t−proper:1 vdash−properties:6 by blast
  AOT-have b: ‹∀ F (⊥[F] ≡ ∃ q (¬q & F = [λy q]))›
    using y−in:2[unvarify z, OF false-den, THEN →E, OF false-def] &E by blast

  AOT-assume ‹⊥Σp›
  AOT-hence ‹⊥[λy p]› by (metis ≡df E &E(2) prop−enc)
  AOT-hence ‹∃ q (¬q & [λy p] = [λy q])›
    using b[THEN ∀E(1), OF prop−prop2:2, THEN ≡E(1)] by blast
  then AOT-obtain q where ‹¬q & [λy p] = [λy q]› using ∃E[rotated] by blast
  AOT-thus ‹¬p›
    using rule=E &E(1) &E(2) id-sym ≡E(2) p−identity−thm2:3 by fast
qed

AOT-act-theorem q−True:4: ‹p ≡ ¬(⊥Σp)›
  using q−True:5
  by (metis deduction−theorem ≡I ≡E(2) ≡E(4) raa−cor:3)

AOT-act-theorem q−True:6: ‹¬p ≡ ¬(⊤Σp)›
  using ≡E(1) oth−class−taut:4:b q−True:3 by blast

AOT-define ExtensionOf :: ‹τ ⇒ φ ⇒ φ› (‹ExtensionOf'(-,-')›)
  exten−p: ‹ExtensionOf(x,p) ≡df A!x &
                                  ∀ F (x[F] → Propositional([F])) &
                                  ∀ q ((xΣq) ≡ (q ≡ p))›

AOT-theorem extof−e: ‹ExtensionOf(x, p) ≡ TruthValueOf(x, p)›
proof (safe intro!: ≡I →I tv−p[THEN ≡df I] exten−p[THEN ≡df I]
            dest!: tv−p[THEN ≡df E] exten−p[THEN ≡df E])
  AOT-assume 1: ‹[A!]x & ∀ F (x[F] → Propositional([F])) & ∀ q (x Σ q ≡ (q ≡ p))›
  AOT-hence ϑ: ‹[A!]x & ∀ F (x[F] → ∃ q(F = [λy q])) & ∀ q (x Σ q ≡ (q ≡ p))›
    by (AOT-subst ‹∃ q(F = [λy q])› ‹Propositional([F])› for: F :: ‹<κ>›)
       (auto simp add: df−rules−formulas[3] df−rules−formulas[4]
                       ≡I prop−prop1)
  AOT-show ‹[A!]x & ∀ F (x[F] ≡ ∃ q ((q ≡ p) & F = [λy q]))›
  proof(safe intro!: &I GEN 1[THEN &E(1), THEN &E(1)] ≡I →I)
    fix F
    AOT-assume 0: ‹x[F]›
    AOT-hence ‹∃ q (F = [λy q])›
      using ϑ[THEN &E(1), THEN &E(2)] ∀E(2) →E by blast
    then AOT-obtain q where q-prop: ‹F = [λy q]› using ∃E[rotated] by blast
    AOT-hence ‹x[λy q]› using 0 rule=E by blast
    AOT-hence ‹xΣq› by (metis ≡df I &I ex:1:a prop−enc rule−ui:3)
    AOT-hence ‹q ≡ p› using ϑ[THEN &E(2)] ∀E(2) ≡E(1) by blast
    AOT-hence ‹(q ≡ p) & F = [λy q]› using q-prop &I by blast
    AOT-thus ‹∃ q ((q ≡ p) & F = [λy q])› by (rule ∃I)
  next
    fix F
    AOT-assume ‹∃ q ((q ≡ p) & F = [λy q])›
    then AOT-obtain q where q-prop: ‹(q ≡ p) & F = [λy q]›
      using ∃E[rotated] by blast
    AOT-hence ‹xΣq› using ϑ[THEN &E(2)] ∀E(2) &E ≡E(2) by blast
    AOT-hence ‹x[λy q]› by (metis ≡df E &E(2) prop−enc)
    AOT-thus ‹x[F]› using q-prop[THEN &E(2), symmetric] rule=E by blast
  qed
next
  AOT-assume 0: ‹[A!]x & ∀ F (x[F] ≡ ∃ q ((q ≡ p) & F = [λy q]))›
  AOT-show ‹[A!]x & ∀ F (x[F] → Propositional([F])) & ∀ q (x Σ q ≡ (q ≡ p))›
  proof(safe intro!: &I 0[THEN &E(1)] GEN →I)
    fix F
    AOT-assume ‹x[F]›
    AOT-hence ‹∃ q ((q ≡ p) & F = [λy q])›
      using 0[THEN &E(2)] ∀E(2) ≡E(1) by blast
```

then AOT-obtain *q* **where** ‹$(q \equiv p)$ & $F = [\lambda y\ q]$›
  using ∃*E*[*rotated*] **by** *blast*
 AOT-hence ‹$F = [\lambda y\ q]$› **using** &*E*(*2*) **by** *blast*
 AOT-hence ‹∃ *q* $F = [\lambda y\ q]$› **by** (*rule* ∃*I*)
 AOT-thus ‹*Propositional*([*F*])› **by** (*metis* $\equiv_{df}I$ *prop*−*prop1*)
next
 AOT-show ‹*x*Σ*r* ≡ (*r* ≡ *p*)› **for** *r*
 proof(*rule* ≡*I*; *rule* →*I*)
  AOT-assume ‹*x*Σ*r*›
  AOT-hence ‹*x*[λ*y r*]› **by** (*metis* $\equiv_{df}E$ &*E*(*2*) *prop*−*enc*)
  AOT-hence ‹∃ *q* ((*q* ≡ *p*) & [λ*y r*] = [λ*y q*])›
   using *0*[*THEN* &*E*(*2*), *THEN* ∀*E*(*1*), *OF prop*−*prop2*:*2*, *THEN* ≡*E*(*1*)] **by** *blast*
  then AOT-obtain *q* **where** ‹(*q* ≡ *p*) & [λ*y r*] = [λ*y q*]›
   using ∃*E*[*rotated*] **by** *blast*
  AOT-thus ‹*r* ≡ *p*›
   by (*metis rule*=*E* &*E*(*1*,*2*) *id-sym* ≡*E*(*2*) *Commutativity of* ≡
     p−*identity*−*thm2*:*3*)
 next
  AOT-assume ‹*r* ≡ *p*›
  AOT-hence ‹(*r* ≡ *p*) & [λ*y r*] = [λ*y r*]›
   by (*metis rule*=*I*:*1* ≡*S*(*1*) ≡*E*(*2*) *Commutativity of* & *prop*−*prop2*:*2*)
  AOT-hence ‹∃ *q* ((*q* ≡ *p*) & [λ*y r*] = [λ*y q*])› **by** (*rule* ∃*I*)
  AOT-hence ‹*x*[λ*y r*]›
   using *0*[*THEN* &*E*(*2*), *THEN* ∀*E*(*1*), *OF prop*−*prop2*:*2*, *THEN* ≡*E*(*2*)] **by** *blast*
  AOT-thus ‹*x*Σ*r*› **by** (*metis* $\equiv_{df}I$ &*I ex*:*1*:*a prop*−*enc rule*−*ui*:*3*)
 qed
qed
qed

AOT-theorem *ext*−*p*−*tv*:*1*: ‹∃!*x ExtensionOf*(*x*, *p*)›
 by (*AOT-subst* ‹*ExtensionOf*(*x*, *p*)› ‹*TruthValueOf*(*x*, *p*)› **for**: *x*)
  (*auto simp*: *extof*−*e p*−*has*−!*tv*:*2*)

AOT-theorem *ext*−*p*−*tv*:*2*: ‹ι*x*(*ExtensionOf*(*x*, *p*))↓›
 using *A*−*Exists*:*2 RA*[*2*] *ext*−*p*−*tv*:*1* ≡*E*(*2*) **by** *blast*

AOT-theorem *ext*−*p*−*tv*:*3*: ‹ι*x*(*ExtensionOf*(*x*, *p*)) = ∘*p*›
proof −
 AOT-have *0*: ‹𝓐∀*x*(*ExtensionOf*(*x*, *p*) ≡ *TruthValueOf*(*x*,*p*))›
  by (*rule RA*[*2*]; *rule GEN*; *rule extof*−*e*)
 AOT-have *1*: ‹∘*p* = ι*x TruthValueOf*(*x*,*p*)›
  using *rule*−*id*−*df*:*1 the*−*tv*−*p uni*−*tv* **by** *blast*
 show *?thesis*
  apply (*rule equiv*−*desc*−*eq*:*1*[*THEN* →*E*, *OF 0*, *THEN* ∀*E*(*1*)[**where** τ=‹«∘*p*»›],
        THEN ≡*E*(*2*), *symmetric*])
  using *1 t*=*t*−*proper*:*1 vdash*−*properties*:*10* **apply** *blast*
  by (*fact 1*)
qed

10 Restricted Variables

locale *AOT-restriction-condition* =
 fixes ψ :: ‹′*a*::*AOT-Term-id-2* ⇒ o›
 assumes *res*−*var*:*2*[*AOT*]: ‹[*v* ⊨ ∃α ψ{α}]›
 assumes *res*−*var*:*3*[*AOT*]: ‹[*v* ⊨ ψ{τ} → τ↓]›

ML‹
fun register-restricted-type (*name*:*string*, *restriction*:*string*) *thy* =
let
val ctxt = *thy*
val ctxt = *setupStrictWorld ctxt*
val trm = *Syntax.check-term ctxt* (*AOT-read-term* @{*nonterminal* φ′} *ctxt restriction*)
val free = *case* (*Term.add-frees trm* []) *of* [*f*] => *f* |

```
  - => raise Term.TERM (Expected single free variable., [trm])
val trm = Term.absfree free trm
val localeTerm = Const (const-name ‹AOT-restriction-condition›, dummyT) $ trm
val localeTerm = Syntax.check-term ctxt localeTerm
fun after-qed thms thy = let
val st = Interpretation.global-interpretation
  (([(@{locale AOT-restriction-condition}, ((name, true),
        (Expression.Named [(ψ, trm)], [])))], [])) [] thy
val st = Proof.refine-insert (flat thms) st
val thy = Proof.global-immediate-proof st

val thy = Local-Theory.background-theory
  (AOT-Constraints.map (Symtab.update
    (name, (term-of (snd free), term-of (snd free))))) thy
val thy = Local-Theory.background-theory
  (AOT-Restriction.map (Symtab.update
    (name, (trm, Const (const-name ‹AOT-term-of-var›, dummyT))))) thy

in thy end
in
Proof.theorem NONE after-qed [[(HOLogic.mk-Trueprop localeTerm, [])]] ctxt
end

val - =
  Outer-Syntax.command
    command-keyword ‹AOT-register-restricted-type›
    Register a restricted type.
    (((Parse.short-ident --| Parse.$$$ :) -- Parse.term) >>
    (Toplevel.local-theory-to-proof NONE NONE o register-restricted-type));
›

locale AOT-rigid-restriction-condition = AOT-restriction-condition +
  assumes rigid[AOT]: ‹[v ⊨ ∀ α(ψ{α} → □ψ{α})]›
begin
lemma rigid-condition[AOT]: ‹[v ⊨ □(ψ{α} → □ψ{α})]›
  using rigid[THEN ∀ E(2)] RN by simp
lemma type-set-nonempty[AOT-no-atp, no-atp]: ‹∃ x . x ∈ { α . [w₀ ⊨ ψ{α}]}›
  by (metis instantiation mem-Collect-eq res−var:2)
end

locale AOT-restricted-type = AOT-rigid-restriction-condition +
  fixes Rep and Abs
  assumes AOT-restricted-type-definition[AOT-no-atp]:
    ‹type-definition Rep Abs { α . [w₀ ⊨ ψ{α}]}›
begin

AOT-theorem restricted-var-condition: ‹ψ{«AOT-term-of-var (Rep α)»}›
proof −
  interpret type-definition Rep Abs { α . [w₀ ⊨ ψ{α}]}
    using AOT-restricted-type-definition.
  AOT-actually {
    AOT-have ‹«AOT-term-of-var (Rep α)»↓› and ‹ψ{«AOT-term-of-var (Rep α)»}›
      using AOT-sem-imp Rep res−var:3 by auto
  }
  moreover AOT-actually {
    AOT-have ‹ψ{α} → □ψ{α}› for α
      using AOT-sem-box rigid-condition by presburger
    AOT-hence ‹ψ{τ} → □ψ{τ}› if ‹τ↓› for τ
      by (metis AOT-model.AOT-term-of-var-cases AOT-sem-denotes that)
  }
  ultimately AOT-show ‹ψ{«AOT-term-of-var (Rep α)»}›
    using AOT-sem-box AOT-sem-imp by blast
qed
```

```
lemmas ψ = restricted-var-condition

AOT-theorem GEN: assumes ‹for arbitrary α: φ{«AOT-term-of-var (Rep α)»}›
  shows ‹∀ α (ψ{α} → φ{α})›
proof(rule GEN; rule →I)
  interpret type-definition Rep Abs { α . [w₀ ⊨ ψ{α}]}
    using AOT-restricted-type-definition.
  fix α
  AOT-assume ‹ψ{α}›
  AOT-hence ‹𝓐ψ{α}›
    by (metis AOT-model-axiom-def AOT-sem-box AOT-sem-imp act-closure rigid-condition)
  hence 0: ‹[w₀ ⊨ ψ{α}]› by (metis AOT-sem-act)
  {
    fix τ
    assume α-def: ‹α = Rep τ›
    AOT-have ‹φ{α}›
      unfolding α-def
      using assms by blast
  }
  AOT-thus ‹φ{α}›
    using Rep-cases[simplified, OF 0]
    by blast
qed
lemmas ∀ I = GEN

end


lemma AOT-restricted-type-intro[AOT-no-atp, no-atp]:
  assumes ‹type-definition Rep Abs { α . [w₀ ⊨ ψ{α}]}›
      and ‹AOT-rigid-restriction-condition ψ›
  shows ‹AOT-restricted-type ψ Rep Abs›
  by (auto intro!: assms AOT-restricted-type-axioms.intro AOT-restricted-type.intro)



ML‹
fun register-rigid-restricted-type (name:string, restriction:string) thy =
let
val ctxt = thy
val ctxt = setupStrictWorld ctxt
val trm = Syntax.check-term ctxt (AOT-read-term @{nonterminal φ'} ctxt restriction)
val free = case (Term.add-frees trm []) of [f] => f
  | - => raise Term.TERM (Expected single free variable., [trm])
val trm = Term.absfree free trm
val localeTerm = HOLogic.mk-Trueprop
  (Const (const-name‹AOT-rigid-restriction-condition›, dummyT) $ trm)
val localeTerm = Syntax.check-prop ctxt localeTerm
val int-bnd = Binding.concealed (Binding.qualify true internal (Binding.name name))
val bnds = {Rep-name = Binding.qualify true name (Binding.name Rep),
            Abs-name = Binding.qualify true Abs int-bnd,
            type-definition-name = Binding.qualify true type-definition int-bnd}

fun after-qed witts thy = let
  val thms = (map (Element.conclude-witness ctxt) (flat witts))

  val typeset = HOLogic.mk-Collect (α, dummyT,
    const‹AOT-model-valid-in› $ const‹w₀› $
    (trm $ (Const (const-name‹AOT-term-of-var›, dummyT) $ Bound 0)))
  val typeset = Syntax.check-term thy typeset
  val nonempty-thm = Drule.OF
    (@{thm AOT-rigid-restriction-condition.type-set-nonempty}, thms)
```

```
  val ((-,st),thy) = Typedef.add-typedef {overloaded=true}
    (Binding.name name, [], Mixfix.NoSyn) typeset (SOME bnds)
    (fn ctxt => (Tactic.resolve-tac ctxt ([nonempty-thm]) 1)) thy
  val ({rep-type = -, abs-type = -, Rep-name = Rep-name, Abs-name = Abs-name,
        axiom-name = -},
      {inhabited = -, type-definition = type-definition, Rep = -,
       Rep-inverse = -, Abs-inverse = -, Rep-inject = -, Abs-inject = -,
       Rep-cases = -, Abs-cases = -, Rep-induct = -, Abs-induct = -}) = st

  val locale-thm = Drule.OF (@{thm AOT-restricted-type-intro}, type-definition::thms)

  val st = Interpretation.global-interpretation (([(@{locale AOT-restricted-type},
    ((name, true), (Expression.Named [
      (ψ, trm),
      (Rep, Const (Rep-name, dummyT)),
      (Abs, Const (Abs-name, dummyT))], [])))
    ], [])) [] thy

  val st = Proof.refine-insert [locale-thm] st
  val thy = Proof.global-immediate-proof st

  val thy = Local-Theory.background-theory (AOT-Constraints.map (
    Symtab.update (name, (term-of (snd free), term-of (snd free))))) thy
  val thy = Local-Theory.background-theory (AOT-Restriction.map (
    Symtab.update (name, (trm, Const (Rep-name, dummyT))))) thy

  in thy end
in
Element.witness-proof after-qed [[localeTerm]] thy
end

val - =
  Outer-Syntax.command
    command-keyword ‹AOT-register-rigid-restricted-type›
    Register a restricted type.
    (((Parse.short-ident --| Parse.$$$ :) -- Parse.term) >>
    (Toplevel.local-theory-to-proof NONE NONE o register-rigid-restricted-type));
›
```

```
ML‹
fun get-instantiated-allI ctxt varname thm = let
val trm = Thm.concl-of thm
val trm = case trm of (@{const Trueprop} $ (@{const AOT-model-valid-in} $ - $ x)) => x
                      | - => raise Term.TERM (Expected simple theorem., [trm])
fun extractVars (Const (const-name ‹AOT-term-of-var›, t) $ (Const rep $ Var v)) =
    (if fst (fst v) = fst varname
     then [Const (const-name ‹AOT-term-of-var›, t) $ (Const rep $ Var v)]
     else []) (* TODO: care about the index *)
  | extractVars (t1 $ t2) = extractVars t1 @ extractVars t2
  | extractVars (Abs (-, -, t)) = extractVars t
  | extractVars - = []
val vars = extractVars trm
val vartrm = hd vars
val vars = fold Term.add-vars vars []
val var = hd vars
val trmty = (case vartrm of (Const (-, Type (fun, [-, ty])) $ -) => ty
                      | - => raise Match)
val varty = snd var
val tyname = fst (Term.dest-Type varty)
val b = tyname^.∀ I (* TODO: better way to find the theorem *)
val thms = fst (Context.map-proof-result (fn ctxt => (Attrib.eval-thms ctxt
    [(Facts.Named ((b,Position.none),NONE),[])], ctxt)) ctxt)
```

```
val allthm = (case thms of (thm::-) => thm
    | - => raise Fail Unknown restricted type.)
val trm = Abs (Term.string-of-vname (fst var), trmty, Term.abstract-over (vartrm, trm))
val trm = Thm.cterm-of (Context.proof-of ctxt) trm
val phi = hd (Term.add-vars (Thm.prop-of allthm) [])
val allthm = Drule.instantiate-normalize (TVars.empty, Vars.make [(phi,trm)]) allthm
in
allthm
end
›


attribute-setup unconstrain =
  ‹Scan.lift (Scan.repeat1 Args.var) >> (fn args => Thm.rule-attribute []
  (fn ctxt => fn thm =>
    let
    val thm = fold (fn arg => fn thm => thm RS get-instantiated-allI ctxt arg thm)
                  args thm
    val thm = fold (fn - => fn thm => thm RS @{thm ∀E(2)}) args thm
    in
     thm
    end))›
  Generalize a statement about restricted variables to a statement about
   unrestricted variables with explicit restriction condition.



context AOT-restricted-type
begin

AOT-theorem rule−ui:
  assumes ‹∀α(ψ{α} → φ{α})›
  shows ‹φ{«AOT-term-of-var (Rep α)»}›
proof −
  AOT-have ‹φ{α}› if ‹ψ{α}› for α using assms[THEN ∀E(2), THEN →E] that by blast
  moreover AOT-have ‹ψ{«AOT-term-of-var (Rep α)»}›
    by (auto simp: ψ)
  ultimately show ?thesis by blast
qed
lemmas ∀E = rule−ui

AOT-theorem instantiation:
  assumes ‹for arbitrary β: φ{«AOT-term-of-var (Rep β)»} ⊢ χ› and ‹∃α (ψ{α} & φ{α})›
  shows ‹χ›
proof −
  AOT-have ‹φ{«AOT-term-of-var (Rep α)»} → χ› for α
    using assms(1)
    by (simp add: deduction−theorem)
  AOT-hence 0: ‹∀α (ψ{α} → (φ{α} → χ))›
    using GEN by simp
  moreover AOT-obtain α where ‹ψ{α} & φ{α}› using assms(2) ∃E[rotated] by blast
  ultimately AOT-show ‹χ› using AOT-PLM.∀E(2)[THEN →E, THEN →E] &E by fast
qed
lemmas ∃E = instantiation

AOT-theorem existential: assumes ‹φ{«AOT-term-of-var (Rep β)»}›
  shows ‹∃ α (ψ{α} & φ{α})›
  by (meson AOT-restricted-type.ψ AOT-restricted-type-axioms assms
          &I existential:2[const-var])
lemmas ∃I = existential
end
```

context *AOT-rigid-restriction-condition*
begin

AOT-theorem *res−var−bound−reas*[*1*]:
‹∀ α(ψ{α} → ∀ β φ{α, β}) ≡ ∀ β∀ α (ψ{α} → φ{α, β})›
proof(*safe intro*!: ≡*I* →*I GEN*)
fix β α
AOT-assume ‹∀ α (ψ{α} → ∀ β φ{α, β})›
AOT-hence ‹ψ{α} → ∀ β φ{α, β}› **using** ∀ *E*(*2*) **by** *blast*
moreover AOT-assume ‹ψ{α}›
ultimately AOT-have ‹∀ β φ{α, β}› **using** →*E* **by** *blast*
AOT-thus ‹φ{α, β}› **using** ∀ *E*(*2*) **by** *blast*
next
fix α β
AOT-assume ‹∀ β∀ α(ψ{α} → φ{α, β})›
AOT-hence ‹∀ α(ψ{α} → φ{α, β})› **using** ∀ *E*(*2*) **by** *blast*
AOT-hence ‹ψ{α} → φ{α, β}› **using** ∀ *E*(*2*) **by** *blast*
moreover AOT-assume ‹ψ{α}›
ultimately AOT-show ‹φ{α, β}› **using** →*E* **by** *blast*
qed

AOT-theorem *res−var−bound−reas*[*BF*]:
‹∀ α(ψ{α} → □φ{α}) → □∀ α(ψ{α} → φ{α})›
proof(*safe intro*!: →*I*)
AOT-assume ‹∀ α(ψ{α} → □φ{α})›
AOT-hence ‹ψ{α} → □φ{α}› **for** α
using ∀ *E*(*2*) **by** *blast*
AOT-hence ‹□(ψ{α} → φ{α})› **for** α
by (*metis sc−eq−box−box*:*6* *rigid-condition vdash−properties*:*6*)
AOT-hence ‹∀ α □(ψ{α} → φ{α})›
by (*rule GEN*)
AOT-thus ‹□∀ α (ψ{α} → φ{α})›
by (*metis BF vdash−properties*:*6*)
qed

AOT-theorem *res−var−bound−reas*[*CBF*]:
‹□∀ α(ψ{α} → φ{α}) → ∀ α(ψ{α} → □φ{α})›
proof(*safe intro*!: →*I GEN*)
fix α
AOT-assume ‹□∀ α (ψ{α} → φ{α})›
AOT-hence ‹∀ α □(ψ{α} → φ{α})›
by (*metis CBF vdash−properties*:*6*)
AOT-hence *1*: ‹□(ψ{α} → φ{α})›
using ∀ *E*(*2*) **by** *blast*
AOT-assume ‹ψ{α}›
AOT-hence ‹□ψ{α}›
by (*metis B◇ T◇ rigid-condition vdash−properties*:*6*)
AOT-thus ‹□φ{α}›
using *1 qml*:*1*[*axiom-inst*, *THEN* →*E*, *THEN* →*E*] **by** *blast*
qed

AOT-theorem *res−var−bound−reas*[*2*]:
‹∀ α (ψ{α} → ***𝒜***φ{α}) → ***𝒜***∀ α (ψ{α} → φ{α})›
proof(*safe intro*!: →*I*)
AOT-assume ‹∀ α (ψ{α} → ***𝒜***φ{α})›
AOT-hence ‹ψ{α} → ***𝒜***φ{α}› **for** α
using ∀ *E*(*2*) **by** *blast*
AOT-hence ‹***𝒜***(ψ{α} → φ{α})› **for** α
by (*metis sc−eq−box−box*:*7* *rigid-condition vdash−properties*:*6*)
AOT-hence ‹∀ α ***𝒜***(ψ{α} → φ{α})›
by (*rule GEN*)
AOT-thus ‹***𝒜***∀ α(ψ{α} → φ{α})›
by (*metis* ≡*E*(*2*) *logic−actual−nec*:*3*[*axiom-inst*])

qed

AOT-theorem *res−var−bound−reas*[*3*]:
‹$\mathcal{A}\forall\,\alpha\ (\psi\{\alpha\} \to \varphi\{\alpha\}) \to \forall\,\alpha\ (\psi\{\alpha\} \to \mathcal{A}\varphi\{\alpha\})$›
proof(*safe intro*!: →*I GEN*)
fix α
AOT-assume ‹$\mathcal{A}\forall\,\alpha\ (\psi\{\alpha\} \to \varphi\{\alpha\})$›
AOT-hence ‹$\forall\,\alpha\ \mathcal{A}(\psi\{\alpha\} \to \varphi\{\alpha\})$›
by (*metis* ≡*E*(*1*) *logic−actual−nec*:*3*[*axiom-inst*])
AOT-hence *1*: ‹$\mathcal{A}(\psi\{\alpha\} \to \varphi\{\alpha\})$› **by** (*metis rule−ui*:*3*)
AOT-assume ‹$\psi\{\alpha\}$›
AOT-hence ‹$\mathcal{A}\psi\{\alpha\}$›
by (*metis nec−imp−act qml*:*2*[*axiom-inst*] *rigid-condition* →*E*)
AOT-thus ‹$\mathcal{A}\varphi\{\alpha\}$›
using *1* **by** (*metis act−cond* →*E*)
qed

AOT-theorem *res−var−bound−reas*[*Buridan*]:
‹$\exists\,\alpha\ (\psi\{\alpha\}\ \&\ \Box\varphi\{\alpha\}) \to \Box\exists\,\alpha\ (\psi\{\alpha\}\ \&\ \varphi\{\alpha\})$›
proof (*rule* →*I*)
AOT-assume ‹$\exists\,\alpha\ (\psi\{\alpha\}\ \&\ \Box\varphi\{\alpha\})$›
then AOT-obtain α **where** ‹$\psi\{\alpha\}\ \&\ \Box\varphi\{\alpha\}$›
using ∃ *E*[*rotated*] **by** *blast*
AOT-hence ‹$\Box(\psi\{\alpha\}\ \&\ \varphi\{\alpha\})$›
by (*metis KBasic*:*11 KBasic*:*3 T*◇ &*I* &*E*(*1*) &*E*(*2*)
≡*E*(*2*) *reductio−aa*:*1 rigid-condition vdash−properties*:*6*)
AOT-hence ‹$\exists\,\alpha\ \Box(\psi\{\alpha\}\ \&\ \varphi\{\alpha\})$›
by (*rule* ∃ *I*)
AOT-thus ‹$\Box\exists\,\alpha\ (\psi\{\alpha\}\ \&\ \varphi\{\alpha\})$›
by (*rule Buridan*[*THEN* →*E*])
qed

AOT-theorem *res−var−bound−reas*[*BF*◇]:
‹$\Diamond\exists\,\alpha\ (\psi\{\alpha\}\ \&\ \varphi\{\alpha\}) \to \exists\,\alpha\ (\psi\{\alpha\}\ \&\ \Diamond\varphi\{\alpha\})$›
proof(*rule* →*I*)
AOT-assume ‹$\Diamond\exists\,\alpha\ (\psi\{\alpha\}\ \&\ \varphi\{\alpha\})$›
AOT-hence ‹$\exists\,\alpha\ \Diamond(\psi\{\alpha\}\ \&\ \varphi\{\alpha\})$›
using *BF*◇[*THEN* →*E*] **by** *blast*
then AOT-obtain α **where** ‹$\Diamond(\psi\{\alpha\}\ \&\ \varphi\{\alpha\})$›
using ∃ *E*[*rotated*] **by** *blast*
AOT-hence ‹$\Diamond\psi\{\alpha\}$› **and** ‹$\Diamond\varphi\{\alpha\}$›
using *KBasic2*:*3* &*E* →*E* **by** *blast*+
moreover AOT-have ‹$\psi\{\alpha\}$›
using *calculation rigid-condition* **by** (*metis B*◇ *K*◇ →*E*)
ultimately AOT-have ‹$\psi\{\alpha\}\ \&\ \Diamond\varphi\{\alpha\}$›
using &*I* **by** *blast*
AOT-thus ‹$\exists\,\alpha\ (\psi\{\alpha\}\ \&\ \Diamond\varphi\{\alpha\})$›
by (*rule* ∃ *I*)
qed

AOT-theorem *res−var−bound−reas*[*CBF*◇]:
‹$\exists\,\alpha\ (\psi\{\alpha\}\ \&\ \Diamond\varphi\{\alpha\}) \to \Diamond\exists\,\alpha\ (\psi\{\alpha\}\ \&\ \varphi\{\alpha\})$›
proof(*rule* →*I*)
AOT-assume ‹$\exists\,\alpha\ (\psi\{\alpha\}\ \&\ \Diamond\varphi\{\alpha\})$›
then AOT-obtain α **where** ‹$\psi\{\alpha\}\ \&\ \Diamond\varphi\{\alpha\}$›
using ∃ *E*[*rotated*] **by** *blast*
AOT-hence ‹$\Box\psi\{\alpha\}$› **and** ‹$\Diamond\varphi\{\alpha\}$›
using *rigid-condition*[*THEN qml*:*2*[*axiom-inst*, *THEN* →*E*], *THEN* →*E*] &*E* **by** *blast*+
AOT-hence ‹$\Diamond(\psi\{\alpha\}\ \&\ \varphi\{\alpha\})$›
by (*metis KBasic*:*16 con−dis−taut*:*5* →*E*)
AOT-hence ‹$\exists\,\alpha\ \Diamond(\psi\{\alpha\}\ \&\ \varphi\{\alpha\})$›
by (*rule* ∃ *I*)

AOT-thus ‹◇∃α (ψ{α} & φ{α})›
using *CBF◇[THEN →E]* **by** *fast*
qed

AOT-theorem *res−var−bound−reas[A−Exists:1]*:
‹𝓐∃!α (ψ{α} & φ{α}) ≡ ∃!α (ψ{α} & 𝓐φ{α})›
proof(*safe intro*!: ≡*I* →*I*)
AOT-assume ‹𝓐∃!α (ψ{α} & φ{α})›
AOT-hence ‹∃!α 𝓐(ψ{α} & φ{α})›
using *A−Exists:1[THEN ≡E(1)]* **by** *blast*
AOT-hence ‹∃!α (𝓐ψ{α} & 𝓐φ{α})›
apply(*AOT-subst* ‹𝓐ψ{α} & 𝓐φ{α}› ‹𝓐(ψ{α} & φ{α})› **for**: α)
apply (*meson Act−Basic:2 intro−elim:3:f oth−class−taut:3:a*)
by *simp*
AOT-thus ‹∃!α (ψ{α} & 𝓐φ{α})›
apply (*AOT-subst* ‹ψ{α}› ‹𝓐ψ{α}› **for**: α)
using *Commutativity of ≡ intro−elim:3:b sc−eq−fur:2*
→*E rigid-condition* **by** *blast*
next
AOT-assume ‹∃!α (ψ{α} & 𝓐φ{α})›
AOT-hence ‹∃!α (𝓐ψ{α} & 𝓐φ{α})›
apply (*AOT-subst* ‹𝓐ψ{α}› ‹ψ{α}› **for**: α)
apply (*meson sc−eq−fur:2 →E rigid-condition*)
by *simp*
AOT-hence ‹∃!α 𝓐(ψ{α} & φ{α})›
apply (*AOT-subst* ‹𝓐(ψ{α} & φ{α})› ‹𝓐ψ{α} & 𝓐φ{α}› **for**: α)
using *Act−Basic:2* **apply** *presburger*
by *simp*
AOT-thus ‹𝓐∃!α (ψ{α} & φ{α})›
by (*metis A−Exists:1 intro−elim:3:b*)
qed

end

theory *AOT-ExtendedRelationComprehension*
imports *AOT-RestrictedVariables*
begin

11 Extended Relation Comprehension

This theory depends on choosing extended models.

interpretation *AOT-ExtendedModel* **by** (*standard*; *auto*)

Auxiliary lemma: negations of denoting relations denote.

AOT-theorem *negation-denotes*: ‹[λx φ{x}]↓ → [λx ¬φ{x}]↓›
proof(*rule* →*I*)
AOT-assume *0*: ‹[λx φ{x}]↓›
AOT-show ‹[λx ¬φ{x}]↓›
proof (*rule safe−ext[axiom-inst, THEN →E, OF &I]*)
AOT-show ‹[λx ¬[λx φ{x}]x]↓› **by** *cqt:2*
next
AOT-have ‹□[λx φ{x}]↓›
using *0 exist−nec[THEN →E]* **by** *blast*
moreover AOT-have ‹□[λx φ{x}]↓ → □∀x (¬[λx φ{x}]x ≡ ¬φ{x})›
by(*rule RM*; *safe intro*!: *GEN* ≡*I* →*I* *β→C(2) β←C(2) cqt:2*)
ultimately AOT-show ‹□∀x (¬[λx φ{x}]x ≡ ¬φ{x})›
using →*E* **by** *blast*
qed
qed

Auxiliary lemma: conjunctions of denoting relations denote.

AOT-theorem *conjunction-denotes*: ‹[λx φ{x}]↓ & [λx ψ{x}]↓ → [λx φ{x} & ψ{x}]↓›

proof(*rule* $\rightarrow I$)
  AOT-assume *0*: ‹$[\lambda x\ \varphi\{x\}]\downarrow$ & $[\lambda x\ \psi\{x\}]\downarrow$›
  AOT-show ‹$[\lambda x\ \varphi\{x\}$ & $\psi\{x\}]\downarrow$›
  proof (*rule safe−ext*[*axiom-inst*, *THEN* $\rightarrow E$, *OF* &*I*])
    AOT-show ‹$[\lambda x\ [\lambda x\ \varphi\{x\}]x$ & $[\lambda x\ \psi\{x\}]x]\downarrow$› **by** *cqt*:*2*
  next
    AOT-have ‹$\Box([\lambda x\ \varphi\{x\}]\downarrow$ & $[\lambda x\ \psi\{x\}]\downarrow)$›
      using *0 exist−nec*[*THEN* $\rightarrow E$] &*E*
          KBasic:*3 df−simplify*:*2 intro−elim*:*3*:*b* **by** *blast*
    moreover AOT-have
      ‹$\Box([\lambda x\ \varphi\{x\}]\downarrow$ & $[\lambda x\ \psi\{x\}]\downarrow) \rightarrow \Box\forall x\ ([\lambda x\ \varphi\{x\}]x$ & $[\lambda x\ \psi\{x\}]x \equiv \varphi\{x\}$ & $\psi\{x\})$›
      by(*rule RM*; *auto intro*!: *GEN* $\equiv I$ $\rightarrow I$ *cqt*:*2* &*I*
                intro: $\beta\leftarrow C$
                dest: &*E* $\beta\rightarrow C$)
    ultimately AOT-show ‹$\Box\forall x\ ([\lambda x\ \varphi\{x\}]x$ & $[\lambda x\ \psi\{x\}]x \equiv \varphi\{x\}$ & $\psi\{x\})$›
      using $\rightarrow E$ **by** *blast*
  qed
qed

AOT-register-rigid-restricted-type
  Ordinary: ‹$O!\kappa$›
proof
  AOT-modally-strict {
    AOT-show ‹$\exists x\ O!x$›
      by (*meson B*$\Diamond$ *T*$\Diamond$ *o−objects−exist*:*1* $\rightarrow E$)
  }
next
  AOT-modally-strict {
    AOT-show ‹$O!\kappa \rightarrow \kappa\downarrow$› **for** κ
      by (*simp add*: $\rightarrow I$ *cqt*:*5*:*a*[*1*][*axiom-inst*, *THEN* $\rightarrow E$, *THEN* &*E*(*2*)])
  }
next
  AOT-modally-strict {
    AOT-show ‹$\forall\alpha(O!\alpha \rightarrow \Box O!\alpha)$›
      by (*simp add*: *GEN oa−facts*:*1*)
  }
qed

AOT-register-variable-names
  Ordinary: *u v r t s*

In PLM this is defined in the Natural Numbers chapter, but since it is helpful for stating the comprehension principles, we already define it here.

AOT-define *eqE* :: ‹$\tau \Rightarrow \tau \Rightarrow \varphi$› (**infixl** ‹$\equiv_E$› *50*)
  eqE: ‹$F \equiv_E G \equiv_{df} F\downarrow$ & $G\downarrow$ & $\forall u\ ([F]u \equiv [G]u)$›

Derive existence claims about relations from the axioms.

AOT-theorem *denotes-all*: ‹$[\lambda x\ \forall G\ (\Box G \equiv_E F \rightarrow x[G])]\downarrow$›
    and *denotes-all-neg*: ‹$[\lambda x\ \forall G\ (\Box G \equiv_E F \rightarrow \neg x[G])]\downarrow$›
proof −
  AOT-have *Aux*: ‹$\forall F\ (\Box F \equiv_E G \rightarrow (x[F] \equiv x[G]))$, $\neg(x[G] \equiv y[G])$
  $\vdash_\Box \exists F([F]x$ & $\neg[F]y)$› **for** *x y G*
  proof −
    AOT-modally-strict {
    AOT-assume *0*: ‹$\forall F\ (\Box F \equiv_E G \rightarrow (x[F] \equiv x[G]))$›
    AOT-assume *G-prop*: ‹$\neg(x[G] \equiv y[G])$›
    {
      AOT-assume ‹$\neg\exists F([F]x$ & $\neg[F]y)$›
      AOT-hence *0*: ‹$\forall F\ \neg([F]x$ & $\neg[F]y)$›
        by (*metis cqt−further*:*4* $\rightarrow E$)
      AOT-have ‹$\forall F\ ([F]x \equiv [F]y)$›
      proof (*rule GEN*; *rule* $\equiv I$; *rule* $\rightarrow I$)
        fix *F*

```
      AOT-assume ‹[F]x›
      moreover AOT-have ‹¬([F]x & ¬[F]y)›
        using 0[THEN ∀E(2)] by blast
      ultimately AOT-show ‹[F]y›
        by (metis &I raa-cor:1)
    next
      fix F
      AOT-assume ‹[F]y›
      AOT-hence ‹¬[λx ¬[F]x]y›
        by (metis ¬¬I β→C(2))
      moreover AOT-have ‹¬([λx ¬[F]x]x & ¬[λx ¬[F]x]y)›
        apply (rule 0[THEN ∀E(1)]) by cqt:2[lambda]
      ultimately AOT-have 1: ‹¬[λx ¬[F]x]x›
        by (metis &I raa-cor:3)
      {
        AOT-assume ‹¬[F]x›
        AOT-hence ‹[λx ¬[F]x]x›
          by (auto intro!: β←C(1) cqt:2)
        AOT-hence ‹p & ¬p› for p
          using 1 by (metis raa-cor:3)
      }
      AOT-thus ‹[F]x› by (metis raa-cor:1)
    qed
    AOT-hence ‹□∀F ([F]x ≡ [F]y)›
      using ind-nec[THEN →E] by blast
    AOT-hence ‹∀F □([F]x ≡ [F]y)›
      by (metis CBF →E)
  } note indistI = this
  {
    AOT-assume G-prop: ‹x[G] & ¬y[G]›
    AOT-hence Ax: ‹A!x›
      using &E(1) ∃I(2) →E encoders-are-abstract by blast

    {
      AOT-assume Ay: ‹A!y›
      {
        fix F
        {
          AOT-assume ‹∀u□([F]u ≡ [G]u)›
          AOT-hence ‹□∀u([F]u ≡ [G]u)›
            using Ordinary.res-var-bound-reas[BF][THEN →E] by simp
          AOT-hence ‹□F ≡E G›
            by (AOT-subst ‹F ≡E G› ‹∀u ([F]u ≡ [G]u)›)
               (auto intro!: eqE[THEN ≡Df, THEN ≡S(1), OF &I] cqt:2)
          AOT-hence ‹x[F] ≡ x[G]›
            using 0[THEN ∀E(2)] ≡E →E by meson
          AOT-hence ‹x[F]›
            using G-prop &E ≡E by blast
        }
        AOT-hence ‹∀u□([F]u ≡ [G]u) → x[F]›
          by (rule →I)
      }
      AOT-hence xprop: ‹∀F(∀u□([F]u ≡ [G]u) → x[F])›
        by (rule GEN)
      moreover AOT-have yprop: ‹¬∀F(∀u□([F]u ≡ [G]u) → y[F])›
      proof (rule raa-cor:2)
        AOT-assume ‹∀F(∀u□([F]u ≡ [G]u) → y[F])›
        AOT-hence ‹∀F(□∀u([F]u ≡ [G]u) → y[F])›
          apply (AOT-subst ‹□∀u([F]u ≡ [G]u)› ‹∀u□([F]u ≡ [G]u)› for: F)
          using Ordinary.res-var-bound-reas[BF]
                Ordinary.res-var-bound-reas[CBF]
                intro-elim:2 apply presburger
          by simp
```

AOT-hence *A*: ‹∀ *F*(□*F* $\equiv_E$ *G* → *y*[*F*])›
by (*AOT-subst* ‹*F* $\equiv_E$ *G*› ‹∀ *u* ([*F*]*u* ≡ [*G*]*u*)› **for**: *F*)
(*auto intro*!: *eqE*[*THEN* ≡*Df*, *THEN* ≡*S*(*1*), *OF* &*I*] *cqt*:*2*)
moreover AOT-have ‹□*G* $\equiv_E$ *G*›
by (*auto intro*!: *eqE*[*THEN* $\equiv_{df}$*I*] *cqt*:*2 RN* &*I GEN* →*I* ≡*I*)
ultimately AOT-have ‹*y*[*G*]› **using** ∀ *E*(*2*) →*E* **by** *blast*
AOT-thus ‹*p* & ¬*p*› **for** *p* **using** *G-prop* &*E* **by** (*metis raa*−*cor*:*3*)
qed
AOT-have ‹∃ *F*([*F*]*x* & ¬[*F*]*y*)›
proof(*rule raa*−*cor*:*1*)
AOT-assume ‹¬∃ *F*([*F*]*x* & ¬[*F*]*y*)›
AOT-hence *indist*: ‹∀ *F* □([*F*]*x* ≡ [*F*]*y*)› **using** *indistI* **by** *blast*
AOT-have ‹∀ *F*(∀ *u*□([*F*]*u* ≡ [*G*]*u*) → *y*[*F*])›
using *indistinguishable-ord-enc-all*[*axiom-inst*, *THEN* →*E*, *OF* &*I*,
OF &*I*, *OF* &*I*, *OF cqt*:*2*[*const-var*][*axiom-inst*],
OF Ax, *OF Ay*, *OF indist*, *THEN* ≡*E*(*1*), *OF xprop*]**.**
AOT-thus ‹∀ *F*(∀ *u*□([*F*]*u* ≡ [*G*]*u*) → *y*[*F*]) & ¬∀ *F*(∀ *u*□([*F*]*u* ≡ [*G*]*u*) → *y*[*F*])›
using *yprop* &*I* **by** *blast*
qed
}
moreover {
AOT-assume *notAy*: ‹¬*A*!*y*›
AOT-have ‹∃ *F*([*F*]*x* & ¬[*F*]*y*)›
apply (*rule* ∃ *I*(*1*)[**where** *τ*=‹«*A*!»›])
using *Ax notAy* &*I* **apply** *blast*
by (*simp add*: *oa*−*exist*:*2*)
}
ultimately AOT-have ‹∃ *F*([*F*]*x* & ¬[*F*]*y*)›
by (*metis raa*−*cor*:*1*)
}
moreover {
AOT-assume *G-prop*: ‹¬*x*[*G*] & *y*[*G*]›
AOT-hence *Ay*: ‹*A*!*y*›
by (*meson* &*E*(*2*) *encoders*−*are*−*abstract existential*:*2*[*const-var*] →*E*)
AOT-hence *notOy*: ‹¬*O*!*y*›
using ≡*E*(*1*) *oa*−*contingent*:*3* **by** *blast*
{
AOT-assume *Ax*: ‹*A*!*x*›
{
fix *F*
{
AOT-assume ‹□∀ *u*([*F*]*u* ≡ [*G*]*u*)›
AOT-hence ‹□*F* $\equiv_E$ *G*›
by (*AOT-subst* ‹*F* $\equiv_E$ *G*› ‹∀ *u*([*F*]*u* ≡ [*G*]*u*)›)
(*auto intro*!: *eqE*[*THEN* ≡*Df*, *THEN* ≡*S*(*1*), *OF* &*I*] *cqt*:*2*)
AOT-hence ‹*x*[*F*] ≡ *x*[*G*]›
using *0*[*THEN* ∀ *E*(*2*)] ≡*E* →*E* **by** *meson*
AOT-hence ‹¬*x*[*F*]›
using *G-prop* &*E* ≡*E* **by** *blast*
}
AOT-hence ‹□∀ *u*([*F*]*u* ≡ [*G*]*u*) → ¬*x*[*F*]›
by (*rule* →*I*)
}
AOT-hence *x-prop*: ‹∀ *F*(□∀ *u*([*F*]*u* ≡ [*G*]*u*) → ¬*x*[*F*])›
by (*rule GEN*)
AOT-have *x-prop*: ‹¬∃ *F*(∀ *u*□([*F*]*u* ≡ [*G*]*u*) & *x*[*F*])›
proof (*rule raa*−*cor*:*2*)
AOT-assume ‹∃ *F*(∀ *u* □([*F*]*u* ≡ [*G*]*u*) & *x*[*F*])›
then AOT-obtain *F* **where** *F-prop*: ‹∀ *u* □([*F*]*u* ≡ [*G*]*u*) & *x*[*F*]›
using ∃ *E*[*rotated*] **by** *blast*
AOT-have ‹□([*F*]*u* ≡ [*G*]*u*)› **for** *u*
using *F-prop*[*THEN* &*E*(*1*), *THEN Ordinary*.∀ *E*]**.**
AOT-hence ‹∀ *u* □([*F*]*u* ≡ [*G*]*u*)›

```
        by (rule Ordinary.GEN)
      AOT-hence ‹□∀ u([F]u ≡ [G]u)›
        by (metis Ordinary.res−var−bound−reas[BF] →E)
      AOT-hence ‹¬x[F]›
        using x-prop[THEN ∀ E(2), THEN →E] by blast
      AOT-thus ‹p & ¬p› for p
        using F-prop[THEN &E(2)] by (metis raa−cor:3)
    qed
    AOT-have y-prop: ‹∃ F(∀ u □([F]u ≡ [G]u) & y[F])›
    proof (rule raa−cor:1)
      AOT-assume ‹¬∃ F (∀ u □([F]u ≡ [G]u) & y[F])›
      AOT-hence 0: ‹∀ F ¬(∀ u □([F]u ≡ [G]u) & y[F])›
        using cqt−further:4[THEN →E] by blast
      {
        fix F
        {
          AOT-assume ‹∀ u □([F]u ≡ [G]u)›
          AOT-hence ‹¬y[F]›
            using 0[THEN ∀ E(2)] &I raa−cor:1 by meson
        }
        AOT-hence ‹(∀ u □([F]u ≡ [G]u) → ¬y[F])›
          by (rule →I)
      }
      AOT-hence A: ‹∀ F(∀ u □([F]u ≡ [G]u) → ¬y[F])›
        by (rule GEN)
      moreover AOT-have ‹∀ u □([G]u ≡ [G]u)›
        by (simp add: RN oth−class−taut:3:a universal−cor →I)
      ultimately AOT-have ‹¬y[G]›
        using ∀ E(2) →E by blast
      AOT-thus ‹p & ¬p› for p
        using G-prop &E by (metis raa−cor:3)
    qed
    AOT-have ‹∃ F([F]x & ¬[F]y)›
    proof(rule raa−cor:1)
      AOT-assume ‹¬∃ F([F]x & ¬[F]y)›
      AOT-hence indist: ‹∀ F □([F]x ≡ [F]y)›
        using indistI by blast
      AOT-thus ‹∃ F(∀ u □([F]u ≡ [G]u) & x[F]) & ¬∃ F(∀ u □([F]u ≡ [G]u) & x[F])›
        using indistinguishable-ord-enc-ex[axiom-inst, THEN →E, OF &I,
                OF &I, OF &I, OF cqt:2[const-var][axiom-inst],
                OF Ax, OF Ay, OF indist, THEN ≡E(2), OF y-prop]
            x-prop &I by blast
    qed
  }
  moreover {
    AOT-assume notAx: ‹¬A!x›
    AOT-hence Ox: ‹O!x›
      using ∨E(3) oa−exist:3 by blast
    AOT-have ‹∃ F([F]x & ¬[F]y)›
      apply (rule ∃ I(1)[where τ=‹«O!»›])
      using Ox notOy &I apply blast
      by (simp add: oa−exist:1)
  }
  ultimately AOT-have ‹∃ F([F]x & ¬[F]y)›
    by (metis raa−cor:1)
}
ultimately AOT-show ‹∃ F([F]x & ¬[F]y)›
  using G-prop by (metis &I →I ≡I raa−cor:1)
}
qed

AOT-modally-strict {
  fix x y
```

```
AOT-assume indist: ‹∀F ([F]x ≡ [F]y)›
AOT-hence nec-indist: ‹□∀F ([F]x ≡ [F]y)›
  using ind−nec vdash−properties:10 by blast
AOT-hence indist-nec: ‹∀F □([F]x ≡ [F]y)›
  using CBF vdash−properties:10 by blast
AOT-assume 0: ‹∀G (□G ≡E F → x[G])›
AOT-hence 1: ‹∀G (□∀u ([G]u ≡ [F]u) → x[G])›
  by (AOT-subst (reverse) ‹∀u ([G]u ≡ [F]u)› ‹G ≡E F› for: G)
     (auto intro!: eqE[THEN ≡Df, THEN ≡S(1), OF &I] cqt:2)
AOT-have ‹x[F]›
  by (safe intro!: 1[THEN ∀E(2), THEN →E] GEN →I RN ≡I)
AOT-have ‹∀G (□G ≡E F → y[G])›
proof(rule raa−cor:1)
  AOT-assume ‹¬∀G (□G ≡E F → y[G])›
  AOT-hence ‹∃G ¬(□G ≡E F → y[G])›
    using cqt−further:2 →E by blast
  then AOT-obtain G where G-prop: ‹¬(□G ≡E F → y[G])›
    using ∃E[rotated] by blast
  AOT-hence 1: ‹□G ≡E F & ¬y[G]›
    by (metis ≡E(1) oth−class−taut:1:b)
  AOT-have xG: ‹x[G]›
    using 0[THEN ∀E(2), THEN →E] 1[THEN &E(1)] by blast
  AOT-hence ‹x[G] & ¬y[G]›
    using 1[THEN &E(2)] &I by blast
  AOT-hence B: ‹¬(x[G] ≡ y[G])›
    using &E(2) ≡E(1) reductio−aa:1 xG by blast
  {
    fix H
    {
      AOT-assume ‹□H ≡E G›
      AOT-hence ‹□(H ≡E G & G ≡E F)›
        using 1 by (metis KBasic:3 con−dis−i−e:1 con−dis−i−e:2:a
                        intro−elim:3:b)
      moreover AOT-have ‹□(H ≡E G & G ≡E F) → □(H ≡E F)›
      proof(rule RM)
        AOT-modally-strict {
          AOT-show ‹H ≡E G & G ≡E F → H ≡E F›
          proof (safe intro!: →I eqE[THEN ≡df I] &I cqt:2 Ordinary.GEN)
            fix u
            AOT-assume ‹H ≡E G & G ≡E F›
            AOT-hence ‹∀u ([H]u ≡ [G]u)› and ‹∀u ([G]u ≡ [F]u)›
              using eqE[THEN ≡df E] &E by blast+
            AOT-thus ‹[H]u ≡ [F]u›
              by (auto dest!: Ordinary.∀E dest: ≡E)
          qed
        }
      qed
      ultimately AOT-have ‹□(H ≡E F)›
        using →E by blast
      AOT-hence ‹x[H]›
        using 0[THEN ∀E(2)] →E by blast
      AOT-hence ‹x[H] ≡ x[G]›
        using xG ≡I →I by blast
    }
    AOT-hence ‹□H ≡E G → (x[H] ≡ x[G])› by (rule →I)
  }
  AOT-hence A: ‹∀H(□H ≡E G → (x[H] ≡ x[G]))›
    by (rule GEN)
  then AOT-obtain F where F-prop: ‹[F]x & ¬[F]y›
    using Aux[OF A, OF B] ∃E[rotated] by blast
  moreover AOT-have ‹[F]y›
    using indist[THEN ∀E(2), THEN ≡E(1), OF F-prop[THEN &E(1)]].
  AOT-thus ‹p & ¬p› for p
```

```
      using F-prop[THEN &E(2)] by (metis raa−cor:3)
  qed
} note 0 = this
AOT-modally-strict {
  fix x y
  AOT-assume ‹∀F ([F]x ≡ [F]y)›
  moreover AOT-have ‹∀F ([F]y ≡ [F]x)›
    by (metis calculation cqt−basic:11 ≡E(2))
  ultimately AOT-have ‹∀G (□G ≡E F → x[G]) ≡ ∀G (□G ≡E F → y[G])›
    using 0 ≡I →I by auto
} note 1 = this
AOT-show ‹[λx ∀G (□G ≡E F → x[G])]↓›
  by (safe intro!: RN GEN →I 1 kirchner−thm:2[THEN ≡E(2)])

AOT-modally-strict {
  fix x y
  AOT-assume indist: ‹∀F ([F]x ≡ [F]y)›
  AOT-hence nec-indist: ‹□∀F ([F]x ≡ [F]y)›
    using ind−nec vdash−properties:10 by blast
  AOT-hence indist-nec: ‹∀F □([F]x ≡ [F]y)›
    using CBF vdash−properties:10 by blast
  AOT-assume 0: ‹∀G (□G ≡E F → ¬x[G])›
  AOT-hence 1: ‹∀G (□∀u ([G]u ≡ [F]u) → ¬x[G])›
    by (AOT-subst (reverse) ‹∀u ([G]u ≡ [F]u)› ‹G ≡E F› for: G)
       (auto intro!: eqE[THEN ≡Df, THEN ≡S(1), OF &I] cqt:2)
  AOT-have ‹¬x[F]›
    by (safe intro!: 1[THEN ∀E(2), THEN →E] GEN →I RN ≡I)
  AOT-have ‹∀G (□G ≡E F → ¬y[G])›
  proof(rule raa−cor:1)
    AOT-assume ‹¬∀G (□G ≡E F → ¬y[G])›
    AOT-hence ‹∃G ¬(□G ≡E F → ¬y[G])›
      using cqt−further:2 →E by blast
    then AOT-obtain G where G-prop: ‹¬(□G ≡E F → ¬y[G])›
      using ∃E[rotated] by blast
    AOT-hence 1: ‹□G ≡E F & ¬¬y[G]›
      by (metis ≡E(1) oth−class−taut:1:b)
    AOT-hence yG: ‹y[G]›
      using G-prop →I raa−cor:3 by blast
    moreover AOT-hence 12: ‹¬x[G]›
      using 0[THEN ∀E(2), THEN →E] 1[THEN &E(1)] by blast
    ultimately AOT-have ‹¬x[G] & y[G]›
      using &I by blast
    AOT-hence B: ‹¬(x[G] ≡ y[G])›
      by (metis 12 ≡E(3) raa−cor:3 yG)
    {
      fix H
      {
        AOT-assume 3: ‹□H ≡E G›
        AOT-hence ‹□(H ≡E G & G ≡E F)›
          using 1
          by (metis KBasic:3 con−dis−i−e:1 →I intro−elim:3:b
                reductio−aa:1 G-prop)
        moreover AOT-have ‹□(H ≡E G & G ≡E F) → □(H ≡E F)›
        proof (rule RM)
          AOT-modally-strict {
            AOT-show ‹H ≡E G & G ≡E F → H ≡E F›
            proof (safe intro!: →I eqE[THEN ≡df I] &I cqt:2 Ordinary.GEN)
              fix u
              AOT-assume ‹H ≡E G & G ≡E F›
              AOT-hence ‹∀u ([H]u ≡ [G]u)› and ‹∀u ([G]u ≡ [F]u)›
                using eqE[THEN ≡df E] &E by blast+
              AOT-thus ‹[H]u ≡ [F]u›
                by (auto dest!: Ordinary.∀E dest: ≡E)
```

qed
}
qed
ultimately AOT-have ‹$\Box(H \equiv_E F)$›
using $\rightarrow E$ **by** *blast*
AOT-hence ‹$\neg x[H]$›
using *0*[*THEN* $\forall E(2)$] $\rightarrow E$ **by** *blast*
AOT-hence ‹$x[H] \equiv x[G]$›
using *12* $\equiv I \rightarrow I$ **by** (*metis raa−cor*:*3*)
}
AOT-hence ‹$\Box H \equiv_E G \rightarrow (x[H] \equiv x[G])$›
by (*rule* $\rightarrow I$)
}
AOT-hence *A*: ‹$\forall H(\Box H \equiv_E G \rightarrow (x[H] \equiv x[G]))$›
by (*rule GEN*)
then AOT-obtain *F* **where** *F-prop*: ‹$[F]x$ & $\neg[F]y$›
using *Aux*[*OF A*, *OF B*] $\exists E$[*rotated*] **by** *blast*
moreover AOT-have ‹$[F]y$›
using *indist*[*THEN* $\forall E(2)$, *THEN* $\equiv E(1)$, *OF F-prop*[*THEN* &*E*(*1*)]]**.**
AOT-thus ‹*p* & $\neg p$› **for** *p*
using *F-prop*[*THEN* &*E*(*2*)] **by** (*metis raa−cor*:*3*)
qed
} note *0 = this*
AOT-modally-strict {
fix *x y*
AOT-assume ‹$\forall F$ ($[F]x \equiv [F]y$)›
moreover AOT-have ‹$\forall F$ ($[F]y \equiv [F]x$)›
by (*metis calculation cqt−basic*:*11* $\equiv E(2)$)
ultimately AOT-have ‹$\forall G$ ($\Box G \equiv_E F \rightarrow \neg x[G]$) $\equiv \forall G$ ($\Box G \equiv_E F \rightarrow \neg y[G]$)›
using *0* $\equiv I \rightarrow I$ **by** *auto*
} note *1 = this*
AOT-show ‹$[\lambda x\ \forall G$ ($\Box G \equiv_E F \rightarrow \neg x[G]$)$]\downarrow$›
by (*safe intro*!: *RN GEN* $\rightarrow I$ *1 kirchner−thm*:*2*[*THEN* $\equiv E(2)$])
qed

Reformulate the existence claims in terms of their negations.

AOT-theorem *denotes-ex*: ‹$[\lambda x\ \exists G$ ($\Box G \equiv_E F$ & $x[G]$)$]\downarrow$›
proof (*rule safe−ext*[*axiom-inst*, *THEN* $\rightarrow E$, *OF* &*I*])
AOT-show ‹$[\lambda x\ \neg\forall G$ ($\Box G \equiv_E F \rightarrow \neg x[G]$)$]\downarrow$›
using *denotes-all-neg*[*THEN negation-denotes*[*THEN* $\rightarrow E$]]**.**
next
AOT-show ‹$\Box\forall x$ ($\neg\forall G$ ($\Box G \equiv_E F \rightarrow \neg x[G]$) $\equiv \exists G$ ($\Box G \equiv_E F$ & $x[G]$))›
by (*AOT-subst* ‹$\Box G \equiv_E F$ & $x[G]$› ‹$\neg(\Box G \equiv_E F \rightarrow \neg x[G])$› **for**: *G x*)
(*auto simp*: *conventions*:*1 rule−eq−df*:*1*
intro: *oth−class−taut*:*4*:*b*[*THEN* $\equiv E(2)$]
intro−elim:*3*:*f*[*OF cqt−further*:*3*, *OF oth−class−taut*:*3*:*b*]
intro!: *RN GEN*)
qed

AOT-theorem *denotes-ex-neg*: ‹$[\lambda x\ \exists G$ ($\Box G \equiv_E F$ & $\neg x[G]$)$]\downarrow$›
proof (*rule safe−ext*[*axiom-inst*, *THEN* $\rightarrow E$, *OF* &*I*])
AOT-show ‹$[\lambda x\ \neg\forall G$ ($\Box G \equiv_E F \rightarrow x[G]$)$]\downarrow$›
using *denotes-all*[*THEN negation-denotes*[*THEN* $\rightarrow E$]]**.**
next
AOT-show ‹$\Box\forall x$ ($\neg\forall G$ ($\Box G \equiv_E F \rightarrow x[G]$) $\equiv \exists G$ ($\Box G \equiv_E F$ & $\neg x[G]$))›
by (*AOT-subst* (*reverse*) ‹$\Box G \equiv_E F$ & $\neg x[G]$› ‹$\neg(\Box G \equiv_E F \rightarrow x[G])$› **for**: *G x*)
(*auto simp*: *oth−class−taut*:*1*:*b*
intro: *oth−class−taut*:*4*:*b*[*THEN* $\equiv E(2)$]
intro−elim:*3*:*f*[*OF cqt−further*:*3*, *OF oth−class−taut*:*3*:*b*]
intro!: *RN GEN*)
qed

Derive comprehension principles.

```
AOT-theorem Comprehension-1:
  shows ‹□∀F∀G(□G ≡E F → (φ{F} ≡ φ{G})) → [λx ∃F (φ{F} & x[F])]↓›
proof(rule →I)
  AOT-assume assm: ‹□∀F∀G(□G ≡E F → (φ{F} ≡ φ{G}))›
  AOT-modally-strict {
    fix x y
    AOT-assume 0: ‹∀F∀G (□G ≡E F → (φ{F} ≡ φ{G}))›
    AOT-assume indist: ‹∀F ([F]x ≡ [F]y)›
    AOT-assume x-prop: ‹∃F (φ{F} & x[F])›
    then AOT-obtain F where F-prop: ‹φ{F} & x[F]›
      using ∃E[rotated] by blast
    AOT-hence ‹□F ≡E F & x[F]›
      by (auto intro!: RN eqE[THEN ≡df I] &I cqt:2 GEN ≡I →I dest: &E)
    AOT-hence ‹∃G(□G ≡E F & x[G])›
      by (rule ∃I)
    AOT-hence ‹[λx ∃G(□G ≡E F & x[G])]x›
      by (safe intro!: β←C denotes-ex cqt:2)
    AOT-hence ‹[λx ∃G(□G ≡E F & x[G])]y›
      using indist[THEN ∀E(1), OF denotes-ex, THEN ≡E(1)] by blast
    AOT-hence ‹∃G(□G ≡E F & y[G])›
      using β→C by blast
    then AOT-obtain G where ‹□G ≡E F & y[G]›
      using ∃E[rotated] by blast
    AOT-hence ‹φ{G} & y[G]›
      using 0[THEN ∀E(2), THEN ∀E(2), THEN →E, THEN ≡E(1)]
            F-prop[THEN &E(1)] &E &I by blast
    AOT-hence ‹∃F (φ{F} & y[F])›
      by (rule ∃I)
  } note 1 = this
  AOT-modally-strict {
    AOT-assume 0: ‹∀F∀G (□G ≡E F → (φ{F} ≡ φ{G}))›
    {
      fix x y
      {
        AOT-assume ‹∀F ([F]x ≡ [F]y)›
        moreover AOT-have ‹∀F ([F]y ≡ [F]x)›
          by (metis calculation cqt−basic:11 ≡E(1))
        ultimately AOT-have ‹∃F (φ{F} & x[F]) ≡ ∃F (φ{F} & y[F])›
          using 0 1[OF 0] ≡I →I by simp
      }
      AOT-hence ‹∀F ([F]x ≡ [F]y) → (∃F (φ{F} & x[F]) ≡ ∃F (φ{F} & y[F]))›
        using →I by blast
    }
    AOT-hence ‹∀x∀y(∀F ([F]x ≡ [F]y) → (∃F (φ{F} & x[F]) ≡  ∃F (φ{F} & y[F])))›
      by (auto intro!: GEN)
  } note 1 = this
  AOT-hence ‹⊢□ ∀F∀G (□G ≡E F → (φ{F} ≡ φ{G})) →
             ∀x∀y(∀F ([F]x ≡ [F]y) → (∃F (φ{F} & x[F]) ≡ ∃F (φ{F} & y[F])))›
    by (rule →I)
  AOT-hence ‹□∀F∀G (□G ≡E F → (φ{F} ≡ φ{G})) →
           □∀x∀y(∀F ([F]x ≡ [F]y) → (∃F (φ{F} & x[F]) ≡ ∃F (φ{F} & y[F])))›
    by (rule RM)
  AOT-hence ‹□∀x∀y(∀F ([F]x ≡ [F]y) → (∃F (φ{F} & x[F]) ≡ ∃F (φ{F} & y[F])))›
    using →E assm by blast
  AOT-thus ‹[λx ∃F (φ{F} & x[F])]↓›
    by (safe intro!: kirchner−thm:2[THEN ≡E(2)])
qed

AOT-theorem Comprehension-2:
  shows ‹□∀F∀G(□G ≡E F → (φ{F} ≡ φ{G})) → [λx ∃F (φ{F} & ¬x[F])]↓›
proof(rule →I)
  AOT-assume assm: ‹□∀F∀G(□G ≡E F → (φ{F} ≡ φ{G}))›
  AOT-modally-strict {
```

fix $x\ y$
AOT-assume *0*: ‹$\forall F \forall G\ (\Box G \equiv_E F \rightarrow (\varphi\{F\} \equiv \varphi\{G\}))$›
AOT-assume *indist*: ‹$\forall F\ ([F]x \equiv [F]y)$›
AOT-assume *x-prop*: ‹$\exists F\ (\varphi\{F\}\ \&\ \neg x[F])$›
then AOT-obtain F **where** *F-prop*: ‹$\varphi\{F\}\ \&\ \neg x[F]$›
using $\exists E$[*rotated*] **by** *blast*
AOT-hence ‹$\Box F \equiv_E F\ \&\ \neg x[F]$›
by (*auto intro*!: *RN eqE*[*THEN* $\equiv_{df}I$] *&I cqt*:*2 GEN* $\equiv I$ $\rightarrow I$ *dest*: *&E*)
AOT-hence ‹$\exists G(\Box G \equiv_E F\ \&\ \neg x[G])$›
by (*rule* $\exists I$)
AOT-hence ‹$[\lambda x\ \exists G(\Box G \equiv_E F\ \&\ \neg x[G])]x$›
by (*safe intro*!: $\beta{\leftarrow}C$ *denotes-ex-neg cqt*:*2*)
AOT-hence ‹$[\lambda x\ \exists G(\Box G \equiv_E F\ \&\ \neg x[G])]y$›
using *indist*[*THEN* $\forall E(1)$, *OF denotes-ex-neg*, *THEN* $\equiv E(1)$] **by** *blast*
AOT-hence ‹$\exists G(\Box G \equiv_E F\ \&\ \neg y[G])$›
using $\beta{\rightarrow}C$ **by** *blast*
then AOT-obtain G **where** ‹$\Box G \equiv_E F\ \&\ \neg y[G]$›
using $\exists E$[*rotated*] **by** *blast*
AOT-hence ‹$\varphi\{G\}\ \&\ \neg y[G]$›
using *0*[*THEN* $\forall E(2)$, *THEN* $\forall E(2)$, *THEN* $\rightarrow E$, *THEN* $\equiv E(1)$]
F-prop[*THEN* *&E*(*1*)] *&E* *&I* **by** *blast*
AOT-hence ‹$\exists F\ (\varphi\{F\}\ \&\ \neg y[F])$›
by (*rule* $\exists I$)
} **note** *1* = *this*
AOT-modally-strict {
AOT-assume *0*: ‹$\forall F \forall G\ (\Box G \equiv_E F \rightarrow (\varphi\{F\} \equiv \varphi\{G\}))$›
{
fix $x\ y$
{
AOT-assume ‹$\forall F\ ([F]x \equiv [F]y)$›
moreover AOT-have ‹$\forall F\ ([F]y \equiv [F]x)$›
by (*metis calculation cqt*−*basic*:*11* $\equiv E(1)$)
ultimately AOT-have ‹$\exists F\ (\varphi\{F\}\ \&\ \neg x[F]) \equiv \exists F\ (\varphi\{F\}\ \&\ \neg y[F])$›
using *0 1*[*OF 0*] $\equiv I$ $\rightarrow I$ **by** *simp*
}
AOT-hence ‹$\forall F\ ([F]x \equiv [F]y) \rightarrow (\exists F\ (\varphi\{F\}\ \&\ \neg x[F]) \equiv \exists F\ (\varphi\{F\}\ \&\ \neg y[F]))$›
using $\rightarrow I$ **by** *blast*
}
AOT-hence ‹$\forall x \forall y(\forall F\ ([F]x \equiv [F]y) \rightarrow (\exists F\ (\varphi\{F\}\ \&\ \neg x[F]) \equiv\ \exists F\ (\varphi\{F\}\ \&\ \neg y[F])))$›
by (*auto intro*!: *GEN*)
} **note** *1* = *this*
AOT-hence ‹$\vdash_\Box \forall F \forall G\ (\Box G \equiv_E F \rightarrow (\varphi\{F\} \equiv \varphi\{G\})) \rightarrow$
$\forall x \forall y(\forall F\ ([F]x \equiv [F]y) \rightarrow (\exists F\ (\varphi\{F\}\ \&\ \neg x[F]) \equiv \exists F\ (\varphi\{F\}\ \&\ \neg y[F])))$›
by (*rule* $\rightarrow I$)
AOT-hence ‹$\Box \forall F \forall G\ (\Box G \equiv_E F \rightarrow (\varphi\{F\} \equiv \varphi\{G\})) \rightarrow$
$\Box \forall x \forall y(\forall F\ ([F]x \equiv [F]y) \rightarrow (\exists F\ (\varphi\{F\}\ \&\ \neg x[F]) \equiv \exists F\ (\varphi\{F\}\ \&\ \neg y[F])))$›
by (*rule RM*)
AOT-hence ‹$\Box \forall x \forall y(\forall F\ ([F]x \equiv [F]y) \rightarrow (\exists F\ (\varphi\{F\}\ \&\ \neg x[F]) \equiv \exists F\ (\varphi\{F\}\ \&\ \neg y[F])))$›
using $\rightarrow E$ *assm* **by** *blast*
AOT-thus ‹$[\lambda x\ \exists F\ (\varphi\{F\}\ \&\ \neg x[F])]\downarrow$›
by (*safe intro*!: *kirchner*−*thm*:*2*[*THEN* $\equiv E(2)$])
qed

Derived variants of the comprehension principles above.

AOT-theorem *Comprehension-1*′:
shows ‹$\Box \forall F \forall G(\Box G \equiv_E F \rightarrow (\varphi\{F\} \equiv \varphi\{G\})) \rightarrow [\lambda x\ \forall F\ (x[F] \rightarrow \varphi\{F\})]\downarrow$›
proof(*rule* $\rightarrow I$)
AOT-assume ‹$\Box \forall F \forall G(\Box G \equiv_E F \rightarrow (\varphi\{F\} \equiv \varphi\{G\}))$›
AOT-hence *0*: ‹$\Box \forall F \forall G(\Box G \equiv_E F \rightarrow (\neg\varphi\{F\} \equiv \neg\varphi\{G\}))$›
by (*AOT-subst* (*reverse*) ‹$\neg\varphi\{F\} \equiv \neg\varphi\{G\}$› ‹$\varphi\{F\} \equiv \varphi\{G\}$› **for**: $F\ G$)
(*auto simp*: *oth*−*class*−*taut*:*4*:*b*)
AOT-show ‹$[\lambda x\ \forall F\ (x[F] \rightarrow \varphi\{F\})]\downarrow$›
proof(*rule safe*−*ext*[*axiom-inst*, *THEN* $\rightarrow E$, *OF* *&I*])

AOT-show ‹$[\lambda x\ \neg\exists F(\neg\varphi\{F\}\ \&\ x[F])]\downarrow$›
using *Comprehension-1*[*THEN* →*E*, *OF 0*, *THEN negation-denotes*[*THEN* →*E*]]**.**
next
AOT-show ‹$\Box\forall x\ (\neg\exists F\ (\neg\varphi\{F\}\ \&\ x[F]) \equiv \forall F\ (x[F] \rightarrow \varphi\{F\}))$›
by (*AOT-subst* (*reverse*) ‹$\neg\varphi\{F\}\ \&\ x[F]$› ‹$\neg(x[F] \rightarrow \varphi\{F\})$› **for**: *F x*)
(*auto simp*: *oth−class−taut*:*1*:*b*[*THEN intro−elim*:*3*:*e*,
OF oth−class−taut:*2*:*a*]
intro: *intro−elim*:*3*:*f*[*OF cqt−further*:*3*, *OF oth−class−taut*:*3*:*a*,
symmetric]
intro!: *RN GEN*)
qed
qed

AOT-theorem *Comprehension-2′*:
shows ‹$\Box\forall F\forall G(\Box G \equiv_E F \rightarrow (\varphi\{F\} \equiv \varphi\{G\})) \rightarrow [\lambda x\ \forall F\ (\varphi\{F\} \rightarrow x[F])]\downarrow$›
proof(*rule* →*I*)
AOT-assume *0*: ‹$\Box\forall F\forall G(\Box G \equiv_E F \rightarrow (\varphi\{F\} \equiv \varphi\{G\}))$›
AOT-show ‹$[\lambda x\ \forall F\ (\varphi\{F\} \rightarrow x[F])]\downarrow$›
proof(*rule safe−ext*[*axiom-inst*, *THEN* →*E*, *OF* &*I*])
AOT-show ‹$[\lambda x\ \neg\exists F(\varphi\{F\}\ \&\ \neg x[F])]\downarrow$›
using *Comprehension-2*[*THEN* →*E*, *OF 0*, *THEN negation-denotes*[*THEN* →*E*]]**.**
next
AOT-show ‹$\Box\forall x\ (\neg\exists F\ (\varphi\{F\}\ \&\ \neg x[F]) \equiv \forall F\ (\varphi\{F\} \rightarrow x[F]))$›
by (*AOT-subst* (*reverse*) ‹$\varphi\{F\}\ \&\ \neg x[F]$› ‹$\neg(\varphi\{F\} \rightarrow x[F])$› **for**: *F x*)
(*auto simp*: *oth−class−taut*:*1*:*b*
intro: *intro−elim*:*3*:*f*[*OF cqt−further*:*3*, *OF oth−class−taut*:*3*:*a*,
symmetric]
intro!: *RN GEN*)
qed
qed

Derive a combined comprehension principles.

AOT-theorem *Comprehension-3*:
‹$\Box\forall F\forall G(\Box G \equiv_E F \rightarrow (\varphi\{F\} \equiv \varphi\{G\})) \rightarrow [\lambda x\ \forall F\ (x[F] \equiv \varphi\{F\})]\downarrow$›
proof(*rule* →*I*)
AOT-assume *0*: ‹$\Box\forall F\forall G(\Box G \equiv_E F \rightarrow (\varphi\{F\} \equiv \varphi\{G\}))$›
AOT-show ‹$[\lambda x\ \forall F\ (x[F] \equiv \varphi\{F\})]\downarrow$›
proof(*rule safe−ext*[*axiom-inst*, *THEN* →*E*, *OF* &*I*])
AOT-show ‹$[\lambda x\ \forall F\ (x[F] \rightarrow \varphi\{F\})\ \&\ \forall F\ (\varphi\{F\} \rightarrow x[F])]\downarrow$›
by (*safe intro*!: *conjunction-denotes*[*THEN* →*E*, *OF* &*I*]
Comprehension-1′[*THEN* →*E*]
Comprehension-2′[*THEN* →*E*] *0*)
next
AOT-show ‹$\Box\forall x\ (\forall F\ (x[F] \rightarrow \varphi\{F\})\ \&\ \forall F\ (\varphi\{F\} \rightarrow x[F]) \equiv \forall F\ (x[F] \equiv \varphi\{F\}))$›
by (*auto intro*!: *RN GEN* ≡*I* →*I* &*I dest*: &*E* ∀*E*(*2*) →*E* ≡*E*(*1*,*2*))
qed
qed

notepad
begin

Verify that the original axioms are equivalent to $\vdash_\Box [\lambda x\ \exists G\ (\Box G \equiv_E F\ \&\ x[G])]\downarrow$ and $\vdash_\Box [\lambda x\ \exists G\ (\Box G \equiv_E F\ \&\ \neg x[G])]\downarrow$.

AOT-modally-strict {
fix *x y H*
AOT-have ‹$A!x\ \&\ A!y\ \&\ \forall F\ \Box([F]x \equiv [F]y) \rightarrow$
$(\forall G\ (\forall z\ (O!z \rightarrow \Box([G]z \equiv [H]z)) \rightarrow x[G]) \equiv$
$\forall G\ (\forall z\ (O!z \rightarrow \Box([G]z \equiv [H]z)) \rightarrow y[G]))$›
proof(*rule* →*I*)
{
fix *x y*
AOT-assume ‹$A!x$›
AOT-assume ‹$A!y$›

```
    AOT-assume indist: ‹∀ F □([F]x ≡ [F]y)›
    AOT-assume ‹∀ G (∀ u □([G]u ≡ [H]u) → x[G])›
    AOT-hence ‹∀ G (□∀ u ([G]u ≡ [H]u) → x[G])›
      using Ordinary.res−var−bound−reas[BF] Ordinary.res−var−bound−reas[CBF]
            intro−elim:2
      by (AOT-subst ‹□∀ u ([G]u ≡ [H]u)› ‹∀ u □([G]u ≡ [H]u)› for: G) auto
    AOT-hence ‹∀ G (□G ≡E H → x[G])›
      by (AOT-subst ‹G ≡E H› ‹∀ u ([G]u ≡ [H]u)› for: G)
         (safe intro!: eqE[THEN ≡Df, THEN ≡S(1), OF &I] cqt:2)
    AOT-hence ‹¬∃ G (□G ≡E H & ¬x[G])›
      by (AOT-subst (reverse) ‹(□G ≡E H & ¬x[G])› ‹¬(□G ≡E H → x[G])› for: G)
        (auto simp: oth−class−taut:1:b cqt−further:3[THEN ≡E(1)])
    AOT-hence ‹¬[λx ∃ G (□G ≡E H & ¬x[G])]x›
      by (auto intro: β→C)
    AOT-hence ‹¬[λx ∃ G (□G ≡E H & ¬x[G])]y›
      using indist[THEN ∀E(1), OF denotes-ex-neg,
                   THEN qml:2[axiom-inst, THEN →E],
                   THEN ≡E(3)] by blast
    AOT-hence ‹¬∃ G (□G ≡E H & ¬y[G])›
      by (safe intro!: β←C denotes-ex-neg cqt:2)
    AOT-hence ‹∀ G ¬(□G ≡E H & ¬y[G])›
      using cqt−further:4[THEN →E] by blast
    AOT-hence ‹∀ G(□G ≡E H → y[G])›
      by (AOT-subst ‹□G ≡E H → y[G]› ‹¬(□G ≡E H & ¬y[G])› for: G)
        (auto simp: oth−class−taut:1:a)
    AOT-hence ‹∀ G (□∀ u([G]u ≡ [H]u) → y[G])›
      by (AOT-subst (reverse) ‹∀ u ([G]u ≡ [H]u)› ‹G ≡E H› for: G)
        (safe intro!: eqE[THEN ≡Df, THEN ≡S(1), OF &I] cqt:2)
    AOT-hence ‹∀ G (∀ u □([G]u ≡ [H]u) → y[G])›
      using Ordinary.res−var−bound−reas[BF] Ordinary.res−var−bound−reas[CBF]
            intro−elim:2
      by (AOT-subst ‹∀ u □([G]u ≡ [H]u)› ‹□∀ u ([G]u ≡ [H]u)› for: G) auto
  } note 0 = this
  AOT-assume ‹A!x & A!y & ∀ F □([F]x ≡ [F]y)›
  AOT-hence ‹A!x› and ‹A!y› and ‹∀ F □([F]x ≡ [F]y)›
    using &E by blast+
  moreover AOT-have ‹∀ F □([F]y ≡ [F]x)›
    using calculation(3)
    apply (safe intro!: CBF[THEN →E] dest!: BF[THEN →E])
    using RM:3 cqt−basic:11 intro−elim:3:b by fast
  ultimately AOT-show ‹∀ G (∀ u □([G]u ≡ [H]u) → x[G]) ≡
                       ∀ G (∀ u □([G]u ≡ [H]u) → y[G])›
    using 0 by (auto intro!: ≡I →I)
qed

AOT-have ‹A!x & A!y & ∀ F □([F]x ≡ [F]y) →
(∃ G (∀ z (O!z → □([G]z ≡ [H]z)) & x[G]) ≡ ∃ G (∀ z (O!z → □([G]z ≡ [H]z)) & y[G]))›
proof(rule →I)
  {
    fix x y
    AOT-assume ‹A!x›
    AOT-assume ‹A!y›
    AOT-assume indist: ‹∀ F □([F]x ≡ [F]y)›
    AOT-assume x-prop: ‹∃ G (∀ u □([G]u ≡ [H]u) & x[G])›
    AOT-hence ‹∃ G (□∀ u ([G]u ≡ [H]u) & x[G])›
      using Ordinary.res−var−bound−reas[BF] Ordinary.res−var−bound−reas[CBF]
            intro−elim:2
      by (AOT-subst ‹□∀ u ([G]u ≡ [H]u)› ‹∀ u □([G]u ≡ [H]u)› for: G) auto
    AOT-hence ‹∃ G (□G ≡E H & x[G])›
      by (AOT-subst ‹G ≡E H› ‹∀ u ([G]u ≡ [H]u)› for: G)
        (safe intro!: eqE[THEN ≡Df, THEN ≡S(1), OF &I] cqt:2)
    AOT-hence ‹[λx ∃ G (□G ≡E H & x[G])]x›
      by (safe intro!: β←C denotes-ex cqt:2)
```

```
      AOT-hence ‹[λx ∃ G (□G ≡E H & x[G])]y›
        using indist[THEN ∀ E(1), OF denotes-ex,
                  THEN qml:2[axiom-inst, THEN →E],
                  THEN ≡E(1)] by blast
      AOT-hence ‹∃ G (□G ≡E H & y[G])›
        by (rule β→C)
      AOT-hence ‹∃ G (□∀ u ([G]u ≡ [H]u) & y[G])›
        by (AOT-subst (reverse) ‹∀ u ([G]u ≡ [H]u)› ‹G ≡E H› for: G)
          (safe intro!: eqE[THEN ≡Df, THEN ≡S(1), OF &I] cqt:2)
      AOT-hence ‹∃ G (∀ u □([G]u ≡ [H]u) & y[G])›
        using Ordinary.res−var−bound−reas[BF]
            Ordinary.res−var−bound−reas[CBF]
            intro−elim:2
        by (AOT-subst ‹∀ u □([G]u ≡ [H]u)› ‹□∀ u ([G]u ≡ [H]u)› for: G) auto
    } note 0 = this
    AOT-assume ‹A!x & A!y & ∀ F □([F]x ≡ [F]y)›
    AOT-hence ‹A!x› and ‹A!y› and ‹∀ F □([F]x ≡ [F]y)›
      using &E by blast+
    moreover AOT-have ‹∀ F □([F]y ≡ [F]x)›
      using calculation(3)
      apply (safe intro!: CBF[THEN →E] dest!: BF[THEN →E])
      using RM:3 cqt−basic:11 intro−elim:3:b by fast
    ultimately AOT-show ‹∃ G (∀ u □([G]u ≡ [H]u) & x[G]) ≡
                    ∃ G (∀ u □([G]u ≡ [H]u) & y[G])›
      using 0 by (auto intro!: ≡I →I)
  qed
}
end
end
```

12 Possible Worlds

```
AOT-define Situation :: ‹τ ⇒ φ› (‹Situation'(-')›)
  situations: ‹Situation(x) ≡df A!x & ∀ F (x[F] → Propositional([F]))›

AOT-theorem T−sit: ‹TruthValue(x) → Situation(x)›
proof(rule →I)
  AOT-assume ‹TruthValue(x)›
  AOT-hence ‹∃ p TruthValueOf(x,p)›
    using T−value[THEN ≡df E] by blast
  then AOT-obtain p where ‹TruthValueOf(x,p)› using ∃ E[rotated] by blast
  AOT-hence ϑ: ‹A!x & ∀ F (x[F] ≡ ∃ q((q ≡ p) & F = [λy q]))›
    using tv−p[THEN ≡df E] by blast
  AOT-show ‹Situation(x)›
  proof(rule situations[THEN ≡df I]; safe intro!: &I GEN →I ϑ[THEN &E(1)])
    fix F
    AOT-assume ‹x[F]›
    AOT-hence ‹∃ q((q ≡ p) & F = [λy q])›
      using ϑ[THEN &E(2), THEN ∀ E(2)[where β=F], THEN ≡E(1)] by argo
    then AOT-obtain q where ‹(q ≡ p) & F = [λy q]› using ∃ E[rotated] by blast
    AOT-hence ‹∃ p F = [λy p]› using &E(2) ∃ I(2) by metis
    AOT-thus ‹Propositional([F])›
      by (metis ≡df I prop−prop1)
  qed
qed

AOT-theorem possit−sit:1: ‹Situation(x) ≡ □Situation(x)›
proof(rule ≡I; rule →I)
  AOT-assume ‹Situation(x)›
  AOT-hence 0: ‹A!x & ∀ F (x[F] → Propositional([F]))›
    using situations[THEN ≡df E] by blast
  AOT-have 1: ‹□(A!x & ∀ F (x[F] → Propositional([F])))›
  proof(rule KBasic:3[THEN ≡E(2)]; rule &I)
```

```
      AOT-show ‹□A!x› using 0[THEN &E(1)] by (metis oa−facts:2[THEN →E])
    next
      AOT-have ‹∀ F (x[F] → Propositional([F])) → □∀ F (x[F] → Propositional([F]))›
        by (AOT-subst ‹Propositional([F])› ‹∃ p (F = [λy p])› for: F :: ‹<κ>›)
           (auto simp: prop−prop1 ≡Df enc−prop−nec:2)
      AOT-thus ‹□∀ F (x[F] → Propositional([F]))›
        using 0[THEN &E(2)] →E by blast
    qed
    AOT-show ‹□Situation(x)›
      by (AOT-subst ‹Situation(x)› ‹A!x & ∀ F (x[F] → Propositional([F]))›)
         (auto simp: 1 ≡Df situations)
  next
    AOT-show ‹Situation(x)› if ‹□Situation(x)›
      using qml:2[axiom-inst, THEN →E, OF that].
  qed

  AOT-theorem possit−sit:2: ‹◇Situation(x) ≡ Situation(x)›
    using possit−sit:1
    by (metis RE◇ S5Basic:2 ≡E(1) ≡E(5) Commutativity of ≡)

  AOT-theorem possit−sit:3: ‹◇Situation(x) ≡ □Situation(x)›
    using possit−sit:1 possit−sit:2 by (meson ≡E(5))

  AOT-theorem possit−sit:4: ‹𝓐Situation(x) ≡ Situation(x)›
    by (meson Act−Basic:5 Act−Sub:2 RA[2] ≡E(1) ≡E(6) possit−sit:2)

  AOT-theorem possit−sit:5: ‹Situation(∘p)›
  proof (safe intro!: situations[THEN ≡df I] &I GEN →I prop−prop1[THEN ≡df I])
    AOT-have ‹∃ F ∘p[F]›
      using tv−id:2[THEN prop−enc[THEN ≡df E], THEN &E(2)]
            existential:1 prop−prop2:2 by blast
    AOT-thus ‹A!∘p›
      by (safe intro!: encoders−are−abstract[unvarify x, THEN →E]
                       t=t−proper:2[THEN →E, OF ext−p−tv:3])
  next
    fix F
    AOT-assume ‹∘p[F]›
    AOT-hence ‹ιx(A!x & ∀ F (x[F] ≡ ∃ q ((q ≡ p) & F = [λy q])))[F]›
      using tv−id:1 rule=E by fast
    AOT-hence ‹𝓐∃ q ((q ≡ p) & F = [λy q])›
      using ≡E(1) desc−nec−encode:1 by fast
    AOT-hence ‹∃ q 𝓐((q ≡ p) & F = [λy q])›
      by (metis Act−Basic:10 ≡E(1))
    then AOT-obtain q where ‹𝓐((q ≡ p) & F = [λy q])› using ∃ E[rotated] by blast
    AOT-hence ‹𝓐F = [λy q]› by (metis Act−Basic:2 con−dis−i−e:2:b intro−elim:3:a)
    AOT-hence ‹F = [λy q]›
      using id−act:1[unvarify β, THEN ≡E(2)] by (metis prop−prop2:2)
    AOT-thus ‹∃ p F = [λy p]›
      using ∃ I by fast
  qed

  AOT-theorem possit−sit:6: ‹Situation(⊤)›
  proof −
    AOT-have true-def: ‹⊢□ ⊤ = ιx (A!x & ∀ F (x[F] ≡ ∃ p(p & F = [λy p])))›
      by (simp add: A−descriptions rule−id−df:1[zero] the−true:1)
    AOT-hence true-den: ‹⊢□ ⊤↓›
      using t=t−proper:1 vdash−properties:6 by blast
    AOT-have ‹𝓐TruthValue(⊤)›
      using actual−desc:2[unvarify x, OF true-den, THEN →E, OF true-def]
      using TV−lem2:1[unvarify x, OF true-den, THEN RA[2],
                      THEN act−cond[THEN →E], THEN →E]
      by blast
    AOT-hence ‹𝓐Situation(⊤)›
```

using *T−sit*[*unvarify x*, *OF true-den*, *THEN RA*[*2*],
THEN act−cond[*THEN* →*E*], *THEN* →*E*] **by** *blast*
AOT-thus ‹*Situation*(⊤)›
using *possit−sit*:*4*[*unvarify x*, *OF true-den*, *THEN* ≡*E*(*1*)] **by** *blast*
qed

AOT-theorem *possit−sit*:*7*: ‹*Situation*(⊥)›
proof −
AOT-have *true-def*: ‹$\vdash_\Box$ ⊥ = ɩ*x* (*A*!*x* & ∀ *F* (*x*[*F*] ≡ ∃ *p*(¬*p* & *F* = [λ*y p*])))›
by (*simp add*: *A−descriptions rule−id−df*:*1*[*zero*] *the−true*:*2*)
AOT-hence *true-den*: ‹$\vdash_\Box$ ⊥↓›
using *t*=*t−proper*:*1 vdash−properties*:*6* **by** *blast*
AOT-have ‹***𝒜****TruthValue*(⊥)›
using *actual−desc*:*2*[*unvarify x*, *OF true-den*, *THEN* →*E*, *OF true-def*]
using *TV−lem2*:*2*[*unvarify x*, *OF true-den*, *THEN RA*[*2*],
THEN act−cond[*THEN* →*E*], *THEN* →*E*]
by *blast*
AOT-hence ‹***𝒜****Situation*(⊥)›
using *T−sit*[*unvarify x*, *OF true-den*, *THEN RA*[*2*],
THEN act−cond[*THEN* →*E*], *THEN* →*E*] **by** *blast*
AOT-thus ‹*Situation*(⊥)›
using *possit−sit*:*4*[*unvarify x*, *OF true-den*, *THEN* ≡*E*(*1*)] **by** *blast*
qed

AOT-register-rigid-restricted-type
Situation: ‹*Situation*(κ)›
proof
AOT-modally-strict {
fix *p*
AOT-obtain *x* **where** ‹*TruthValueOf*(*x*,*p*)›
by (*metis instantiation p−has−*!*tv*:*1*)
AOT-hence ‹∃ *p TruthValueOf*(*x*,*p*)› **by** (*rule* ∃ *I*)
AOT-hence ‹*TruthValue*(*x*)› **by** (*metis* $\equiv_{df}$*I T−value*)
AOT-hence ‹*Situation*(*x*)› **using** *T−sit*[*THEN* →*E*] **by** *blast*
AOT-thus ‹∃ *x Situation*(*x*)› **by** (*rule* ∃ *I*)
}
next
AOT-modally-strict {
AOT-show ‹*Situation*(κ) → κ↓› **for** κ
proof (*rule* →*I*)
AOT-assume ‹*Situation*(κ)›
AOT-hence ‹*A*!κ› **by** (*metis* $\equiv_{df}$*E* &*E*(*1*) *situations*)
AOT-thus ‹κ↓› **by** (*metis russell−axiom*[*exe*,*1*].ψ*-denotes-asm*)
qed
}
next
AOT-modally-strict {
AOT-show ‹∀ α(*Situation*(α) → □*Situation*(α))›
using *possit−sit*:*1*[*THEN conventions*:*3*[*THEN* $\equiv_{df}$*E*],
THEN &*E*(*1*)] *GEN* **by** *fast*
}
qed

AOT-register-variable-names
Situation: *s*

AOT-define *TruthInSituation* :: ‹τ ⇒ φ ⇒ φ› (‹(- ⊨/ -)› [*100*, *40*] *100*)
true−in−s: ‹*s* ⊨ *p* $\equiv_{df}$ *s***Σ***p*›

notepad
begin

fix *x p q*

```
  have ‹«x ⊨ p → q» = «(x ⊨ p) → q»›
    by simp
  have ‹«x ⊨ p & q» = «(x ⊨ p) & q»›
    by simp
  have ‹«x ⊨ ¬p» = «x ⊨ (¬p)»›
    by simp
  have ‹«x ⊨ □p» = «x ⊨ (□p)»›
    by simp
  have ‹«x ⊨ 𝓐p» = «x ⊨ (𝓐p)»›
    by simp
  have ‹«□x ⊨ p» = «□(x ⊨ p)»›
    by simp
  have ‹«¬x ⊨ p» = «¬(x ⊨ p)»›
    by simp
end


AOT-theorem lem1: ‹Situation(x) → (x ⊨ p ≡ x[λy p])›
proof (rule →I; rule ≡I; rule →I)
  AOT-assume ‹Situation(x)›
  AOT-assume ‹x ⊨ p›
  AOT-hence ‹xΣp›
    using true−in−s[THEN ≡df E] &E by blast
  AOT-thus ‹x[λy p]› using prop−enc[THEN ≡df E] &E by blast
next
  AOT-assume 1: ‹Situation(x)›
  AOT-assume ‹x[λy p]›
  AOT-hence ‹xΣp›
    using prop−enc[THEN ≡df I, OF &I, OF cqt:2(1)] by blast
  AOT-thus ‹x ⊨ p›
    using true−in−s[THEN ≡df I] 1 &I by blast
qed

AOT-theorem lem2:1: ‹s ⊨ p ≡ □s ⊨ p›
proof −
  AOT-have sit: ‹Situation(s)›
    by (simp add: Situation.ψ)
  AOT-have ‹s ⊨ p ≡ s[λy p]›
    using lem1[THEN →E, OF sit] by blast
  also AOT-have ‹. . . ≡ □s[λy p]›
    by (rule en−eq:2[1][unvarify F]) cqt:2[lambda]
  also AOT-have ‹. . . ≡ □s ⊨ p›
    using lem1[THEN RM, THEN →E, OF possit−sit:1[THEN ≡E(1), OF sit]]
    by (metis KBasic:6 ≡E(2) Commutativity of ≡ →E)
  finally show ?thesis.
qed

AOT-theorem lem2:2: ‹◇s ⊨ p ≡ s ⊨ p›
proof −
  AOT-have ‹□(s ⊨ p → □s ⊨ p)›
    using possit−sit:1[THEN ≡E(1), OF Situation.ψ]
          lem2:1[THEN conventions:3[THEN ≡df E, THEN &E(1)]]
          RM[OF →I, THEN →E] by blast
  thus ?thesis by (metis B◇ S5Basic:13 T◇ ≡I ≡E(1) →E)
qed

AOT-theorem lem2:3: ‹◇s ⊨ p ≡ □s ⊨ p›
  using lem2:1 lem2:2 by (metis ≡E(5))

AOT-theorem lem2:4: ‹𝓐(s ⊨ p) ≡ s ⊨ p›
proof −
  AOT-have ‹□(s ⊨ p → □s ⊨ p)›
    using possit−sit:1[THEN ≡E(1), OF Situation.ψ]
```

```
    lem2:1[THEN conventions:3[THEN ≡df E, THEN &E(1)]]
    RM[OF →I, THEN →E] by blast
  thus ?thesis
    using sc−eq−fur:2[THEN →E] by blast
qed

AOT-theorem lem2:5: ‹¬s ⊨ p ≡ □¬s ⊨ p›
  by (metis KBasic2:1 contraposition:1[2] →I ≡I ≡E(3) ≡E(4) lem2:2)

AOT-theorem sit−identity: ‹s = s′ ≡ ∀p(s ⊨ p ≡ s′ ⊨ p)›
proof(rule ≡I; rule →I)
  AOT-assume ‹s = s′›
  moreover AOT-have ‹∀p(s ⊨ p ≡ s ⊨ p)›
    by (simp add: oth−class−taut:3:a universal−cor)
  ultimately AOT-show ‹∀p(s ⊨ p ≡ s′ ⊨ p)›
    using rule=E by fast
next
  AOT-assume a: ‹∀p (s ⊨ p ≡ s′ ⊨ p)›
  AOT-show ‹s = s′›
  proof(safe intro!: ab−obey:1[THEN →E, THEN →E] &I GEN ≡I →I)
    AOT-show ‹A!s› using Situation.ψ ≡df E &E(1) situations by blast
  next
    AOT-show ‹A!s′› using Situation.ψ ≡df E &E(1) situations by blast
  next
    fix F
    AOT-assume 0: ‹s[F]›
    AOT-hence ‹∃p (F = [λy p])›
      using Situation.ψ[THEN situations[THEN ≡df E], THEN &E(2),
                  THEN ∀E(2)[where β=F], THEN →E]
          prop−prop1[THEN ≡df E] by blast
    then AOT-obtain p where F-def: ‹F = [λy p]›
      using ∃E by metis
    AOT-hence ‹s[λy p]›
      using 0 rule=E by blast
    AOT-hence ‹s ⊨ p›
      using lem1[THEN →E, OF Situation.ψ, THEN ≡E(2)] by blast
    AOT-hence ‹s′ ⊨ p›
      using a[THEN ∀E(2)[where β=p], THEN ≡E(1)] by blast
    AOT-hence ‹s′[λy p]›
      using lem1[THEN →E, OF Situation.ψ, THEN ≡E(1)] by blast
    AOT-thus ‹s′[F]›
      using F-def[symmetric] rule=E by blast
  next
    fix F
    AOT-assume 0: ‹s′[F]›
    AOT-hence ‹∃p (F = [λy p])›
      using Situation.ψ[THEN situations[THEN ≡df E], THEN &E(2),
                  THEN ∀E(2)[where β=F], THEN →E]
          prop−prop1[THEN ≡df E] by blast
    then AOT-obtain p where F-def: ‹F = [λy p]›
      using ∃E by metis
    AOT-hence ‹s′[λy p]›
      using 0 rule=E by blast
    AOT-hence ‹s′ ⊨ p›
      using lem1[THEN →E, OF Situation.ψ, THEN ≡E(2)] by blast
    AOT-hence ‹s ⊨ p›
      using a[THEN ∀E(2)[where β=p], THEN ≡E(2)] by blast
    AOT-hence ‹s[λy p]›
      using lem1[THEN →E, OF Situation.ψ, THEN ≡E(1)] by blast
    AOT-thus ‹s[F]›
      using F-def[symmetric] rule=E by blast
  qed
qed
```

```
AOT-define PartOfSituation :: ‹τ ⇒ τ ⇒ φ› (infixl ‹⊴› 80)
  sit−part−whole: ‹s ⊴ s′ ≡df ∀ p (s |= p → s′ |= p)›

AOT-theorem part:1: ‹s ⊴ s›
  by (rule sit−part−whole[THEN ≡df I])
     (safe intro!: &I Situation.ψ GEN →I)

AOT-theorem part:2: ‹s ⊴ s′ & s ≠ s′ → ¬(s′ ⊴ s)›
proof(rule →I; frule &E(1); drule &E(2); rule raa−cor:2)
  AOT-assume 0: ‹s ⊴ s′›
  AOT-hence a: ‹s |= p → s′ |= p› for p
    using ∀ E(2) sit−part−whole[THEN ≡df E] &E by blast
  AOT-assume ‹s′ ⊴ s›
  AOT-hence b: ‹s′ |= p → s |= p› for p
    using ∀ E(2) sit−part−whole[THEN ≡df E] &E by blast
  AOT-have ‹∀ p (s |= p ≡ s′ |= p)›
    using a b by (simp add: ≡I universal−cor)
  AOT-hence 1: ‹s = s′›
    using sit−identity[THEN ≡E(2)] by metis
  AOT-assume ‹s ≠ s′›
  AOT-hence ‹¬(s = s′)›
    by (metis ≡df E =−infix)
  AOT-thus ‹s = s′ & ¬(s = s′)›
    using 1 &I by blast
qed

AOT-theorem part:3: ‹s ⊴ s′ & s′ ⊴ s′′ → s ⊴ s′′›
proof(rule →I; frule &E(1); drule &E(2);
      safe intro!: &I GEN →I sit−part−whole[THEN ≡df I] Situation.ψ)
  fix p
  AOT-assume ‹s |= p›
  moreover AOT-assume ‹s ⊴ s′›
  ultimately AOT-have ‹s′ |= p›
    using sit−part−whole[THEN ≡df E, THEN &E(2),
                         THEN ∀ E(2)[where β=p], THEN →E] by blast
  moreover AOT-assume ‹s′ ⊴ s′′›
  ultimately AOT-show ‹s′′ |= p›
    using sit−part−whole[THEN ≡df E, THEN &E(2),
                         THEN ∀ E(2)[where β=p], THEN →E] by blast
qed

AOT-theorem sit−identity2:1: ‹s = s′ ≡ s ⊴ s′ & s′ ⊴ s›
proof (safe intro!: ≡I &I →I)
  AOT-show ‹s ⊴ s′› if ‹s = s′›
    using rule=E part:1 that by blast
next
  AOT-show ‹s′ ⊴ s› if ‹s = s′›
    using rule=E part:1 that[symmetric] by blast
next
  AOT-assume ‹s ⊴ s′ & s′ ⊴ s›
  AOT-thus ‹s = s′› using part:2[THEN →E, OF &I]
    by (metis ≡df I &E(1) &E(2) =−infix raa−cor:3)
qed

AOT-theorem sit−identity2:2: ‹s = s′ ≡ ∀ s′′ (s′′ ⊴ s ≡ s′′ ⊴ s′)›
proof(safe intro!: ≡I →I Situation.GEN sit−identity[THEN ≡E(2)]
                   GEN[where ′a=o])
  AOT-show ‹s′′ ⊴ s′› if ‹s′′ ⊴ s› and ‹s = s′› for s′′
    using rule=E that by blast
next
  AOT-show ‹s′′ ⊴ s› if ‹s′′ ⊴ s′› and ‹s = s′› for s′′
    using rule=E id-sym that by blast
```

```
next
  AOT-show ‹s' ⊨ p› if ‹s ⊨ p› and ‹∀ s'' (s'' ⊴ s ≡ s'' ⊴ s')› for p
    using sit−part−whole[THEN ≡df E, THEN &E(2),
              OF that(2)[THEN Situation.∀ E, THEN ≡E(1), OF part:1],
              THEN ∀ E(2), THEN →E, OF that(1)].
next
  AOT-show ‹s ⊨ p› if ‹s' ⊨ p› and ‹∀ s'' (s'' ⊴ s ≡ s'' ⊴ s')› for p
    using sit−part−whole[THEN ≡df E, THEN &E(2),
          OF that(2)[THEN Situation.∀ E, THEN ≡E(2), OF part:1],
          THEN ∀ E(2), THEN →E, OF that(1)].
qed

AOT-define Persistent :: ‹φ ⇒ φ› (‹Persistent'(-')›)
  persistent: ‹Persistent(p) ≡df ∀ s (s ⊨ p → ∀ s' (s ⊴ s' → s' ⊨ p))›

AOT-theorem pers−prop: ‹∀ p Persistent(p)›
  by (safe intro!: GEN[where 'a=o] Situation.GEN persistent[THEN ≡df I] →I)
     (simp add: sit−part−whole[THEN ≡df E, THEN &E(2), THEN ∀ E(2), THEN →E])

AOT-define NullSituation :: ‹τ ⇒ φ› (‹NullSituation'(-')›)
  df−null−trivial:1: ‹NullSituation(s) ≡df ¬∃ p s ⊨ p›

AOT-define TrivialSituation :: ‹τ ⇒ φ› (‹TrivialSituation'(-')›)
  df−null−trivial:2: ‹TrivialSituation(s) ≡df ∀ p s ⊨ p›

AOT-theorem thm−null−trivial:1: ‹∃ !x NullSituation(x)›
proof (AOT-subst ‹NullSituation(x)› ‹A!x & ∀ F (x[F] ≡ F ≠ F)› for: x)
  AOT-modally-strict {
    AOT-show ‹NullSituation(x) ≡ A!x & ∀ F (x[F] ≡ F ≠ F)› for x
    proof (safe intro!: ≡I →I df−null−trivial:1[THEN ≡df I]
              dest!: df−null−trivial:1[THEN ≡df E])
      AOT-assume 0: ‹Situation(x) & ¬∃ p x ⊨ p›
      AOT-have 1: ‹A!x›
        using 0[THEN &E(1), THEN situations[THEN ≡df E], THEN &E(1)].
      AOT-have 2: ‹x[F] → ∃ p F = [λy p]› for F
        using 0[THEN &E(1), THEN situations[THEN ≡df E],
              THEN &E(2), THEN ∀ E(2)]
        by (metis ≡df E →I prop−prop1 →E)
      AOT-show ‹A!x & ∀ F (x[F] ≡ F ≠ F)›
      proof (safe intro!: &I 1 GEN ≡I →I)
        fix F
        AOT-assume ‹x[F]›
        moreover AOT-obtain p where ‹F = [λy p]›
          using calculation 2[THEN →E] ∃ E[rotated] by blast
        ultimately AOT-have ‹x[λy p]›
          by (metis rule=E)
        AOT-hence ‹x ⊨ p›
          using lem1[THEN →E, OF 0[THEN &E(1)], THEN ≡E(2)] by blast
        AOT-hence ‹∃ p (x ⊨ p)›
          by (rule ∃ I)
        AOT-thus ‹F ≠ F›
          using 0[THEN &E(2)] raa−cor:1 &I by blast
      next
        fix F :: ‹<κ> AOT-var›
        AOT-assume ‹F ≠ F›
        AOT-hence ‹¬(F = F)› by (metis ≡df E =−infix)
        moreover AOT-have ‹F = F›
          by (simp add: id−eq:1)
        ultimately AOT-show ‹x[F]› using &I raa−cor:1 by blast
      qed
    next
      AOT-assume 0: ‹A!x & ∀ F (x[F] ≡ F ≠ F)›
      AOT-hence ‹x[F] ≡ F ≠ F› for F
```

using ∀ *E* &*E* **by** *blast*
AOT-hence *1*: ‹¬*x*[*F*]› **for** *F*
using $\equiv_{df}E$ *id−eq*:*1* *=−infix reductio−aa*:*1* ≡*E*(*1*) **by** *blast*
AOT-show ‹*Situation*(*x*) & ¬∃ *p* *x* ⊨ *p*›
proof (*safe intro*!: &*I situations*[*THEN* $\equiv_{df}I$] *0*[*THEN* &*E*(*1*)] *GEN* →*I*)
AOT-show ‹*Propositional*([*F*])› **if** ‹*x*[*F*]› **for** *F*
using *that 1* &*I raa−cor*:*1* **by** *fast*
next
AOT-show ‹¬∃ *p* *x* ⊨ *p*›
proof(*rule raa−cor*:*2*)
AOT-assume ‹∃ *p* *x* ⊨ *p*›
then AOT-obtain *p* **where** ‹*x* ⊨ *p*› **using** ∃ *E*[*rotated*] **by** *blast*
AOT-hence ‹*x*[λ*y* *p*]›
using $\equiv_{df}E$ &*E*(*1*) ≡*E*(*1*) *lem1 modus−tollens*:*1*
raa−cor:*3 true−in−s* **by** *fast*
moreover AOT-have ‹¬*x*[λ*y* *p*]›
by (*rule 1*[*unvarify F*]) *cqt*:*2*[*lambda*]
ultimately AOT-show ‹*p* & ¬*p*› **for** *p* **using** &*I raa−cor*:*1* **by** *blast*
qed
qed
qed
}
next
AOT-show ‹∃ !*x* ([*A*!]*x* & ∀ *F* (*x*[*F*] ≡ *F* ≠ *F*))›
by (*simp add*: *A−objects*!)
qed

AOT-theorem *thm−null−trivial*:*2*: ‹∃ !*x TrivialSituation*(*x*)›
proof (*AOT-subst* ‹*TrivialSituation*(*x*)› ‹*A*!*x* & ∀ *F* (*x*[*F*] ≡ ∃ *p* *F* = [λ*y* *p*])› **for**: *x*)
AOT-modally-strict {
AOT-show ‹*TrivialSituation*(*x*) ≡ *A*!*x* & ∀ *F* (*x*[*F*] ≡ ∃ *p* *F* = [λ*y* *p*])› **for** *x*
proof (*safe intro*!: ≡*I* →*I df−null−trivial*:*2*[*THEN* $\equiv_{df}I$]
dest!: *df−null−trivial*:*2*[*THEN* $\equiv_{df}E$])
AOT-assume *0*: ‹*Situation*(*x*) & ∀ *p* *x* ⊨ *p*›
AOT-have *1*: ‹*A*!*x*›
using *0*[*THEN* &*E*(*1*), *THEN situations*[*THEN* $\equiv_{df}E$], *THEN* &*E*(*1*)]**.**
AOT-have *2*: ‹*x*[*F*] → ∃ *p* *F* = [λ*y* *p*]› **for** *F*
using *0*[*THEN* &*E*(*1*), *THEN situations*[*THEN* $\equiv_{df}E$],
THEN &*E*(*2*), *THEN* ∀ *E*(*2*)]
by (*metis* $\equiv_{df}E$ *deduction−theorem prop−prop1* →*E*)
AOT-show ‹*A*!*x* & ∀ *F* (*x*[*F*] ≡ ∃ *p* *F* = [λ*y* *p*])›
proof (*safe intro*!: &*I 1 GEN* ≡*I* →*I 2*)
fix *F*
AOT-assume ‹∃ *p* *F* = [λ*y* *p*]›
then AOT-obtain *p* **where** ‹*F* = [λ*y* *p*]›
using ∃ *E*[*rotated*] **by** *blast*
moreover AOT-have ‹*x* ⊨ *p*›
using *0*[*THEN* &*E*(*2*)] ∀ *E* **by** *blast*
ultimately AOT-show ‹*x*[*F*]›
by (*metis 0 rule*=*E* &*E*(*1*) *id-sym* ≡*E*(*2*) *lem1*
Commutativity of ≡ →*E*)
qed
next
AOT-assume *0*: ‹*A*!*x* & ∀ *F* (*x*[*F*] ≡ ∃ *p* *F* = [λ*y* *p*])›
AOT-hence *1*: ‹*x*[*F*] ≡ ∃ *p* *F* = [λ*y* *p*]› **for** *F*
using ∀ *E* &*E* **by** *blast*
AOT-have *2*: ‹*Situation*(*x*)›
proof (*safe intro*!: &*I situations*[*THEN* $\equiv_{df}I$] *0*[*THEN* &*E*(*1*)] *GEN* →*I*)
AOT-show ‹*Propositional*([*F*])› **if** ‹*x*[*F*]› **for** *F*
using *1*[*THEN* ≡*E*(*1*), *OF that*]
by (*metis* $\equiv_{df}I$ *prop−prop1*)
qed

AOT-show ‹*Situation*(*x*) & ∀ *p* (*x* ⊨ *p*)›
proof (*safe intro*!: &*I 2 0*[*THEN* &*E*(*1*)] *GEN* →*I*)
AOT-have ‹*x*[λ*y p*] ≡ ∃ *q* [λ*y p*] = [λ*y q*]› **for** *p*
by (*rule 1*[*unvarify F*, **where** τ=«[λ*y p*]»]) *cqt*:*2*[*lambda*]
moreover AOT-have ‹∃ *q* [λ*y p*] = [λ*y q*]› **for** *p*
by (*rule* ∃ *I*(*2*)[**where** β=*p*])
(*simp add*: *rule*=*I*:*1 prop*−*prop2*:*2*)
ultimately AOT-have ‹*x*[λ*y p*]› **for** *p* **by** (*metis* ≡*E*(*2*))
AOT-thus ‹*x* ⊨ *p*› **for** *p*
by (*metis 2* ≡*E*(*2*) *lem1* →*E*)
qed
qed
}
next
AOT-show ‹∃ !*x* ([*A*!]*x* & ∀ *F* (*x*[*F*] ≡ ∃ *p F* = [λ*y p*]))›
by (*simp add*: *A*−*objects*!)
qed

AOT-theorem *thm*−*null*−*trivial*:*3*: ‹ι*x NullSituation*(*x*)↓›
by (*meson A*−*Exists*:*2 RA*[*2*] ≡*E*(*2*) *thm*−*null*−*trivial*:*1*)

AOT-theorem *thm*−*null*−*trivial*:*4*: ‹ι*x TrivialSituation*(*x*)↓›
using *A*−*Exists*:*2 RA*[*2*] ≡*E*(*2*) *thm*−*null*−*trivial*:*2* **by** *blast*

AOT-define *TheNullSituation* :: ‹κ_s› (‹$\mathbf{s}_\emptyset$›)
df−*the*−*null*−*sit*:*1*: ‹$\mathbf{s}_\emptyset =_{df}$ ι*x NullSituation*(*x*)›

AOT-define *TheTrivialSituation* :: ‹κ_s› (‹$\mathbf{s}_V$›)
df−*the*−*null*−*sit*:*2*: ‹$\mathbf{s}_V =_{df}$ ι*x TrivialSituation*(*x*)›

AOT-theorem *null*−*triv*−*sc*:*1*: ‹*NullSituation*(*x*) → □*NullSituation*(*x*)›
proof(*safe intro*!: →*I dest*!: *df*−*null*−*trivial*:*1*[*THEN* $\equiv_{df}$*E*];
frule &*E*(*1*); *drule* &*E*(*2*))
AOT-assume *1*: ‹¬∃ *p* (*x* ⊨ *p*)›
AOT-assume *0*: ‹*Situation*(*x*)›
AOT-hence ‹□*Situation*(*x*)› **by** (*metis* ≡*E*(*1*) *possit*−*sit*:*1*)
moreover AOT-have ‹□¬∃ *p* (*x* ⊨ *p*)›
proof(*rule raa*−*cor*:*1*)
AOT-assume ‹¬□¬∃ *p* (*x* ⊨ *p*)›
AOT-hence ‹◇∃ *p* (*x* ⊨ *p*)›
by (*metis* $\equiv_{df}$*I conventions*:*5*)
AOT-hence ‹∃ *p* ◇(*x* ⊨ *p*)› **by** (*metis BF*◇ →*E*)
then AOT-obtain *p* **where** ‹◇(*x* ⊨ *p*)› **using** ∃ *E*[*rotated*] **by** *blast*
AOT-hence ‹*x* ⊨ *p*›
by (*metis* ≡*E*(*1*) *lem2*:*2*[*unconstrain s*, *THEN* →*E*, *OF 0*])
AOT-hence ‹∃ *p x* ⊨ *p*› **using** ∃ *I* **by** *fast*
AOT-thus ‹∃ *p x* ⊨ *p* & ¬∃ *p x* ⊨ *p*› **using** *1* &*I* **by** *blast*
qed
ultimately AOT-have *2*: ‹□(*Situation*(*x*) & ¬∃ *p x* ⊨ *p*)›
by (*metis KBasic*:*3* &*I* ≡*E*(*2*))
AOT-show ‹□*NullSituation*(*x*)›
by (*AOT-subst* ‹*NullSituation*(*x*)› ‹*Situation*(*x*) & ¬∃ *p x* ⊨ *p*›)
(*auto simp*: *df*−*null*−*trivial*:*1* ≡*Df 2*)
qed

AOT-theorem *null*−*triv*−*sc*:*2*: ‹*TrivialSituation*(*x*) → □*TrivialSituation*(*x*)›
proof(*safe intro*!: →*I dest*!: *df*−*null*−*trivial*:*2*[*THEN* $\equiv_{df}$*E*];
frule &*E*(*1*); *drule* &*E*(*2*))
AOT-assume *0*: ‹*Situation*(*x*)›
AOT-hence *1*: ‹□*Situation*(*x*)› **by** (*metis* ≡*E*(*1*) *possit*−*sit*:*1*)
AOT-assume ‹∀ *p x* ⊨ *p*›
AOT-hence ‹*x* ⊨ *p*› **for** *p*

```
    using ∀E by blast
  AOT-hence ‹□x ⊨ p› for p
    using 0 ≡E(1) lem2:1[unconstrain s, THEN →E] by blast
  AOT-hence ‹∀p □x ⊨ p›
    by (rule GEN)
  AOT-hence ‹□∀p x ⊨ p›
    by (rule BF[THEN →E])
  AOT-hence 2: ‹□(Situation(x) & ∀p x ⊨ p)›
    using 1 by (metis KBasic:3 &I ≡E(2))
  AOT-show ‹□TrivialSituation(x)›
    by (AOT-subst ‹TrivialSituation(x)› ‹Situation(x) & ∀p x ⊨ p›)
       (auto simp: df−null−trivial:2 ≡Df 2)
qed

AOT-theorem null−triv−sc:3: ‹NullSituation(s∅)›
  by (safe intro!: df−the−null−sit:1[THEN =dfI(2)] thm−null−trivial:3
       rule=I:1[OF thm−null−trivial:3]
       !box−desc:2[THEN →E, THEN →E, rotated, OF thm−null−trivial:1,
                   OF ∀I, OF null−triv−sc:1, THEN ∀E(1), THEN →E])

AOT-theorem null−triv−sc:4: ‹TrivialSituation(sV)›
  by (safe intro!: df−the−null−sit:2[THEN =dfI(2)] thm−null−trivial:4
       rule=I:1[OF thm−null−trivial:4]
       !box−desc:2[THEN →E, THEN →E, rotated, OF thm−null−trivial:2,
                   OF ∀I, OF null−triv−sc:2, THEN ∀E(1), THEN →E])

AOT-theorem null−triv−facts:1: ‹NullSituation(x) ≡ Null(x)›
proof (safe intro!: ≡I →I df−null−uni:1[THEN ≡dfI]
                    df−null−trivial:1[THEN ≡dfI]
            dest!: df−null−uni:1[THEN ≡dfE] df−null−trivial:1[THEN ≡dfE])
  AOT-assume 0: ‹Situation(x) & ¬∃p x ⊨ p›
  AOT-have 1: ‹x[F] → ∃p F = [λy p]› for F
    using 0[THEN &E(1), THEN situations[THEN ≡dfE], THEN &E(2), THEN ∀E(2)]
    by (metis ≡dfE deduction−theorem prop−prop1 →E)
  AOT-show ‹A!x & ¬∃F x[F]›
  proof (safe intro!: &I 0[THEN &E(1), THEN situations[THEN ≡dfE],
                          THEN &E(1)];
         rule raa−cor:2)
    AOT-assume ‹∃F x[F]›
    then AOT-obtain F where F-prop: ‹x[F]›
      using ∃E[rotated] by blast
    AOT-hence ‹∃p F = [λy p]›
      using 1[THEN →E] by blast
    then AOT-obtain p where ‹F = [λy p]›
      using ∃E[rotated] by blast
    AOT-hence ‹x[λy p]›
      by (metis rule=E F-prop)
    AOT-hence ‹x ⊨ p›
      using lem1[THEN →E, OF 0[THEN &E(1)], THEN ≡E(2)] by blast
    AOT-hence ‹∃p x ⊨ p›
      by (rule ∃I)
    AOT-thus ‹∃p x ⊨ p & ¬∃p x ⊨ p›
      using 0[THEN &E(2)] &I by blast
  qed
next
  AOT-assume 0: ‹A!x & ¬∃F x[F]›
  AOT-have ‹Situation(x)›
    apply (rule situations[THEN ≡dfI, OF &I, OF 0[THEN &E(1)]]; rule GEN)
    using 0[THEN &E(2)] by (metis →I existential:2[const-var] raa−cor:3)
  moreover AOT-have ‹¬∃p x ⊨ p›
  proof (rule raa−cor:2)
    AOT-assume ‹∃p x ⊨ p›
    then AOT-obtain p where ‹x ⊨ p› by (metis instantiation)
```

    AOT-hence ‹$x[\lambda y\ p]$› **by** (*metis* $\equiv_{df}E$ &*E*(*2*) *prop−enc true−in−s*)
    AOT-hence ‹$\exists F\ x[F]$› **by** (*rule* $\exists I$) *cqt*:*2*[*lambda*]
    AOT-thus ‹$\exists F\ x[F]\ \&\ \neg\exists F\ x[F]$› **using** *0*[*THEN* &*E*(*2*)] &*I* **by** *blast*
  qed
  ultimately AOT-show ‹*Situation*(x) & $\neg\exists p\ x \models p$› **using** &*I* **by** *blast*
qed

AOT-theorem *null−triv−facts*:*2*: ‹$\mathbf{s}_\emptyset = a_\emptyset$›
  apply (*rule* $=_{df}I$(*2*)[*OF df−the−null−sit*:*1*])
   apply (*fact thm−null−trivial*:*3*)
  apply (*rule* $=_{df}I$(*2*)[*OF df−null−uni−terms*:*1*])
   apply (*fact null−uni−uniq*:*3*)
  apply (*rule equiv−desc−eq*:*3*[*THEN* →*E*])
  apply (*rule* &*I*)
   apply (*fact thm−null−trivial*:*3*)
  by (*rule RN*; *rule GEN*; *rule null−triv−facts*:*1*)

AOT-theorem *null−triv−facts*:*3*: ‹$\mathbf{s}_V \neq a_V$›
proof(*rule* =−*infix*[*THEN* $\equiv_{df}I$])
  AOT-have ‹*Universal*(a_V)›
    by (*simp add*: *null−uni−facts*:*4*)
  AOT-hence *0*: ‹$a_V[A!]$›
    using *df−null−uni*:*2*[*THEN* $\equiv_{df}E$] &*E* ∀*E*(*1*)
    by (*metis cqt*:*5*:*a vdash−properties*:*10 vdash−properties*:*1*[*2*])
  moreover AOT-have *1*: ‹$\neg\mathbf{s}_V[A!]$›
  proof(*rule raa−cor*:*2*)
    AOT-have ‹*Situation*($\mathbf{s}_V$)›
      using $\equiv_{df}E$ &*E*(*1*) *df−null−trivial*:*2 null−triv−sc*:*4* **by** *blast*
    AOT-hence ‹$\forall F\ (\mathbf{s}_V[F] \rightarrow$ *Propositional*([*F*]))›
      by (*metis* $\equiv_{df}E$ &*E*(*2*) *situations*)
    moreover AOT-assume ‹$\mathbf{s}_V[A!]$›
    ultimately AOT-have ‹*Propositional*(*A*!)›
      using ∀*E*(*1*)[*rotated*, *OF oa−exist*:*2*] →*E* **by** *blast*
    AOT-thus ‹*Propositional*(*A*!) & ¬*Propositional*(*A*!)›
      using *prop−in−f*:*4*:*d* &*I* **by** *blast*
  qed
  AOT-show ‹$\neg(\mathbf{s}_V = a_V)$›
  proof (*rule raa−cor*:*2*)
    AOT-assume ‹$\mathbf{s}_V = a_V$›
    AOT-hence ‹$\mathbf{s}_V[A!]$› **using** *0 rule=E id-sym* **by** *fast*
    AOT-thus ‹$\mathbf{s}_V[A!]\ \&\ \neg\mathbf{s}_V[A!]$› **using** *1* &*I* **by** *blast*
  qed
qed

definition *ConditionOnPropositionalProperties* :: ‹(<κ> ⇒ o) ⇒ *bool*› **where**
  cond−prop: ‹*ConditionOnPropositionalProperties* ≡ λ φ . ∀ *v* .
                [$v \models \forall F\ (\varphi\{F\} \rightarrow$ *Propositional*([*F*]))]›

syntax *ConditionOnPropositionalProperties* :: ‹*id-position* ⇒ *AOT-prop*›
  (‹*CONDITION*′*-ON*′*-PROPOSITIONAL*′*-PROPERTIES*′(-′)›)

AOT-theorem *cond−prop*[*E*]:
  assumes ‹*CONDITION-ON-PROPOSITIONAL-PROPERTIES*(φ)›
  shows ‹$\forall F\ (\varphi\{F\} \rightarrow$ *Propositional*([*F*]))›
  using *assms*[*unfolded cond−prop*] **by** *auto*

AOT-theorem *cond−prop*[*I*]:
  assumes ‹$\vdash_\Box \forall F\ (\varphi\{F\} \rightarrow$ *Propositional*([*F*]))›
  shows ‹*CONDITION-ON-PROPOSITIONAL-PROPERTIES*(φ)›
  using *assms cond−prop* **by** *metis*

AOT-theorem *pre−comp−sit*:
  assumes ‹*CONDITION-ON-PROPOSITIONAL-PROPERTIES*(φ)›

shows ‹(*Situation*(*x*) & ∀ *F* (*x*[*F*] ≡ φ{*F*})) ≡ (*A*!*x* & ∀ *F* (*x*[*F*] ≡ φ{*F*}))›
proof(*rule* ≡*I*; *rule* →*I*)
AOT-assume ‹*Situation*(*x*) & ∀ *F* (*x*[*F*] ≡ φ{*F*})›
AOT-thus ‹*A*!*x* & ∀ *F* (*x*[*F*] ≡ φ{*F*})›
using &*E situations*[*THEN* $\equiv_{df}$*E*] &*I* **by** *blast*
next
AOT-assume *0*: ‹*A*!*x* & ∀ *F* (*x*[*F*] ≡ φ{*F*})›
AOT-show ‹*Situation*(*x*) & ∀ *F* (*x*[*F*] ≡ φ{*F*})›
proof (*safe intro*!: *situations*[*THEN* $\equiv_{df}$*I*] &*I*)
AOT-show ‹*A*!*x*› **using** *0*[*THEN* &*E*(*1*)].
next
AOT-show ‹∀ *F* (*x*[*F*] → *Propositional*([*F*]))›
proof(*rule GEN*; *rule* →*I*)
fix *F*
AOT-assume ‹*x*[*F*]›
AOT-hence ‹φ{*F*}›
using *0*[*THEN* &*E*(*2*)] ∀ *E* ≡*E* **by** *blast*
AOT-thus ‹*Propositional*([*F*])›
using *cond−prop*[*E*][*OF assms*] ∀ *E* →*E* **by** *blast*
qed
next
AOT-show ‹∀ *F* (*x*[*F*] ≡ φ{*F*})› **using** *0* &*E* **by** *blast*
qed
qed

AOT-theorem *comp−sit*:*1*:
assumes ‹*CONDITION-ON-PROPOSITIONAL-PROPERTIES*(φ)›
shows ‹∃ *s* ∀ *F*(*s*[*F*] ≡ φ{*F*})›
by (*AOT-subst* ‹*Situation*(*x*) & ∀ *F*(*x*[*F*] ≡ φ{*F*})› ‹*A*!*x* & ∀ *F* (*x*[*F*] ≡ φ{*F*})› **for**: *x*)
(*auto simp*: *pre−comp−sit*[*OF assms*] *A−objects*[**where** φ=φ, *axiom-inst*])

AOT-theorem *comp−sit*:*2*:
assumes ‹*CONDITION-ON-PROPOSITIONAL-PROPERTIES*(φ)›
shows ‹∃ !*s* ∀ *F*(*s*[*F*] ≡ φ{*F*})›
by (*AOT-subst* ‹*Situation*(*x*) & ∀ *F*(*x*[*F*] ≡ φ{*F*})› ‹*A*!*x* & ∀ *F* (*x*[*F*] ≡ φ{*F*})› **for**: *x*)
(*auto simp*: *assms pre−comp−sit pre−comp−sit*[*OF assms*] *A−objects*!)

AOT-theorem *can−sit−desc*:*1*:
assumes ‹*CONDITION-ON-PROPOSITIONAL-PROPERTIES*(φ)›
shows ‹ι*s*(∀ *F* (*s*[*F*] ≡ φ{*F*}))↓›
using *comp−sit*:*2*[*OF assms*] *A−Exists*:*2 RA*[*2*] ≡*E*(*2*) **by** *blast*

AOT-theorem *can−sit−desc*:*2*:
assumes ‹*CONDITION-ON-PROPOSITIONAL-PROPERTIES*(φ)›
shows ‹ι*s*(∀ *F* (*s*[*F*] ≡ φ{*F*})) = ι*x*(*A*!*x* & ∀ *F* (*x*[*F*] ≡ φ{*F*}))›
by (*auto intro*!: *equiv−desc−eq*:*2*[*THEN* →*E*, *OF* &*I*,
OF can−sit−desc:*1*[*OF assms*]]
RA[*2*] *GEN pre−comp−sit*[*OF assms*])

AOT-theorem *strict−sit*:
assumes ‹*RIGID-CONDITION*(φ)›
and ‹*CONDITION-ON-PROPOSITIONAL-PROPERTIES*(φ)›
shows ‹*y* = ι*s*(∀ *F* (*s*[*F*] ≡ φ{*F*})) → ∀ *F* (*y*[*F*] ≡ φ{*F*})›
using *rule*=*E*[*rotated*, *OF can−sit−desc*:*2*[*OF assms*(*2*), *symmetric*]]
box−phi−a:*2*[*OF assms*(*1*)] →*E* →*I* &*E* **by** *fast*

AOT-define *actual* :: ‹τ ⇒ φ› (‹*Actual*′(-′)›)
‹*Actual*(*s*) $\equiv_{df}$ ∀ *p* (*s* ⊨ *p* → *p*)›

AOT-theorem *act−and−not−pos*: ‹∃ *s* (*Actual*(*s*) & ◇¬*Actual*(*s*))›
proof −

```
  AOT-obtain q_1 where q_1-prop: ‹q_1 & ◊¬q_1›
    by (metis ≡_df E instantiation cont−tf:1 cont−tf−thm:1)
  AOT-have ‹∃ s (∀ F (s[F] ≡ F = [λy q_1]))›
  proof (safe intro!: comp−sit:1 cond−prop[I] GEN →I)
    AOT-modally-strict {
      AOT-show ‹Propositional([F])› if ‹F = [λy q_1]› for F
        using ≡_df I existential:2[const-var] prop−prop1 that by fastforce
    }
  qed
  then AOT-obtain s_1 where s-prop: ‹∀ F (s_1[F] ≡ F = [λy q_1])›
    using Situation.∃ E[rotated] by meson
  AOT-have ‹Actual(s_1)›
  proof(safe intro!: actual[THEN ≡_df I] &I GEN →I s-prop Situation.ψ)
    fix p
    AOT-assume ‹s_1 ⊨ p›
    AOT-hence ‹s_1[λy p]›
      by (metis ≡_df E &E(2) prop−enc true−in−s)
    AOT-hence ‹[λy p] = [λy q_1]›
      by (rule s-prop[THEN ∀ E(1), THEN ≡E(1), rotated]) cqt:2[lambda]
    AOT-hence ‹p = q_1› by (metis ≡E(2) p−identity−thm2:3)
    AOT-thus ‹p› using q_1-prop[THEN &E(1)] rule=E id-sym by fast
  qed
  moreover AOT-have ‹◊¬Actual(s_1)›
  proof(rule raa−cor:1; drule KBasic:12[THEN ≡E(2)])
    AOT-assume ‹□Actual(s_1)›
    AOT-hence ‹□(Situation(s_1) & ∀ p (s_1 ⊨ p → p))›
      using actual[THEN ≡Df, THEN conventions:3[THEN ≡_df E],
                  THEN &E(1), THEN RM, THEN →E] by blast
    AOT-hence ‹□∀ p (s_1 ⊨ p → p)›
      by (metis RM:1 Conjunction Simplification(2) →E)
    AOT-hence ‹∀ p □(s_1 ⊨ p → p)›
      by (metis CBF vdash−properties:10)
    AOT-hence ‹□(s_1 ⊨ q_1 → q_1)›
      using ∀ E by blast
    AOT-hence ‹□s_1 ⊨ q_1 → □q_1›
      by (metis →E qml:1 vdash−properties:1[2])
    moreover AOT-have ‹s_1 ⊨ q_1›
      using s-prop[THEN ∀ E(1), THEN ≡E(2),
                  THEN lem1[THEN →E, OF Situation.ψ, THEN ≡E(2)]]
            rule=I:1 prop−prop2:2 by blast
    ultimately AOT-have ‹□q_1›
      using ≡_df E &E(1) ≡E(1) lem2:1 true−in−s →E by fast
    AOT-thus ‹◊¬q_1 & ¬◊¬q_1›
      using KBasic:12[THEN ≡E(1)] q_1-prop[THEN &E(2)] &I by blast
  qed
  ultimately AOT-have ‹(Actual(s_1) & ◊¬Actual(s_1))›
    using s-prop &I by blast
  thus ?thesis
    by (rule Situation.∃ I)
qed

AOT-theorem actual−s:1: ‹∃ s Actual(s)›
proof −
  AOT-obtain s where ‹(Actual(s) & ◊¬Actual(s))›
    using act−and−not−pos Situation.∃ E[rotated] by meson
  AOT-hence ‹Actual(s)› using &E &I by metis
  thus ?thesis by (rule Situation.∃ I)
qed

AOT-theorem actual−s:2: ‹∃ s ¬Actual(s)›
proof(rule ∃ I(1)[where τ=‹«s_V»›]; (rule &I)?)
  AOT-show ‹Situation(s_V)›
    using ≡_df E &E(1) df−null−trivial:2 null−triv−sc:4 by blast
```

next
  AOT-show ‹¬*Actual*($\mathbf{s}_V$)›
  proof(*rule raa−cor*:*2*)
    AOT-assume *0*: ‹*Actual*($\mathbf{s}_V$)›
    AOT-obtain p_1 **where** *notp*$_1$: ‹¬p_1›
      by (*metis* ∃*E* ∃*I*(*1*) *log−prop−prop*:*2 non−contradiction*)
    AOT-have ‹$\mathbf{s}_V$ ⊨ p_1›
      using *null−triv−sc*:*4*[*THEN* ≡$_{df}$*E*[*OF df−null−trivial*:*2*], *THEN* &*E*(*2*)]
        ∀*E* **by** *blast*
    AOT-hence ‹p_1›
      using *0*[*THEN actual*[*THEN* ≡$_{df}$*E*], *THEN* &*E*(*2*), *THEN* ∀*E*(*2*), *THEN* →*E*]
      by *blast*
    AOT-thus ‹*p* & ¬*p*› **for** *p* **using** *notp*$_1$ **by** (*metis raa−cor*:*3*)
  qed
next
  AOT-show ‹$\mathbf{s}_V$↓›
    using *df−the−null−sit*:*2 rule−id−df*:*2*:*b*[*zero*] *thm−null−trivial*:*4* **by** *blast*
qed

AOT-theorem *actual−s*:*3*: ‹∃*p*∀*s*(*Actual*(*s*) → ¬*s* ⊨ *p*)›
proof −
  AOT-obtain p_1 **where** *notp*$_1$: ‹¬p_1›
    by (*metis* ∃*E* ∃*I*(*1*) *log−prop−prop*:*2 non−contradiction*)
  AOT-have ‹∀*s* (*Actual*(*s*) → ¬(*s* ⊨ p_1))›
  proof (*rule Situation.GEN*; *rule* →*I*; *rule raa−cor*:*2*)
    fix *s*
    AOT-assume ‹*Actual*(*s*)›
    moreover AOT-assume ‹*s* ⊨ p_1›
    ultimately AOT-have ‹p_1›
      using *actual*[*THEN* ≡$_{df}$*E*, *THEN* &*E*(*2*), *THEN* ∀*E*(*2*), *THEN* →*E*] **by** *blast*
    AOT-thus ‹p_1 & ¬p_1›
      using *notp*$_1$ &*I* **by** *simp*
  qed
  thus *?thesis* **by** (*rule* ∃*I*)
qed

AOT-theorem *comp*:
  ‹∃*s* (*s*′ ⊴ *s* & *s*′′ ⊴ *s* & ∀*s*′′′ (*s*′ ⊴ *s*′′′ & *s*′′ ⊴ *s*′′′ → *s* ⊴ *s*′′′))›
proof −
  have *cond-prop*: ‹*ConditionOnPropositionalProperties* (λ Π . «*s*′[Π] ∨ *s*′′[Π]»)›
  proof(*safe intro*!: *cond−prop*[*I*] *GEN oth−class−taut*:*8*:*c*[*THEN* →*E*, *THEN* →*E*];
      rule →*I*)
    AOT-modally-strict {
      fix *F*
      AOT-have ‹*Situation*(*s*′)›
        by (*simp add*: *Situation.restricted-var-condition*)
      AOT-hence ‹*s*′[*F*] → *Propositional*([*F*])›
        using *situations*[*THEN* ≡$_{df}$*E*, *THEN* &*E*(*2*), *THEN* ∀*E*(*2*)] **by** *blast*
      moreover AOT-assume ‹*s*′[*F*]›
      ultimately AOT-show ‹*Propositional*([*F*])›
        using →*E* **by** *blast*
    }
  next
    AOT-modally-strict {
      fix *F*
      AOT-have ‹*Situation*(*s*′′)›
        by (*simp add*: *Situation.restricted-var-condition*)
      AOT-hence ‹*s*′′[*F*] → *Propositional*([*F*])›
        using *situations*[*THEN* ≡$_{df}$*E*, *THEN* &*E*(*2*), *THEN* ∀*E*(*2*)] **by** *blast*
      moreover AOT-assume ‹*s*′′[*F*]›
      ultimately AOT-show ‹*Propositional*([*F*])›
        using →*E* **by** *blast*
    }

qed
AOT-obtain s_3 **where** ϑ: ‹$\forall F\ (s_3[F] \equiv s'[F] \vee s''[F])$›
using *comp–sit:1*[*OF cond-prop*] *Situation.∃E*[*rotated*] **by** *meson*
AOT-have ‹$s' \trianglelefteq s_3\ \&\ s'' \trianglelefteq s_3\ \&\ \forall s'''\ (s' \trianglelefteq s'''\ \&\ s'' \trianglelefteq s''' \rightarrow s_3 \trianglelefteq s''')$›
proof(*safe intro*!: &*I* $\equiv_{df}$*I*[*OF true–in–s*] $\equiv_{df}$*I*[*OF prop–enc*]
Situation.GEN GEN[**where** '*a*=o] →*I*
sit–part–whole[*THEN* $\equiv_{df}$*I*]
Situation.ψ cqt:2[*const-var*][*axiom-inst*])
fix *p*
AOT-assume ‹$s' \models p$›
AOT-hence ‹$s'[\lambda x\ p]$›
by (*metis* &*E*(*2*) *prop–enc* $\equiv_{df}$*E true–in–s*)
AOT-thus ‹$s_3[\lambda x\ p]$›
using ϑ[*THEN* ∀*E*(*1*),*OF prop–prop2:2*, *THEN* ≡*E*(*2*), *OF* ∨*I*(*1*)] **by** *blast*
next
fix *p*
AOT-assume ‹$s'' \models p$›
AOT-hence ‹$s''[\lambda x\ p]$›
by (*metis* &*E*(*2*) *prop–enc* $\equiv_{df}$*E true–in–s*)
AOT-thus ‹$s_3[\lambda x\ p]$›
using ϑ[*THEN* ∀*E*(*1*),*OF prop–prop2:2*, *THEN* ≡*E*(*2*), *OF* ∨*I*(*2*)] **by** *blast*
next
fix *s p*
AOT-assume *0*: ‹$s' \trianglelefteq s\ \&\ s'' \trianglelefteq s$›
AOT-assume ‹$s_3 \models p$›
AOT-hence ‹$s_3[\lambda x\ p]$›
by (*metis* &*E*(*2*) *prop–enc* $\equiv_{df}$*E true–in–s*)
AOT-hence ‹$s'[\lambda x\ p] \vee s''[\lambda x\ p]$›
using ϑ[*THEN* ∀*E*(*1*),*OF prop–prop2:2*, *THEN* ≡*E*(*1*)] **by** *blast*
moreover {
AOT-assume ‹$s'[\lambda x\ p]$›
AOT-hence ‹$s' \models p$›
by (*safe intro*!: *prop–enc*[*THEN* $\equiv_{df}$*I*] *true–in–s*[*THEN* $\equiv_{df}$*I*] &*I*
Situation.ψ cqt:2[*const-var*][*axiom-inst*])
moreover AOT-have ‹$s' \models p \rightarrow s \models p$›
using *sit–part–whole*[*THEN* $\equiv_{df}$*E*, *THEN* &*E*(*2*)] *0*[*THEN* &*E*(*1*)]
∀*E*(*2*) **by** *blast*
ultimately AOT-have ‹$s \models p$›
using →*E* **by** *blast*
AOT-hence ‹$s[\lambda x\ p]$›
using *true–in–s*[*THEN* $\equiv_{df}$*E*] *prop–enc*[*THEN* $\equiv_{df}$*E*] &*E* **by** *blast*
}
moreover {
AOT-assume ‹$s''[\lambda x\ p]$›
AOT-hence ‹$s'' \models p$›
by (*safe intro*!: *prop–enc*[*THEN* $\equiv_{df}$*I*] *true–in–s*[*THEN* $\equiv_{df}$*I*] &*I*
Situation.ψ cqt:2[*const-var*][*axiom-inst*])
moreover AOT-have ‹$s'' \models p \rightarrow s \models p$›
using *sit–part–whole*[*THEN* $\equiv_{df}$*E*, *THEN* &*E*(*2*)] *0*[*THEN* &*E*(*2*)]
∀*E*(*2*) **by** *blast*
ultimately AOT-have ‹$s \models p$›
using →*E* **by** *blast*
AOT-hence ‹$s[\lambda x\ p]$›
using *true–in–s*[*THEN* $\equiv_{df}$*E*] *prop–enc*[*THEN* $\equiv_{df}$*E*] &*E* **by** *blast*
}
ultimately AOT-show ‹$s[\lambda x\ p]$›
by (*metis* ∨*E*(*1*) →*I*)
qed
thus *?thesis*
using *Situation.∃I* **by** *fast*
qed

AOT-theorem *act–sit:1*: ‹$Actual(s) \rightarrow (s \models p \rightarrow [\lambda y\ p]s)$›

proof (*safe intro*!: →*I*)
  AOT-assume ‹*Actual*(*s*)›
  AOT-hence *p* **if** ‹*s* ⊨ *p*›
    using *actual*[*THEN* $\equiv_{df}E$, *THEN* &*E*(*2*), *THEN* ∀*E*(*2*), *THEN* →*E*] *that* **by** *blast*
  moreover AOT-assume ‹*s* ⊨ *p*›
  ultimately AOT-have *p* **by** *blast*
  AOT-thus ‹[λ*y p*]*s*›
    by (*safe intro*!: β←*C*(*1*) *cqt*:*2*)
qed

AOT-theorem *act−sit*:*2*:
  ‹(*Actual*(*s*′) & *Actual*(*s*″)) → ∃*x* (*Actual*(*x*) & *s*′ ⊴ *x* & *s*″ ⊴ *x*)›
proof(*rule* →*I*; *frule* &*E*(*1*); *drule* &*E*(*2*))
  AOT-assume *act-s*′: ‹*Actual*(*s*′)›
  AOT-assume *act-s*″: ‹*Actual*(*s*″)›
  have *cond−prop*: ‹*ConditionOnPropositionalProperties*
                      (λ Π . «∃*p* (Π = [λ*y p*] & (*s*′ ⊨ *p* ∨ *s*″ ⊨ *p*))»)›
  proof (*safe intro*!: *cond−prop*[*I*] ∀*I* →*I prop−prop1*[*THEN* $\equiv_{df}I$])
    AOT-modally-strict {
      fix β
      AOT-assume ‹∃*p* (β = [λ*y p*] & (*s*′ ⊨ *p* ∨ *s*″ ⊨ *p*))›
      then AOT-obtain *p* **where** ‹β = [λ*y p*]› **using** ∃*E*[*rotated*] &*E* **by** *blast*
      AOT-thus ‹∃*p* β = [λ*y p*]› **by** (*rule* ∃*I*)
    }
  qed
  have *rigid*: ‹*rigid-condition* (λ Π . «∃*p* (Π = [λ*y p*] & (*s*′ ⊨ *p* ∨ *s*″ ⊨ *p*))»)›
  proof(*safe intro*!: *strict−can*:*1*[*I*] →*I GEN*)
    AOT-modally-strict {
      fix *F*
      AOT-assume ‹∃*p* (*F* = [λ*y p*] & (*s*′ ⊨ *p* ∨ *s*″ ⊨ *p*))›
      then AOT-obtain p_1 **where** p_1-*prop*: ‹*F* = [λ*y* p_1] & (*s*′ ⊨ p_1 ∨ *s*″ ⊨ p_1)›
        using ∃*E*[*rotated*] **by** *blast*
      AOT-hence ‹□(*F* = [λ*y* p_1])›
        using &*E*(*1*) *id−nec*:*2 vdash−properties*:*10* **by** *blast*
      moreover AOT-have ‹□(*s*′ ⊨ p_1 ∨ *s*″ ⊨ p_1)›
      proof(*rule* ∨*E*; (*rule* →*I*; *rule KBasic*:*15*[*THEN* →*E*])*?*)
        AOT-show ‹*s*′ ⊨ p_1 ∨ *s*″ ⊨ p_1› **using** p_1-*prop* &*E* **by** *blast*
      next
        AOT-show ‹□*s*′ ⊨ p_1 ∨ □*s*″ ⊨ p_1› **if** ‹*s*′ ⊨ p_1›
          apply (*rule* ∨*I*(*1*))
          using $\equiv_{df}E$ &*E*(*1*) ≡*E*(*1*) *lem2*:*1 that true−in−s* **by** *blast*
      next
        AOT-show ‹□*s*′ ⊨ p_1 ∨ □*s*″ ⊨ p_1› **if** ‹*s*″ ⊨ p_1›
          apply (*rule* ∨*I*(*2*))
          using $\equiv_{df}E$ &*E*(*1*) ≡*E*(*1*) *lem2*:*1 that true−in−s* **by** *blast*
      qed
      ultimately AOT-have ‹□(*F* = [λ*y* p_1] & (*s*′ ⊨ p_1 ∨ *s*″ ⊨ p_1))›
        by (*metis KBasic*:*3* &*I* ≡*E*(*2*))
      AOT-hence ‹∃*p* □(*F* = [λ*y p*] & (*s*′ ⊨ *p* ∨ *s*″ ⊨ *p*))› **by** (*rule* ∃*I*)
      AOT-thus ‹□∃*p* (*F* = [λ*y p*] & (*s*′ ⊨ *p* ∨ *s*″ ⊨ *p*))›
        using *Buridan*[*THEN* →*E*] **by** *fast*
    }
  qed

  AOT-have *desc-den*: ‹ι*s*(∀*F* (*s*[*F*] ≡ ∃*p* (*F* = [λ*y p*] & (*s*′ ⊨ *p* ∨ *s*″ ⊨ *p*))))↓›
    by (*rule can−sit−desc*:*1*[*OF cond−prop*])
  AOT-obtain x_0
    where x_0-*prop1*: ‹x_0 = ι*s*(∀*F* (*s*[*F*] ≡ ∃*p* (*F* = [λ*y p*] & (*s*′ ⊨ *p* ∨ *s*″ ⊨ *p*))))›
    by (*metis* (*no-types*, *lifting*) ∃*E rule*=*I*:*1 desc-den* ∃*I*(*1*) *id-sym*)
  AOT-hence x_0-*sit*: ‹*Situation*(x_0)›
    using *actual−desc*:*3*[*THEN* →*E*] *Act−Basic*:*2* &*E*(*1*) ≡*E*(*1*)
          possit−sit:*4* **by** *blast*

AOT-have *1*: ‹$\forall F\ (x_0[F] \equiv \exists p\ (F = [\lambda y\ p]\ \&\ (s' \models p \vee s'' \models p)))$›
using *strict−sit*[*OF rigid*, *OF cond−prop*, *THEN* →*E*, *OF* x_0*-prop1*].
AOT-have *2*: ‹$(x_0 \models p) \equiv (s' \models p \vee s'' \models p)$› **for** *p*
proof (*rule* ≡*I*; *rule* →*I*)
AOT-assume ‹$x_0 \models p$›
AOT-hence ‹$x_0[\lambda y\ p]$› **using** *lem1*[*THEN* →*E*, *OF* x_0*-sit*, *THEN* ≡*E*(*1*)] **by** *blast*
then AOT-obtain *q* **where** ‹$[\lambda y\ p] = [\lambda y\ q]\ \&\ (s' \models q \vee s'' \models q)$›
using *1*[*THEN* ∀*E*(*1*)[**where** τ=«[λy p]»], *OF prop−prop2*:*2*, *THEN* ≡*E*(*1*)]
∃*E*[*rotated*] **by** *blast*
AOT-thus ‹$s' \models p \vee s'' \models p$›
by (*metis rule*=*E* &*E*(*1*) &*E*(*2*) ∨*I*(*1*) ∨*I*(*2*)
∨*E*(*1*) *deduction−theorem id-sym* ≡*E*(*2*) *p−identity−thm2*:*3*)
next
AOT-assume ‹$s' \models p \vee s'' \models p$›
AOT-hence ‹$[\lambda y\ p] = [\lambda y\ p]\ \&\ (s' \models p \vee s'' \models p)$›
by (*metis rule*=*I*:*1* &*I prop−prop2*:*2*)
AOT-hence ‹$\exists q\ ([\lambda y\ p] = [\lambda y\ q]\ \&\ (s' \models q \vee s'' \models q))$›
by (*rule* ∃*I*)
AOT-hence ‹$x_0[\lambda y\ p]$›
using *1*[*THEN* ∀*E*(*1*), *OF prop−prop2*:*2*, *THEN* ≡*E*(*2*)] **by** *blast*
AOT-thus ‹$x_0 \models p$›
by (*metis* $\equiv_{df}$*I* &*I ex*:*1*:*a prop−enc rule−ui*:*2*[*const-var*]
x_0*-sit true−in−s*)
qed

AOT-have ‹$Actual(x_0)\ \&\ s' \trianglelefteq x_0\ \&\ s'' \trianglelefteq x_0$›
proof(*safe intro*!: →*I* &*I* ∃*I*(*1*) *actual*[*THEN* $\equiv_{df}$*I*] x_0*-sit GEN*
sit−part−whole[*THEN* $\equiv_{df}$*I*])
fix *p*
AOT-assume ‹$x_0 \models p$›
AOT-hence ‹$s' \models p \vee s'' \models p$›
using *2* ≡*E*(*1*) **by** *metis*
AOT-thus ‹*p*›
using *act-s′ act-s″*
actual[*THEN* $\equiv_{df}$*E*, *THEN* &*E*(*2*), *THEN* ∀*E*(*2*), *THEN* →*E*]
by (*metis* ∨*E*(*3*) *reductio−aa*:*1*)
next
AOT-show ‹$x_0 \models p$› **if** ‹$s' \models p$› **for** *p*
using *2*[*THEN* ≡*E*(*2*), *OF* ∨*I*(*1*), *OF that*].
next
AOT-show ‹$x_0 \models p$› **if** ‹$s'' \models p$› **for** *p*
using *2*[*THEN* ≡*E*(*2*), *OF* ∨*I*(*2*), *OF that*].
next
AOT-show ‹$Situation(s')$›
using *act-s′*[*THEN actual*[*THEN* $\equiv_{df}$*E*]] &*E* **by** *blast*
next
AOT-show ‹$Situation(s'')$›
using *act-s″*[*THEN actual*[*THEN* $\equiv_{df}$*E*]] &*E* **by** *blast*
qed
AOT-thus ‹$\exists x\ (Actual(x)\ \&\ s' \trianglelefteq x\ \&\ s'' \trianglelefteq x)$›
by (*rule* ∃*I*)
qed

AOT-define *Consistent* :: ‹τ ⇒ φ› (‹*Consistent*′(-′)›)
cons: ‹$Consistent(s) \equiv_{df} \neg\exists p\ (s \models p\ \&\ s \models \neg p)$›

AOT-theorem *sit−cons*: ‹$Actual(s) \rightarrow Consistent(s)$›
proof(*safe intro*!: →*I cons*[*THEN* $\equiv_{df}$*I*] &*I Situation.ψ*
dest!: *actual*[*THEN* $\equiv_{df}$*E*]; *frule* &*E*(*1*); *drule* &*E*(*2*))
AOT-assume *0*: ‹$\forall p\ (s \models p \rightarrow p)$›
AOT-show ‹$\neg\exists p\ (s \models p\ \&\ s \models \neg p)$›
proof (*rule raa−cor*:*2*)
AOT-assume ‹$\exists p\ (s \models p\ \&\ s \models \neg p)$›

```
    then AOT-obtain p where ‹s ⊨ p & s ⊨ ¬p›
      using ∃ E[rotated] by blast
    AOT-hence ‹p & ¬p›
      using 0[THEN ∀ E(1)[where τ=‹«¬p»›, THEN →E], OF log−prop−prop:2]
            0[THEN ∀ E(2)[where β=p], THEN →E] &E &I by blast
    AOT-thus ‹p & ¬p› for p by (metis raa−cor:1)
  qed
qed

AOT-theorem cons−rigid:1: ‹¬Consistent(s) ≡ □¬Consistent(s)›
proof (rule ≡I; rule →I)
  AOT-assume ‹¬Consistent(s)›
  AOT-hence ‹∃ p (s ⊨ p & s ⊨ ¬p)›
    using cons[THEN ≡df I, OF &I, OF Situation.ψ]
    by (metis raa−cor:3)
  then AOT-obtain p where p-prop: ‹s ⊨ p & s ⊨ ¬p›
    using ∃ E[rotated] by blast
  AOT-hence ‹□s ⊨ p›
    using &E(1) ≡E(1) lem2:1 by blast
  moreover AOT-have ‹□s ⊨ ¬p›
    using p-prop T◇ &E ≡E(1)
      modus−tollens:1 raa−cor:3 lem2:3[unvarify p]
      log−prop−prop:2 by metis
  ultimately AOT-have ‹□(s ⊨ p & s ⊨ ¬p)›
    by (metis KBasic:3 &I ≡E(2))
  AOT-hence ‹∃ p □(s ⊨ p & s ⊨ ¬p)›
    by (rule ∃ I)
  AOT-hence ‹□∃ p(s ⊨ p & s ⊨ ¬p)›
    by (metis Buridan vdash−properties:10)
  AOT-thus ‹□¬Consistent(s)›
    apply (rule qml:1[axiom-inst, THEN →E, THEN →E, rotated])
    apply (rule RN)
    using ≡df E &E(2) cons deduction−theorem raa−cor:3 by blast
next
  AOT-assume ‹□¬Consistent(s)›
  AOT-thus ‹¬Consistent(s)› using qml:2[axiom-inst, THEN →E] by auto
qed

AOT-theorem cons−rigid:2: ‹◇Consistent(x) ≡ Consistent(x)›
proof(rule ≡I; rule →I)
  AOT-assume 0: ‹◇Consistent(x)›
  AOT-have ‹◇(Situation(x) & ¬∃ p (x ⊨ p & x ⊨ ¬p))›
    apply (AOT-subst ‹Situation(x) & ¬∃ p (x ⊨ p & x ⊨ ¬p)› ‹Consistent(x)›)
     using cons ≡E(2) Commutativity of ≡ ≡Df apply blast
    by (simp add: 0)
  AOT-hence ‹◇Situation(x)› and 1: ‹◇¬∃ p (x ⊨ p & x ⊨ ¬p)›
    using RM◇ Conjunction Simplification(1) Conjunction Simplification(2)
          modus−tollens:1 raa−cor:3 by blast+
  AOT-hence 2: ‹Situation(x)› by (metis ≡E(1) possit−sit:2)
  AOT-have 3: ‹¬□∃ p (x ⊨ p & x ⊨ ¬p)›
    using 2 using 1 KBasic:11 ≡E(2) by blast
  AOT-show ‹Consistent(x)›
  proof (rule raa−cor:1)
    AOT-assume ‹¬Consistent(x)›
    AOT-hence ‹∃ p (x ⊨ p & x ⊨ ¬p)›
      using 0 ≡df E conventions:5 2 cons−rigid:1[unconstrain s, THEN →E]
            modus−tollens:1 raa−cor:3 ≡E(4) by meson
    then AOT-obtain p where ‹x ⊨ p› and 4: ‹x ⊨ ¬p›
      using ∃ E[rotated] &E by blast
    AOT-hence ‹□x ⊨ p›
      by (metis 2 ≡E(1) lem2:1[unconstrain s, THEN →E])
    moreover AOT-have ‹□x ⊨ ¬p›
      using 4 lem2:1[unconstrain s, unvarify p, THEN →E]
```

```
      by (metis 2 ≡E(1) log−prop−prop:2)
    ultimately AOT-have ‹□(x ⊨ p & x ⊨ ¬p)›
      by (metis KBasic:3 &I ≡E(3) raa−cor:3)
    AOT-hence ‹∃ p □(x ⊨ p & x ⊨ ¬p)›
      by (metis existential:1 log−prop−prop:2)
    AOT-hence ‹□∃ p (x ⊨ p & x ⊨ ¬p)›
      by (metis Buridan vdash−properties:10)
    AOT-thus ‹p & ¬p› for p
      using 3 &I by (metis raa−cor:3)
  qed
next
  AOT-show ‹◇Consistent(x)› if ‹Consistent(x)›
    using T◇ that vdash−properties:10 by blast
qed

AOT-define possible :: ‹τ ⇒ φ› (‹Possible'(-')›)
  pos: ‹Possible(s) ≡df ◇Actual(s)›

AOT-theorem sit−pos:1: ‹Actual(s) → Possible(s)›
  apply(rule →I; rule pos[THEN ≡dfI]; rule &I)
  apply (meson ≡dfE actual &E(1))
  using T◇ vdash−properties:10 by blast

AOT-theorem sit−pos:2: ‹∃ p ((s ⊨ p) & ¬◇p) → ¬Possible(s)›
proof(rule →I)
  AOT-assume ‹∃ p ((s ⊨ p) & ¬◇p)›
  then AOT-obtain p where a: ‹(s ⊨ p) & ¬◇p›
    using ∃ E[rotated] by blast
  AOT-hence ‹□(s ⊨ p)›
    using &E by (metis T◇ ≡E(1) lem2:3 vdash−properties:10)
  moreover AOT-have ‹□¬p›
    using a[THEN &E(2)] by (metis KBasic2:1 ≡E(2))
  ultimately AOT-have ‹□(s ⊨ p & ¬p)›
    by (metis KBasic:3 &I ≡E(3) raa−cor:3)
  AOT-hence ‹∃ p □(s ⊨ p & ¬p)›
    by (rule ∃ I)
  AOT-hence 1: ‹□∃ q (s ⊨ q & ¬q)›
    by (metis Buridan vdash−properties:10)
  AOT-have ‹□¬∀ q (s ⊨ q → q)›
    apply (AOT-subst ‹s ⊨ q → q› ‹¬(s ⊨ q & ¬q)› for: q)
     apply (simp add: oth−class−taut:1:a)
    apply (AOT-subst ‹¬∀ q ¬(s ⊨ q & ¬q)› ‹∃ q (s ⊨ q & ¬q)›)
    by (auto simp: conventions:4 df−rules−formulas[3] df−rules−formulas[4] ≡I 1)
  AOT-hence 0: ‹¬◇∀ q (s ⊨ q → q)›
    by (metis ≡dfE conventions:5 raa−cor:3)
  AOT-show ‹¬Possible(s)›
    apply (AOT-subst ‹Possible(s)› ‹Situation(s) & ◇Actual(s)›)
     apply (simp add: pos ≡Df)
    apply (AOT-subst ‹Actual(s)› ‹Situation(s) & ∀ q (s ⊨ q → q)›)
     using actual ≡Df apply presburger
    by (metis 0 KBasic2:3 &E(2) raa−cor:3 vdash−properties:10)
qed

AOT-theorem pos−cons−sit:1: ‹Possible(s) → Consistent(s)›
  by (auto simp: sit−cons[THEN RM◇, THEN →E,
                          THEN cons−rigid:2[THEN ≡E(1)]]
          intro!: →I dest!: pos[THEN ≡dfE] &E(2))

AOT-theorem pos−cons−sit:2: ‹∃ s (Consistent(s) & ¬Possible(s))›
proof −
  AOT-obtain q1 where ‹q1 & ◇¬q1›
    using ≡dfE instantiation cont−tf:1 cont−tf−thm:1 by blast
  have cond−prop: ‹ConditionOnPropositionalProperties
```

```
                 (λ Π . «Π = [λy q1 & ¬q1]»)›
  by (auto intro!: cond−prop[I] GEN →I prop−prop1[THEN ≡dfI]
                 ∃ I(1)[where τ=‹«q1 & ¬q1»›, rotated, OF log−prop−prop:2])
have rigid: ‹rigid-condition (λ Π . «Π = [λy q1 & ¬q1]»)›
  by (auto intro!: strict−can:1[I] GEN →I simp: id−nec:2[THEN →E])

AOT-obtain x where x-prop: ‹x = ιs (∀ F (s[F] ≡ F = [λy q1 & ¬q1]))›
  using ex:1:b[THEN ∀ E(1), OF can−sit−desc:1, OF cond−prop]
      ∃ E[rotated] by blast
AOT-hence 0: ‹𝓐(Situation(x) & ∀ F (x[F] ≡ F = [λy q1 & ¬q1]))›
  using →E actual−desc:2 by blast
AOT-hence ‹𝓐(Situation(x))› by (metis Act−Basic:2 &E(1) ≡E(1))
AOT-hence s-sit: ‹Situation(x)› by (metis ≡E(1) possit−sit:4)
AOT-have s-enc-prop: ‹∀ F (x[F] ≡ F = [λy q1 & ¬q1])›
  using strict−sit[OF rigid, OF cond−prop, THEN →E, OF x-prop].
AOT-hence ‹x[λy q1 & ¬q1]›
  using ∀ E(1)[rotated, OF prop−prop2:2]
      rule=I:1[OF prop−prop2:2] ≡E by blast
AOT-hence ‹x ⊨ (q1 & ¬q1)›
  using lem1[THEN →E, OF s-sit, unvarify p, THEN ≡E(2), OF log−prop−prop:2]
  by blast
AOT-hence ‹□(x ⊨ (q1 & ¬q1))›
  using lem2:1[unconstrain s, THEN →E, OF s-sit, unvarify p,
               OF log−prop−prop:2, THEN ≡E(1)] by blast
moreover AOT-have ‹□(x ⊨ (q1 & ¬q1) → ¬Actual(x))›
proof(rule RN; rule →I; rule raa−cor:2)
  AOT-modally-strict {
    AOT-assume ‹Actual(x)›
    AOT-hence ‹∀ p (x ⊨ p → p)›
      using actual[THEN ≡dfE, THEN &E(2)] by blast
    moreover AOT-assume ‹x ⊨ (q1 & ¬q1)›
    ultimately AOT-show ‹q1 & ¬q1›
      using ∀ E(1)[rotated, OF log−prop−prop:2] →E by metis
  }
qed
ultimately AOT-have nec-not-actual-s: ‹□¬Actual(x)›
  using qml:1[axiom-inst, THEN →E, THEN →E] by blast
AOT-have 1: ‹¬∃ p (x ⊨ p & x ⊨ ¬p)›
proof (rule raa−cor:2)
  AOT-assume ‹∃ p (x ⊨ p & x ⊨ ¬p)›
  then AOT-obtain p where ‹x ⊨ p & x ⊨ ¬p›
    using ∃ E[rotated] by blast
  AOT-hence ‹x[λy p] & x[λy ¬p]›
    using lem1[unvarify p, THEN →E, OF log−prop−prop:2,
             OF s-sit, THEN ≡E(1)] &I &E by metis
  AOT-hence ‹[λy p] = [λy q1 & ¬q1]› and ‹[λy ¬p] = [λy q1 & ¬q1]›
    by (auto intro!: prop−prop2:2 s-enc-prop[THEN ∀ E(1), THEN ≡E(1)]
           elim: &E)
  AOT-hence i: ‹[λy p] = [λy ¬p]› by (metis rule=E id-sym)
  {
    AOT-assume 0: ‹p›
    AOT-have ‹[λy p]x› for x
      by (auto intro!: β←C(1) cqt:2 0)
    AOT-hence ‹[λy ¬p]x› for x using i rule=E by fast
    AOT-hence ‹¬p›
      using β→C(1) by auto
  }
  moreover {
    AOT-assume 0: ‹¬p›
    AOT-have ‹[λy ¬p]x› for x
      by (auto intro!: β←C(1) cqt:2 0)
    AOT-hence ‹[λy p]x› for x using i[symmetric] rule=E by fast
    AOT-hence ‹p›
```

```
      using β→C(1) by auto
    }
    ultimately AOT-show ‹p & ¬p› for p by (metis raa−cor:1 raa−cor:3)
  qed
  AOT-have 2: ‹¬Possible(x)›
  proof(rule raa−cor:2)
    AOT-assume ‹Possible(x)›
    AOT-hence ‹◇Actual(x)›
      by (metis ≡df E &E(2) pos)
    moreover AOT-have ‹¬◇Actual(x)› using nec-not-actual-s
      using ≡df E conventions:5 reductio−aa:2 by blast
    ultimately AOT-show ‹◇Actual(x) & ¬◇Actual(x)› by (rule &I)
  qed
  show ?thesis
    by(rule ∃I(2)[where β=x]; safe intro!: &I 2 s-sit cons[THEN ≡df I] 1)
qed

AOT-theorem sit−classical:1: ‹∀p (s |= p ≡ p) → ∀q(s |= ¬q ≡ ¬s |= q)›
proof(rule →I; rule GEN)
  fix q
  AOT-assume ‹∀p (s |= p ≡ p)›
  AOT-hence ‹s |= q ≡ q› and ‹s |= ¬q ≡ ¬q›
    using ∀E(1)[rotated, OF log−prop−prop:2] by blast+
  AOT-thus ‹s |= ¬q ≡ ¬s |= q›
    by (metis deduction−theorem ≡I ≡E(1) ≡E(2) ≡E(4))
qed

AOT-theorem sit−classical:2:
  ‹∀p (s |= p ≡ p) → ∀q∀r((s |= (q → r)) ≡ (s |= q → s |= r))›
proof(rule →I; rule GEN; rule GEN)
  fix q r
  AOT-assume ‹∀p (s |= p ≡ p)›
  AOT-hence ϑ: ‹s |= q ≡ q› and ξ: ‹s |= r ≡ r› and ζ: ‹(s |= (q → r)) ≡ (q → r)›
    using ∀E(1)[rotated, OF log−prop−prop:2] by blast+
  AOT-show ‹(s |= (q → r)) ≡ (s |= q → s |= r)›
  proof (safe intro!: ≡I →I)
    AOT-assume ‹s |= (q → r)›
    moreover AOT-assume ‹s |= q›
    ultimately AOT-show ‹s |= r›
      using ϑ ξ ζ by (metis ≡E(1) ≡E(2) vdash−properties:10)
  next
    AOT-assume ‹s |= q → s |= r›
    AOT-thus ‹s |= (q → r)›
      using ϑ ξ ζ by (metis deduction−theorem ≡E(1) ≡E(2) →E)
  qed
qed

AOT-theorem sit−classical:3:
  ‹∀p (s |= p ≡ p) → ((s |= ∀α φ{α}) ≡ ∀α s |= φ{α})›
proof (rule →I)
  AOT-assume ‹∀p (s |= p ≡ p)›
  AOT-hence ϑ: ‹s |= φ{α} ≡ φ{α}› and ξ: ‹s |= ∀α φ{α} ≡ ∀α φ{α}› for α
    using ∀E(1)[rotated, OF log−prop−prop:2] by blast+
  AOT-show ‹s |= ∀α φ{α} ≡ ∀α s |= φ{α}›
  proof (safe intro!: ≡I →I GEN)
    fix α
    AOT-assume ‹s |= ∀α φ{α}›
    AOT-hence ‹φ{α}› using ξ ∀E(2) ≡E(1) by blast
    AOT-thus ‹s |= φ{α}› using ϑ ≡E(2) by blast
  next
    AOT-assume ‹∀α s |= φ{α}›
    AOT-hence ‹s |= φ{α}› for α using ∀E(2) by blast
    AOT-hence ‹φ{α}› for α using ϑ ≡E(1) by blast
```

```
    AOT-hence ‹∀ α φ{α}› by (rule GEN)
    AOT-thus ‹s ⊨ ∀ α φ{α}› using ξ ≡E(2) by blast
  qed
qed

AOT-theorem sit-classical:4: ‹∀ p (s ⊨ p ≡ p) → ∀ q (s ⊨ □q → □s ⊨ q)›
proof(rule →I; rule GEN; rule →I)
  fix q
  AOT-assume ‹∀ p (s ⊨ p ≡ p)›
  AOT-hence ϑ: ‹s ⊨ q ≡ q› and ξ: ‹s ⊨ □q ≡ □q›
    using ∀ E(1)[rotated, OF log-prop-prop:2] by blast+
  AOT-assume ‹s ⊨ □q›
  AOT-hence ‹□q› using ξ ≡E(1) by blast
  AOT-hence ‹q› using qml:2[axiom-inst, THEN →E] by blast
  AOT-hence ‹s ⊨ q› using ϑ ≡E(2) by blast
  AOT-thus ‹□s ⊨ q› using ≡df E &E(1) ≡E(1) lem2:1 true-in-s by blast
qed

AOT-theorem sit-classical:5:
  ‹∀ p (s ⊨ p ≡ p) → ∃ q(□(s ⊨ q) & ¬(s ⊨ □ q))›
proof (rule →I)
  AOT-obtain r where A: ‹r› and ‹◇¬r›
    by (metis &E(1) &E(2) ≡df E instantiation cont-tf:1 cont-tf-thm:1)
  AOT-hence B: ‹¬□r›
    using KBasic:11 ≡E(2) by blast
  moreover AOT-assume asm: ‹∀  p (s ⊨ p ≡ p)›
  AOT-hence ‹s ⊨ r›
    using ∀ E(2) A ≡E(2) by blast
  AOT-hence 1: ‹□s ⊨ r›
    using ≡df E &E(1) ≡E(1) lem2:1 true-in-s by blast
  AOT-have ‹s ⊨ ¬□r›
    using asm[THEN ∀ E(1)[rotated, OF log-prop-prop:2], THEN ≡E(2)] B by blast
  AOT-hence ‹¬s ⊨ □r›
    using sit-classical:1[THEN →E, OF asm,
              THEN ∀ E(1)[rotated, OF log-prop-prop:2], THEN ≡E(1)] by blast
  AOT-hence ‹□s ⊨ r & ¬s ⊨ □r›
    using 1 &I by blast
  AOT-thus ‹∃ r (□s ⊨ r & ¬s ⊨ □r)›
    by (rule ∃ I)
qed

AOT-theorem sit-classical:6:
  ‹∃ s ∀ p (s ⊨ p ≡ p)›
proof -
  have cond-prop: ‹ConditionOnPropositionalProperties
                      (λ Π . «∃ q (q & Π = [λy q])»)›
  proof (safe intro!: cond-prop[I] GEN →I)
    fix F
    AOT-modally-strict {
      AOT-assume ‹∃ q (q & F = [λy q])›
      then AOT-obtain q where ‹q & F = [λy q]›
        using ∃ E[rotated] by blast
      AOT-hence ‹F = [λy q]›
        using &E by blast
      AOT-hence ‹∃ q F = [λy q]›
        by (rule ∃ I)
      AOT-thus ‹Propositional([F])›
        by (metis ≡df I prop-prop1)
    }
  qed
  AOT-have ‹∃ s ∀ F (s[F] ≡ ∃ q (q & F = [λy q]))›
    using comp-sit:1[OF cond-prop].
  then AOT-obtain s0 where s0-prop: ‹∀ F (s0[F] ≡ ∃ q (q & F = [λy q]))›
```

```
    using Situation.∃E[rotated] by meson
  AOT-have ‹∀p (s0 ⊨ p ≡ p)›
  proof(safe intro!: GEN ≡I →I)
    fix p
    AOT-assume ‹s0 ⊨ p›
    AOT-hence ‹s0[λy p]›
      using lem1[THEN →E, OF Situation.ψ, THEN ≡E(1)] by blast
    AOT-hence ‹∃q (q & [λy p] = [λy q])›
      using s0-prop[THEN ∀E(1)[rotated, OF prop−prop2:2], THEN ≡E(1)] by blast
    then AOT-obtain q1 where q1-prop: ‹q1 & [λy p] = [λy q1]›
      using ∃E[rotated] by blast
    AOT-hence ‹p = q1›
      by (metis &E(2) ≡E(2) p−identity−thm2:3)
    AOT-thus ‹p›
      using q1-prop[THEN &E(1)] rule=E id-sym by fast
  next
    fix p
    AOT-assume ‹p›
    moreover AOT-have ‹[λy p] = [λy p]›
      by (simp add: rule=I:1[OF prop−prop2:2])
    ultimately AOT-have ‹p & [λy p] = [λy p]›
      using &I by blast
    AOT-hence ‹∃q (q & [λy p] = [λy q])›
      by (rule ∃I)
    AOT-hence ‹s0[λy p]›
      using s0-prop[THEN ∀E(1)[rotated, OF prop−prop2:2], THEN ≡E(2)] by blast
    AOT-thus ‹s0 ⊨ p›
      using lem1[THEN →E, OF Situation.ψ, THEN ≡E(2)] by blast
  qed
  AOT-hence ‹∀p (s0 ⊨ p ≡ p)›
    using &I by blast
  AOT-thus ‹∃s ∀p (s ⊨ p ≡ p)›
    by (rule Situation.∃I)
qed

AOT-define PossibleWorld :: ‹τ ⇒ φ› (‹PossibleWorld'(-')›)
  world:1: ‹PossibleWorld(x) ≡df Situation(x) & ◇∀p(x ⊨ p ≡ p)›

AOT-theorem world:2: ‹∃x PossibleWorld(x)›
proof −
  AOT-obtain s where s-prop: ‹∀p (s ⊨ p ≡ p)›
    using sit−classical:6 Situation.∃E[rotated] by meson
  AOT-have ‹∀p (s ⊨ p ≡ p)›
  proof(safe intro!: GEN ≡I →I)
    fix p
    AOT-assume ‹s ⊨ p›
    AOT-thus ‹p›
      using s-prop[THEN ∀E(2), THEN ≡E(1)] by blast
  next
    fix p
    AOT-assume ‹p›
    AOT-thus ‹s ⊨ p›
      using s-prop[THEN ∀E(2), THEN ≡E(2)] by blast
  qed
  AOT-hence ‹◇∀p (s ⊨ p ≡ p)›
    by (metis T◇[THEN →E])
  AOT-hence ‹◇∀p (s ⊨ p ≡ p)›
    using s-prop &I by blast
  AOT-hence ‹PossibleWorld(s)›
    using world:1[THEN ≡dfI] Situation.ψ &I by blast
  AOT-thus ‹∃x PossibleWorld(x)›
    by (rule ∃I)
qed
```

AOT-theorem *world*:*3*: ‹*PossibleWorld*(*κ*) → *κ*↓›
proof (*rule* →*I*)
  AOT-assume ‹*PossibleWorld*(*κ*)›
  AOT-hence ‹*Situation*(*κ*)›
    using *world*:*1*[*THEN* $\equiv_{df}E$] &*E* **by** *blast*
  AOT-hence ‹*A*!*κ*›
    by (*metis* $\equiv_{df}E$ &*E*(*1*) *situations*)
  AOT-thus ‹*κ*↓›
    by (*metis russell*−*axiom*[*exe*,*1*].*ψ*-*denotes*-*asm*)
qed

AOT-theorem *rigid*−*pw*:*1*: ‹*PossibleWorld*(*x*) ≡ □*PossibleWorld*(*x*)›
proof(*safe intro*!: ≡*I* →*I*)
  AOT-assume ‹*PossibleWorld*(*x*)›
  AOT-hence ‹*Situation*(*x*) & ◇∀ *p*(*x* ⊨ *p* ≡ *p*)›
    using *world*:*1*[*THEN* $\equiv_{df}E$] **by** *blast*
  AOT-hence ‹□*Situation*(*x*) & □◇∀ *p*(*x* ⊨ *p* ≡ *p*)›
    by (*metis S5Basic*:*1* &*I* &*E*(*1*) &*E*(*2*) ≡*E*(*1*) *possit*−*sit*:*1*)
  AOT-hence *0*: ‹□(*Situation*(*x*) & ◇∀ *p*(*x* ⊨ *p* ≡ *p*))›
    by (*metis KBasic*:*3* ≡*E*(*2*))
  AOT-show ‹□*PossibleWorld*(*x*)›
    by (*AOT-subst* ‹*PossibleWorld*(*x*)› ‹*Situation*(*x*) & ◇∀ *p*(*x* ⊨ *p* ≡ *p*)›)
      (*auto simp*: ≡*Df world*:*1 0*)
next
  AOT-show ‹*PossibleWorld*(*x*)› **if** ‹□*PossibleWorld*(*x*)›
    using *that qml*:*2*[*axiom-inst*, *THEN* →*E*] **by** *blast*
qed

AOT-theorem *rigid*−*pw*:*2*: ‹◇*PossibleWorld*(*x*) ≡ *PossibleWorld*(*x*)›
  using *rigid*−*pw*:*1*
  by (*meson RE*◇ *S5Basic*:*2* ≡*E*(*2*) ≡*E*(*6*) *Commutativity of* ≡)

AOT-theorem *rigid*−*pw*:*3*: ‹◇*PossibleWorld*(*x*) ≡ □*PossibleWorld*(*x*)›
  using *rigid*−*pw*:*1 rigid*−*pw*:*2* **by** (*meson* ≡*E*(*5*))

AOT-theorem *rigid*−*pw*:*4*: ‹***𝒜****PossibleWorld*(*x*) ≡ *PossibleWorld*(*x*)›
  by (*metis Act*−*Sub*:*3* →*I* ≡*I* ≡*E*(*6*) *nec*−*imp*−*act rigid*−*pw*:*1 rigid*−*pw*:*2*)

AOT-register-rigid-restricted-type
  PossibleWorld: ‹*PossibleWorld*(*κ*)›
proof
  AOT-modally-strict {
    AOT-show ‹∃ *x PossibleWorld*(*x*)› **using** *world*:*2*.
  }
next
  AOT-modally-strict {
    AOT-show ‹*PossibleWorld*(*κ*) → *κ*↓› **for** *κ* **using** *world*:*3*.
  }
next
  AOT-modally-strict {
    AOT-show ‹∀ *α*(*PossibleWorld*(*α*) → □*PossibleWorld*(*α*))›
      by (*meson GEN* →*I* ≡*E*(*1*) *rigid*−*pw*:*1*)
  }
qed
AOT-register-variable-names
  PossibleWorld: *w*

AOT-theorem *world*−*pos*: ‹*Possible*(*w*)›
proof (*safe intro*!: $\equiv_{df}E$[*OF world*:*1*, *OF PossibleWorld*.*ψ*, *THEN* &*E*(*1*)]
    pos[*THEN* $\equiv_{df}I$] &*I*)
  AOT-have ‹◇∀ *p* (*w* ⊨ *p* ≡ *p*)›
    using *world*:*1*[*THEN* $\equiv_{df}E$, *OF PossibleWorld*.*ψ*, *THEN* &*E*(*2*)].

AOT-hence ‹◇∀ *p* (*w* ⊨ *p* → *p*)›
proof (*rule RM*◇[*THEN* →*E*, *rotated*]; *safe intro*!: →*I GEN*)
AOT-modally-strict {
fix *p*
AOT-assume ‹∀ *p* (*w* ⊨ *p* ≡ *p*)›
AOT-hence ‹*w* ⊨ *p* ≡ *p*› **using** ∀ *E*(*2*) **by** *blast*
moreover AOT-assume ‹*w* ⊨ *p*›
ultimately AOT-show *p* **using** ≡*E*(*1*) **by** *blast*
}
qed
AOT-hence *0*: ‹◇(*Situation*(*w*) & ∀ *p* (*w* ⊨ *p* → *p*))›
using *world*:*1*[*THEN* $\equiv_{df}$*E*, *OF PossibleWorld*.*ψ*, *THEN* &*E*(*1*),
THEN possit−sit:*1*[*THEN* ≡*E*(*1*)]]
by (*metis KBasic*:*16* &*I vdash−properties*:*10*)
AOT-show ‹◇*Actual*(*w*)›
by (*AOT-subst* ‹*Actual*(*w*)› ‹*Situation*(*w*) & ∀ *p* (*w* ⊨ *p* → *p*)›)
(*auto simp*: *actual* ≡*Df 0*)
qed

AOT-theorem *world−cons*:*1*: ‹*Consistent*(*w*)›
using *world−pos*
using *pos−cons−sit*:*1*[*unconstrain s*, *THEN* →*E*, *THEN* →*E*]
by (*meson* $\equiv_{df}$*E* &*E*(*1*) *pos*)

AOT-theorem *world−cons*:*2*: ‹¬ *TrivialSituation*(*w*)›
proof(*rule raa−cor*:*2*)
AOT-assume ‹ *TrivialSituation*(*w*)›
AOT-hence ‹*Situation*(*w*) & ∀ *p w* ⊨ *p*›
using *df−null−trivial*:*2*[*THEN* $\equiv_{df}$*E*] **by** *blast*
AOT-hence *0*: ‹□*w* ⊨ (∃ *p* (*p* & ¬*p*))›
using &*E*
by (*metis Buridan*◇ *T*◇ &*E*(*2*) ≡*E*(*1*) *lem2*:*3*[*unconstrain s*, *THEN* →*E*]
log−prop−prop:*2 rule−ui*:*1 universal−cor* →*E*)
AOT-have ‹◇∀ *p* (*w* ⊨ *p* ≡ *p*)›
using *PossibleWorld*.*ψ world*:*1*[*THEN* $\equiv_{df}$*E*, *THEN* &*E*(*2*)] **by** *metis*
AOT-hence ‹∀ *p* ◇(*w* ⊨ *p* ≡ *p*)›
using *Buridan*◇[*THEN* →*E*] **by** *blast*
AOT-hence ‹◇(*w* ⊨ (∃ *p* (*p* & ¬*p*)) ≡ (∃ *p* (*p* & ¬*p*)))›
by (*metis log−prop−prop*:*2 rule−ui*:*1*)
AOT-hence ‹◇(*w* ⊨ (∃ *p* (*p* & ¬*p*)) → (∃ *p* (*p* & ¬*p*)))›
using *RM*◇[*THEN* →*E*] →*I* ≡*E*(*1*) **by** *meson*
AOT-hence ‹◇(∃ *p* (*p* & ¬*p*))› **using** *0*
by (*metis KBasic2*:*4* ≡*E*(*1*) →*E*)
moreover AOT-have ‹¬◇(∃ *p* (*p* & ¬*p*))›
by (*metis instantiation KBasic2*:*1 RN* ≡*E*(*1*) *raa−cor*:*2*)
ultimately AOT-show ‹◇(∃ *p* (*p* & ¬*p*)) & ¬◇(∃ *p* (*p* & ¬*p*))›
using &*I* **by** *blast*
qed

AOT-theorem *rigid−truth−at*:*1*: ‹*w* ⊨ *p* ≡ □*w* ⊨ *p*›
using *lem2*:*1*[*unconstrain s*, *THEN* →*E*,
OF PossibleWorld.*ψ*[*THEN world*:*1*[*THEN* $\equiv_{df}$*E*], *THEN* &*E*(*1*)]].

AOT-theorem *rigid−truth−at*:*2*: ‹◇*w* ⊨ *p* ≡ *w* ⊨ *p*›
using *lem2*:*2*[*unconstrain s*, *THEN* →*E*,
OF PossibleWorld.*ψ*[*THEN world*:*1*[*THEN* $\equiv_{df}$*E*], *THEN* &*E*(*1*)]].

AOT-theorem *rigid−truth−at*:*3*: ‹◇*w* ⊨ *p* ≡ □*w* ⊨ *p*›
using *lem2*:*3*[*unconstrain s*, *THEN* →*E*,
OF PossibleWorld.*ψ*[*THEN world*:*1*[*THEN* $\equiv_{df}$*E*], *THEN* &*E*(*1*)]].

AOT-theorem *rigid−truth−at*:*4*: ‹$\mathcal{A}$*w* ⊨ *p* ≡ *w* ⊨ *p*›
using *lem2*:*4*[*unconstrain s*, *THEN* →*E*,

```
                OF PossibleWorld.ψ[THEN world:1[THEN ≡dfE], THEN &E(1)]].

AOT-theorem rigid−truth−at:5: ‹¬w ⊨ p ≡ □¬w ⊨ p›
  using lem2:5[unconstrain s, THEN →E,
                OF PossibleWorld.ψ[THEN world:1[THEN ≡dfE], THEN &E(1)]].

AOT-define Maximal :: ‹τ ⇒ φ› (‹Maximal′(-′)›)
  max: ‹Maximal(s) ≡df ∀p (s ⊨ p ∨ s ⊨ ¬p)›

AOT-theorem world−max: ‹Maximal(w)›
proof(safe intro!: PossibleWorld.ψ[THEN ≡dfE[OF world:1], THEN &E(1)]
                GEN ≡dfI[OF max] &I )
  fix q
  AOT-have ‹◇(w ⊨ q ∨ w ⊨ ¬q)›
  proof(rule RM◇[THEN →E]; (rule →I)?)
    AOT-modally-strict {
      AOT-assume ‹∀p (w ⊨ p ≡ p)›
      AOT-hence ‹w ⊨ q ≡ q› and ‹w ⊨ ¬q ≡ ¬q›
        using ∀E(1)[rotated, OF log−prop−prop:2] by blast+
      AOT-thus ‹w ⊨ q ∨ w ⊨ ¬q›
        by (metis ∨I(1) ∨I(2) ≡E(3) reductio−aa:1)
    }
  next
    AOT-show ‹◇∀p (w ⊨ p ≡ p)›
      using PossibleWorld.ψ[THEN ≡dfE[OF world:1], THEN &E(2)].
  qed
  AOT-hence ‹◇w ⊨ q ∨ ◇w ⊨ ¬q›
    using KBasic2:2[THEN ≡E(1)] by blast
  AOT-thus ‹w ⊨ q ∨ w ⊨ ¬q›
    using lem2:2[unconstrain s, THEN →E, unvarify p,
                OF PossibleWorld.ψ[THEN ≡dfE[OF world:1], THEN &E(1)],
                THEN ≡E(1), OF log−prop−prop:2]
    by (metis ∨I(1) ∨I(2) ∨E(3) raa−cor:2)
qed

AOT-theorem world=maxpos:1: ‹Maximal(x) → □Maximal(x)›
proof (AOT-subst ‹Maximal(x)› ‹Situation(x) & ∀p (x ⊨ p ∨ x ⊨ ¬p)›;
      safe intro!: max ≡Df →I; frule &E(1); drule &E(2))
  AOT-assume sit-x: ‹Situation(x)›
  AOT-hence nec-sit-x: ‹□Situation(x)›
    by (metis ≡E(1) possit−sit:1)
  AOT-assume ‹∀p (x ⊨ p ∨ x ⊨ ¬p)›
  AOT-hence ‹x ⊨ p ∨ x ⊨ ¬p› for p
    using ∀E(1)[rotated, OF log−prop−prop:2] by blast
  AOT-hence ‹□x ⊨ p ∨ □x ⊨ ¬p› for p
    using lem2:1[unconstrain s, THEN →E, OF sit-x, unvarify p,
                OF log−prop−prop:2, THEN ≡E(1)]
    by (metis ∨I(1) ∨I(2) ∨E(2) raa−cor:1)
  AOT-hence ‹□(x ⊨ p ∨ x ⊨ ¬p)› for p
    by (metis KBasic:15 →E)
  AOT-hence ‹∀p □(x ⊨ p ∨ x ⊨ ¬p)›
    by (rule GEN)
  AOT-hence ‹□∀p (x ⊨ p ∨ x ⊨ ¬p)›
    by (rule BF[THEN →E])
  AOT-thus ‹□(Situation(x) & ∀p (x ⊨ p ∨ x ⊨ ¬p))›
    using nec-sit-x by (metis KBasic:3 &I ≡E(2))
qed

AOT-theorem world=maxpos:2: ‹PossibleWorld(x) ≡ Maximal(x) & Possible(x)›
proof(safe intro!: ≡I →I &I world−pos[unconstrain w, THEN →E]
                world−max[unconstrain w, THEN →E];
     frule &E(2); drule &E(1))
  AOT-assume pos-x: ‹Possible(x)›
```

AOT-have ‹◊(*Situation*(*x*) & ∀*p*(*x* ⊨ *p* → *p*))›
  apply (*AOT-subst* (*reverse*) ‹*Situation*(*x*) & ∀*p*(*x* ⊨ *p* → *p*)› ‹*Actual*(*x*)›)
   using *actual* ≡*Df* **apply** *presburger*
  using $\equiv_{df}E$ &*E*(*2*) *pos pos-x* **by** *blast*
AOT-hence *0*: ‹◊∀*p*(*x* ⊨ *p* → *p*)›
  by (*metis KBasic2*:*3* &*E*(*2*) *vdash−properties*:*6*)
AOT-assume *max-x*: ‹*Maximal*(*x*)›
AOT-hence *sit-x*: ‹*Situation*(*x*)› **by** (*metis* $\equiv_{df}E$ *max-x* &*E*(*1*) *max*)
AOT-have ‹□*Maximal*(*x*)› **using** *world*=*maxpos*:*1*[*THEN* →*E*, *OF max-x*] **by** *simp*
moreover AOT-have ‹□*Maximal*(*x*) → □(∀*p*(*x* ⊨ *p* → *p*) → ∀*p* (*x* ⊨ *p* ≡ *p*))›
proof(*safe intro*!: →*I RM GEN*)
  AOT-modally-strict {
    fix *p*
    AOT-assume *0*: ‹*Maximal*(*x*)›
    AOT-assume *1*: ‹∀*p* (*x* ⊨ *p* → *p*)›
    AOT-show ‹*x* ⊨ *p* ≡ *p*›
    proof(*safe intro*!: ≡*I* →*I* *1*[*THEN* ∀*E*(*2*), *THEN* →*E*]; *rule raa−cor*:*1*)
      AOT-assume ‹¬*x* ⊨ *p*›
      AOT-hence ‹*x* ⊨ ¬*p*›
        using *0*[*THEN* $\equiv_{df}E$[*OF max*], *THEN* &*E*(*2*), *THEN* ∀*E*(*2*)]
              1 **by** (*metis* ∨*E*(*2*))
      AOT-hence ‹¬*p*›
        using *1*[*THEN* ∀*E*(*1*), *OF log−prop−prop*:*2*, *THEN* →*E*] **by** *blast*
      moreover AOT-assume *p*
      ultimately AOT-show ‹*p* & ¬*p*› **using** &*I* **by** *blast*
    qed
  }
qed
ultimately AOT-have ‹□(∀*p*(*x* ⊨ *p* → *p*) → ∀*p* (*x* ⊨ *p* ≡ *p*))›
  using →*E* **by** *blast*
AOT-hence ‹◊∀*p*(*x* ⊨ *p* → *p*) → ◊∀*p*(*x* ⊨ *p* ≡ *p*)›
  by (*metis KBasic*:*13*[*THEN* →*E*])
AOT-hence ‹◊∀*p*(*x* ⊨ *p* ≡ *p*)›
  using *0* →*E* **by** *blast*
AOT-thus ‹*PossibleWorld*(*x*)›
  using $\equiv_{df}I$[*OF world*:*1*, *OF* &*I*, *OF sit-x*] **by** *blast*
qed

AOT-define *NecImpl* :: ‹φ ⇒ φ ⇒ φ› (**infixl** ‹⇒› *26*)
  nec−impl−p:*1*: ‹*p* ⇒ *q* $\equiv_{df}$ □(*p* → *q*)›
AOT-define *NecEquiv* :: ‹φ ⇒ φ ⇒ φ› (**infixl** ‹⇔› *21*)
  nec−impl−p:*2*: ‹*p* ⇔ *q* $\equiv_{df}$ (*p* ⇒ *q*) & (*q* ⇒ *p*)›

AOT-theorem *nec−equiv−nec−im*: ‹*p* ⇔ *q* ≡ □(*p* ≡ *q*)›
proof(*safe intro*!: ≡*I* →*I*)
  AOT-assume ‹*p* ⇔ *q*›
  AOT-hence ‹(*p* ⇒ *q*)› **and** ‹(*q* ⇒ *p*)›
    using *nec−impl−p*:*2*[*THEN* $\equiv_{df}E$] &*E* **by** *blast*+
  AOT-hence ‹□(*p* → *q*)› **and** ‹□(*q* → *p*)›
    using *nec−impl−p*:*1*[*THEN* $\equiv_{df}E$] **by** *blast*+
  AOT-thus ‹□(*p* ≡ *q*)› **by** (*metis KBasic*:*4* &*I* ≡*E*(*2*))
next
  AOT-assume ‹□(*p* ≡ *q*)›
  AOT-hence ‹□(*p* → *q*)› **and** ‹□(*q* → *p*)›
    using *KBasic*:*4* &*E* ≡*E*(*1*) **by** *blast*+
  AOT-hence ‹(*p* ⇒ *q*)› **and** ‹(*q* ⇒ *p*)›
    using *nec−impl−p*:*1*[*THEN* $\equiv_{df}I$] **by** *blast*+
  AOT-thus ‹*p* ⇔ *q*›
    using *nec−impl−p*:*2*[*THEN* $\equiv_{df}I$] &*I* **by** *blast*
qed

AOT-theorem *world-closed-lem-1-a*:

‹$(s \models (\varphi \;\&\; \psi)) \rightarrow (\forall p\ (s \models p \equiv p) \rightarrow (s \models \varphi \;\&\; s \models \psi))$›
proof(*safe intro*!: →*I*)
  AOT-assume ‹$\forall\ p\ (s \models p \equiv p)$›
  AOT-hence ‹$s \models (\varphi \;\&\; \psi) \equiv (\varphi \;\&\; \psi)$› **and** ‹$s \models \varphi \equiv \varphi$› **and** ‹$s \models \psi \equiv \psi$›
    using ∀ *E*(*1*)[*rotated*, *OF log−prop−prop*:*2*] **by** *blast*+
  moreover AOT-assume ‹$s \models (\varphi \;\&\; \psi)$›
  ultimately AOT-show ‹$s \models \varphi \;\&\; s \models \psi$›
    by (*metis* &*I* &*E*(*1*) &*E*(*2*) ≡*E*(*1*) ≡*E*(*2*))
qed

AOT-theorem *world-closed-lem-1-b*:
  ‹$(s \models \varphi \;\&\; (\varphi \rightarrow q)) \rightarrow (\forall p\ (s \models p \equiv p) \rightarrow s \models q)$›
proof(*safe intro*!: →*I*)
  AOT-assume ‹$\forall\ p\ (s \models p \equiv p)$›
  AOT-hence ‹$s \models \varphi \equiv \varphi$› **for** φ
    using ∀ *E*(*1*)[*rotated*, *OF log−prop−prop*:*2*] **by** *blast*
  moreover AOT-assume ‹$s \models \varphi \;\&\; (\varphi \rightarrow q)$›
  ultimately AOT-show ‹$s \models q$›
    by (*metis* &*E*(*1*) &*E*(*2*) ≡*E*(*1*) ≡*E*(*2*) →*E*)
qed

AOT-theorem *world-closed-lem-1-c*:
  ‹$(s \models \varphi \;\&\; s \models (\varphi \rightarrow \psi)) \rightarrow (\forall p\ (s \models p \equiv p) \rightarrow s \models \psi)$›
proof(*safe intro*!: →*I*)
  AOT-assume ‹$\forall\ p\ (s \models p \equiv p)$›
  AOT-hence ‹$s \models \varphi \equiv \varphi$› **for** φ
    using ∀ *E*(*1*)[*rotated*, *OF log−prop−prop*:*2*] **by** *blast*
  moreover AOT-assume ‹$s \models \varphi \;\&\; s \models (\varphi \rightarrow \psi)$›
  ultimately AOT-show ‹$s \models \psi$›
    by (*metis* &*E*(*1*) &*E*(*2*) ≡*E*(*1*) ≡*E*(*2*) →*E*)
qed

AOT-theorem *world−closed−lem*:*1*[*0*]:
  ‹$q \rightarrow (\forall p\ (s \models p \equiv p) \rightarrow s \models q)$›
  by (*meson* →*I* ≡*E*(*2*) *log−prop−prop*:*2 rule−ui*:*1*)

AOT-theorem *world−closed−lem*:*1*[*1*]:
  ‹$s \models p_1 \;\&\; (p_1 \rightarrow q) \rightarrow (\forall p\ (s \models p \equiv p) \rightarrow s \models q)$›
  using *world-closed-lem-1-b*.

AOT-theorem *world−closed−lem*:*1*[*2*]:
  ‹$s \models p_1 \;\&\; s \models p_2 \;\&\; ((p_1 \;\&\; p_2) \rightarrow q) \rightarrow (\forall p\ (s \models p \equiv p) \rightarrow s \models q)$›
  using *world-closed-lem-1-b world-closed-lem-1-a*
  by (*metis* (*full-types*) &*I* &*E* →*I* →*E*)

AOT-theorem *world−closed−lem*:*1*[*3*]:
  ‹$s \models p_1 \;\&\; s \models p_2 \;\&\; s \models p_3 \;\&\; ((p_1 \;\&\; p_2 \;\&\; p_3) \rightarrow q) \rightarrow (\forall p\ (s \models p \equiv p) \rightarrow s \models q)$›
  using *world-closed-lem-1-b world-closed-lem-1-a*
  by (*metis* (*full-types*) &*I* &*E* →*I* →*E*)

AOT-theorem *world−closed−lem*:*1*[*4*]:
  ‹$s \models p_1 \;\&\; s \models p_2 \;\&\; s \models p_3 \;\&\; s \models p_4 \;\&\; ((p_1 \;\&\; p_2 \;\&\; p_3 \;\&\; p_4) \rightarrow q) \rightarrow$
   $(\forall p\ (s \models p \equiv p) \rightarrow s \models q)$›
  using *world-closed-lem-1-b world-closed-lem-1-a*
  by (*metis* (*full-types*) &*I* &*E* →*I* →*E*)

AOT-theorem *coherent*:*1*: ‹$w \models \neg p \equiv \neg w \models p$›
proof(*safe intro*!: ≡*I* →*I*)
  AOT-assume *1*: ‹$w \models \neg p$›
  AOT-show ‹$\neg w \models p$›
  proof(*rule raa−cor*:*2*)
    AOT-assume ‹$w \models p$›
    AOT-hence ‹$w \models p \;\&\; w \models \neg p$› **using** *1* &*I* **by** *blast*

AOT-hence ‹∃ *q* (*w* ⊨ *q* & *w* ⊨ ¬*q*)› **by** (*rule* ∃ *I*)
moreover AOT-have ‹¬∃ *q* (*w* ⊨ *q* & *w* ⊨ ¬*q*)›
using *world−cons*:*1*[*THEN* $\equiv_{df}$*E*[*OF cons*], *THEN* &*E*(*2*)].
ultimately AOT-show ‹∃ *q* (*w* ⊨ *q* & *w* ⊨ ¬*q*) & ¬∃ *q* (*w* ⊨ *q* & *w* ⊨ ¬*q*)›
using &*I* **by** *blast*
qed
next
AOT-assume ‹¬*w* ⊨ *p*›
AOT-thus ‹*w* ⊨ ¬*p*›
using *world−max*[*THEN* $\equiv_{df}$*E*[*OF max*], *THEN* &*E*(*2*)]
by (*metis* ∨*E*(*2*) *log−prop−prop*:*2* *rule−ui*:*1*)
qed

AOT-theorem *coherent*:*2*: ‹*w* ⊨ *p* ≡ ¬*w* ⊨ ¬*p*›
by (*metis coherent*:*1* *deduction−theorem* ≡*I* ≡*E*(*1*) ≡*E*(*2*) *raa−cor*:*3*)

AOT-theorem *act−world*:*1*: ‹∃ *w* ∀ *p* (*w* ⊨ *p* ≡ *p*)›
proof −
AOT-obtain *s* **where** *s-prop*: ‹∀ *p* (*s* ⊨ *p* ≡ *p*)›
using *sit−classical*:*6* *Situation*.∃ *E*[*rotated*] **by** *meson*
AOT-hence ‹◇∀ *p* (*s* ⊨ *p* ≡ *p*)›
by (*metis T*◇ *vdash−properties*:*10*)
AOT-hence ‹*PossibleWorld*(*s*)›
using *world*:*1*[*THEN* $\equiv_{df}$*I*] *Situation*.*ψ* &*I* **by** *blast*
AOT-hence ‹*PossibleWorld*(*s*) & ∀ *p* (*s* ⊨ *p* ≡ *p*)›
using &*I* *s-prop* **by** *blast*
thus *?thesis* **by** (*rule* ∃ *I*)
qed

AOT-theorem *act−world*:*2*: ‹∃!*w* *Actual*(*w*)›
proof −
AOT-obtain *w* **where** *w-prop*: ‹∀ *p* (*w* ⊨ *p* ≡ *p*)›
using *act−world*:*1* *PossibleWorld*.∃ *E*[*rotated*] **by** *meson*
AOT-have *sit-s*: ‹*Situation*(*w*)›
using *PossibleWorld*.*ψ* *world*:*1*[*THEN* $\equiv_{df}$*E*, *THEN* &*E*(*1*)] **by** *blast*
show *?thesis*
proof (*safe intro*!: *uniqueness*:*1*[*THEN* $\equiv_{df}$*I*] ∃ *I*(*2*) &*I* *GEN* →*I*
PossibleWorld.*ψ* *actual*[*THEN* $\equiv_{df}$*I*] *sit-s*
sit−identity[*unconstrain s*, *unconstrain s*′, *THEN* →*E*,
THEN →*E*, *THEN* ≡*E*(*2*)] ≡*I*
w-prop[*THEN* ∀ *E*(*2*), *THEN* ≡*E*(*1*)])
AOT-show ‹*PossibleWorld*(*w*)› **using** *PossibleWorld*.*ψ*.
next
AOT-show ‹*Situation*(*w*)›
by (*simp add*: *sit-s*)
next
fix *y p*
AOT-assume *w-asm*: ‹*PossibleWorld*(*y*) & *Actual*(*y*)›
AOT-assume ‹*w* ⊨ *p*›
AOT-hence *p*: ‹*p*›
using *w-prop*[*THEN* ∀ *E*(*2*), *THEN* ≡*E*(*1*)] **by** *blast*
AOT-show ‹*y* ⊨ *p*›
proof(*rule raa−cor*:*1*)
AOT-assume ‹¬*y* ⊨ *p*›
AOT-hence ‹*y* ⊨ ¬*p*›
by (*metis coherent*:*1*[*unconstrain w*, *THEN* →*E*] &*E*(*1*) ≡*E*(*2*) *w-asm*)
AOT-hence ‹¬*p*›
using *w-asm*[*THEN* &*E*(*2*), *THEN* *actual*[*THEN* $\equiv_{df}$*E*], *THEN* &*E*(*2*),
THEN ∀ *E*(*1*), *rotated*, *OF log−prop−prop*:*2*]
→*E* **by** *blast*
AOT-thus ‹*p* & ¬*p*› **using** *p* &*I* **by** *blast*
qed
next

```
    AOT-show ‹w ⊨ p› if ‹y ⊨ p› and ‹PossibleWorld(y) & Actual(y)› for p y
      using that(2)[THEN &E(2), THEN actual[THEN ≡df E], THEN &E(2),
                    THEN ∀E(2), THEN →E, OF that(1)]
            w-prop[THEN ∀E(2), THEN ≡E(2)] by blast
  next
    AOT-show ‹Situation(y)› if ‹PossibleWorld(y) & Actual(y)› for y
      using that[THEN &E(1)] world:1[THEN ≡df E, THEN &E(1)] by blast
  next
    AOT-show ‹Situation(w)›
      using sit-s by blast
  qed(simp)
qed

AOT-theorem pre−walpha: ‹ιw Actual(w)↓›
  using A−Exists:2 RA[2] act−world:2 ≡E(2) by blast

AOT-define TheActualWorld :: ‹κs› (‹wα›)
  w−alpha: ‹wα =df ιw Actual(w)›


AOT-theorem true-in-truth-act-true: ‹⊤ ⊨ p ≡ 𝓐p›
proof(safe intro!: ≡I →I)
  AOT-have true-def: ‹⊢□ ⊤ = ιx (A!x & ∀F (x[F] ≡ ∃p(p & F = [λy p])))›
    by (simp add: A−descriptions rule−id−df:1[zero] the−true:1)
  AOT-hence true-den: ‹⊢□ ⊤↓›
    using t=t−proper:1 vdash−properties:6 by blast
  {
    AOT-assume ‹⊤ ⊨ p›
    AOT-hence ‹⊤[λy p]›
      by (metis ≡df E con−dis−i−e:2:b prop−enc true−in−s)
    AOT-hence ‹ιx(A!x & ∀F (x[F] ≡ ∃q (q & F = [λy q])))[λy p]›
      using rule=E true-def true-den by fast
    AOT-hence ‹𝓐∃q (q & [λy p] = [λy q])›
      using ≡E(1) desc−nec−encode:1[unvarify F] prop−prop2:2 by fast
    AOT-hence ‹∃q 𝓐(q & [λy p] = [λy q])›
      by (metis Act−Basic:10 ≡E(1))
    then AOT-obtain q where ‹𝓐(q & [λy p] = [λy q])›
      using ∃E[rotated] by blast
    AOT-hence actq: ‹𝓐q› and ‹𝓐[λy p] = [λy q]›
      using Act−Basic:2 intro−elim:3:a &E by blast+
    AOT-hence ‹[λy p] = [λy q]›
      using id−act:1[unvarify α β, THEN ≡E(2)] prop−prop2:2 by blast
    AOT-hence ‹p = q›
      by (metis intro−elim:3:b p−identity−thm2:3)
    AOT-thus ‹𝓐p›
      using actq rule=E id-sym by blast
  }
  {
    AOT-assume ‹𝓐p›
    AOT-hence ‹𝓐(p & [λy p] = [λy p])›
      by (auto intro!: Act−Basic:2[THEN ≡E(2)] &I
               intro: RA[2] =I(1)[OF prop−prop2:2])
    AOT-hence ‹∃q 𝓐(q & [λy p] = [λy q])›
      using ∃I by fast
    AOT-hence ‹𝓐∃q (q & [λy p] = [λy q])›
      by (metis Act−Basic:10 ≡E(2))
    AOT-hence ‹ιx(A!x & ∀F (x[F] ≡ ∃q (q & F = [λy q])))[λy p]›
      using ≡E(2) desc−nec−encode:1[unvarify F] prop−prop2:2 by fast
    AOT-hence ‹⊤[λy p]›
      using rule=E true-def true-den id-sym by fast
    AOT-thus ‹⊤ ⊨ p›
      by (safe intro!: true−in−s[THEN ≡df I] &I possit−sit:6
                    prop−enc[THEN ≡df I] true-den)
```

}
qed

AOT-theorem *T−world*: ‹⊤ = $\mathbf{w}_\alpha$›
proof −
AOT-have *true-den*: ‹$\vdash_\Box$ ⊤↓›
using *Situation.res−var*:*3* *possit−sit*:*6* →*E* **by** *blast*
AOT-have ‹$\mathcal{A}$∀ *p* (⊤ ⊨ *p* → *p*)›
proof (*safe intro*!: *logic−actual−nec*:*3*[*axiom-inst*, *THEN* ≡*E*(*2*)] *GEN*
logic−actual−nec:*2*[*axiom-inst*, *THEN* ≡*E*(*2*)] →*I*)
fix *p*
AOT-assume ‹$\mathcal{A}$⊤ ⊨ *p*›
AOT-hence ‹⊤ ⊨ *p*›
using *lem2*:*4*[*unconstrain s*, *unvarify β*, *OF true-den*,
THEN →*E*, *OF possit−sit*:*6*] ≡*E*(*1*) **by** *blast*
AOT-thus ‹$\mathcal{A}$*p*› **using** *true-in-truth-act-true* ≡*E*(*1*) **by** *blast*
qed
moreover AOT-have ‹$\mathcal{A}$(*Situation*(κ) & ∀ *p* (κ ⊨ *p* → *p*)) → $\mathcal{A}$*Actual*(κ)› **for** κ
using *actual*[*THEN* ≡*Df*, *THEN conventions*:*3*[*THEN* $\equiv_{df}$*E*, *THEN* &*E*(*2*)],
THEN RA[*2*], *THEN act−cond*[*THEN* →*E*]].
ultimately AOT-have *act-act-true*: ‹$\mathcal{A}$*Actual*(⊤)›
using *possit−sit*:*4*[*unvarify x*, *OF true-den*, *THEN* ≡*E*(*2*), *OF possit−sit*:*6*]
Act−Basic:*2*[*THEN* ≡*E*(*2*), *OF* &*I*] →*E* **by** *blast*
AOT-hence ‹◇*Actual*(⊤)› **by** (*metis Act−Sub*:*3* *vdash−properties*:*10*)
AOT-hence ‹*Possible*(⊤)›
by (*safe intro*!: *pos*[*THEN* $\equiv_{df}$*I*] &*I possit−sit*:*6*)
moreover AOT-have ‹*Maximal*(⊤)›
proof (*safe intro*!: *max*[*THEN* $\equiv_{df}$*I*] &*I possit−sit*:*6* *GEN*)
fix *p*
AOT-have ‹$\mathcal{A}$*p* ∨ $\mathcal{A}$¬*p*›
by (*simp add*: *Act−Basic*:*1*)
moreover AOT-have ‹⊤ ⊨ *p*› **if** ‹$\mathcal{A}$*p*›
using *that true-in-truth-act-true*[*THEN* ≡*E*(*2*)] **by** *blast*
moreover AOT-have ‹⊤ ⊨ ¬*p*› **if** ‹$\mathcal{A}$¬*p*›
using *that true-in-truth-act-true*[*unvarify p*, *THEN* ≡*E*(*2*)]
log−prop−prop:*2* **by** *blast*
ultimately AOT-show ‹⊤ ⊨ *p* ∨ ⊤ ⊨ ¬*p*›
using ∨*I*(*3*) →*I* **by** *blast*
qed
ultimately AOT-have ‹*PossibleWorld*(⊤)›
by (*safe intro*!: *world*=*maxpos*:*2*[*unvarify x*, *OF true-den*, *THEN* ≡*E*(*2*)] &*I*)
AOT-hence ‹$\mathcal{A}$*PossibleWorld*(⊤)›
using *rigid−pw*:*4*[*unvarify x*, *OF true-den*, *THEN* ≡*E*(*2*)] **by** *blast*
AOT-hence *1*: ‹$\mathcal{A}$(*PossibleWorld*(⊤) & *Actual*(⊤))›
using *act-act-true Act−Basic*:*2* *df−simplify*:*2* *intro−elim*:*3*:*b* **by** *blast*
AOT-have ‹$\mathbf{w}_\alpha$ = ι*w*(*Actual*(*w*))›
using *rule−id−df*:*1*[*zero*][*OF w−alpha*, *OF pre−walpha*] **by** *simp*
moreover AOT-have *w-act-den*: ‹$\mathbf{w}_\alpha$↓›
using *calculation t*=*t−proper*:*1* →*E* **by** *blast*
ultimately AOT-have ‹∀ *z* ($\mathcal{A}$(*PossibleWorld*(*z*) & *Actual*(*z*)) → *z* = $\mathbf{w}_\alpha$)›
using *nec−hintikka−scheme*[*unvarify x*] ≡*E*(*1*) &*E* **by** *blast*
AOT-thus ‹⊤ = $\mathbf{w}_\alpha$›
using ∀ *E*(*1*)[*rotated*, *OF true-den*] *1* →*E* **by** *blast*
qed

AOT-act-theorem *truth−at−alpha*:*1*: ‹*p* ≡ $\mathbf{w}_\alpha$ = ι*x* (*ExtensionOf*(*x*, *p*))›
by (*metis rule*=*E T−world deduction−theorem ext−p−tv*:*3* *id-sym* ≡*I*
≡*E*(*1*) ≡*E*(*2*) *q−True*:*1*)

AOT-act-theorem *truth−at−alpha*:*2*: ‹*p* ≡ $\mathbf{w}_\alpha$ ⊨ *p*›
proof −
AOT-have ‹*PossibleWorld*($\mathbf{w}_\alpha$)›
using &*E*(*1*) *pre−walpha rule−id−df*:*2*:*b*[*zero*] →*E*

w−alpha y−in:3 **by** *blast*
AOT-hence *sit-w-alpha*: ‹*Situation*($\mathbf{w}_\alpha$)›
by (*metis* $\equiv_{df}E$ &*E*(*1*) *world*:*1*)
AOT-have *w-alpha-den*: ‹$\mathbf{w}_\alpha\downarrow$›
using *pre−walpha rule−id−df*:*2*:*b*[*zero*] *w−alpha* **by** *blast*
AOT-have ‹$p \equiv \top\mathbf{\Sigma}p$›
using *q−True*:*3* **by** *force*
moreover AOT-have ‹$\top = \mathbf{w}_\alpha$›
using *T−world* **by** *auto*
ultimately AOT-have ‹$p \equiv \mathbf{w}_\alpha\mathbf{\Sigma}p$›
using *rule=E* **by** *fast*
moreover AOT-have ‹$\mathbf{w}_\alpha\ \mathbf{\Sigma}\ p \equiv \mathbf{w}_\alpha \models p$›
using *lem1*[*unvarify x*, *OF w-alpha-den*, *THEN* →*E*, *OF sit-w-alpha*]
using ≡*S*(*1*) ≡*E*(*1*) *Commutativity of* ≡ ≡*Df sit-w-alpha true−in−s* **by** *blast*
ultimately AOT-show ‹$p \equiv \mathbf{w}_\alpha \models p$›
by (*metis* ≡*E*(*5*))
qed

AOT-theorem *alpha−world*:*1*: ‹*PossibleWorld*($\mathbf{w}_\alpha$)›
proof −
AOT-have *0*: ‹$\mathbf{w}_\alpha = \iota w\ Actual(w)$›
using *pre−walpha rule−id−df*:*1*[*zero*] *w−alpha* **by** *blast*
AOT-hence *walpha-den*: ‹$\mathbf{w}_\alpha\downarrow$›
by (*metis t=t−proper*:*1 vdash−properties*:*6*)
AOT-have ‹$\mathcal{A}$(*PossibleWorld*($\mathbf{w}_\alpha$) & *Actual*($\mathbf{w}_\alpha$))›
by (*rule actual−desc*:*2*[*unvarify x*, *OF walpha-den*, *THEN* →*E*]) (*fact 0*)
AOT-hence ‹$\mathcal{A}$*PossibleWorld*($\mathbf{w}_\alpha$)›
by (*metis Act−Basic*:*2* &*E*(*1*) ≡*E*(*1*))
AOT-thus ‹*PossibleWorld*($\mathbf{w}_\alpha$)›
using *rigid−pw*:*4*[*unvarify x*, *OF walpha-den*, *THEN* ≡*E*(*1*)]
by *blast*
qed

AOT-theorem *alpha−world*:*2*: ‹*Maximal*($\mathbf{w}_\alpha$)›
proof −
AOT-have ‹$\mathbf{w}_\alpha\downarrow$›
using *pre−walpha rule−id−df*:*2*:*b*[*zero*] *w−alpha* **by** *blast*
then AOT-obtain *x* **where** *x-def*: ‹$x = \mathbf{w}_\alpha$›
by (*metis instantiation rule=I*:*1 existential*:*1 id-sym*)
AOT-hence ‹*PossibleWorld*(*x*)› **using** *alpha−world*:*1 rule=E id-sym* **by** *fast*
AOT-hence ‹*Maximal*(*x*)› **by** (*metis world−max*[*unconstrain w*, *THEN* →*E*])
AOT-thus ‹*Maximal*($\mathbf{w}_\alpha$)› **using** *x-def rule=E* **by** *blast*
qed

AOT-theorem *t−at−alpha−strict*: ‹$\mathbf{w}_\alpha \models p \equiv \mathcal{A}p$›
proof −
AOT-have *0*: ‹$\mathbf{w}_\alpha = \iota w\ Actual(w)$›
using *pre−walpha rule−id−df*:*1*[*zero*] *w−alpha* **by** *blast*
AOT-hence *walpha-den*: ‹$\mathbf{w}_\alpha\downarrow$›
by (*metis t=t−proper*:*1 vdash−properties*:*6*)
AOT-have *1*: ‹$\mathcal{A}$(*PossibleWorld*($\mathbf{w}_\alpha$) & *Actual*($\mathbf{w}_\alpha$))›
by (*rule actual−desc*:*2*[*unvarify x*, *OF walpha-den*, *THEN* →*E*]) (*fact 0*)
AOT-have *walpha-sit*: ‹*Situation*($\mathbf{w}_\alpha$)›
by (*meson* $\equiv_{df}E$ *alpha−world*:*2* &*E*(*1*) *max*)
{
fix *p*
AOT-have *2*: ‹$Situation(x) \to (\mathcal{A}x \models p \equiv x \models p)$› **for** *x*
using *lem2*:*4*[*unconstrain s*] **by** *blast*
AOT-assume ‹$\mathbf{w}_\alpha \models p$›
AOT-hence *ϑ*: ‹$\mathcal{A}\mathbf{w}_\alpha \models p$›
using *2*[*unvarify x*, *OF walpha-den*, *THEN* →*E*, *OF walpha-sit*, *THEN* ≡*E*(*2*)]
by *argo*
AOT-have *3*: ‹$\mathcal{A}$*Actual*($\mathbf{w}_\alpha$)›

```
      using 1 Act−Basic:2 &E(2) ≡E(1) by blast
    AOT-have ‹𝓐(Situation(wα) & ∀ q(wα ⊨ q → q))›
      apply (AOT-subst (reverse) ‹Situation(wα) & ∀ q(wα ⊨ q → q)› ‹Actual(wα)›)
       using actual ≡Df apply blast
      by (fact 3)
    AOT-hence ‹𝓐∀ q(wα ⊨ q → q)› by (metis Act−Basic:2 &E(2) ≡E(1))
    AOT-hence ‹∀ q 𝓐(wα ⊨ q → q)›
      using logic−actual−nec:3[axiom-inst, THEN ≡E(1)] by blast
    AOT-hence ‹𝓐(wα ⊨ p → p)› using ∀ E(2) by blast
    AOT-hence ‹𝓐(wα ⊨ p) → 𝓐p› by (metis act−cond vdash−properties:10)
    AOT-hence ‹𝓐p› using ϑ →E by blast
  }
  AOT-hence 2: ‹wα ⊨ p → 𝓐p› for p by (rule →I)
  AOT-have walpha-sit: ‹Situation(wα)›
    using ≡df E alpha−world:2 &E(1) max by blast
  show ?thesis
  proof(safe intro!: ≡I →I 2)
    AOT-assume actp: ‹𝓐p›
    AOT-show ‹wα ⊨ p›
    proof(rule raa−cor:1)
      AOT-assume ‹¬wα ⊨ p›
      AOT-hence ‹wα ⊨ ¬p›
        using alpha−world:2[THEN max[THEN ≡df E], THEN &E(2),
                            THEN ∀ E(1), OF log−prop−prop:2]
        by (metis ∨E(2))
      AOT-hence ‹𝓐¬p›
        using 2[unvarify p, OF log−prop−prop:2, THEN →E] by blast
      AOT-hence ‹¬𝓐p› by (metis ¬¬I Act−Sub:1 ≡E(4))
      AOT-thus ‹𝓐p & ¬𝓐p› using actp &I by blast
    qed
  qed
qed

AOT-act-theorem not−act: ‹w ≠ wα → ¬Actual(w)›
proof (rule →I; rule raa−cor:2)
  AOT-assume ‹w ≠ wα›
  AOT-hence 0: ‹¬(w = wα)› by (metis ≡df E =−infix)
  AOT-have walpha-den: ‹wα↓›
    using pre−walpha rule−id−df:2:b[zero] w−alpha by blast
  AOT-have walpha-sit: ‹Situation(wα)›
    using ≡df E alpha−world:2 &E(1) max by blast
  AOT-assume act-w: ‹Actual(w)›
  AOT-hence w-sit: ‹Situation(w)› by (metis ≡df E actual &E(1))
  AOT-have sid: ‹Situation(x') → (w = x' ≡ ∀ p (w ⊨ p ≡ x' ⊨ p))› for x'
    using sit−identity[unconstrain s', unconstrain s, THEN →E, OF w-sit]
    by blast
  AOT-have ‹w = wα›
  proof(safe intro!: GEN sid[unvarify x', OF walpha-den, THEN →E,
                          OF walpha-sit, THEN ≡E(2)] ≡I →I)
    fix p
    AOT-assume ‹w ⊨ p›
    AOT-hence ‹p›
      using actual[THEN ≡df E, OF act-w, THEN &E(2), THEN ∀ E(2), THEN →E]
      by blast
    AOT-hence ‹𝓐p›
      by (metis RA[1])
    AOT-thus ‹wα ⊨ p›
      using t−at−alpha−strict[THEN ≡E(2)] by blast
  next
    fix p
    AOT-assume ‹wα ⊨ p›
    AOT-hence ‹𝓐p›
      using t−at−alpha−strict[THEN ≡E(1)] by blast
```

```
    AOT-hence p: ‹p›
      using logic−actual[act-axiom-inst, THEN →E] by blast
    AOT-show ‹w ⊨ p›
    proof(rule raa−cor:1)
      AOT-assume ‹¬w ⊨ p›
      AOT-hence ‹w ⊨ ¬p›
        by (metis coherent:1 ≡E(2))
      AOT-hence ‹¬p›
        using actual[THEN ≡_df E, OF act-w, THEN &E(2), THEN ∀E(1),
                    OF log−prop−prop:2, THEN →E] by blast
      AOT-thus ‹p & ¬p› using p &I by blast
    qed
  qed
  AOT-thus ‹w = w_α & ¬(w = w_α)› using 0 &I by blast
qed

AOT-act-theorem w−alpha−part: ‹Actual(s) ≡ s ⊴ w_α›
proof(safe intro!: ≡I →I sit−part−whole[THEN ≡_df I] &I GEN
          dest!:  sit−part−whole[THEN ≡_df E])
  AOT-show ‹Situation(s)› if ‹Actual(s)›
    using ≡_df E actual &E(1) that by blast
next
  AOT-show ‹Situation(w_α)›
    using ≡_df E alpha−world:2 &E(1) max by blast
next
  fix p
  AOT-assume ‹Actual(s)›
  moreover AOT-assume ‹s ⊨ p›
  ultimately AOT-have ‹p›
    using actual[THEN ≡_df E, THEN &E(2), THEN ∀E(2), THEN →E] by blast
  AOT-thus ‹w_α ⊨ p›
     by (metis ≡E(1) truth−at−alpha:2)
next
  AOT-assume 0: ‹Situation(s) & Situation(w_α) & ∀p (s ⊨ p → w_α ⊨ p)›
  AOT-hence ‹s ⊨ p → w_α ⊨ p› for p
    using &E ∀E(2) by blast
  AOT-hence ‹s ⊨ p → p› for p
    by (metis →I ≡E(2) truth−at−alpha:2 →E)
  AOT-hence ‹∀p (s ⊨ p → p)› by (rule GEN)
  AOT-thus ‹Actual(s)›
    using actual[THEN ≡_df I, OF &I, OF 0[THEN &E(1), THEN &E(1)]] by blast
qed

AOT-act-theorem act−world2:1: ‹w_α ⊨ p ≡ [λy p]w_α›
  apply (AOT-subst ‹[λy p]w_α› p)
   apply (rule beta−C−meta[THEN →E, OF prop−prop2:2, unvarify ν_1ν_n])
  using pre−walpha rule−id−df:2:b[zero] w−alpha apply blast
  using ≡E(2) Commutativity of ≡ truth−at−alpha:2 by blast

AOT-act-theorem act−world2:2: ‹p ≡ w_α ⊨ [λy p]w_α›
proof −
  AOT-have ‹p ≡ [λy p]w_α›
    apply (rule beta−C−meta[THEN →E, OF prop−prop2:2,
                              unvarify ν_1ν_n, symmetric])
    using pre−walpha rule−id−df:2:b[zero] w−alpha by blast
  also AOT-have ‹... ≡ w_α ⊨ [λy p]w_α›
    by (meson log−prop−prop:2 rule−ui:1 truth−at−alpha:2 universal−cor)
  finally show ?thesis.
qed

AOT-theorem fund−lem:1: ‹◇p → ◇∃w (w ⊨ p)›
proof (rule RM◇; rule →I; rule raa−cor:1)
  AOT-modally-strict {
```

```
    AOT-obtain w where w-prop: ‹∀ q (w ⊨ q ≡ q)›
      using act−world:1 PossibleWorld.∃ E[rotated] by meson
    AOT-assume p: ‹p›
    AOT-assume 0: ‹¬∃ w (w ⊨ p)›
    AOT-have ‹∀ w ¬(w ⊨ p)›
      apply (AOT-subst ‹PossibleWorld(x) → ¬x ⊨ p›
                   ‹¬(PossibleWorld(x) & x ⊨ p)› for: x)
      apply (metis &I &E(1) &E(2) →I ≡I modus−tollens:2)
      using 0 cqt−further:4 vdash−properties:10 by blast
    AOT-hence ‹¬(w ⊨ p)›
      using PossibleWorld.ψ rule−ui:3 →E by blast
    AOT-hence ‹¬p›
      using w-prop[THEN ∀ E(2), THEN ≡E(2)]
      by (metis raa−cor:3)
    AOT-thus ‹p & ¬p›
      using p &I by blast
  }
qed

AOT-theorem fund−lem:2: ‹◇∃ w (w ⊨ p) → ∃ w (w ⊨ p)›
proof (rule →I)
  AOT-assume ‹◇∃ w (w ⊨ p)›
  AOT-hence ‹∃ w ◇(w ⊨ p)›
    using PossibleWorld.res−var−bound−reas[BF◇][THEN →E] by auto
  then AOT-obtain w where ‹◇(w ⊨ p)›
    using PossibleWorld.∃ E[rotated] by meson
  moreover AOT-have ‹Situation(w)›
    by (metis ≡df E &E(1) pos world−pos)
  ultimately AOT-have ‹w ⊨ p›
    using lem2:2[unconstrain s, THEN →E]  ≡E by blast
  AOT-thus ‹∃ w w ⊨ p›
    by (rule PossibleWorld.∃ I)
qed

AOT-theorem fund−lem:3: ‹p → ∀ s(∀ q (s ⊨ q ≡ q) → s ⊨ p)›
proof(safe intro!: →I Situation.GEN)
  fix s
  AOT-assume ‹p›
  moreover AOT-assume ‹∀ q (s ⊨ q ≡ q)›
  ultimately AOT-show ‹s ⊨ p›
    using ∀ E(2) ≡E(2) by blast
qed

AOT-theorem fund−lem:4: ‹□p → □∀ s(∀ q (s ⊨ q ≡ q) → s ⊨ p)›
  using fund−lem:3 by (rule RM)

AOT-theorem fund−lem:5: ‹□∀ s φ{s} → ∀ s □φ{s}›
proof(safe intro!: →I Situation.GEN)
  fix s
  AOT-assume ‹□∀ s φ{s}›
  AOT-hence ‹∀ s □φ{s}›
    using Situation.res−var−bound−reas[CBF][THEN →E] by blast
  AOT-thus ‹□φ{s}›
    using Situation.∀ E by fast
qed
```

Note: not explicit in PLM.

```
AOT-theorem fund−lem:5[world]: ‹□∀ w φ{w} → ∀ w □φ{w}›
proof(safe intro!: →I PossibleWorld.GEN)
  fix w
  AOT-assume ‹□∀ w φ{w}›
  AOT-hence ‹∀ w □φ{w}›
    using PossibleWorld.res−var−bound−reas[CBF][THEN →E] by blast
```

```
  AOT-thus ‹□φ{w}›
    using PossibleWorld.∀E by fast
qed

AOT-theorem fund−lem:6: ‹∀w w ⊨ p → □∀w w ⊨ p›
proof(rule →I)
  AOT-assume ‹∀w (w ⊨ p)›
  AOT-hence 1: ‹PossibleWorld(w) → (w ⊨ p)› for w
    using ∀E(2) by blast
  AOT-show ‹□∀w w ⊨ p›
  proof(rule raa−cor:1)
    AOT-assume ‹¬□∀w w ⊨ p›
    AOT-hence ‹◇¬∀w w ⊨ p›
      by (metis KBasic:11 ≡E(1))
    AOT-hence ‹◇∃x (¬(PossibleWorld(x) → x ⊨ p))›
      apply (rule RM◇[THEN →E, rotated])
      by (simp add: cqt−further:2)
    AOT-hence ‹∃x ◇(¬(PossibleWorld(x) → x ⊨ p))›
      by (metis BF◇ vdash−properties:10)
    then AOT-obtain x where x-prop: ‹◇¬(PossibleWorld(x) → x ⊨ p)›
      using ∃E[rotated] by blast
    AOT-have ‹◇(PossibleWorld(x) & ¬x ⊨ p)›
      apply (AOT-subst ‹PossibleWorld(x) & ¬x ⊨ p›
                  ‹¬(PossibleWorld(x) → x ⊨ p)›)
       apply (meson ≡E(6) oth−class−taut:1:b oth−class−taut:3:a)
      by(fact x-prop)
    AOT-hence 2: ‹◇PossibleWorld(x) & ◇¬x ⊨ p›
      by (metis KBasic2:3 vdash−properties:10)
    AOT-hence ‹PossibleWorld(x)›
      using &E(1) ≡E(1) rigid−pw:2 by blast
    AOT-hence ‹□(x ⊨ p)›
      using 2[THEN &E(2)]  1[unconstrain w, THEN →E] →E
            rigid−truth−at:1[unconstrain w, THEN →E]
      by (metis ≡E(1))
    moreover AOT-have ‹¬□(x ⊨ p)›
      using 2[THEN &E(2)] by (metis ¬¬I KBasic:12 ≡E(4))
    ultimately AOT-show ‹p & ¬p› for p
      by (metis raa−cor:3)
  qed
qed

AOT-theorem fund−lem:7: ‹□∀w(w ⊨ p) → □p›
proof(rule RM; rule →I)
  AOT-modally-strict {
    AOT-obtain w where w-prop: ‹∀p (w ⊨ p ≡ p)›
      using act−world:1 PossibleWorld.∃E[rotated] by meson
    AOT-assume ‹∀w (w ⊨ p)›
    AOT-hence ‹w ⊨ p›
      using PossibleWorld.∀E by fast
    AOT-thus ‹p›
      using w-prop[THEN ∀E(2), THEN ≡E(1)] by blast
  }
qed

AOT-theorem fund:1: ‹◇p ≡ ∃w w ⊨ p›
proof (rule ≡I; rule →I)
  AOT-assume ‹◇p›
  AOT-thus ‹∃w w ⊨ p›
    by (metis fund−lem:1 fund−lem:2 →E)
next
  AOT-assume ‹∃w w ⊨ p›
  then AOT-obtain w where w-prop: ‹w ⊨ p›
    using PossibleWorld.∃E[rotated] by meson
```

AOT-hence ‹◇∀ *p* (*w* ⊨ *p* ≡ *p*)›
  using *world*:*1*[*THEN* $\equiv_{df}$*E*, *THEN* &*E*(*2*)] *PossibleWorld*.*ψ* &*E* **by** *blast*
AOT-hence ‹∀ *p* ◇(*w* ⊨ *p* ≡ *p*)›
  by (*metis Buridan*◇ →*E*)
AOT-hence *1*: ‹◇(*w* ⊨ *p* ≡ *p*)›
  by (*metis log*−*prop*−*prop*:*2 rule*−*ui*:*1*)
AOT-have ‹◇((*w* ⊨ *p* → *p*) & (*p* → *w* ⊨ *p*))›
  apply (*AOT-subst* ‹(*w* ⊨ *p* → *p*) & (*p* → *w* ⊨ *p*)› ‹*w* ⊨ *p* ≡ *p*›)
   apply (*meson conventions*:*3* ≡*E*(*6*) *oth*−*class*−*taut*:*3*:*a* ≡*Df*)
  by (*fact 1*)
AOT-hence ‹◇(*w* ⊨ *p* → *p*)›
  by (*metis RM*◇ *Conjunction Simplification*(*1*) →*E*)
moreover AOT-have ‹□(*w* ⊨ *p*)›
  using *w-prop* **by** (*metis* ≡*E*(*1*) *rigid*−*truth*−*at*:*1*)
ultimately AOT-show ‹◇*p*›
  by (*metis KBasic2*:*4* ≡*E*(*1*) →*E*)
qed

AOT-theorem *fund*:*2*: ‹□*p* ≡ ∀ *w* (*w* ⊨ *p*)›
proof −
AOT-have *0*: ‹∀ *w* (*w* ⊨ ¬*p* ≡ ¬*w* ⊨ *p*)›
  apply (*rule PossibleWorld*.*GEN*)
  using *coherent*:*1* **by** *blast*
AOT-have ‹◇¬*p* ≡ ∃ *w* (*w* ⊨ ¬*p*)›
  using *fund*:*1*[*unvarify p*, *OF log*−*prop*−*prop*:*2*] **by** *blast*
also AOT-have ‹. . . ≡ ∃ *w* ¬(*w* ⊨ *p*)›
proof(*safe intro*!: ≡*I* →*I*)
  AOT-assume ‹∃ *w w* ⊨ ¬*p*›
  then AOT-obtain *w* **where** *w-prop*: ‹*w* ⊨ ¬*p*›
    using *PossibleWorld*.∃ *E*[*rotated*] **by** *meson*
  AOT-hence ‹¬*w* ⊨ *p*›
    using *0*[*THEN PossibleWorld*.∀ *E*, *THEN* ≡*E*(*1*)] &*E* **by** *blast*
  AOT-thus ‹∃ *w* ¬*w* ⊨ *p*›
    by (*rule PossibleWorld*.∃ *I*)
next
  AOT-assume ‹∃ *w* ¬*w* ⊨ *p*›
  then AOT-obtain *w* **where** *w-prop*: ‹¬*w* ⊨ *p*›
    using *PossibleWorld*.∃ *E*[*rotated*] **by** *meson*
  AOT-hence ‹*w* ⊨ ¬*p*›
    using *0*[*THEN* ∀ *E*(*2*), *THEN* →*E*, *THEN* ≡*E*(*1*)] &*E*
    by (*metis coherent*:*1* ≡*E*(*2*))
  AOT-thus ‹∃ *w w* ⊨ ¬*p*›
    by (*rule PossibleWorld*.∃ *I*)
qed
finally AOT-have ‹¬◇¬*p* ≡ ¬∃ *w* ¬*w* ⊨ *p*›
  by (*meson* ≡*E*(*1*) *oth*−*class*−*taut*:*4*:*b*)
AOT-hence ‹□*p* ≡ ¬∃ *w* ¬*w* ⊨ *p*›
  by (*metis KBasic*:*12* ≡*E*(*5*))
also AOT-have ‹. . . ≡ ∀ *w w* ⊨ *p*›
proof(*safe intro*!: ≡*I* →*I*)
  AOT-assume ‹¬∃ *w* ¬*w* ⊨ *p*›
  AOT-hence *0*: ‹∀ *x* (¬(*PossibleWorld*(*x*) & ¬*x* ⊨ *p*))›
    by (*metis cqt*−*further*:*4* →*E*)
  AOT-show ‹∀ *w w* ⊨ *p*›
    apply (*AOT-subst* ‹*PossibleWorld*(*x*) → *x* ⊨ *p*›
                ‹¬(*PossibleWorld*(*x*) & ¬*x* ⊨ *p*)› **for**: *x*)
     using *oth*−*class*−*taut*:*1*:*a* **apply** *presburger*
    by (*fact 0*)
next
  AOT-assume *0*: ‹∀ *w w* ⊨ *p*›
  AOT-have ‹∀ *x* (¬(*PossibleWorld*(*x*) & ¬*x* ⊨ *p*))›
    by (*AOT-subst* (*reverse*) ‹¬(*PossibleWorld*(*x*) & ¬*x* ⊨ *p*)›
                ‹*PossibleWorld*(*x*) → *x* ⊨ *p*› **for**: *x*)

(*auto simp*: *oth−class−taut*:*1*:*a* *0*)
AOT-thus ‹¬∃ *w* ¬*w* ⊨ *p*›
by (*metis* ∃ *E* *raa−cor*:*3* *rule−ui*:*3*)
qed
finally AOT-show ‹□*p* ≡ ∀ *w* *w* ⊨ *p*›.
qed

AOT-theorem *fund*:*3*: ‹¬◇*p* ≡ ¬∃ *w* *w* ⊨ *p*›
by (*metis* (*full-types*) *contraposition*:*1*[*1*] →*I* *fund*:*1* ≡*I* ≡*E*(*1*,*2*))

AOT-theorem *fund*:*4*: ‹¬□*p* ≡ ∃ *w* ¬*w* ⊨*p*›
apply (*AOT-subst* ‹∃ *w* ¬*w* ⊨ *p*› ‹¬ ∀ *w* *w* ⊨ *p*›)
apply (*AOT-subst* ‹*PossibleWorld*(*x*) → *x* ⊨ *p*›
‹¬(*PossibleWorld*(*x*) & ¬*x* ⊨ *p*)› **for**: *x*)
by (*auto simp add*: *oth−class−taut*:*1*:*a* *conventions*:*4* ≡*Df* *RN*
fund:*2* *rule−sub−lem*:*1*:*a*)

AOT-theorem *nec−dia−w*:*1*: ‹□*p* ≡ ∃ *w* *w* ⊨ □*p*›
proof −
AOT-have ‹□*p* ≡ ◇□*p*›
using *S5Basic*:*2* **by** *blast*
also AOT-have ‹... ≡ ∃ *w* *w* ⊨ □*p*›
using *fund*:*1*[*unvarify p*, *OF log−prop−prop*:*2*] **by** *blast*
finally show *?thesis*.
qed

AOT-theorem *nec−dia−w*:*2*: ‹□*p* ≡ ∀ *w* *w* ⊨ □*p*›
proof −
AOT-have ‹□*p* ≡ □□*p*›
using *4* *qml*:*2*[*axiom-inst*] ≡*I* **by** *blast*
also AOT-have ‹... ≡ ∀ *w* *w* ⊨ □*p*›
using *fund*:*2*[*unvarify p*, *OF log−prop−prop*:*2*] **by** *blast*
finally show *?thesis*.
qed

AOT-theorem *nec−dia−w*:*3*: ‹◇*p* ≡ ∃ *w* *w* ⊨ ◇*p*›
proof −
AOT-have ‹◇*p* ≡ ◇◇*p*›
by (*simp add*: *4*◇ *T*◇ ≡*I*)
also AOT-have ‹... ≡ ∃ *w* *w* ⊨ ◇*p*›
using *fund*:*1*[*unvarify p*, *OF log−prop−prop*:*2*] **by** *blast*
finally show *?thesis*.
qed

AOT-theorem *nec−dia−w*:*4*: ‹◇*p* ≡ ∀ *w* *w* ⊨ ◇*p*›
proof −
AOT-have ‹◇*p* ≡ □◇*p*›
by (*simp add*: *S5Basic*:*1*)
also AOT-have ‹... ≡ ∀ *w* *w* ⊨ ◇*p*›
using *fund*:*2*[*unvarify p*, *OF log−prop−prop*:*2*] **by** *blast*
finally show *?thesis*.
qed

AOT-theorem *conj−dist−w*:*1*: ‹*w* ⊨ (*p* & *q*) ≡ ((*w* ⊨ *p*) & (*w* ⊨ *q*))›
proof(*safe intro*!: ≡*I* →*I*)
AOT-assume ‹*w* ⊨ (*p* & *q*)›
AOT-hence *0*: ‹□*w* ⊨ (*p* & *q*)›
using *rigid−truth−at*:*1*[*unvarify p*, *THEN* ≡*E*(*1*), *OF log−prop−prop*:*2*]
by *blast*
AOT-modally-strict {
AOT-have ‹∀ *p* (*w* ⊨ *p* ≡ *p*) → ((*w* ⊨ (φ & ψ)) → (*w* ⊨ φ & *w* ⊨ ψ))› **for** *w* φ ψ
proof(*safe intro*!: →*I*)
AOT-assume ‹∀ *p* (*w* ⊨ *p* ≡ *p*)›

AOT-hence ‹$w \models (\varphi \ \& \ \psi) \equiv (\varphi \ \& \ \psi)$› **and** ‹$w \models \varphi \equiv \varphi$› **and** ‹$w \models \psi \equiv \psi$›
using $\forall E(1)$[*rotated*, *OF log−prop−prop*:*2*] **by** *blast+*
moreover AOT-assume ‹$w \models (\varphi \ \& \ \psi)$›
ultimately AOT-show ‹$w \models \varphi \ \& \ w \models \psi$›
by (*metis* &*I* &*E*(*1*) &*E*(*2*) ≡*E*(*1*) ≡*E*(*2*))
qed
}
AOT-hence ‹$\Diamond\forall p \ (w \models p \equiv p) \rightarrow \Diamond(w \models (\varphi \ \& \ \psi) \rightarrow w \models \varphi \ \& \ w \models \psi)$› **for** $w \ \varphi \ \psi$
by (*rule RM*◇)
moreover AOT-have *pos*: ‹$\Diamond\forall p \ (w \models p \equiv p)$›
using *world*:*1*[*THEN* $\equiv_{df}E$, *OF PossibleWorld*.ψ] &*E* **by** *blast*
ultimately AOT-have ‹$\Diamond(w \models (p \ \& \ q) \rightarrow w \models p \ \& \ w \models q)$› **using** →*E* **by** *blast*
AOT-hence ‹$\Diamond(w \models p) \ \& \ \Diamond(w \models q)$›
by (*metis 0 KBasic2*:*3 KBasic2*:*4* ≡*E*(*1*) *vdash−properties*:*10*)
AOT-thus ‹$w \models p \ \& \ w \models q$›
using *rigid−truth−at*:*2*[*unvarify p*, *THEN* ≡*E*(*1*), *OF log−prop−prop*:*2*]
&*E* &*I* **by** *meson*
next
AOT-assume ‹$w \models p \ \& \ w \models q$›
AOT-hence ‹$\Box w \models p \ \& \ \Box w \models q$›
using *rigid−truth−at*:*1*[*unvarify p*, *THEN* ≡*E*(*1*), *OF log−prop−prop*:*2*]
&*E* &*I* **by** *blast*
AOT-hence *0*: ‹$\Box(w \models p \ \& \ w \models q)$›
by (*metis KBasic*:*3* ≡*E*(*2*))
AOT-modally-strict {
AOT-have ‹$\forall p \ (w \models p \equiv p) \rightarrow ((w \models \varphi \ \& \ w \models \psi) \rightarrow (w \models (\varphi \ \& \ \psi)))$› **for** $w \ \varphi \ \psi$
proof(*safe intro*!: →*I*)
AOT-assume ‹$\forall \ p \ (w \models p \equiv p)$›
AOT-hence ‹$w \models (\varphi \ \& \ \psi) \equiv (\varphi \ \& \ \psi)$› **and** ‹$w \models \varphi \equiv \varphi$› **and** ‹$w \models \psi \equiv \psi$›
using $\forall E(1)$[*rotated*, *OF log−prop−prop*:*2*] **by** *blast+*
moreover AOT-assume ‹$w \models \varphi \ \& \ w \models \psi$›
ultimately AOT-show ‹$w \models (\varphi \ \& \ \psi)$›
by (*metis* &*I* &*E*(*1*) &*E*(*2*) ≡*E*(*1*) ≡*E*(*2*))
qed
}
AOT-hence ‹$\Diamond\forall p \ (w \models p \equiv p) \rightarrow \Diamond((w \models \varphi \ \& \ w \models \psi) \rightarrow w \models (\varphi \ \& \ \psi))$› **for** $w \ \varphi \ \psi$
by (*rule RM*◇)
moreover AOT-have *pos*: ‹$\Diamond\forall p \ (w \models p \equiv p)$›
using *world*:*1*[*THEN* $\equiv_{df}E$, *OF PossibleWorld*.ψ] &*E* **by** *blast*
ultimately AOT-have ‹$\Diamond((w \models p \ \& \ w \models q) \rightarrow w \models (p \ \& \ q))$›
using →*E* **by** *blast*
AOT-hence ‹$\Diamond(w \models (p \ \& \ q))$›
by (*metis 0 KBasic2*:*4* ≡*E*(*1*) *vdash−properties*:*10*)
AOT-thus ‹$w \models (p \ \& \ q)$›
using *rigid−truth−at*:*2*[*unvarify p*, *THEN* ≡*E*(*1*), *OF log−prop−prop*:*2*]
by *blast*
qed

AOT-theorem *conj−dist−w*:*2*: ‹$w \models (p \rightarrow q) \equiv ((w \models p) \rightarrow (w \models q))$›
proof(*safe intro*!: ≡*I* →*I*)
AOT-assume ‹$w \models (p \rightarrow q)$›
AOT-hence *0*: ‹$\Box w \models (p \rightarrow q)$›
using *rigid−truth−at*:*1*[*unvarify p*, *THEN* ≡*E*(*1*), *OF log−prop−prop*:*2*]
by *blast*
AOT-assume ‹$w \models p$›
AOT-hence *1*: ‹$\Box w \models p$›
by (*metis T*◇ ≡*E*(*1*) *rigid−truth−at*:*3* →*E*)
AOT-modally-strict {
AOT-have ‹$\forall p \ (w \models p \equiv p) \rightarrow ((w \models (\varphi \rightarrow \psi)) \rightarrow (w \models \varphi \rightarrow w \models \psi))$› **for** $w \ \varphi \ \psi$
proof(*safe intro*!: →*I*)
AOT-assume ‹$\forall \ p \ (w \models p \equiv p)$›
AOT-hence ‹$w \models (\varphi \rightarrow \psi) \equiv (\varphi \rightarrow \psi)$› **and** ‹$w \models \varphi \equiv \varphi$› **and** ‹$w \models \psi \equiv \psi$›
using $\forall E(1)$[*rotated*, *OF log−prop−prop*:*2*] **by** *blast+*

```
      moreover AOT-assume ‹w ⊨ (φ → ψ)›
      moreover AOT-assume ‹w ⊨ φ›
      ultimately AOT-show ‹w ⊨ ψ›
        by (metis ≡E(1) ≡E(2) →E)
    qed
  }
  AOT-hence ‹◇∀p (w ⊨ p ≡ p) → ◇(w ⊨ (φ → ψ) → (w ⊨ φ → w ⊨ ψ))› for w φ ψ
    by (rule RM◇)
  moreover AOT-have pos: ‹◇∀p (w ⊨ p ≡ p)›
    using world:1[THEN ≡dfE, OF PossibleWorld.ψ] &E by blast
  ultimately AOT-have ‹◇(w ⊨ (p → q) → (w ⊨ p → w ⊨ q))›
    using →E by blast
  AOT-hence ‹◇(w ⊨ p → w ⊨ q)›
    by (metis 0 KBasic2:4 ≡E(1) →E)
  AOT-hence ‹◇w ⊨ q›
    by (metis 1 KBasic2:4 ≡E(1) →E)
  AOT-thus ‹w ⊨ q›
    using rigid−truth−at:2[unvarify p, THEN ≡E(1), OF log−prop−prop:2]
          &E &I by meson
next
  AOT-assume ‹w ⊨ p → w ⊨ q›
  AOT-hence ‹¬(w ⊨ p) ∨ w ⊨ q›
    by (metis ∨I(1) ∨I(2) reductio−aa:1 →E)
  AOT-hence ‹w ⊨ ¬p ∨ w ⊨ q›
    by (metis coherent:1 ∨I(1) ∨I(2) ∨E(2) ≡E(2) reductio−aa:1)
  AOT-hence 0: ‹□(w ⊨ ¬p ∨ w ⊨ q)›
    using rigid−truth−at:1[unvarify p, THEN ≡E(1), OF log−prop−prop:2]
    by (metis KBasic:15 ∨I(1) ∨I(2) ∨E(2) reductio−aa:1 →E)
  AOT-modally-strict {
    AOT-have ‹∀p (w ⊨ p ≡ p) → ((w ⊨ ¬φ ∨ w ⊨ ψ) → (w ⊨ (φ → ψ)))› for w φ ψ
    proof(safe intro!: →I)
      AOT-assume ‹∀ p (w ⊨ p ≡ p)›
      moreover AOT-assume ‹w ⊨ ¬φ ∨ w ⊨ ψ›
      ultimately AOT-show ‹w ⊨ (φ → ψ)›
        by (metis ∨E(2) →I ≡E(1) ≡E(2) log−prop−prop:2
                reductio−aa:1 rule−ui:1)
    qed
  }
  AOT-hence ‹◇∀p (w ⊨ p ≡ p) → ◇((w ⊨ ¬φ ∨ w ⊨ ψ) → w ⊨ (φ → ψ))› for w φ ψ
    by (rule RM◇)
  moreover AOT-have pos: ‹◇∀p (w ⊨ p ≡ p)›
    using world:1[THEN ≡dfE, OF PossibleWorld.ψ] &E by blast
  ultimately AOT-have ‹◇((w ⊨ ¬p ∨ w ⊨ q) → w ⊨ (p → q))›
    using →E by blast
  AOT-hence ‹◇(w ⊨ (p → q))›
    by (metis 0 KBasic2:4 ≡E(1) →E)
  AOT-thus ‹w ⊨ (p → q)›
    using rigid−truth−at:2[unvarify p, THEN ≡E(1), OF log−prop−prop:2]
    by blast
qed

AOT-theorem conj−dist−w:3: ‹w ⊨ (p ∨ q) ≡ ((w ⊨ p) ∨ (w ⊨ q))›
proof(safe intro!: ≡I →I)
  AOT-assume ‹w ⊨ (p ∨ q)›
  AOT-hence 0: ‹□w ⊨ (p ∨ q)›
    using rigid−truth−at:1[unvarify p, THEN ≡E(1), OF log−prop−prop:2]
    by blast
  AOT-modally-strict {
    AOT-have ‹∀p (w ⊨ p ≡ p) → ((w ⊨ (φ ∨ ψ)) → (w ⊨ φ ∨ w ⊨ ψ))› for w φ ψ
    proof(safe intro!: →I)
      AOT-assume ‹∀ p (w ⊨ p ≡ p)›
      AOT-hence ‹w ⊨ (φ ∨ ψ) ≡ (φ ∨ ψ)› and ‹w ⊨ φ ≡ φ› and ‹w ⊨ ψ ≡ ψ›
        using ∀E(1)[rotated, OF log−prop−prop:2] by blast+
```

    moreover AOT-assume ‹$w \models (\varphi \vee \psi)$›
    ultimately AOT-show ‹$w \models \varphi \vee w \models \psi$›
      by (*metis* $\vee I(1)$ $\vee I(2)$ $\vee E(3)$ $\equiv E(1)$ $\equiv E(2)$ *reductio−aa*:*1*)
  qed
}
AOT-hence ‹$\Diamond\forall p\ (w \models p \equiv p) \rightarrow \Diamond(w \models (\varphi \vee \psi) \rightarrow (w \models \varphi \vee w \models \psi))$› **for** $w\ \varphi\ \psi$
  by (*rule RM*$\Diamond$)
moreover AOT-have *pos*: ‹$\Diamond\forall p\ (w \models p \equiv p)$›
  using *world*:*1*[*THEN* $\equiv_{df}E$, *OF PossibleWorld*.ψ] &*E* **by** *blast*
ultimately AOT-have ‹$\Diamond(w \models (p \vee q) \rightarrow (w \models p \vee w \models q))$› **using** $\rightarrow E$ **by** *blast*
AOT-hence ‹$\Diamond(w \models p \vee w \models q)$›
  by (*metis 0 KBasic2*:*4* $\equiv E(1)$ *vdash−properties*:*10*)
AOT-hence ‹$\Diamond w \models p \vee \Diamond w \models q$›
  using *KBasic2*:*2*[*THEN* $\equiv E(1)$] **by** *blast*
AOT-thus ‹$w \models p \vee w \models q$›
  using *rigid−truth−at*:*2*[*unvarify p*, *THEN* $\equiv E(1)$, *OF log−prop−prop*:*2*]
  by (*metis* $\vee I(1)$ $\vee I(2)$ $\vee E(2)$ *reductio−aa*:*1*)
next
AOT-assume ‹$w \models p \vee w \models q$›
AOT-hence *0*: ‹$\Box(w \models p \vee w \models q)$›
  using *rigid−truth−at*:*1*[*unvarify p*, *THEN* $\equiv E(1)$, *OF log−prop−prop*:*2*]
  by (*metis KBasic*:*15* $\vee I(1)$ $\vee I(2)$ $\vee E(2)$ *reductio−aa*:*1* $\rightarrow E$)
AOT-modally-strict {
  AOT-have ‹$\forall p\ (w \models p \equiv p) \rightarrow ((w \models \varphi \vee w \models \psi) \rightarrow (w \models (\varphi \vee \psi)))$› **for** $w\ \varphi\ \psi$
  proof(*safe intro*!: $\rightarrow I$)
    AOT-assume ‹$\forall\ p\ (w \models p \equiv p)$›
    moreover AOT-assume ‹$w \models \varphi \vee w \models \psi$›
    ultimately AOT-show ‹$w \models (\varphi \vee \psi)$›
      by (*metis* $\vee I(1)$ $\vee I(2)$ $\vee E(2)$ $\equiv E(1)$ $\equiv E(2)$
            log−prop−prop:*2* *reductio−aa*:*1* *rule−ui*:*1*)
  qed
}
AOT-hence ‹$\Diamond\forall p\ (w \models p \equiv p) \rightarrow \Diamond((w \models \varphi \vee w \models \psi) \rightarrow w \models (\varphi \vee \psi))$› **for** $w\ \varphi\ \psi$
  by (*rule RM*$\Diamond$)
moreover AOT-have *pos*: ‹$\Diamond\forall p\ (w \models p \equiv p)$›
  using *world*:*1*[*THEN* $\equiv_{df}E$, *OF PossibleWorld*.ψ] &*E* **by** *blast*
ultimately AOT-have ‹$\Diamond((w \models p \vee w \models q) \rightarrow w \models (p \vee q))$›
  using $\rightarrow E$ **by** *blast*
AOT-hence ‹$\Diamond(w \models (p \vee q))$›
  by (*metis 0 KBasic2*:*4* $\equiv E(1)$ $\rightarrow E$)
AOT-thus ‹$w \models (p \vee q)$›
  using *rigid−truth−at*:*2*[*unvarify p*, *THEN* $\equiv E(1)$, *OF log−prop−prop*:*2*]
  by *blast*
qed

AOT-theorem *conj−dist−w*:*4*: ‹$w \models (p \equiv q) \equiv ((w \models p) \equiv (w \models q))$›
proof(*rule* $\equiv I$; *rule* $\rightarrow I$)
  AOT-assume ‹$w \models (p \equiv q)$›
  AOT-hence *0*: ‹$\Box w \models (p \equiv q)$›
    using *rigid−truth−at*:*1*[*unvarify p*, *THEN* $\equiv E(1)$, *OF log−prop−prop*:*2*]
    by *blast*
  AOT-modally-strict {
    AOT-have ‹$\forall p\ (w \models p \equiv p) \rightarrow ((w \models (\varphi \equiv \psi)) \rightarrow (w \models \varphi \equiv w \models \psi))$› **for** $w\ \varphi\ \psi$
    proof(*safe intro*!: $\rightarrow I$)
      AOT-assume ‹$\forall\ p\ (w \models p \equiv p)$›
      AOT-hence ‹$w \models (\varphi \equiv \psi) \equiv (\varphi \equiv \psi)$› **and** ‹$w \models \varphi \equiv \varphi$› **and** ‹$w \models \psi \equiv \psi$›
        using $\forall E(1)$[*rotated*, *OF log−prop−prop*:*2*] **by** *blast+*
      moreover AOT-assume ‹$w \models (\varphi \equiv \psi)$›
      ultimately AOT-show ‹$w \models \varphi \equiv w \models \psi$›
        by (*metis* $\equiv E(2)$ $\equiv E(5)$ *Commutativity of* $\equiv$)
    qed
  }
  AOT-hence ‹$\Diamond\forall p\ (w \models p \equiv p) \rightarrow \Diamond(w \models (\varphi \equiv \psi) \rightarrow (w \models \varphi \equiv w \models \psi))$› **for** $w\ \varphi\ \psi$

```
      by (rule RM◇)
    moreover AOT-have pos: ‹◇∀p (w ⊨ p ≡ p)›
      using world:1[THEN ≡df E, OF PossibleWorld.ψ] &E by blast
    ultimately AOT-have ‹◇(w ⊨ (p ≡ q) → (w ⊨ p ≡ w ⊨ q))›
      using →E by blast
    AOT-hence 1: ‹◇(w ⊨ p ≡ w ⊨ q)›
      by (metis 0 KBasic2:4 ≡E(1) vdash−properties:10)
    AOT-have ‹◇((w ⊨ p → w ⊨ q) & (w ⊨ q → w ⊨ p))›
      apply (AOT-subst ‹(w ⊨ p → w ⊨ q) & (w ⊨ q → w ⊨ p)› ‹w ⊨ p ≡ w ⊨ q›)
       apply (meson ≡df E conventions:3 →I df−rules−formulas[4] ≡I)
      by (fact 1)
    AOT-hence 2: ‹◇(w ⊨ p → w ⊨ q) & ◇(w ⊨ q → w ⊨ p)›
      by (metis KBasic2:3 vdash−properties:10)
    AOT-have ‹◇(¬w ⊨ p ∨ w ⊨ q)› and ‹◇(¬w ⊨ q ∨ w ⊨ p)›
       apply (AOT-subst (reverse) ‹¬w ⊨ p ∨ w ⊨ q› ‹w ⊨ p → w ⊨ q›)
        apply (simp add: oth−class−taut:1:c)
       apply (fact 2[THEN &E(1)])
      apply (AOT-subst (reverse) ‹¬w ⊨ q ∨ w ⊨ p› ‹w ⊨ q → w ⊨ p›)
       apply (simp add: oth−class−taut:1:c)
      by (fact 2[THEN &E(2)])
    AOT-hence ‹◇(¬w ⊨ p) ∨ ◇w ⊨ q› and ‹◇¬w ⊨ q ∨ ◇w ⊨ p›
      using KBasic2:2 ≡E(1) by blast+
    AOT-hence ‹¬□w ⊨ p ∨ ◇w ⊨ q› and ‹¬□w ⊨ q ∨ ◇w ⊨ p›
      by (metis KBasic:11 ∨I(1) ∨I(2) ∨E(2) ≡E(2) raa−cor:1)+
    AOT-thus ‹w ⊨ p ≡ w ⊨ q›
      using rigid−truth−at:2[unvarify p, THEN ≡E(1), OF log−prop−prop:2]
      by (metis ¬¬I T◇ ∨E(2) →I ≡I ≡E(1) rigid−truth−at:3)
  next
    AOT-have ‹□PossibleWorld(w)›
      using ≡E(1) rigid−pw:1 PossibleWorld.ψ by blast
    moreover {
      fix p
      AOT-modally-strict {
        AOT-have ‹PossibleWorld(w) → (w ⊨ p → □w ⊨ p)›
          using rigid−truth−at:1 →I
          by (metis ≡E(1))
      }
      AOT-hence ‹□PossibleWorld(w) → □(w ⊨ p → □w ⊨ p)›
        by (rule RM)
    }
    ultimately AOT-have 1: ‹□(w ⊨ p → □w ⊨ p)› for p
      by (metis →E)
    AOT-assume ‹w ⊨ p ≡ w ⊨ q›
    AOT-hence 0: ‹□(w ⊨ p ≡ w ⊨ q)›
      using sc−eq−box−box:5[THEN →E, THEN qml:2[axiom-inst, THEN →E],
                            THEN →E, OF &I]
          by (metis 1)
    AOT-modally-strict {
      AOT-have ‹∀p (w ⊨ p ≡ p) → ((w ⊨ φ ≡ w ⊨ ψ) → (w ⊨ (φ ≡ ψ)))› for w φ ψ
      proof(safe intro!: →I)
        AOT-assume ‹∀ p (w ⊨ p ≡ p)›
        moreover AOT-assume ‹w ⊨ φ ≡ w ⊨ ψ›
        ultimately AOT-show ‹w ⊨ (φ ≡ ψ)›
          by (metis ≡E(2) ≡E(6) log−prop−prop:2 rule−ui:1)
      qed
    }
    AOT-hence ‹◇∀p (w ⊨ p ≡ p) → ◇((w ⊨ φ ≡ w ⊨ ψ) → w ⊨ (φ ≡ ψ))› for w φ ψ
      by (rule RM◇)
    moreover AOT-have pos: ‹◇∀p (w ⊨ p ≡ p)›
      using world:1[THEN ≡df E, OF PossibleWorld.ψ] &E by blast
    ultimately AOT-have ‹◇((w ⊨ p ≡  w ⊨ q) → w ⊨ (p ≡ q))›
      using →E by blast
    AOT-hence ‹◇(w ⊨ (p ≡ q))›
```

  by (*metis 0 KBasic2:4* $\equiv E(1)$ $\rightarrow E$)
 AOT-thus ‹$w \models (p \equiv q)$›
  using *rigid−truth−at:2*[*unvarify p*, *THEN* $\equiv E(1)$, *OF log−prop−prop:2*]
  by *blast*
qed

AOT-theorem *conj−dist−w:5*: ‹$w \models (\forall \alpha\ \varphi\{\alpha\}) \equiv (\forall\ \alpha\ (w \models \varphi\{\alpha\}))$›
proof(*safe intro*!: $\equiv I$ $\rightarrow I$ *GEN*)
 AOT-assume ‹$w \models (\forall \alpha\ \varphi\{\alpha\})$›
 AOT-hence *0*: ‹$\Box w \models (\forall \alpha\ \varphi\{\alpha\})$›
  using *rigid−truth−at:1*[*unvarify p*, *THEN* $\equiv E(1)$, *OF log−prop−prop:2*]
  by *blast*
 AOT-modally-strict {
  AOT-have ‹$\forall p\ (w \models p \equiv p) \rightarrow ((w \models (\forall \alpha\ \varphi\{\alpha\})) \rightarrow (\forall \alpha\ w \models \varphi\{\alpha\}))$› **for** w
  proof(*safe intro*!: $\rightarrow I$ *GEN*)
   AOT-assume ‹$\forall p\ (w \models p \equiv p)$›
   moreover AOT-assume ‹$w \models (\forall \alpha\ \varphi\{\alpha\})$›
   ultimately AOT-show ‹$w \models \varphi\{\alpha\}$› **for** α
    by (*metis* $\equiv E(1)$ $\equiv E(2)$ *log−prop−prop:2 rule−ui:1 rule−ui:3*)
  qed
 }
 AOT-hence ‹$\Diamond\forall p\ (w \models p \equiv p) \rightarrow \Diamond(w \models (\forall \alpha\ \varphi\{\alpha\}) \rightarrow (\forall \alpha\ w \models \varphi\{\alpha\}))$› **for** w
  by (*rule RM*$\Diamond$)
 moreover AOT-have *pos*: ‹$\Diamond\forall p\ (w \models p \equiv p)$›
  using *world:1*[*THEN* $\equiv_{df}E$, *OF PossibleWorld.*ψ] $\&E$ **by** *blast*
 ultimately AOT-have ‹$\Diamond(w \models (\forall \alpha\ \varphi\{\alpha\}) \rightarrow (\forall \alpha\ w \models \varphi\{\alpha\}))$› **using** $\rightarrow E$ **by** *blast*
 AOT-hence ‹$\Diamond(\forall \alpha\ w \models \varphi\{\alpha\})$›
  by (*metis 0 KBasic2:4* $\equiv E(1)$ $\rightarrow E$)
 AOT-hence ‹$\forall \alpha\ \Diamond w \models \varphi\{\alpha\}$›
  by (*metis Buridan*$\Diamond$ $\rightarrow E$)
 AOT-thus ‹$w \models \varphi\{\alpha\}$› **for** α
  using *rigid−truth−at:2*[*unvarify p*, *THEN* $\equiv E(1)$, *OF log−prop−prop:2*]
     $\forall E(2)$ **by** *blast*
next
 AOT-assume ‹$\forall \alpha\ w \models \varphi\{\alpha\}$›
 AOT-hence ‹$w \models \varphi\{\alpha\}$› **for** α **using** $\forall E(2)$ **by** *blast*
 AOT-hence ‹$\Box w \models \varphi\{\alpha\}$› **for** α
  using *rigid−truth−at:1*[*unvarify p*, *THEN* $\equiv E(1)$, *OF log−prop−prop:2*]
     $\&E$ $\&I$ **by** *blast*
 AOT-hence ‹$\forall \alpha\ \Box w \models \varphi\{\alpha\}$› **by** (*rule GEN*)
 AOT-hence *0*: ‹$\Box\forall \alpha\ w \models \varphi\{\alpha\}$› **by** (*rule BF*[*THEN* $\rightarrow E$])
 AOT-modally-strict {
  AOT-have ‹$\forall p\ (w \models p \equiv p) \rightarrow ((\forall \alpha\ w \models \varphi\{\alpha\}) \rightarrow (w \models (\forall \alpha\ \varphi\{\alpha\})))$› **for** w
  proof(*safe intro*!: $\rightarrow I$)
   AOT-assume ‹$\forall\ p\ (w \models p \equiv p)$›
   moreover AOT-assume ‹$\forall \alpha\ w \models \varphi\{\alpha\}$›
   ultimately AOT-show ‹$w \models (\forall \alpha\ \varphi\{\alpha\})$›
    by (*metis* $\equiv E(1)$ $\equiv E(2)$ *log−prop−prop:2 rule−ui:1*
       rule−ui:3 universal−cor)
  qed
 }
 AOT-hence ‹$\Diamond\forall p\ (w \models p \equiv p) \rightarrow \Diamond((\forall \alpha\ w \models \varphi\{\alpha\}) \rightarrow w \models (\forall \alpha\ \varphi\{\alpha\}))$› **for** w
  by (*rule RM*$\Diamond$)
 moreover AOT-have *pos*: ‹$\Diamond\forall p\ (w \models p \equiv p)$›
  using *world:1*[*THEN* $\equiv_{df}E$, *OF PossibleWorld.*ψ] $\&E$ **by** *blast*
 ultimately AOT-have ‹$\Diamond((\forall \alpha\ w \models \varphi\{\alpha\}) \rightarrow w \models (\forall \alpha\ \varphi\{\alpha\}))$›
  using $\rightarrow E$ **by** *blast*
 AOT-hence ‹$\Diamond(w \models (\forall \alpha\ \varphi\{\alpha\}))$›
  by (*metis 0 KBasic2:4* $\equiv E(1)$ $\rightarrow E$)
 AOT-thus ‹$w \models (\forall \alpha\ \varphi\{\alpha\})$›
  using *rigid−truth−at:2*[*unvarify p*, *THEN* $\equiv E(1)$, *OF log−prop−prop:2*]
  by *blast*
qed

AOT-theorem *conj−dist−w*:*6*: ‹$w \models (\exists\alpha\ \varphi\{\alpha\}) \equiv (\exists\ \alpha\ (w \models \varphi\{\alpha\}))$›
proof(*safe intro*!: ≡*I* →*I GEN*)
  AOT-assume ‹$w \models (\exists\alpha\ \varphi\{\alpha\})$›
  AOT-hence *0*: ‹$\Box w \models (\exists\alpha\ \varphi\{\alpha\})$›
    using *rigid−truth−at*:*1*[*unvarify p*, *THEN* ≡*E*(*1*), *OF log−prop−prop*:*2*]
    by *blast*
  AOT-modally-strict {
    AOT-have ‹$\forall p\ (w \models p \equiv p) \rightarrow ((w \models (\exists\alpha\ \varphi\{\alpha\})) \rightarrow (\exists\alpha\ w \models \varphi\{\alpha\}))$› **for** *w*
    proof(*safe intro*!: →*I GEN*)
      AOT-assume ‹$\forall p\ (w \models p \equiv p)$›
      moreover AOT-assume ‹$w \models (\exists\alpha\ \varphi\{\alpha\})$›
      ultimately AOT-show ‹$\exists\ \alpha\ (w \models \varphi\{\alpha\})$›
        by (*metis* ∃*E* ∃*I*(*2*) ≡*E*(*1*,*2*) *log−prop−prop*:*2 rule−ui*:*1*)
    qed
  }
  AOT-hence ‹$\Diamond\forall p\ (w \models p \equiv p) \rightarrow \Diamond(w \models (\exists\alpha\ \varphi\{\alpha\}) \rightarrow (\exists\alpha\ w \models \varphi\{\alpha\}))$› **for** *w*
    by (*rule RM*◇)
  moreover AOT-have *pos*: ‹$\Diamond\forall p\ (w \models p \equiv p)$›
    using *world*:*1*[*THEN* $\equiv_{df}$*E*, *OF PossibleWorld*.*ψ*] &*E* **by** *blast*
  ultimately AOT-have ‹$\Diamond(w \models (\exists\alpha\ \varphi\{\alpha\}) \rightarrow (\exists\alpha\ w \models \varphi\{\alpha\}))$› **using** →*E* **by** *blast*
  AOT-hence ‹$\Diamond(\exists\alpha\ w \models \varphi\{\alpha\})$›
    by (*metis 0 KBasic2*:*4* ≡*E*(*1*) →*E*)
  AOT-hence ‹$\exists\alpha\ \Diamond w \models \varphi\{\alpha\}$›
    by (*metis BF*◇ →*E*)
  then AOT-obtain *α* **where** ‹$\Diamond w \models \varphi\{\alpha\}$›
    using ∃*E*[*rotated*] **by** *blast*
  AOT-hence ‹$w \models \varphi\{\alpha\}$›
    using *rigid−truth−at*:*2*[*unvarify p*, *THEN* ≡*E*(*1*), *OF log−prop−prop*:*2*] **by** *blast*
  AOT-thus ‹$\exists\ \alpha\ w \models \varphi\{\alpha\}$› **by** (*rule* ∃*I*)
next
  AOT-assume ‹$\exists\alpha\ w \models \varphi\{\alpha\}$›
  then AOT-obtain *α* **where** ‹$w \models \varphi\{\alpha\}$› **using** ∃*E*[*rotated*] **by** *blast*
  AOT-hence ‹$\Box w \models \varphi\{\alpha\}$›
    using *rigid−truth−at*:*1*[*unvarify p*, *THEN* ≡*E*(*1*), *OF log−prop−prop*:*2*]
        &*E* &*I* **by** *blast*
  AOT-hence ‹$\exists\alpha\ \Box w \models \varphi\{\alpha\}$›
    by (*rule* ∃*I*)
  AOT-hence *0*: ‹$\Box\exists\alpha\ w \models \varphi\{\alpha\}$›
    by (*metis Buridan* →*E*)
  AOT-modally-strict {
    AOT-have ‹$\forall p\ (w \models p \equiv p) \rightarrow ((\exists\alpha\ w \models \varphi\{\alpha\}) \rightarrow (w \models (\exists\alpha\ \varphi\{\alpha\})))$› **for** *w*
    proof(*safe intro*!: →*I*)
      AOT-assume ‹$\forall\ p\ (w \models p \equiv p)$›
      moreover AOT-assume ‹$\exists\alpha\ w \models \varphi\{\alpha\}$›
      then AOT-obtain *α* **where** ‹$w \models \varphi\{\alpha\}$›
        using ∃*E*[*rotated*] **by** *blast*
      ultimately AOT-show ‹$w \models (\exists\alpha\ \varphi\{\alpha\})$›
        by (*metis* ∃*I*(*2*) ≡*E*(*1*,*2*) *log−prop−prop*:*2 rule−ui*:*1*)
    qed
  }
  AOT-hence ‹$\Diamond\forall p\ (w \models p \equiv p) \rightarrow \Diamond((\exists\alpha\ w \models \varphi\{\alpha\}) \rightarrow w \models (\exists\alpha\ \varphi\{\alpha\}))$› **for** *w*
    by (*rule RM*◇)
  moreover AOT-have *pos*: ‹$\Diamond\forall p\ (w \models p \equiv p)$›
    using *world*:*1*[*THEN* $\equiv_{df}$*E*, *OF PossibleWorld*.*ψ*] &*E* **by** *blast*
  ultimately AOT-have ‹$\Diamond((\exists\alpha\ w \models \varphi\{\alpha\}) \rightarrow w \models (\exists\alpha\ \varphi\{\alpha\}))$›
    using →*E* **by** *blast*
  AOT-hence ‹$\Diamond(w \models (\exists\alpha\ \varphi\{\alpha\}))$›
    by (*metis 0 KBasic2*:*4* ≡*E*(*1*) →*E*)
  AOT-thus ‹$w \models (\exists\alpha\ \varphi\{\alpha\})$›
    using *rigid−truth−at*:*2*[*unvarify p*, *THEN* ≡*E*(*1*), *OF log−prop−prop*:*2*]
    by *blast*
qed

AOT-theorem *conj−dist−w:7*: ‹$(w \models \Box p) \rightarrow \Box w \models p$›
proof(*rule* →*I*)
  AOT-assume ‹$w \models \Box p$›
  AOT-hence ‹$\exists\, w\ w \models \Box p$› **by** (*rule PossibleWorld.∃ I*)
  AOT-hence ‹$\Diamond\Box p$› **using** *fund:1*[*unvarify p*, *OF log−prop−prop:2*, *THEN* ≡*E*(*2*)]
    by *blast*
  AOT-hence ‹$\Box p$›
    by (*metis 5*◇ →*E*)
  AOT-hence *1*: ‹$\Box\Box p$›
    by (*metis S5Basic:6* ≡*E*(*1*))
  AOT-have ‹$\Box\forall\, w\ w \models p$›
    by (*AOT-subst* (*reverse*) ‹$\forall\, w\ w \models p$› ‹$\Box p$›)
      (*auto simp add*: *fund:2 1*)
  AOT-hence ‹$\forall\, w\ \Box w \models p$›
    using *fund−lem:5*[*world*][*THEN* →*E*] **by** *simp*
  AOT-thus ‹$\Box w \models p$›
    using →*E PossibleWorld.*∀ *E* **by** *fast*
qed

AOT-theorem *conj−dist−w:8*: ‹$\exists\, w \exists\, p((\Box w \models p)\ \&\ \neg w \models \Box p)$›
proof −
  AOT-obtain *r* **where** *A*: *r* **and** ‹$\Diamond\neg r$›
    by (*metis* &*E*(*1*) &*E*(*2*) $\equiv_{df}E$ ∃ *E cont−tf:1 cont−tf−thm:1*)
  AOT-hence *B*: ‹$\neg\Box r$›
    by (*metis KBasic:11* ≡*E*(*2*))
  AOT-have ‹$\Diamond r$›
    using *A T*◇[*THEN* →*E*] **by** *simp*
  AOT-hence ‹$\exists\, w\ w \models r$›
    using *fund:1*[*THEN* ≡*E*(*1*)] **by** *blast*
  then AOT-obtain *w* **where** *w*: ‹$w \models r$›
    using *PossibleWorld.*∃ *E*[*rotated*] **by** *meson*
  AOT-hence ‹$\Box w \models r$›
    by (*metis T*◇ ≡*E*(*1*) *rigid−truth−at:3 vdash−properties:10*)
  moreover AOT-have ‹$\neg w \models \Box r$›
  proof(*rule raa−cor:2*)
    AOT-assume ‹$w \models \Box r$›
    AOT-hence ‹$\exists\, w\ w \models \Box r$›
      by (*rule PossibleWorld.∃ I*)
    AOT-hence ‹$\Box r$›
      by (*metis* ≡*E*(*2*) *nec−dia−w:1*)
    AOT-thus ‹$\Box r\ \&\ \neg\Box r$›
      using *B* &*I* **by** *blast*
  qed
  ultimately AOT-have ‹$\Box w \models r\ \&\ \neg w \models \Box r$›
    by (*rule* &*I*)
  AOT-hence ‹$\exists\, p\ (\Box w \models p\ \&\ \neg w \models \Box p)$›
    by (*rule* ∃ *I*)
  thus *?thesis*
    by (*rule PossibleWorld.∃ I*)
qed

AOT-theorem *conj−dist−w:9*: ‹$(\Diamond w \models p) \rightarrow w \models \Diamond p$›
proof(*rule* →*I*; *rule raa−cor:1*)
  AOT-assume ‹$\Diamond w \models p$›
  AOT-hence *0*: ‹$w \models p$›
    by (*metis* ≡*E*(*1*) *rigid−truth−at:2*)
  AOT-assume ‹$\neg w \models \Diamond p$›
  AOT-hence *1*: ‹$w \models \neg\Diamond p$›
    using *coherent:1*[*unvarify p*, *THEN* ≡*E*(*2*), *OF log−prop−prop:2*] **by** *blast*
  moreover AOT-have ‹$w \models (\neg\Diamond p \rightarrow \neg p)$›
    using *T*◇[*THEN contraposition:1*[*1*], *THEN RN*]
        fund:2[*unvarify p*, *OF log−prop−prop:2*, *THEN* ≡*E*(*1*), *THEN* ∀ *E*(*2*),

```
              THEN →E, rotated, OF PossibleWorld.ψ]
        by blast
  ultimately AOT-have ‹w ⊨ ¬p›
    using conj−dist−w:2[unvarify p q, OF log−prop−prop:2, OF log−prop−prop:2,
                        THEN ≡E(1), THEN →E]
    by blast
  AOT-hence ‹w ⊨ p & w ⊨ ¬p› using 0 &I by blast
  AOT-thus ‹p & ¬p›
    by (metis coherent:1 Conjunction Simplification(1,2) ≡E(4)
          modus−tollens:1 raa−cor:3)
qed

AOT-theorem conj−dist−w:10: ‹∃ w∃ p((w ⊨ ◇p) & ¬◇w ⊨ p)›
proof −
  AOT-obtain w where w: ‹∀ p (w ⊨ p ≡ p)›
    using act−world:1 PossibleWorld.∃ E[rotated] by meson
  AOT-obtain r where ‹¬r› and ‹◇r›
    using cont−tf−thm:2 cont−tf:2[THEN ≡df E] &E ∃ E[rotated] by metis
  AOT-hence ‹w ⊨ ¬r› and 0: ‹w ⊨ ◇r›
    using w[THEN ∀ E(1), OF log−prop−prop:2, THEN ≡E(2)] by blast+
  AOT-hence ‹¬w ⊨ r› using coherent:1[THEN ≡E(1)] by blast
  AOT-hence ‹¬◇w ⊨ r› by (metis ≡E(4) rigid−truth−at:2)
  AOT-hence ‹w ⊨ ◇r & ¬◇w ⊨ r› using 0 &I by blast
  AOT-hence ‹∃ p (w ⊨ ◇p & ¬◇w ⊨ p)› by (rule ∃ I)
  thus ?thesis by (rule PossibleWorld.∃ I)
qed

AOT-theorem two−worlds−exist:1: ‹∃ p(ContingentlyTrue(p)) → ∃ w (¬Actual(w))›
proof(rule →I)
  AOT-assume ‹∃ p ContingentlyTrue(p)›
  then AOT-obtain p where ‹ContingentlyTrue(p)›
    using ∃ E[rotated] by blast
  AOT-hence p: ‹p & ◇¬p›
    by (metis ≡df E cont−tf:1)
  AOT-hence ‹∃ w w ⊨ ¬p›
    using fund:1[unvarify p, OF log−prop−prop:2, THEN ≡E(1)] &E by blast
  then AOT-obtain w where w: ‹w ⊨ ¬p›
    using PossibleWorld.∃ E[rotated] by meson
  AOT-have ‹¬Actual(w)›
  proof(rule raa−cor:2)
    AOT-assume ‹Actual(w)›
    AOT-hence ‹w ⊨ p›
      using p[THEN &E(1)] actual[THEN ≡df E, THEN &E(2)]
      by (metis log−prop−prop:2 raa−cor:3 rule−ui:1 →E w)
    moreover AOT-have ‹¬(w ⊨ p)›
      by (metis coherent:1 ≡E(4) reductio−aa:2 w)
    ultimately AOT-show ‹w ⊨ p & ¬(w ⊨ p)›
      using &I by blast
  qed
  AOT-thus ‹∃ w ¬Actual(w)›
    by (rule PossibleWorld.∃ I)
qed


AOT-theorem two−worlds−exist:2: ‹∃ p(ContingentlyFalse(p)) → ∃ w (¬Actual(w))›
proof(rule →I)
  AOT-assume ‹∃ p ContingentlyFalse(p)›
  then AOT-obtain p where ‹ContingentlyFalse(p)›
    using ∃ E[rotated] by blast
  AOT-hence p: ‹¬p & ◇p›
    by (metis ≡df E cont−tf:2)
  AOT-hence ‹∃ w w ⊨ p›
    using fund:1[unvarify p, OF log−prop−prop:2, THEN ≡E(1)] &E by blast
```

```
  then AOT-obtain w where w: ‹w ⊨ p›
    using PossibleWorld.∃E[rotated] by meson
  moreover AOT-have ‹¬Actual(w)›
  proof(rule raa−cor:2)
    AOT-assume ‹Actual(w)›
    AOT-hence ‹w ⊨ ¬p›
      using p[THEN &E(1)] actual[THEN ≡df E, THEN &E(2)]
      by (metis log−prop−prop:2 raa−cor:3 rule−ui:1 →E w)
    moreover AOT-have ‹¬(w ⊨ p)›
      using calculation by (metis coherent:1 ≡E(4) reductio−aa:2)
    AOT-thus ‹w ⊨ p & ¬(w ⊨ p)›
      using &I w by metis
  qed
  AOT-thus ‹∃w ¬Actual(w)›
    by (rule PossibleWorld.∃I)
qed

AOT-theorem two−worlds−exist:3: ‹∃w ¬Actual(w)›
  using cont−tf−thm:1 two−worlds−exist:1 →E by blast

AOT-theorem two−worlds−exist:4: ‹∃w∃w′(w ≠ w′)›
proof −
  AOT-obtain w where w: ‹Actual(w)›
    using act−world:2[THEN uniqueness:1[THEN ≡df E],
                      THEN cqt−further:5[THEN →E]]
          PossibleWorld.∃E[rotated] &E
    by blast
  moreover AOT-obtain w′ where w′: ‹¬Actual(w′)›
    using two−worlds−exist:3 PossibleWorld.∃E[rotated] by meson
  AOT-have ‹¬(w = w′)›
  proof(rule raa−cor:2)
    AOT-assume ‹w = w′›
    AOT-thus ‹p & ¬p› for p
      using w w′ &E by (metis rule=E raa−cor:3)
  qed
  AOT-hence ‹w ≠ w′›
    by (metis ≡df I =−infix)
  AOT-hence ‹∃w′ w ≠ w′›
    by (rule PossibleWorld.∃I)
  thus ?thesis
    by (rule PossibleWorld.∃I)
qed


AOT-theorem w−rel:1: ‹[λx φ{x}]↓ → [λx w ⊨ φ{x}]↓›
proof(rule →I)
  AOT-assume ‹[λx φ{x}]↓›
  AOT-hence ‹□[λx φ{x}]↓›
    by (metis exist−nec →E)
  moreover AOT-have
    ‹□[λx φ{x}]↓ → □∀x∀y(∀F([F]x ≡ [F]y) → ((w ⊨ φ{x}) ≡ ( w ⊨ φ{y})))›
  proof (rule RM; rule →I; rule GEN; rule GEN; rule →I)
    AOT-modally-strict {
      fix x y
      AOT-assume ‹[λx φ{x}]↓›
      AOT-hence ‹∀x∀y (∀F ([F]x ≡ [F]y) → □(φ{x} ≡ φ{y}))›
        using &E kirchner−thm−cor:1[THEN →E] by blast
      AOT-hence ‹∀F ([F]x ≡ [F]y) → □(φ{x} ≡ φ{y})›
        using ∀E(2) by blast
      moreover AOT-assume ‹∀F ([F]x ≡ [F]y)›
      ultimately AOT-have ‹□(φ{x} ≡ φ{y})›
        using →E by blast
```

```
      AOT-hence ‹∀ w (w ⊨ (φ{x} ≡ φ{y}))›
        using fund:2[unvarify p, OF log−prop−prop:2, THEN ≡E(1)] by blast
      AOT-hence ‹w ⊨ (φ{x} ≡ φ{y})›
        using ∀ E(2) using PossibleWorld.ψ →E by blast
      AOT-thus ‹(w ⊨ φ{x}) ≡ (w ⊨ φ{y})›
        using conj−dist−w:4[unvarify p q, OF log−prop−prop:2,
                        OF log−prop−prop:2, THEN ≡E(1)] by blast
    }
  qed
  ultimately AOT-have ‹□∀ x∀ y(∀ F([F]x ≡ [F]y) → ((w ⊨ φ{x}) ≡ ( w ⊨ φ{y})))›
    using →E by blast
  AOT-thus ‹[λx w ⊨ φ{x}]↓›
    using kirchner−thm:1[THEN ≡E(2)] by fast
qed

AOT-theorem w−rel:2: ‹[λx1...xn φ{x1...xn}]↓ → [λx1...xn w ⊨ φ{x1...xn}]↓›
proof(rule →I)
  AOT-assume ‹[λx1...xn φ{x1...xn}]↓›
  AOT-hence ‹□[λx1...xn φ{x1...xn}]↓›
    by (metis exist−nec →E)
  moreover AOT-have ‹□[λx1...xn φ{x1...xn}]↓ → □∀ x1...∀ xn∀ y1...∀ yn(
    ∀ F([F]x1...xn ≡ [F]y1...yn) → ((w ⊨ φ{x1...xn}) ≡ ( w ⊨ φ{y1...yn})))›
  proof (rule RM; rule →I; rule GEN; rule GEN; rule →I)
    AOT-modally-strict {
      fix x1xn y1yn
      AOT-assume ‹[λx1...xn φ{x1...xn}]↓›
      AOT-hence ‹∀ x1...∀ xn∀ y1...∀ yn (
        ∀ F ([F]x1...xn ≡ [F]y1...yn) → □(φ{x1...xn} ≡ φ{y1...yn}))›
        using &E kirchner−thm−cor:2[THEN →E] by blast
      AOT-hence ‹∀ F ([F]x1...xn ≡ [F]y1...yn) → □(φ{x1...xn} ≡ φ{y1...yn})›
        using ∀ E(2) by blast
      moreover AOT-assume ‹∀ F ([F]x1...xn ≡ [F]y1...yn)›
      ultimately AOT-have ‹□(φ{x1...xn} ≡ φ{y1...yn})›
        using →E by blast
      AOT-hence ‹∀ w (w ⊨ (φ{x1...xn} ≡ φ{y1...yn}))›
        using fund:2[unvarify p, OF log−prop−prop:2, THEN ≡E(1)] by blast
      AOT-hence ‹w ⊨ (φ{x1...xn} ≡ φ{y1...yn})›
        using ∀ E(2) using PossibleWorld.ψ →E by blast
      AOT-thus ‹(w ⊨ φ{x1...xn}) ≡ (w ⊨ φ{y1...yn})›
        using conj−dist−w:4[unvarify p q, OF log−prop−prop:2,
                        OF log−prop−prop:2, THEN ≡E(1)] by blast
    }
  qed
  ultimately AOT-have ‹□∀ x1...∀ xn∀ y1...∀ yn(
      ∀ F([F]x1...xn ≡ [F]y1...yn) → ((w ⊨ φ{x1...xn}) ≡ ( w ⊨ φ{y1...yn})))›
    using →E by blast
  AOT-thus ‹[λx1...xn w ⊨ φ{x1...xn}]↓›
    using kirchner−thm:2[THEN ≡E(2)] by fast
qed

AOT-theorem w−rel:3: ‹[λx1...xn w ⊨ [F]x1...xn]↓›
  by (rule w−rel:2[THEN →E]) cqt:2[lambda]

AOT-define WorldIndexedRelation :: ‹Π ⇒ τ ⇒ Π› (‹-−›)
  w−index: ‹[F]w =df [λx1...xn w ⊨ [F]x1...xn]›

AOT-define Rigid :: ‹τ ⇒ φ› (‹Rigid′(-′)›)
  df−rigid−rel:1:
‹Rigid(F) ≡df F↓ & □∀ x1...∀ xn([F]x1...xn → □[F]x1...xn)›

AOT-define Rigidifies :: ‹τ ⇒ τ ⇒ φ› (‹Rigidifies′(-,-′)›)
  df−rigid−rel:2:
‹Rigidifies(F, G) ≡df Rigid(F) & ∀ x1...∀ xn([F]x1...xn ≡ [G]x1...xn)›
```

```
AOT-theorem rigid−der:1: ‹[[F]_w]x_1...x_n ≡ w ⊨ [F]x_1...x_n›
  apply (rule rule−id−df:2:b[2][where τ=λ (Π, κ). «[Π]_κ» and
                                σ=λ(Π, κ). «[λx_1...x_n κ ⊨ [Π]x_1...x_n]»,
                             simplified, OF w−index])
   apply (fact w−rel:3)
  apply (rule beta−C−meta[THEN →E])
  by (fact w−rel:3)

AOT-theorem rigid−der:2: ‹Rigid([G]_w)›
proof(safe intro!: ≡_df I[OF df−rigid−rel:1] &I)
  AOT-show ‹[G]_w↓›
    by (rule rule−id−df:2:b[2][where τ=λ (Π, κ). «[Π]_κ» and
                                  σ=λ(Π, κ). «[λx_1...x_n κ ⊨ [Π]x_1...x_n]»,
                               simplified, OF w−index])
       (fact w−rel:3)+
next
  AOT-have ‹□∀ x_1...∀ x_n ([[G]_w]x_1...x_n → □[[G]_w]x_1...x_n)›
  proof(rule RN; safe intro!: →I GEN)
    AOT-modally-strict {
      AOT-have assms: ‹PossibleWorld(w)› using PossibleWorld.ψ.
      AOT-hence nec-pw-w: ‹□PossibleWorld(w)›
        using ≡E(1) rigid−pw:1 by blast
      fix x_1x_n
      AOT-assume ‹[[G]_w]x_1...x_n›
      AOT-hence ‹[λx_1...x_n w ⊨ [G]x_1...x_n]x_1...x_n›
        using rule−id−df:2:a[2][where τ=λ (Π, κ). «[Π]_κ» and
                                 σ=λ(Π, κ). «[λx_1...x_n κ ⊨ [Π]x_1...x_n]»,
                                 simplified, OF w−index, OF w−rel:3]
        by fast
      AOT-hence ‹w ⊨ [G]x_1...x_n›
        by (metis β→C(1))
      AOT-hence ‹□w ⊨ [G]x_1...x_n›
        using rigid−truth−at:1[unvarify p, OF log−prop−prop:2, THEN ≡E(1)]
        by blast
      moreover AOT-have ‹□w ⊨ [G]x_1...x_n → □[λx_1...x_n w ⊨ [G]x_1...x_n]x_1...x_n›
      proof (rule RM; rule →I)
        AOT-modally-strict {
          AOT-assume ‹w ⊨ [G]x_1...x_n›
          AOT-thus ‹[λx_1...x_n w ⊨ [G]x_1...x_n]x_1...x_n›
            by (auto intro!: β←C(1) simp: w−rel:3 cqt:2)
        }
      qed
      ultimately AOT-have 1: ‹□[λx_1...x_n w ⊨ [G]x_1...x_n]x_1...x_n›
        using →E by blast
      AOT-show ‹□[[G]_w]x_1...x_n›
        by (rule rule−id−df:2:b[2][where τ=λ (Π, κ). «[Π]_κ» and
                                      σ=λ(Π, κ). «[λx_1...x_n κ ⊨ [Π]x_1...x_n]»,
                                   simplified, OF w−index])
           (auto simp: 1 w−rel:3)
    }
  qed
  AOT-thus ‹□∀ x_1...∀ x_n ([[G]_w]x_1...x_n → □[[G]_w]x_1...x_n)›
    using →E by blast
qed

AOT-theorem rigid−der:3: ‹∃ F Rigidifies(F, G)›
proof −
  AOT-obtain w where w: ‹∀ p (w ⊨ p ≡ p)›
    using act−world:1 PossibleWorld.∃ E[rotated] by meson
  show ?thesis
  proof (rule ∃ I(1)[where τ=‹«[G]_w»›])
    AOT-show ‹Rigidifies([G]_w, [G])›
```

```
    proof(safe intro!: ≡dfI[OF df−rigid−rel:2] &I GEN)
      AOT-show ‹Rigid([G]w)›
        using rigid−der:2 by blast
    next
      fix x1xn
      AOT-have ‹[[G]w]x1...xn ≡ [λx1...xn w ⊨ [G]x1...xn]x1...xn›
      proof(rule ≡I; rule →I)
        AOT-assume ‹[[G]w]x1...xn›
        AOT-thus ‹[λx1...xn w ⊨ [G]x1...xn]x1...xn›
          by (rule rule−id−df:2:a[2]
                    [where τ=λ (Π, κ). «[Π]κ» and
                           σ=λ(Π, κ). «[λx1...xn κ ⊨ [Π]x1...xn]»,
                     simplified, OF w−index, OF w−rel:3])
      next
        AOT-assume ‹[λx1...xn w ⊨ [G]x1...xn]x1...xn›
        AOT-thus ‹[[G]w]x1...xn›
          by (rule rule−id−df:2:b[2]
                    [where τ=λ (Π, κ). «[Π]κ» and
                           σ=λ(Π, κ). «[λx1...xn κ ⊨ [Π]x1...xn]»,
                     simplified, OF w−index, OF w−rel:3])
      qed
      also AOT-have ‹... ≡ w ⊨ [G]x1...xn›
        by (rule beta−C−meta[THEN →E])
           (fact w−rel:3)
      also AOT-have ‹... ≡ [G]x1...xn›
        using w[THEN ∀E(1), OF log−prop−prop:2] by blast
      finally AOT-show ‹[[G]w]x1...xn ≡ [G]x1...xn›.
    qed
  next
    AOT-show ‹[G]w↓›
      by (rule rule−id−df:2:b[2][where τ=λ (Π, κ). «[Π]κ»
                                  and σ=λ(Π, κ). «[λx1...xn κ ⊨ [Π]x1...xn]»,
                                simplified, OF w−index])
         (auto simp: w−rel:3)
  qed
qed

AOT-theorem rigid−rel−thms:1:
  ‹□(∀x1...∀xn ([F]x1...xn → □[F]x1...xn)) ≡ ∀x1...∀xn(◊[F]x1...xn → □[F]x1...xn)›
proof(safe intro!: ≡I →I GEN)
  fix x1xn
  AOT-assume ‹□∀x1...∀xn ([F]x1...xn → □[F]x1...xn)›
  AOT-hence ‹∀x1...∀xn □([F]x1...xn → □[F]x1...xn)›
    by (metis →E GEN RM cqt−orig:3)
  AOT-hence ‹□([F]x1...xn → □[F]x1...xn)›
    using ∀E(2) by blast
  AOT-hence ‹◊[F]x1...xn → □[F]x1...xn›
    by (metis ≡E(1) sc−eq−box−box:1)
  moreover AOT-assume ‹◊[F]x1...xn›
  ultimately AOT-show ‹□[F]x1...xn›
    using →E by blast
next
  AOT-assume ‹∀x1...∀xn (◊[F]x1...xn → □[F]x1...xn)›
  AOT-hence ‹◊[F]x1...xn → □[F]x1...xn› for x1xn
    using ∀E(2) by blast
  AOT-hence ‹□([F]x1...xn → □[F]x1...xn)› for x1xn
    by (metis ≡E(2) sc−eq−box−box:1)
  AOT-hence 0: ‹∀x1...∀xn □([F]x1...xn → □[F]x1...xn)›
    by (rule GEN)
  AOT-thus ‹□(∀x1...∀xn ([F]x1...xn → □[F]x1...xn))›
    using BF vdash−properties:10 by blast
qed
```

```
AOT-theorem rigid−rel−thms:2:
  ‹□(∀x1...∀xn ([F]x1...xn → □[F]x1...xn)) ≡ ∀x1...∀xn(□[F]x1...xn ∨ □¬[F]x1...xn)›
proof(safe intro!: ≡I →I)
  AOT-assume ‹□(∀x1...∀xn ([F]x1...xn → □[F]x1...xn))›
  AOT-hence 0: ‹∀x1...∀xn □([F]x1...xn → □[F]x1...xn)›
    using CBF[THEN →E] by blast
  AOT-show ‹∀x1...∀xn(□[F]x1...xn ∨ □¬[F]x1...xn)›
  proof(rule GEN)
    fix x1xn
    AOT-have 1: ‹□([F]x1...xn → □[F]x1...xn)›
      using 0[THEN ∀E(2)].
    AOT-hence 2: ‹◇[F]x1...xn → [F]x1...xn›
      using B◇ Hypothetical Syllogism K◇ vdash−properties:10 by blast
    AOT-have ‹[F]x1...xn ∨ ¬[F]x1...xn›
      using exc−mid.
    moreover {
      AOT-assume ‹[F]x1...xn›
      AOT-hence ‹□[F]x1...xn›
        using 1[THEN qml:2[axiom-inst, THEN →E], THEN →E] by blast
    }
    moreover {
      AOT-assume 3: ‹¬[F]x1...xn›
      AOT-have ‹□¬[F]x1...xn›
      proof(rule raa−cor:1)
        AOT-assume ‹¬□¬[F]x1...xn›
        AOT-hence ‹◇[F]x1...xn›
          by (AOT-subst-def conventions:5)
        AOT-hence ‹[F]x1...xn› using 2[THEN →E] by blast
        AOT-thus ‹[F]x1...xn & ¬[F]x1...xn›
          using 3 &I by blast
      qed
    }
    ultimately AOT-show ‹□[F]x1...xn ∨ □¬[F]x1...xn›
      by (metis ∨I(1,2) raa−cor:1)
  qed
next
  AOT-assume 0: ‹∀x1...∀xn(□[F]x1...xn ∨ □¬[F]x1...xn)›
  {
    fix x1xn
    AOT-have ‹□[F]x1...xn ∨ □¬[F]x1...xn› using 0[THEN ∀E(2)] by blast
    moreover {
      AOT-assume ‹□[F]x1...xn›
      AOT-hence ‹□□[F]x1...xn›
        using S5Basic:6[THEN ≡E(1)] by blast
      AOT-hence ‹□([F]x1...xn → □[F]x1...xn)›
        using KBasic:1[THEN →E] by blast
    }
    moreover {
      AOT-assume ‹□¬[F]x1...xn›
      AOT-hence ‹□([F]x1...xn → □[F]x1...xn)›
        using KBasic:2[THEN →E] by blast
    }
    ultimately AOT-have ‹□([F]x1...xn → □[F]x1...xn)›
      using con−dis−i−e:4:b raa−cor:1 by blast
  }
  AOT-hence ‹∀x1...∀xn □([F]x1...xn → □[F]x1...xn)›
    by (rule GEN)
  AOT-thus ‹□(∀x1...∀xn ([F]x1...xn → □[F]x1...xn))›
    using BF[THEN →E] by fast
qed

AOT-theorem rigid−rel−thms:3: ‹Rigid(F) ≡ ∀x1...∀xn (□[F]x1...xn ∨ □¬[F]x1...xn)›
  by (AOT-subst-thm df−rigid−rel:1[THEN ≡Df, THEN ≡S(1), OF cqt:2(1)];
```

AOT-subst-thm rigid−rel−thms:2)
(*simp add: oth−class−taut:3:a*)

13 Natural Numbers

AOT-define *CorrelatesOneToOne* :: ‹τ ⇒ τ ⇒ τ ⇒ φ› (‹- |: - $_{1-1}$⟷ -›)
1−1−cor: ‹R |: F $_{1-1}$⟷ G $\equiv_{df}$ R↓ & F↓ & G↓ &
∀x ([F]x → ∃!y([G]y & [R]xy)) &
∀y ([G]y → ∃!x([F]x & [R]xy))›

AOT-define *MapsTo* :: ‹τ ⇒ τ ⇒ τ ⇒ φ› (‹- |: - ⟶ -›)
fFG:1: ‹R |: F ⟶ G $\equiv_{df}$ R↓ & F↓ & G↓ & ∀x ([F]x → ∃!y([G]y & [R]xy))›

AOT-define *MapsToOneToOne* :: ‹τ ⇒ τ ⇒ τ ⇒ φ› (‹- |: - $_{1-1}$⟶ -›)
fFG:2: ‹R |: F $_{1-1}$⟶ G $\equiv_{df}$
R |: F ⟶ G & ∀x∀y∀z (([F]x & [F]y & [G]z) → ([R]xz & [R]yz → x = y))›

AOT-define *MapsOnto* :: ‹τ ⇒ τ ⇒ τ ⇒ φ› (‹- |: - $⟶_{onto}$ -›)
fFG:3: ‹R |: F $⟶_{onto}$ G $\equiv_{df}$ R |: F ⟶ G & ∀y ([G]y → ∃x([F]x & [R]xy))›

AOT-define *MapsOneToOneOnto* :: ‹τ ⇒ τ ⇒ τ ⇒ φ› (‹- |: - $_{1-1}⟶_{onto}$ -›)
fFG:4: ‹R |: F $_{1-1}⟶_{onto}$ G $\equiv_{df}$ R |: F $_{1-1}$⟶ G & R |: F $⟶_{onto}$ G›

AOT-theorem *eq−1−1*: ‹R |: F $_{1-1}$⟷ G ≡ R |: F $_{1-1}⟶_{onto}$ G›
proof(*rule ≡I*; *rule →I*)
AOT-assume ‹R |: F $_{1-1}$⟷ G›
AOT-hence *A*: ‹∀x ([F]x → ∃!y([G]y & [R]xy))›
and *B*: ‹∀y ([G]y → ∃!x([F]x & [R]xy))›
using $\equiv_{df}$*E*[*OF 1−1−cor*] *&E* **by** *blast+*
AOT-have *C*: ‹R |: F ⟶ G›
proof (*rule* $\equiv_{df}$*I*[*OF fFG:1*]; *rule &I*)
AOT-show ‹R↓ & F↓ & G↓›
using *cqt:2*[*const-var*][*axiom-inst*] *&I* **by** *metis*
next
AOT-show ‹∀x ([F]x → ∃!y([G]y & [R]xy))› **by** (*rule A*)
qed
AOT-show ‹R |: F $_{1-1}⟶_{onto}$ G›
proof (*rule* $\equiv_{df}$*I*[*OF fFG:4*]; *rule &I*)
AOT-show ‹R |: F $_{1-1}$⟶ G›
proof (*rule* $\equiv_{df}$*I*[*OF fFG:2*]; *rule &I*)
AOT-show ‹R |: F ⟶ G› **using** *C*.
next
AOT-show ‹∀x∀y∀z ([F]x & [F]y & [G]z → ([R]xz & [R]yz → x = y))›
proof(*rule GEN*; *rule GEN*; *rule GEN*; *rule →I*; *rule →I*)
fix *x y z*
AOT-assume *1*: ‹[F]x & [F]y & [G]z›
moreover AOT-assume *2*: ‹[R]xz & [R]yz›
ultimately AOT-have *3*: ‹∃!x ([F]x & [R]xz)›
using *B &E ∀E →E* **by** *fast*
AOT-show ‹x = y›
by (*rule uni−most*[*THEN →E*, *OF 3*, *THEN ∀E(2)*[**where** *β=x*],
THEN ∀E(2)[**where** *β=y*], *THEN →E*])
(*metis &I &E 1 2*)
qed
qed
next
AOT-show ‹R |: F $⟶_{onto}$ G›
proof (*rule* $\equiv_{df}$*I*[*OF fFG:3*]; *rule &I*)
AOT-show ‹R |: F ⟶ G› **using** *C*.
next
AOT-show ‹∀y ([G]y → ∃x ([F]x & [R]xy))›
proof(*rule GEN*; *rule →I*)

```
        fix y
        AOT-assume ‹[G]y›
        AOT-hence ‹∃!x ([F]x & [R]xy)›
          using B[THEN ∀E(2), THEN →E] by blast
        AOT-hence ‹∃x ([F]x & [R]xy & ∀β (([F]β & [R]βy) → β = x))›
          using uniqueness:1[THEN ≡df E] by blast
        then AOT-obtain x where ‹[F]x & [R]xy›
          using ∃E[rotated] &E by blast
        AOT-thus ‹∃x ([F]x & [R]xy)› by (rule ∃I)
      qed
    qed
  qed
next
  AOT-assume ‹R |: F 1−1⟶onto G›
  AOT-hence ‹R |: F 1−1⟶ G› and ‹R |: F ⟶onto G›
    using ≡df E[OF fFG:4] &E by blast+
  AOT-hence C: ‹R |: F ⟶ G›
    and D: ‹∀x∀y∀z ([F]x & [F]y & [G]z → ([R]xz & [R]yz → x = y))›
    and E: ‹∀y ([G]y → ∃x ([F]x & [R]xy))›
    using ≡df E[OF fFG:2] ≡df E[OF fFG:3] &E by blast+
  AOT-show ‹R |: F 1−1⟷ G›
  proof(rule 1−1−cor[THEN ≡df I]; safe intro!: &I cqt:2[const-var][axiom-inst])
    AOT-show ‹∀x ([F]x → ∃!y ([G]y & [R]xy))›
      using ≡df E[OF fFG:1, OF C] &E by blast
  next
    AOT-show ‹∀y ([G]y → ∃!x ([F]x & [R]xy))›
    proof (rule GEN; rule →I)
      fix y
      AOT-assume 0: ‹[G]y›
      AOT-hence ‹∃x ([F]x & [R]xy)›
        using E ∀E →E by fast
      then AOT-obtain a where a-prop: ‹[F]a & [R]ay›
        using ∃E[rotated] by blast
      moreover AOT-have ‹∀z ([F]z & [R]zy → z = a)›
      proof (rule GEN; rule →I)
        fix z
        AOT-assume ‹[F]z & [R]zy›
        AOT-thus ‹z = a›
          using D[THEN ∀E(2)[where β=z], THEN ∀E(2)[where β=a],
                  THEN ∀E(2)[where β=y], THEN →E, THEN →E]
                a-prop 0 &E &I by metis
      qed
      ultimately AOT-have ‹∃x ([F]x & [R]xy & ∀z ([F]z & [R]zy → z = x))›
        using &I ∃I(2) by fast
      AOT-thus ‹∃!x ([F]x & [R]xy)›
        using uniqueness:1[THEN ≡df I] by fast
    qed
  qed
qed
```

We have already introduced the restricted type of Ordinary objects in the Extended Relation Comprehension theory. However, make sure all variable names are defined as expected (avoiding conflicts with situations of possible world theory).

```
AOT-register-variable-names
  Ordinary: u v r t s

AOT-theorem equi:1: ‹∃!u φ{u} ≡ ∃u (φ{u} & ∀v (φ{v} → v =E u))›
proof(rule ≡I; rule →I)
  AOT-assume ‹∃!u φ{u}›
  AOT-hence ‹∃!x (O!x & φ{x})›.
  AOT-hence ‹∃x (O!x & φ{x} & ∀β (O!β & φ{β} → β = x))›
    using uniqueness:1[THEN ≡df E] by blast
  then AOT-obtain x where x-prop: ‹O!x & φ{x} & ∀β (O!β & φ{β} → β = x)›
```

```
    using ∃E[rotated] by blast
  {
    fix β
    AOT-assume beta-ord: ‹O!β›
    moreover AOT-assume ‹φ{β}›
    ultimately AOT-have ‹β = x›
      using x-prop[THEN &E(2), THEN ∀E(2)[where β=β]] &I →E by blast
    AOT-hence ‹β =⇩E x›
      using ord-=E=:1[THEN →E, OF ∨I(1)[OF beta-ord],
                      THEN qml:2[axiom-inst, THEN →E],
                      THEN ≡E(1)]
      by blast
  }
  AOT-hence ‹(O!β → (φ{β} → β =⇩E x))› for β
    using →I by blast
  AOT-hence ‹∀β(O!β → (φ{β} → β =⇩E x))›
    by (rule GEN)
  AOT-hence ‹O!x & φ{x} & ∀y (O!y → (φ{y} → y =⇩E x))›
    using x-prop[THEN &E(1)] &I by blast
  AOT-hence ‹O!x & (φ{x} & ∀y (O!y → (φ{y} → y =⇩E x)))›
    using &E &I by meson
  AOT-thus ‹∃u (φ{u} & ∀v (φ{v} → v =⇩E u))›
    using ∃I by fast
next
  AOT-assume ‹∃u (φ{u} & ∀v (φ{v} → v =⇩E u))›
  AOT-hence ‹∃x (O!x & (φ{x} & ∀y (O!y → (φ{y} → y =⇩E x))))›
    by blast
  then AOT-obtain x where x-prop: ‹O!x & (φ{x} & ∀y (O!y → (φ{y} → y =⇩E x)))›
    using ∃E[rotated] by blast
  AOT-have ‹∀y ([O!]y & φ{y} → y = x)›
  proof(rule GEN; rule →I)
    fix y
    AOT-assume ‹O!y & φ{y}›
    AOT-hence ‹y =⇩E x›
      using x-prop[THEN &E(2), THEN &E(2), THEN ∀E(2)[where β=y]]
         →E &E by blast
    AOT-thus ‹y = x›
      using ord-=E=:1[THEN →E, OF ∨I(2)[OF x-prop[THEN &E(1)]],
                      THEN qml:2[axiom-inst, THEN →E], THEN ≡E(2)] by blast
  qed
  AOT-hence ‹[O!]x & φ{x} & ∀y ([O!]y & φ{y} → y = x)›
    using x-prop &E &I by meson
  AOT-hence ‹∃x ([O!]x & φ{x} & ∀y ([O!]y & φ{y} → y = x))›
    by (rule ∃I)
  AOT-hence ‹∃!x (O!x & φ{x})›
    by (rule uniqueness:1[THEN ≡⇩d⇩fI])
  AOT-thus ‹∃!u φ{u}›.
qed

AOT-define CorrelatesEOneToOne :: ‹τ ⇒ τ ⇒ τ ⇒ φ› (‹- |: - ⇩1⇩-⇩1⟷⇩E -›)
  equi:2: ‹R |: F ⇩1⇩-⇩1⟷⇩E G ≡⇩d⇩f R↓ & F↓ & G↓ &
                              ∀u ([F]u → ∃!v([G]v & [R]uv)) &
                              ∀v ([G]v → ∃!u([F]u & [R]uv))›

AOT-define EquinumerousE :: ‹τ ⇒ τ ⇒ φ› (infixl ‹≈⇩E› 50)
  equi:3: ‹F ≈⇩E G ≡⇩d⇩f ∃R (R |: F ⇩1⇩-⇩1⟷⇩E G)›
```

Note: not explicitly in PLM.

```
AOT-theorem eq-den-1: ‹Π↓› if ‹Π ≈⇩E Π'›
proof -
  AOT-have ‹∃R (R |: Π ⇩1⇩-⇩1⟷⇩E Π')›
    using equi:3[THEN ≡⇩d⇩fE] that by blast
  then AOT-obtain R where ‹R |: Π ⇩1⇩-⇩1⟷⇩E Π'›
```

```
    using ∃E[rotated] by blast
  AOT-thus ‹Π↓›
    using equi:2[THEN ≡dfE] &E by blast
qed
```

Note: not explicitly in PLM.

```
AOT-theorem eq-den-2: ‹Π'↓› if ‹Π ≈E Π'›
proof -
  AOT-have ‹∃R (R |: Π 1-1⟷E Π')›
    using equi:3[THEN ≡dfE] that by blast
  then AOT-obtain R where ‹R |: Π 1-1⟷E Π'›
    using ∃E[rotated] by blast
  AOT-thus ‹Π'↓›
    using equi:2[THEN ≡dfE] &E by blast+
qed

AOT-theorem eq-part:1: ‹F ≈E F›
proof (safe intro!: &I GEN →I cqt:2[const-var][axiom-inst]
                    ≡dfI[OF equi:3] ≡dfI[OF equi:2] ∃I(1))
  fix x
  AOT-assume 1: ‹O!x›
  AOT-assume 2: ‹[F]x›
  AOT-show ‹∃!v ([F]v & x =E v)›
  proof(rule equi:1[THEN ≡E(2)];
        rule ∃I(2)[where β=x];
        safe dest!: &E(2)
             intro!:  &I →I 1 2 Ordinary.GEN ord=Eequiv:1[THEN →E, OF 1])
    AOT-show ‹v =E x› if ‹x =E v› for v
      by (metis that ord=Eequiv:2[THEN →E])
  qed
next
  fix y
  AOT-assume 1: ‹O!y›
  AOT-assume 2: ‹[F]y›
  AOT-show ‹∃!u ([F]u & u =E y)›
    by(safe dest!: &E(2)
           intro!: equi:1[THEN ≡E(2)] ∃I(2)[where β=y]
                   &I →I 1 2 GEN ord=Eequiv:1[THEN →E, OF 1])
qed(auto simp: =E[denotes])


AOT-theorem eq-part:2: ‹F ≈E G → G ≈E F›
proof (rule →I)
  AOT-assume ‹F ≈E G›
  AOT-hence ‹∃R R |: F 1-1⟷E G›
    using equi:3[THEN ≡dfE] by blast
  then AOT-obtain R where ‹R |: F 1-1⟷E G›
    using ∃E[rotated] by blast
  AOT-hence 0: ‹R↓ & F↓ & G↓ & ∀u ([F]u → ∃!v([G]v & [R]uv)) &
                         ∀v ([G]v → ∃!u([F]u & [R]uv))›
    using equi:2[THEN ≡dfE] by blast

  AOT-have ‹[λxy [R]yx]↓ & G↓ & F↓ & ∀u ([G]u → ∃!v([F]v & [λxy [R]yx]uv)) &
                         ∀v ([F]v → ∃!u([G]u & [λxy [R]yx]uv))›
  proof (AOT-subst ‹[λxy [R]yx]yx› ‹[R]xy› for: x y;
        (safe intro!: &I cqt:2[const-var][axiom-inst] 0[THEN &E(2)]
                      0[THEN &E(1), THEN &E(2)]; cqt:2[lambda])?)
    AOT-modally-strict {
      AOT-have ‹[λxy [R]yx]xy› if ‹[R]yx› for y x
        by (auto intro!: β←C(1) cqt:2
                 simp: &I ex:1:a prod-denotesI rule-ui:3 that)
      moreover AOT-have ‹[R]yx› if ‹[λxy [R]yx]xy› for y x
        using β→C(1)[where φ=λ(x,y). - (x,y) and κ1κn=(-,-),
```

```
                  simplified, OF that, simplified].
    ultimately AOT-show ‹[λxy [R]yx]αβ ≡ [R]βα› for α β
      by (metis deduction−theorem ≡I)
   }
  qed
  AOT-hence ‹[λxy [R]yx] |: G 1−1⟷E F›
    using equi:2[THEN ≡dfI] by blast
  AOT-hence ‹∃ R R |: G 1−1⟷E F›
    by (rule ∃ I(1)) cqt:2[lambda]
  AOT-thus ‹G ≈E F›
    using equi:3[THEN ≡dfI] by blast
qed
```

Note: not explicitly in PLM.

```
AOT-theorem eq−part:2[terms]: ‹Π ≈E Π′ → Π′ ≈E Π›
  using eq−part:2[unvarify F G] eq-den-1 eq-den-2 →I by meson
declare eq−part:2[terms][THEN →E, sym]

AOT-theorem eq−part:3: ‹(F ≈E G & G ≈E H) → F ≈E H›
proof (rule →I)
  AOT-assume ‹F ≈E G & G ≈E H›
  then AOT-obtain R1 and R2 where
       ‹R1 |: F 1−1⟷E G›
   and ‹R2 |: G 1−1⟷E H›
    using equi:3[THEN ≡dfE] &E ∃ E[rotated] by metis
  AOT-hence ϑ: ‹∀ u ([F]u → ∃ !v([G]v & [R1]uv)) & ∀ v ([G]v → ∃ !u([F]u & [R1]uv))›
        and ξ: ‹∀ u ([G]u → ∃ !v([H]v & [R2]uv)) & ∀ v ([H]v → ∃ !u([G]u & [R2]uv))›
    using equi:2[THEN ≡dfE, THEN &E(2)]
          equi:2[THEN ≡dfE, THEN &E(1), THEN &E(2)]
          &I by blast+
  AOT-have ‹∃ R R = [λxy O!x & O!y & ∃ v ([G]v & [R1]xv & [R2]vy)]›
    by (rule free−thms:3[lambda]) cqt-2-lambda-inst-prover
  then AOT-obtain R where R-def: ‹R = [λxy O!x & O!y & ∃ v ([G]v & [R1]xv & [R2]vy)]›
    using ∃ E[rotated] by blast
  AOT-have 1: ‹∃!v (([H]v & [R]uv))› if a: ‹[O!]u› and b: ‹[F]u› for u
  proof (rule ≡E(2)[OF equi:1])
    AOT-obtain b where
      b-prop: ‹[O!]b & ([G]b & [R1]ub & ∀ v ([G]v & [R1]uv → v =E b))›
      using ϑ[THEN &E(1), THEN ∀ E(2), THEN →E, THEN →E,
              OF a b, THEN ≡E(1)[OF equi:1]]
            ∃ E[rotated] by blast
    AOT-obtain c where
      c-prop: [O!]c & ([H]c & [R2]bc & ∀ v ([H]v & [R2]bv → v =E c))
      using ξ[THEN &E(1), THEN ∀ E(2)[where β=b], THEN →E,
              OF b-prop[THEN &E(1)], THEN →E,
              OF b-prop[THEN &E(2), THEN &E(1), THEN &E(1)],
              THEN ≡E(1)[OF equi:1]]
    ∃ E[rotated] by blast
    AOT-show ‹∃ v ([H]v & [R]uv & ∀ v′ ([H]v′ & [R]uv′ → v′ =E v))›
    proof (safe intro!: &I GEN →I ∃ I(2)[where β=c])
      AOT-show ‹O!c› using c-prop &E by blast
    next
      AOT-show ‹[H]c› using c-prop &E by blast
    next
      AOT-have 0: ‹[O!]u & [O!]c & ∃ v ([G]v & [R1]uv & [R2]vc)›
        by (safe intro!: &I a c-prop[THEN &E(1)] ∃ I(2)[where β=b]
                         b-prop[THEN &E(1)] b-prop[THEN &E(2), THEN &E(1)]
                         c-prop[THEN &E(2), THEN &E(1), THEN &E(2)])
      AOT-show ‹[R]uc›
        by (auto intro: rule=E[rotated, OF R-def[symmetric]]
                 intro!: β←C(1) cqt:2
                 simp: &I ex:1:a prod-denotesI rule−ui:3 0)
    next
```

```
    fix x
    AOT-assume ordx: ‹O!x›
    AOT-assume ‹[H]x & [R]ux›
    AOT-hence hx: ‹[H]x› and ‹[R]ux› using &E by blast+
    AOT-hence ‹[λxy O!x & O!y & ∃ v ([G]v & [R1]xv & [R2]vy)]ux›
      using rule=E[rotated, OF R-def] by fast
    AOT-hence ‹O!u & O!x & ∃ v ([G]v & [R1]uv & [R2]vx)›
      by (rule β→C(1)[where φ=λ(κ,κ'). - κ κ' and κ1κn=(-,-), simplified])
    then AOT-obtain z where z-prop: ‹O!z & ([G]z & [R1]uz & [R2]zx)›
      using &E ∃ E[rotated] by blast
    AOT-hence ‹z =E b›
      using b-prop[THEN &E(2), THEN &E(2), THEN ∀ E(2)[where β=z]]
      using &E →E by metis
    AOT-hence ‹z = b›
      by (metis =E−simple:2[THEN →E])
    AOT-hence ‹[R2]bx›
      using z-prop[THEN &E(2), THEN &E(2)] rule=E by fast
    AOT-thus ‹x =E c›
      using c-prop[THEN &E(2), THEN &E(2), THEN ∀ E(2)[where β=x],
                  THEN →E, THEN →E, OF ordx]
            hx &I by blast
  qed
qed
AOT-have 2: ‹∃!u (([F]u & [R]uv))› if a: ‹[O!]v› and b: ‹[H]v› for v
proof (rule ≡E(2)[OF equi:1])
  AOT-obtain b where
    b-prop: ‹[O!]b & ([G]b & [R2]bv & ∀ u ([G]u & [R2]uv → u =E b))›
    using ξ[THEN &E(2), THEN ∀ E(2), THEN →E, THEN →E,
            OF a b, THEN ≡E(1)[OF equi:1]]
          ∃ E[rotated] by blast
  AOT-obtain c where
    c-prop: [O!]c & ([F]c & [R1]cb & ∀ v ([F]v & [R1]vb → v =E c))
    using ϑ[THEN &E(2), THEN ∀ E(2)[where β=b], THEN →E,
            OF b-prop[THEN &E(1)], THEN →E,
            OF b-prop[THEN &E(2), THEN &E(1), THEN &E(1)],
            THEN ≡E(1)[OF equi:1]]
  ∃ E[rotated] by blast
  AOT-show ‹∃ u ([F]u & [R]uv & ∀ v' ([F]v' & [R]v'v → v' =E u))›
  proof (safe intro!: &I GEN →I ∃ I(2)[where β=c])
    AOT-show ‹O!c› using c-prop &E by blast
  next
    AOT-show ‹[F]c› using c-prop &E by blast
  next
    AOT-have ‹[O!]c & [O!]v & ∃ u ([G]u & [R1]cu & [R2]uv)›
      by (safe intro!: &I a ∃ I(2)[where β=b]
                  c-prop[THEN &E(1)] b-prop[THEN &E(1)]
                  b-prop[THEN &E(2), THEN &E(1), THEN &E(1)]
                  b-prop[THEN &E(2), THEN &E(1), THEN &E(2)]
                  c-prop[THEN &E(2), THEN &E(1), THEN &E(2)])
    AOT-thus ‹[R]cv›
      by (auto intro: rule=E[rotated, OF R-def[symmetric]]
              intro!: β←C(1) cqt:2
              simp: &I ex:1:a prod-denotesI rule−ui:3)
  next
    fix x
    AOT-assume ordx: ‹O!x›
    AOT-assume ‹[F]x & [R]xv›
    AOT-hence hx: ‹[F]x› and ‹[R]xv› using &E by blast+
    AOT-hence ‹[λxy O!x & O!y & ∃ v ([G]v & [R1]xv & [R2]vy)]xv›
      using rule=E[rotated, OF R-def] by fast
    AOT-hence ‹O!x & O!v & ∃ u ([G]u & [R1]xu & [R2]uv)›
      by (rule β→C(1)[where φ=λ(κ,κ'). - κ κ' and κ1κn=(-,-), simplified])
    then AOT-obtain z where z-prop: ‹O!z & ([G]z & [R1]xz & [R2]zv)›
```

using &*E* ∃ *E*[*rotated*] **by** *blast*
AOT-hence ‹$z =_E b$›
using *b-prop*[*THEN* &*E*(*2*), *THEN* &*E*(*2*), *THEN* ∀ *E*(*2*)[**where** *β*=*z*]]
using &*E* →*E* &*I* **by** *metis*
AOT-hence ‹*z* = *b*›
by (*metis* =*E*−*simple*:*2*[*THEN* →*E*])
AOT-hence ‹$[R_1]xb$›
using *z-prop*[*THEN* &*E*(*2*), *THEN* &*E*(*1*), *THEN* &*E*(*2*)] *rule*=*E* **by** *fast*
AOT-thus ‹$x =_E c$›
using *c-prop*[*THEN* &*E*(*2*), *THEN* &*E*(*2*), *THEN* ∀ *E*(*2*)[**where** *β*=*x*],
THEN →*E*, *THEN* →*E*, *OF ordx*]
hx &*I* **by** *blast*
qed
qed
AOT-show ‹$F \approx_E H$›
apply (*rule equi*:*3*[*THEN* $\equiv_{df}$*I*])
apply (*rule* ∃ *I*(*2*)[**where** *β*=*R*])
by (*auto intro*!: *1 2 equi*:*2*[*THEN* $\equiv_{df}$*I*] &*I cqt*:*2*[*const-var*][*axiom-inst*]
Ordinary.*GEN* →*I Ordinary*.*ψ*)
qed

Note: not explicitly in PLM.

AOT-theorem *eq*−*part*:*3*[*terms*]: ‹$\Pi \approx_E \Pi''$› **if** ‹$\Pi \approx_E \Pi'$› **and** ‹$\Pi' \approx_E \Pi''$›
using *eq*−*part*:*3*[*unvarify F G H*, *THEN* →*E*] *eq-den-1 eq-den-2* →*I* &*I*
by (*metis that*(*1*) *that*(*2*))
declare *eq*−*part*:*3*[*terms*][*trans*]

AOT-theorem *eq*−*part*:*4*: ‹$F \approx_E G \equiv \forall H\ (H \approx_E F \equiv H \approx_E G)$›
proof(*rule* ≡*I*; *rule* →*I*)
AOT-assume *0*: ‹$F \approx_E G$›
AOT-hence *1*: ‹$G \approx_E F$› **using** *eq*−*part*:*2*[*THEN* →*E*] **by** *blast*
AOT-show ‹$\forall H\ (H \approx_E F \equiv H \approx_E G)$›
proof (*rule GEN*; *rule* ≡*I*; *rule* →*I*)
AOT-show ‹$H \approx_E G$› **if** ‹$H \approx_E F$› **for** *H* **using** *0*
by (*meson* &*I eq*−*part*:*3 that vdash*−*properties*:*6*)
next
AOT-show ‹$H \approx_E F$› **if** ‹$H \approx_E G$› **for** *H* **using** *1*
by (*metis* &*I eq*−*part*:*3 that vdash*−*properties*:*6*)
qed
next
AOT-assume ‹$\forall H\ (H \approx_E F \equiv H \approx_E G)$›
AOT-hence ‹$F \approx_E F \equiv F \approx_E G$› **using** ∀ *E* **by** *blast*
AOT-thus ‹$F \approx_E G$› **using** *eq*−*part*:*1* ≡*E* **by** *blast*
qed

AOT-define *MapsE* :: ‹τ ⇒ τ ⇒ τ ⇒ φ› (‹- |: - ⟶*E* -›)
equi−*rem*:*1*:
‹*R* |: *F* ⟶*E* *G* $\equiv_{df}$ *R*↓ & *F*↓ & *G*↓ & ∀ *u* ([*F*]*u* → ∃!*v* ([*G*]*v* & [*R*]*uv*))›

AOT-define *MapsEOneToOne* :: ‹τ ⇒ τ ⇒ τ ⇒ φ› (‹- |: - $_{1-1}$⟶*E* -›)
equi−*rem*:*2*:
‹*R* |: *F* $_{1-1}$⟶*E* *G* $\equiv_{df}$
R |: *F* ⟶*E* *G* & ∀ *t*∀ *u*∀ *v* ((([*F*]*t* & [*F*]*u* & [*G*]*v*) → ([*R*]*tv* & [*R*]*uv* → $t =_E u$))›

AOT-define *MapsEOnto* :: ‹τ ⇒ τ ⇒ τ ⇒ φ› (‹- |: - $\longrightarrow_{onto}$*E* -›)
equi−*rem*:*3*:
‹*R* |: *F* $\longrightarrow_{onto}$*E* *G* $\equiv_{df}$ *R* |: *F* ⟶*E* *G* & ∀ *v* ([*G*]*v* → ∃ *u* ([*F*]*u* & [*R*]*uv*))›

AOT-define *MapsEOneToOneOnto* :: ‹τ ⇒ τ ⇒ τ ⇒ φ› (‹- |: - $_{1-1}\longrightarrow_{onto}$*E* -›)
equi−*rem*:*4*:
‹*R* |: *F* $_{1-1}\longrightarrow_{onto}$*E* *G* $\equiv_{df}$ *R* |: *F* $_{1-1}$⟶*E* *G* & *R* |: *F* $\longrightarrow_{onto}$*E* *G*›

AOT-theorem *equi*−*rem*−*thm*:

```
  ‹R |: F 1−1⟷E G ≡ R |: F 1−1⟶ontoE G›
proof −
  AOT-have ‹R |: F 1−1⟷E G ≡ R |: [λx O!x & [F]x] 1−1⟷ [λx O!x & [G]x]›
  proof(safe intro!: ≡I →I &I)
    AOT-assume ‹R |: F 1−1⟷E G›
    AOT-hence ‹∀u ([F]u → ∃!v ([G]v & [R]uv))›
          and ‹∀v ([G]v → ∃!u ([F]u & [R]uv))›
      using equi:2[THEN ≡df E] &E by blast+
    AOT-hence a: ‹([F]u → ∃!v ([G]v & [R]uv))›
          and b: ‹([G]v → ∃!u ([F]u & [R]uv))› for u v
      using Ordinary.∀E by fast+
    AOT-have ‹([λx [O!]x & [F]x]x → ∃!y ([λx [O!]x & [G]x]y & [R]xy))› for x
      apply (AOT-subst ‹[λx [O!]x & [F]x]x› ‹[O!]x & [F]x›)
       apply (rule beta−C−meta[THEN →E])
       apply cqt:2[lambda]
      apply (AOT-subst ‹[λx [O!]x & [G]x]x› ‹[O!]x & [G]x› for: x)
       apply (rule beta−C−meta[THEN →E])
       apply cqt:2[lambda]
      apply (AOT-subst ‹O!y & [G]y & [R]xy› ‹O!y & ([G]y & [R]xy)› for: y)
       apply (meson ≡E(6) Associativity of & oth−class−taut:3:a)
      apply (rule →I) apply (frule &E(1)) apply (drule &E(2))
      by (fact a[unconstrain u, THEN →E, THEN →E, of x])
    AOT-hence A: ‹∀x ([λx [O!]x & [F]x]x → ∃!y ([λx [O!]x & [G]x]y & [R]xy))›
      by (rule GEN)
    AOT-have ‹([λx [O!]x & [G]x]y → ∃!x ([λx [O!]x & [F]x]x & [R]xy))› for y
      apply (AOT-subst ‹[λx [O!]x & [G]x]y› ‹[O!]y & [G]y›)
       apply (rule beta−C−meta[THEN →E])
       apply cqt:2[lambda]
      apply (AOT-subst ‹[λx [O!]x & [F]x]x› ‹[O!]x & [F]x› for: x)
       apply (rule beta−C−meta[THEN →E])
       apply cqt:2[lambda]
      apply (AOT-subst ‹O!x & [F]x & [R]xy› ‹O!x & ([F]x & [R]xy)› for: x)
       apply (meson ≡E(6) Associativity of & oth−class−taut:3:a)
      apply (rule →I) apply (frule &E(1)) apply (drule &E(2))
      by (fact b[unconstrain v, THEN →E, THEN →E, of y])
    AOT-hence B: ‹∀y ([λx [O!]x & [G]x]y → ∃!x ([λx [O!]x & [F]x]x & [R]xy))›
      by (rule GEN)
    AOT-show ‹R |: [λx [O!]x & [F]x] 1−1⟷ [λx [O!]x & [G]x]›
      by (safe intro!: 1−1−cor[THEN ≡df I] &I
                        cqt:2[const-var][axiom-inst] A B)
          cqt:2[lambda]+
  next
    AOT-assume ‹R |: [λx [O!]x & [F]x] 1−1⟷ [λx [O!]x & [G]x]›
    AOT-hence a: ‹([λx [O!]x & [F]x]x → ∃!y ([λx [O!]x & [G]x]y & [R]xy))› and
              b: ‹([λx [O!]x & [G]x]y → ∃!x ([λx [O!]x & [F]x]x & [R]xy))› for x y
      using 1−1−cor[THEN ≡df E] &E ∀E(2) by blast+
    AOT-have ‹[F]u → ∃!v ([G]v & [R]uv)› for u
    proof (safe intro!: →I)
      AOT-assume fu: ‹[F]u›
      AOT-have 0: ‹[λx [O!]x & [F]x]u›
        by (auto intro!: β←C(1) cqt:2 cqt:2[const-var][axiom-inst]
                          Ordinary.ψ fu &I)
      AOT-show ‹∃!v ([G]v & [R]uv)›
        apply (AOT-subst ‹[O!]x & ([G]x & [R]ux)›
                          ‹([O!]x & [G]x) & [R]ux› for: x)
         apply (simp add: Associativity of &)
        apply (AOT-subst (reverse) ‹[O!]x & [G]x›
                                    ‹[λx [O!]x & [G]x]x› for: x)
         apply (rule beta−C−meta[THEN →E])
         apply cqt:2[lambda]
        using a[THEN →E, OF 0] by blast
    qed
    AOT-hence A: ‹∀u ([F]u → ∃!v ([G]v & [R]uv))›
```

by (*rule Ordinary.GEN*)
AOT-have ‹$[G]v \to \exists!u$ ($[F]u$ & $[R]uv$)› **for** v
proof (*safe intro*!: $\to I$)
AOT-assume *gu*: ‹$[G]v$›
AOT-have *0*: ‹$[\lambda x$ $[O!]x$ & $[G]x]v$›
by (*auto intro*!: $\beta{\leftarrow}C(1)$ *cqt*:*2* *cqt*:*2*[*const-var*][*axiom-inst*]
Ordinary.ψ *gu* &*I*)
AOT-show ‹$\exists!u$ ($[F]u$ & $[R]uv$)›
apply (*AOT-subst* ‹$[O!]x$ & ($[F]x$ & $[R]xv$)› ‹(($[O!]x$ & $[F]x$) & $[R]xv$› **for**: x)
apply (*simp add*: *Associativity of* &)
apply (*AOT-subst* (*reverse*) ‹$[O!]x$ & $[F]x$›‹$[\lambda x$ $[O!]x$ & $[F]x]x$› **for**: x)
apply (*rule beta−C−meta*[*THEN* $\to E$])
apply *cqt*:*2*[*lambda*]
using *b*[*THEN* $\to E$, *OF 0*] **by** *blast*
qed
AOT-hence *B*: ‹$\forall v$ ($[G]v \to \exists!u$ ($[F]u$ & $[R]uv$))› **by** (*rule Ordinary.GEN*)
AOT-show ‹R |: F $_{1-1}\longleftrightarrow_E$ G›
by (*safe intro*!: *equi*:*2*[*THEN* $\equiv_{df} I$] &*I* *A* *B* *cqt*:*2*[*const-var*][*axiom-inst*])
qed
also AOT-have ‹... $\equiv$ R |: F $_{1-1}\longrightarrow_{onto} E$ G›
proof(*safe intro*!: $\equiv I$ $\to I$ &*I*)
AOT-assume ‹R |: $[\lambda x$ $[O!]x$ & $[F]x]$ $_{1-1}\longleftrightarrow$ $[\lambda x$ $[O!]x$ & $[G]x]$›
AOT-hence *a*: ‹($[\lambda x$ $[O!]x$ & $[F]x]x \to \exists!y$ ($[\lambda x$ $[O!]x$ & $[G]x]y$ & $[R]xy$))› **and**
b: ‹($[\lambda x$ $[O!]x$ & $[G]x]y \to \exists!x$ ($[\lambda x$ $[O!]x$ & $[F]x]x$ & $[R]xy$))› **for** x y
using *1−1−cor*[*THEN* $\equiv_{df} E$] &*E* $\forall E(2)$ **by** *blast*+
AOT-show ‹R |: F $_{1-1}\longrightarrow_{onto} E$ G›
proof (*safe intro*!: *equi−rem*:*4*[*THEN* $\equiv_{df} I$] &*I* *equi−rem*:*3*[*THEN* $\equiv_{df} I$]
equi−rem:*2*[*THEN* $\equiv_{df} I$] *equi−rem*:*1*[*THEN* $\equiv_{df} I$]
cqt:*2*[*const-var*][*axiom-inst*] *Ordinary.GEN* $\to I$)
fix u
AOT-assume *fu*: ‹$[F]u$›
AOT-have *0*: ‹$[\lambda x$ $[O!]x$ & $[F]x]u$›
by (*auto intro*!: $\beta{\leftarrow}C(1)$ *cqt*:*2* *cqt*:*2*[*const-var*][*axiom-inst*]
Ordinary.ψ *fu* &*I*)
AOT-hence *1*: ‹$\exists!y$ ($[\lambda x$ $[O!]x$ & $[G]x]y$ & $[R]uy$)›
using *a*[*THEN* $\to E$] **by** *blast*
AOT-show ‹$\exists!v$ ($[G]v$ & $[R]uv$)›
apply (*AOT-subst* ‹$[O!]x$ & ($[G]x$ & $[R]ux$)› ‹(($[O!]x$ & $[G]x$) & $[R]ux$› **for**: x)
apply (*simp add*: *Associativity of* &)
apply (*AOT-subst* (*reverse*) ‹$[O!]x$ & $[G]x$› ‹$[\lambda x$ $[O!]x$ & $[G]x]x$› **for**: x)
apply (*rule beta−C−meta*[*THEN* $\to E$])
apply *cqt*:*2*[*lambda*]
by (*fact 1*)
next
fix t u v
AOT-assume ‹$[F]t$ & $[F]u$ & $[G]v$› **and** *rtv-tuv*: ‹$[R]tv$ & $[R]uv$›
AOT-hence *oft*: ‹$[\lambda x$ $O!x$ & $[F]x]t$› **and**
ofu: ‹$[\lambda x$ $O!x$ & $[F]x]u$› **and**
ogv: ‹$[\lambda x$ $O!x$ & $[G]x]v$›
by (*auto intro*!: $\beta{\leftarrow}C(1)$ *cqt*:*2* &*I*
simp: *Ordinary*.ψ *dest*: &*E*)
AOT-hence ‹$\exists!x$ ($[\lambda x$ $[O!]x$ & $[F]x]x$ & $[R]xv$)›
using *b*[*THEN* $\to E$] **by** *blast*
then AOT-obtain a **where**
a-prop: ‹$[\lambda x$ $[O!]x$ & $[F]x]a$ & $[R]av$ &
$\forall x$ (($[\lambda x$ $[O!]x$ & $[F]x]x$ & $[R]xv$) $\to$ $x = a$)›
using *uniqueness*:*1*[*THEN* $\equiv_{df} E$] $\exists E$[*rotated*] **by** *blast*
AOT-hence *ua*: ‹$u = a$›
using *ofu* *rtv-tuv*[*THEN* &*E*(*2*)] $\forall E(2)$ $\to E$ &*I* &*E*(*2*) **by** *blast*
moreover AOT-have *ta*: ‹$t = a$›
using *a-prop* *oft* *rtv-tuv*[*THEN* &*E*(*1*)] $\forall E(2)$ $\to E$ &*I* &*E*(*2*) **by** *blast*
ultimately AOT-have ‹$t = u$› **by** (*metis rule*=*E* *id-sym*)
AOT-thus ‹$t =_E u$›

```
      using rule=E id-sym ord=Eequiv:1 Ordinary.ψ ta ua →E by fast
  next
    fix u
    AOT-assume ‹[F]u›
    AOT-hence ‹[λx O!x & [F]x]u›
      by (auto intro!: β←C(1) cqt:2 &I
              simp: cqt:2[const-var][axiom-inst]  Ordinary.ψ)
    AOT-hence ‹∃!y ([λx [O!]x & [G]x]y & [R]uy)›
      using a[THEN →E] by blast
    then AOT-obtain a where
      a-prop: ‹[λx [O!]x & [G]x]a & [R]ua &
              ∀x (([λx [O!]x & [G]x]x & [R]ux) → x = a)›
      using uniqueness:1[THEN ≡df E] ∃E[rotated] by blast
    AOT-have ‹O!a & [G]a›
      by (rule β→C(1)) (auto simp: a-prop[THEN &E(1), THEN &E(1)])
    AOT-hence ‹O!a› and ‹[G]a› using &E by blast+
    moreover AOT-have ‹∀v ([G]v & [R]uv → v =E a)›
    proof(safe intro!: Ordinary.GEN →I; frule &E(1); drule &E(2))
      fix v
      AOT-assume ‹[G]v› and ruv: ‹[R]uv›
      AOT-hence ‹[λx [O!]x & [G]x]v›
        by (auto intro!: β←C(1) cqt:2 &I simp: Ordinary.ψ)
      AOT-hence ‹v = a›
        using a-prop[THEN &E(2), THEN ∀E(2), THEN →E, OF &I] ruv by blast
      AOT-thus ‹v =E a›
        using rule=E ord=Eequiv:1 Ordinary.ψ →E by fast
    qed
    ultimately AOT-have ‹O!a & ([G]a & [R]ua & ∀v' ([G]v' & [R]uv' → v' =E a))›
      using ∃I &I a-prop[THEN &E(1), THEN &E(2)] by simp
    AOT-hence ‹∃v ([G]v & [R]uv & ∀v' ([G]v' & [R]uv' → v' =E v))›
      by (rule ∃I)
    AOT-thus ‹∃!v ([G]v & [R]uv)›
      by (rule equi:1[THEN ≡E(2)])
  next
    fix v
    AOT-assume ‹[G]v›
    AOT-hence ‹[λx O!x & [G]x]v›
      by (auto intro!: β←C(1) cqt:2 &I Ordinary.ψ)
    AOT-hence ‹∃!x ([λx [O!]x & [F]x]x & [R]xv)›
      using b[THEN →E] by blast
    then AOT-obtain a where
      a-prop: ‹[λx [O!]x & [F]x]a & [R]av &
              ∀y ([λx [O!]x & [F]x]y & [R]yv → y = a)›
      using uniqueness:1[THEN ≡df E, THEN ∃E[rotated]] by blast
    AOT-have ‹O!a & [F]a›
      by (rule β→C(1)) (auto simp: a-prop[THEN &E(1), THEN &E(1)])
    AOT-hence ‹O!a & ([F]a & [R]av)›
      using a-prop[THEN &E(1), THEN &E(2)] &E &I by metis
    AOT-thus ‹∃u ([F]u & [R]uv)›
      by (rule ∃I)
  qed
next
  AOT-assume ‹R |: F 1−1⟶onto E G›
  AOT-hence 1: ‹R |: F 1−1⟶E G›
       and 2: ‹R |: F ⟶onto E G›
    using equi−rem:4[THEN ≡df E] &E by blast+
  AOT-hence 3: ‹R |: F ⟶E G›
       and A: ‹∀t ∀u ∀v ([F]t & [F]u & [G]v → ([R]tv & [R]uv → t =E u))›
    using equi−rem:2[THEN ≡df E, OF 1] &E by blast+
  AOT-hence B: ‹∀u ([F]u → ∃!v ([G]v & [R]uv))›
    using equi−rem:1[THEN ≡df E] &E by blast
  AOT-have C: ‹∀v ([G]v → ∃u ([F]u & [R]uv))›
    using equi−rem:3[THEN ≡df E, OF 2] &E by blast
```

```
AOT-show ‹R |: [λx [O!]x & [F]x] 1−1⟷ [λx [O!]x & [G]x]›
proof (rule 1−1−cor[THEN ≡df I];
      safe intro!: &I cqt:2 GEN →I)
  fix x
  AOT-assume 1: ‹[λx [O!]x & [F]x]x›
  AOT-have ‹O!x & [F]x›
    by (rule β→C(1)) (auto simp: 1)
  AOT-hence ‹∃!v ([G]v & [R]xv)›
    using B[THEN ∀E(2), THEN →E, THEN →E] &E by blast
  then AOT-obtain y where
    y-prop: ‹O!y & ([G]y & [R]xy & ∀u ([G]u & [R]xu → u =E y))›
    using equi:1[THEN ≡E(1)] ∃E[rotated] by fastforce
  AOT-hence ‹[λx O!x & [G]x]y›
    by (auto intro!: β←C(1) cqt:2 &I dest: &E)
  moreover AOT-have ‹∀z ([λx O!x & [G]x]z & [R]xz → z = y)›
  proof(safe intro!: GEN →I; frule &E(1); drule &E(2))
    fix z
    AOT-assume 1: ‹[λx [O!]x & [G]x]z›
    AOT-have 2: ‹O!z & [G]z›
      by (rule β→C(1)) (auto simp: 1)
    moreover AOT-assume ‹[R]xz›
    ultimately AOT-have ‹z =E y›
      using y-prop[THEN &E(2), THEN &E(2), THEN ∀E(2),
                THEN →E, THEN →E, rotated, OF &I] &E
      by blast
    AOT-thus ‹z = y›
      using 2[THEN &E(1)] by (metis =E−simple:2 →E)
  qed
  ultimately AOT-have ‹[λx O!x & [G]x]y & [R]xy &
                  ∀z ([λx O!x & [G]x]z & [R]xz → z = y)›
    using y-prop[THEN &E(2), THEN &E(1), THEN &E(2)] &I by auto
  AOT-hence ‹∃y ([λx O!x & [G]x]y & [R]xy &
            ∀z ([λx O!x & [G]x]z & [R]xz → z = y))›
    by (rule ∃I)
  AOT-thus ‹∃!y ([λx [O!]x & [G]x]y & [R]xy)›
    using uniqueness:1[THEN ≡df I] by fast
next
  fix y
  AOT-assume 1: ‹[λx [O!]x & [G]x]y›
  AOT-have oy-gy: ‹O!y & [G]y›
    by (rule β→C(1)) (auto simp: 1)
  AOT-hence ‹∃u ([F]u & [R]uy)›
    using C[THEN ∀E(2), THEN →E, THEN →E] &E by blast
  then AOT-obtain x where x-prop: ‹O!x & ([F]x & [R]xy)›
    using ∃E[rotated] by blast
  AOT-hence ofx: ‹[λx O!x & [F]x]x›
    by (auto intro!: β←C(1) cqt:2 &I dest: &E)
  AOT-have ‹∃α ([λx [O!]x & [F]x]α & [R]αy &
            ∀β ([λx [O!]x & [F]x]β & [R]βy → β = α))›
  proof (safe intro!: ∃I(2)[where β=x] &I GEN →I)
    AOT-show ‹[λx O!x & [F]x]x› using ofx.
  next
    AOT-show ‹[R]xy› using x-prop[THEN &E(2), THEN &E(2)].
  next
    fix z
    AOT-assume 1: ‹[λx [O!]x & [F]x]z & [R]zy›
    AOT-have oz-fz: ‹O!z & [F]z›
      by (rule β→C(1)) (auto simp: 1[THEN &E(1)])
    AOT-have ‹z =E x›
      using A[THEN ∀E(2)[where β=z], THEN →E, THEN ∀E(2)[where β=x],
            THEN →E, THEN ∀E(2)[where β=y], THEN →E,
            THEN →E, THEN →E, OF oz-fz[THEN &E(1)],
            OF x-prop[THEN &E(1)], OF oy-gy[THEN &E(1)], OF &I, OF &I,
```

OF oz-fz[*THEN* &*E*(*2*)], *OF x-prop*[*THEN* &*E*(*2*), *THEN* &*E*(*1*)],
OF oy-gy[*THEN* &*E*(*2*)], *OF* &*I*, *OF 1*[*THEN* &*E*(*2*)],
OF x-prop[*THEN* &*E*(*2*), *THEN* &*E*(*2*)]]**.**
AOT-thus ‹$z = x$›
by (*metis =E−simple*:*2 vdash−properties*:*10*)
qed
AOT-thus ‹$\exists!x$ ([λx [$O!$]x & [F]x]x & [R]xy)›
by (*rule uniqueness*:*1*[*THEN* $\equiv_{df}I$])
qed
qed
finally show *?thesis***.**
qed

AOT-theorem *empty−approx*:*1*: ‹($\neg\exists u$ [F]u & $\neg\exists v$ [H]v) → $F \approx_E H$›
proof(*rule* →*I*; *frule* &*E*(*1*); *drule* &*E*(*2*))
AOT-assume *0*: ‹$\neg\exists u$ [F]u› **and** *1*: ‹$\neg\exists v$ [H]v›
AOT-have ‹$\forall u$ ([F]u → $\exists!v$ ([H]v & [R]uv))› **for** R
proof(*rule Ordinary.GEN*; *rule* →*I*; *rule raa−cor*:*1*)
fix u
AOT-assume ‹[F]u›
AOT-hence ‹$\exists u$ [F]u› **using** *Ordinary.∃I* &*I* **by** *fast*
AOT-thus ‹$\exists u$ [F]u & $\neg\exists u$ [F]u› **using** &*I 0* **by** *blast*
qed
moreover AOT-have ‹$\forall v$ ([H]v → $\exists!u$ ([F]u & [R]uv))› **for** R
proof(*rule Ordinary.GEN*; *rule* →*I*; *rule raa−cor*:*1*)
fix v
AOT-assume ‹[H]v›
AOT-hence ‹$\exists v$ [H]v› **using** *Ordinary.∃I* &*I* **by** *fast*
AOT-thus ‹$\exists v$ [H]v & $\neg\exists v$ [H]v› **using** *1* &*I* **by** *blast*
qed
ultimately AOT-have ‹R |: F $_{1-1}\!\!\longleftrightarrow_E$ H› **for** R
apply (*safe intro*!: *equi*:*2*[*THEN* $\equiv_{df}I$] &*I GEN cqt*:*2*[*const-var*][*axiom-inst*])
using ∀*E* **by** *blast+*
AOT-hence ‹$\exists R$ R |: F $_{1-1}\!\!\longleftrightarrow_E$ H› **by** (*rule* ∃*I*)
AOT-thus ‹$F \approx_E H$›
by (*rule equi*:*3*[*THEN* $\equiv_{df}I$])
qed

AOT-theorem *empty−approx*:*2*: ‹($\exists u$ [F]u & $\neg\exists v$ [H]v) → $\neg$($F \approx_E H$)›
proof(*rule* →*I*; *frule* &*E*(*1*); *drule* &*E*(*2*); *rule raa−cor*:*2*)
AOT-assume *1*: ‹$\exists u$ [F]u› **and** *2*: ‹$\neg\exists v$ [H]v›
AOT-obtain b **where** *b-prop*: ‹$O!b$ & [F]b›
using *1* ∃*E*[*rotated*] **by** *blast*
AOT-assume ‹$F \approx_E H$›
AOT-hence ‹$\exists R$ R |: F $_{1-1}\!\!\longleftrightarrow_E$ H›
by (*rule equi*:*3*[*THEN* $\equiv_{df}E$])
then AOT-obtain R **where** ‹R |: F $_{1-1}\!\!\longleftrightarrow_E$ H›
using ∃*E*[*rotated*] **by** *blast*
AOT-hence ϑ: ‹$\forall u$ ([F]u → $\exists!v$ ([H]v & [R]uv))›
using *equi*:*2*[*THEN* $\equiv_{df}E$] &*E* **by** *blast+*
AOT-have ‹$\exists!v$ ([H]v & [R]bv)› **for** u
using ϑ[*THEN* ∀*E*(*2*)[**where** β=b], *THEN* →*E*, *THEN* →*E*,
OF b-prop[*THEN* &*E*(*1*)], *OF b-prop*[*THEN* &*E*(*2*)]]**.**
AOT-hence ‹$\exists v$ ([H]v & [R]bv & $\forall u$ ([H]u & [R]bu → $u =_E v$))›
by (*rule equi*:*1*[*THEN* ≡*E*(*1*)])
then AOT-obtain x **where** ‹$O!x$ & ([H]x & [R]bx & $\forall u$ ([H]u & [R]bu → $u =_E x$))›
using ∃*E*[*rotated*] **by** *blast*
AOT-hence ‹$O!x$ & [H]x› **using** &*E* &*I* **by** *blast*
AOT-hence ‹$\exists v$ [H]v› **by** (*rule* ∃*I*)
AOT-thus ‹$\exists v$ [H]v & $\neg\exists v$ [H]v› **using** *2* &*I* **by** *blast*
qed

AOT-define *FminusU* :: ‹Π ⇒ τ ⇒ Π› (‹-$^{--}$›)
F−u: ‹$[F]^{-x}$ $=_{df}$ [λz [F]z & z $\neq_E$ x]›

Note: not explicitly in PLM.

AOT-theorem *F−u[den]*: ‹$[F]^{-x}$↓›
by (*rule* $=_{df}$*I(1)[OF F−u,* **where** $\tau_1\tau_n$=(-,-), *simplified*]; *cqt:2[lambda]*)
AOT-theorem *F−u[equiv]*: ‹$[[F]^{-x}]$y ≡ ([F]y & y $\neq_E$ x)›
by (*auto intro*: *F−u[THEN* $=_{df}$*I(1)*, **where** $\tau_1\tau_n$=(-,-), *simplified*]
intro!: *cqt:2 beta−C−cor:2[THEN →E, THEN ∀ E(2)]*)

AOT-theorem *eqP′*: ‹F $\approx_E$ G & [F]u & [G]v → $[F]^{-u}$ $\approx_E$ $[G]^{-v}$›
proof (*rule →I*; *frule &E(2)*; *drule &E(1)*; *frule &E(2)*; *drule &E(1)*)
AOT-assume ‹F $\approx_E$ G›
AOT-hence ‹∃ R R |: F $_{1-1}$⟷$_E$ G›
using *equi:3[THEN* $\equiv_{df}$*E]* **by** *blast*
then AOT-obtain R **where** *R-prop*: ‹R |: F $_{1-1}$⟷$_E$ G›
using ∃ *E[rotated]* **by** *blast*
AOT-hence *A*: ‹∀ u ([F]u → ∃!v ([G]v & [R]uv))›
and *B*: ‹∀ v ([G]v → ∃!u ([F]u & [R]uv))›
using *equi:2[THEN* $\equiv_{df}$*E] &E* **by** *blast+*
AOT-have ‹R |: F $_{1-1}$⟶$_{onto}$E G›
using *equi−rem−thm[THEN ≡E(1), OF R-prop]*.
AOT-hence ‹R |: F $_{1-1}$⟶E G & R |: F ⟶$_{onto}$E G›
using *equi−rem:4[THEN* $\equiv_{df}$*E]* **by** *blast*
AOT-hence *C*: ‹∀ t∀ u∀ v (([F]t & [F]u & [G]v) → ([R]tv & [R]uv → t $=_E$ u))›
using *equi−rem:2[THEN* $\equiv_{df}$*E] &E* **by** *blast*
AOT-assume *fu*: ‹[F]u›
AOT-assume *gv*: ‹[G]v›
AOT-have ‹[λz [Π]z & z $\neq_E$ κ]↓› **for** Π κ
by *cqt:2[lambda]*
note Π*-minus-κI = rule−id−df:2:b[2][*
where τ=‹(λ(Π, κ). «$[\Pi]^{-\kappa}$»)›, *simplified*, *OF F−u*, *simplified*, *OF this*]
and Π*-minus-κE = rule−id−df:2:a[2][*
where τ=‹(λ(Π, κ). «$[\Pi]^{-\kappa}$»)›, *simplified*, *OF F−u*, *simplified*, *OF this*]
AOT-have Π*-minus-κ-den*: ‹$[\Pi]^{-\kappa}$↓› **for** Π κ
by (*rule* Π*-minus-κI*) *cqt:2[lambda]+*
{
fix R
AOT-assume *R-prop*: ‹R |: F $_{1-1}$⟷$_E$ G›
AOT-hence *A*: ‹∀ u ([F]u → ∃!v ([G]v & [R]uv))›
and *B*: ‹∀ v ([G]v → ∃!u ([F]u & [R]uv))›
using *equi:2[THEN* $\equiv_{df}$*E] &E* **by** *blast+*
AOT-have ‹R |: F $_{1-1}$⟶$_{onto}$E G›
using *equi−rem−thm[THEN ≡E(1), OF R-prop]*.
AOT-hence ‹R |: F $_{1-1}$⟶E G & R |: F ⟶$_{onto}$E G›
using *equi−rem:4[THEN* $\equiv_{df}$*E]* **by** *blast*
AOT-hence *C*: ‹∀ t∀ u∀ v (([F]t & [F]u & [G]v) → ([R]tv & [R]uv → t $=_E$ u))›
using *equi−rem:2[THEN* $\equiv_{df}$*E] &E* **by** *blast*

AOT-assume *Ruv*: ‹[R]uv›
AOT-have ‹R |: $[F]^{-u}$ $_{1-1}$⟷$_E$ $[G]^{-v}$›
proof(*safe intro*!: *equi:2[THEN* $\equiv_{df}$*I] &I cqt:2[const-var][axiom-inst]*
Π*-minus-κ-den Ordinary.GEN →I*)
fix u′
AOT-assume ‹$[[F]^{-u}]$u′›
AOT-hence *0*: ‹[λz [F]z & z $\neq_E$ u]u′›
using Π*-minus-κE* **by** *fast*
AOT-have *0*: ‹[F]u′ & u′ $\neq_E$ u›
by (*rule β→C(1)*[**where** $\kappa_1\kappa_n$=*AOT-term-of-var* (*Ordinary.Rep u′*)]) (*fact 0*)
AOT-have ‹∃!v ([G]v & [R]u′v)›
using *A[THEN Ordinary.∀ E[***where** α=u′], *THEN →E*, *OF 0[THEN &E(1)]]*.
then AOT-obtain v′ **where**
v′-prop: ‹[G]v′ & [R]u′v′ & ∀ t ([G]t & [R]u′t → t $=_E$ v′)›

```
    using equi:1[THEN ≡E(1)] Ordinary.∃ E[rotated] by fastforce

  AOT-show ‹∃!v' ([[G]⁻ᵛ]v' & [R]u'v')›
  proof (safe intro!: equi:1[THEN ≡E(2)] Ordinary.∃ I[where β=v']
                  &I Ordinary.GEN →I)
    AOT-show ‹[[G]⁻ᵛ]v'›
    proof (rule Π-minus-κI;
          safe intro!: β←C(1) cqt:2 &I thm−neg=E[THEN ≡E(2)])
      AOT-show ‹[G]v'› using v'-prop &E by blast
    next
      AOT-show ‹¬v' =E v›
      proof (rule raa−cor:2)
        AOT-assume ‹v' =E v›
        AOT-hence ‹v' = v› by (metis =E−simple:2 →E)
        AOT-hence Ruv': ‹[R]uv'› using rule=E Ruv id-sym by fast
        AOT-have ‹u' =E u›
          by (rule C[THEN Ordinary.∀ E, THEN Ordinary.∀ E,
                  THEN Ordinary.∀ E[where α=v'], THEN →E, THEN →E])
            (safe intro!: &I 0[THEN &E(1)] fu
                       v'-prop[THEN &E(1), THEN &E(1)]
                       Ruv' v'-prop[THEN &E(1), THEN &E(2)])
        moreover AOT-have ‹¬(u' =E u)›
          using 0 &E(2) ≡E(1) thm−neg=E by blast
        ultimately AOT-show ‹u' =E u & ¬u' =E u› using &I by blast
      qed
    qed
  next
    AOT-show ‹[R]u'v'› using v'-prop &E by blast
  next
    fix t
    AOT-assume t-prop: ‹[[G]⁻ᵛ]t & [R]u't›
    AOT-have gt-t-noteq-v: ‹[G]t & t ≠E v›
      apply (rule β→C(1)[where κ₁κₙ=AOT-term-of-var (Ordinary.Rep t)])
      apply (rule Π-minus-κE)
      by (fact t-prop[THEN &E(1)])
    AOT-show ‹t =E v'›
      using v'-prop[THEN &E(2), THEN Ordinary.∀ E, THEN →E,
                 OF &I, OF gt-t-noteq-v[THEN &E(1)],
                 OF t-prop[THEN &E(2)]].
  qed
next
  fix v'
  AOT-assume G-minus-v-v': ‹[[G]⁻ᵛ]v'›
  AOT-have gt-t-noteq-v: ‹[G]v' & v' ≠E v›
    apply (rule β→C(1)[where κ₁κₙ=AOT-term-of-var (Ordinary.Rep v')])
    apply (rule Π-minus-κE)
    by (fact G-minus-v-v')
  AOT-have ‹∃!u([F]u & [R]uv')›
    using B[THEN Ordinary.∀ E, THEN →E, OF gt-t-noteq-v[THEN &E(1)]].
  then AOT-obtain u' where
    u'-prop: ‹[F]u' & [R]u'v' & ∀ t ([F]t & [R]tv' → t =E u')›
    using equi:1[THEN ≡E(1)] Ordinary.∃ E[rotated] by fastforce
  AOT-show ‹∃!u' ([[F]⁻ᵘ]u' & [R]u'v')›
  proof (safe intro!: equi:1[THEN ≡E(2)] Ordinary.∃ I[where β=u'] &I
                  u'-prop[THEN &E(1), THEN &E(2)] Ordinary.GEN →I)
    AOT-show ‹[[F]⁻ᵘ]u'›
    proof (rule Π-minus-κI;
          safe intro!: β←C(1) cqt:2 &I thm−neg=E[THEN ≡E(2)]
          u'-prop[THEN &E(1), THEN &E(1)]; rule raa−cor:2)
      AOT-assume u'-eq-u: ‹u' =E u›
      AOT-hence ‹u' = u›
        using =E−simple:2 vdash−properties:10 by blast
      AOT-hence Ru'v: ‹[R]u'v› using rule=E Ruv id-sym by fast
```

```
        AOT-have ‹v' ≠E v›
          using &E(2) gt-t-noteq-v by blast
        AOT-hence v'-noteq-v: ‹¬(v' =E v)› by (metis ≡E(1) thm−neg=E)
        AOT-have ‹∃ u ([G]u & [R]u'u & ∀ v ([G]v & [R]u'v → v =E u))›
          using A[THEN Ordinary.∀ E, THEN →E,
                OF u'-prop[THEN &E(1), THEN &E(1)],
                THEN equi:1[THEN ≡E(1)]].
        then AOT-obtain t where
          t-prop: ‹[G]t & [R]u't & ∀ v ([G]v & [R]u'v → v =E t)›
          using Ordinary.∃ E[rotated] by meson
        AOT-have ‹v =E t› if ‹[G]v› and ‹[R]u'v› for v
          using t-prop[THEN &E(2), THEN Ordinary.∀ E, THEN →E,
                  OF &I, OF that].
        AOT-hence ‹v' =E t› and ‹v =E t›
          by (auto simp: gt-t-noteq-v[THEN &E(1)] Ru'v gv
                     u'-prop[THEN &E(1), THEN &E(2)])
        AOT-hence ‹v' =E v›
          using rule=E =E−simple:2 id-sym →E by fast
        AOT-thus ‹v' =E v & ¬v' =E v›
          using v'-noteq-v &I by blast
      qed
    next
      fix t
      AOT-assume 0: ‹[[F]^−u]t & [R]tv'›
      moreover AOT-have ‹[F]t & t ≠E u›
        apply (rule β→C(1)[where κ1κn=AOT-term-of-var (Ordinary.Rep t)])
        apply (rule Π-minus-κE)
        by (fact 0[THEN &E(1)])
      ultimately AOT-show ‹t =E u'›
        using u'-prop[THEN &E(2), THEN Ordinary.∀ E, THEN →E, OF &I]
            &E by blast
    qed
  qed
  AOT-hence ‹∃ R R |: [F]^−u 1−1⟷E [G]^−v›
    by (rule ∃ I)
} note 1 = this
moreover {
  AOT-assume not-Ruv: ‹¬[R]uv›
  AOT-have ‹∃!v ([G]v & [R]uv)›
    using A[THEN Ordinary.∀ E, THEN →E, OF fu].
  then AOT-obtain b where
    b-prop: ‹O!b & ([G]b & [R]ub & ∀ t([G]t & [R]ut → t =E b))›
    using equi:1[THEN ≡E(1)] ∃ E[rotated] by fastforce
  AOT-hence ob: ‹O!b› and gb: ‹[G]b› and Rub: ‹[R]ub›
    using &E by blast+
  AOT-have ‹O!t → ([G]t & [R]ut → t =E b)› for t
    using b-prop &E(2) ∀ E(2) by blast
  AOT-hence b-unique: ‹t =E b› if ‹O!t› and ‹[G]t› and ‹[R]ut› for t
    by (metis Adjunction modus−tollens:1 reductio−aa:1 that)
  AOT-have not-v-eq-b: ‹¬(v =E b)›
  proof(rule raa−cor:2)
    AOT-assume ‹v =E b›
    AOT-hence 0: ‹v = b›
      by (metis =E−simple:2 →E)
    AOT-have ‹[R]uv›
      using b-prop[THEN &E(2), THEN &E(1), THEN &E(2)]
          rule=E[rotated, OF 0[symmetric]] by fast
    AOT-thus ‹[R]uv & ¬[R]uv›
      using not-Ruv &I by blast
  qed
  AOT-have not-b-eq-v: ‹¬(b =E v)›
    using modus−tollens:1 not-v-eq-b ord=Eequiv:2 by blast
  AOT-have ‹∃!u ([F]u & [R]uv)›
```

using B[*THEN Ordinary*.$\forall E$, *THEN* $\rightarrow E$, *OF gv*]**.**
then AOT-obtain a **where**
a-prop: ‹$O!a$ & ([F]a & [R]av & $\forall t$([F]t & [R]$tv \rightarrow t =_E a$))›
using *equi*:*1*[*THEN* $\equiv E(1)$] $\exists E$[*rotated*] **by** *fastforce*
AOT-hence *Oa*: ‹$O!a$› **and** *fa*: ‹[F]a› **and** *Rav*: ‹[R]av›
using &E **by** *blast+*
AOT-have ‹$O!t \rightarrow$ ([F]t & [R]$tv \rightarrow t =_E a$)› **for** t
using *a-prop* &E $\forall E(2)$ **by** *blast*
AOT-hence *a-unique*: ‹$t =_E a$› **if** ‹$O!t$› **and** ‹[F]t› **and** ‹[R]tv› **for** t
by (*metis Adjunction modus−tollens*:*1 reductio−aa*:*1 that*)
AOT-have *not-u-eq-a*: ‹$\neg(u =_E a)$›
proof(*rule raa−cor*:*2*)
AOT-assume ‹$u =_E a$›
AOT-hence *0*: ‹$u = a$›
by (*metis* $=E$*−simple*:*2* $\rightarrow E$)
AOT-have ‹[R]uv›
using *a-prop*[*THEN* &$E(2)$, *THEN* &$E(1)$, *THEN* &$E(2)$]
rule=E[*rotated*, *OF 0*[*symmetric*]] **by** *fast*
AOT-thus ‹[R]uv & $\neg$[R]uv›
using *not-Ruv* &I **by** *blast*
qed
AOT-have *not-a-eq-u*: ‹$\neg(a =_E u)$›
using *modus−tollens*:*1 not-u-eq-a ord*=*Eequiv*:*2* **by** *blast*
let *?R* = ‹«[$\lambda u'v'$ ($u' \neq_E u$ & $v' \neq_E v$ & [R]$u'v'$) $\vee$
($u' =_E a$ & $v' =_E b$) $\vee$
($u' =_E u$ & $v' =_E v$)]»›
AOT-have ‹[«*?R*»]$\downarrow$› **by** *cqt*:*2*[*lambda*]
AOT-hence ‹$\exists\ \beta\ \beta$ = [«*?R*»]›
using *free−thms*:*1* $\equiv E(1)$ **by** *fast*
then AOT-obtain R_1 **where** R_1*-def*: ‹R_1 = [«*?R*»]›
using $\exists E$[*rotated*] **by** *blast*
AOT-have *Rxy1*: ‹[R]xy› **if** ‹[R_1]xy› **and** ‹$x \neq_E u$› **and** ‹$x \neq_E a$› **for** x y
proof −
AOT-have *0*: ‹[«*?R*»]xy›
by (*rule rule*=E[*rotated*, *OF* R_1*-def*]) (*fact that*(*1*))
AOT-have ‹($x \neq_E u$ & $y \neq_E v$ & [R]xy) $\vee$ ($x =_E a$ & $y =_E b$) $\vee$ ($x =_E u$ & $y =_E v$)›
using $\beta{\rightarrow}C(1)$[*OF 0*] **by** *simp*
AOT-hence ‹$x \neq_E u$ & $y \neq_E v$ & [R]xy› **using** *that*(*2*,*3*)
by (*metis* $\vee E(3)$ *Conjunction Simplification*(*1*) $\equiv E(1)$
modus−tollens:*1 thm−neg*=E)
AOT-thus ‹[R]xy› **using** &E **by** *blast+*
qed
AOT-have *Rxy2*: ‹[R]xy› **if** ‹[R_1]xy› **and** ‹$y \neq_E v$› **and** ‹$y \neq_E b$› **for** x y
proof −
AOT-have *0*: ‹[«*?R*»]xy›
by (*rule rule*=E[*rotated*, *OF* R_1*-def*]) (*fact that*(*1*))
AOT-have ‹($x \neq_E u$ & $y \neq_E v$ & [R]xy) $\vee$ ($x =_E a$ & $y =_E b$) $\vee$ ($x =_E u$ & $y =_E v$)›
using $\beta{\rightarrow}C(1)$[*OF 0*] **by** *simp*
AOT-hence ‹$x \neq_E u$ & $y \neq_E v$ & [R]xy›
using *that*(*2*,*3*)
by (*metis* $\vee E(3)$ *Conjunction Simplification*(*2*) $\equiv E(1)$
modus−tollens:*1 thm−neg*=E)
AOT-thus ‹[R]xy› **using** &E **by** *blast+*
qed
AOT-have R_1xy: ‹[R_1]xy› **if** ‹[R]xy› **and** ‹$x \neq_E u$› **and** ‹$y \neq_E v$› **for** x y
by (*rule rule*=E[*rotated*, *OF* R_1*-def*[*symmetric*]])
(*auto intro*!: $\beta{\leftarrow}C(1)$ *cqt*:*2*
simp: &I *ex*:*1*:*a prod-denotesI rule−ui*:*3 that* $\vee I(1)$)
AOT-have R_1ab: ‹[R_1]ab›
apply (*rule rule*=E[*rotated*, *OF* R_1*-def*[*symmetric*]])
apply (*safe intro*!: $\beta{\leftarrow}C(1)$ *cqt*:*2 prod-denotesI* &I)
by (*meson a-prop b-prop* &I &$E(1)$ $\vee I(1)$ $\vee I(2)$ *ord*=*Eequiv*:*1* $\rightarrow E$)
AOT-have R_1uv: ‹[R_1]uv›

```
    apply (rule rule=E[rotated, OF R₁-def[symmetric]])
    apply (safe intro!: β←C(1) cqt:2 prod-denotesI &I)
    by (meson &I ∨I(2) ord=Eequiv:1 Ordinary.ψ →E)
  moreover AOT-have ‹R₁ |: F 1-1⟷E G›
  proof (safe intro!: equi:2[THEN ≡df I] &I cqt:2 Ordinary.GEN →I)
    fix u'
    AOT-assume fu': ‹[F]u'›
    {
      AOT-assume not-u'-eq-u: ‹¬(u' =E u)› and not-u'-eq-a: ‹¬(u' =E a)›
      AOT-hence u'-noteq-u: ‹u' ≠E u› and u'-noteq-a: ‹u' ≠E a›
        by (metis ≡E(2) thm−neg=E)+
      AOT-have ‹∃!v ([G]v & [R]u'v)›
        using A[THEN Ordinary.∀E, THEN →E, OF fu'].
      AOT-hence ‹∃v ([G]v & [R]u'v & ∀t ([G]t & [R]u't → t =E v))›
        using equi:1[THEN ≡E(1)] by simp
      then AOT-obtain v' where
        v'-prop: ‹[G]v' & [R]u'v' & ∀t ([G]t & [R]u't → t =E v')›
        using Ordinary.∃E[rotated] by meson
      AOT-hence gv': ‹[G]v'› and Ru'v': ‹[R]u'v'›
        using &E by blast+
      AOT-have not-v'-eq-v: ‹¬v' =E v›
      proof (rule raa−cor:2)
        AOT-assume ‹v' =E v›
        AOT-hence ‹v' = v›
          by (metis =E−simple:2 →E)
        AOT-hence Ru'v: ‹[R]u'v›
          using rule=E Ru'v' by fast
        AOT-have ‹u' =E a›
          using a-unique[OF Ordinary.ψ, OF fu', OF Ru'v].
        AOT-thus ‹u' =E a & ¬u' =E a›
          using not-u'-eq-a &I by blast
      qed
      AOT-hence v'-noteq-v: ‹v' ≠E v›
        using ≡E(2) thm−neg=E by blast
      AOT-have ‹∀t ([G]t & [R]u't → t =E v')›
        using v'-prop &E by blast
      AOT-hence ‹[G]t & [R]u't → t =E v'› for t
        using Ordinary.∀E by meson
      AOT-hence v'-unique: ‹t =E v'› if ‹[G]t› and ‹[R]u't› for t
        by (metis &I that →E)

      AOT-have ‹[G]v' & [R₁]u'v' & ∀t ([G]t & [R₁]u't → t =E v')›
      proof (safe intro!: &I gv' R₁xy Ru'v' u'-noteq-u u'-noteq-a →I
                          Ordinary.GEN thm−neg=E[THEN ≡E(2)] not-v'-eq-v)
        fix t
        AOT-assume 1: ‹[G]t & [R₁]u't›
        AOT-have ‹[R]u't›
          using Rxy1[OF 1[THEN &E(2)], OF u'-noteq-u, OF u'-noteq-a].
        AOT-thus ‹t =E v'›
          using v'-unique 1[THEN &E(1)] by blast
      qed
      AOT-hence ‹∃v ([G]v & [R₁]u'v & ∀t ([G]t & [R₁]u't → t =E v))›
        by (rule Ordinary.∃I)
      AOT-hence ‹∃!v ([G]v & [R₁]u'v)›
        by (rule equi:1[THEN ≡E(2)])
    }
    moreover {
      AOT-assume 0: ‹u' =E u›
      AOT-hence u'-eq-u: ‹u' = u›
        using =E−simple:2 →E by blast
      AOT-have ‹∃!v ([G]v & [R₁]u'v)›
      proof (safe intro!: equi:1[THEN ≡E(2)] Ordinary.∃I[where β=v]
                          &I Ordinary.GEN →I gv)
```

```
      AOT-show ‹[R1]u'v›
        apply (rule rule=E[rotated, OF R1-def[symmetric]])
        apply (safe intro!: β←C(1) cqt:2 &I prod-denotesI)
        by (safe intro!: ∨I(2) &I 0 ord=Eequiv:1[THEN →E, OF Ordinary.ψ])
    next
      fix v'
      AOT-assume ‹[G]v' & [R1]u'v'›
      AOT-hence 0: ‹[R1]uv'›
        using rule=E[rotated, OF u'-eq-u] &E(2) by fast
      AOT-have 1: ‹[«?R»]uv'›
        by (rule rule=E[rotated, OF R1-def]) (fact 0)
      AOT-have 2: ‹(u ≠E u & v' ≠E v & [R]uv') ∨
                  (u =E a & v' =E b) ∨
                  (u =E u & v' =E v)›
        using β→C(1)[OF 1] by simp
      AOT-have ‹¬u ≠E u›
        using ≡E(4) modus−tollens:1 ord=Eequiv:1 Ordinary.ψ
              reductio−aa:2 thm−neg=E by blast
      AOT-hence ‹¬((u ≠E u & v' ≠E v & [R]uv') ∨ (u =E a & v' =E b))›
        using not-u-eq-a
        by (metis ∨E(2) Conjunction Simplification(1)
                  modus−tollens:1 reductio−aa:1)
      AOT-hence ‹(u =E u & v' =E v)›
        using 2 by (metis ∨E(2))
      AOT-thus ‹v' =E v›
        using &E by blast
    qed
  }
  moreover {
    AOT-assume 0: ‹u' =E a›
    AOT-hence u'-eq-a: ‹u' = a›
      using =E−simple:2 →E by blast
    AOT-have ‹∃!v ([G]v & [R1]u'v)›
    proof (safe intro!: equi:1[THEN ≡E(2)] ∃I(2)[where β=b] &I
                        Ordinary.GEN →I b-prop[THEN &E(1)]
                        b-prop[THEN &E(2), THEN &E(1), THEN &E(1)])
      AOT-show ‹[R1]u'b›
        apply (rule rule=E[rotated, OF R1-def[symmetric]])
        apply (safe intro!: β←C(1) cqt:2 &I prod-denotesI)
        apply (rule ∨I(1); rule ∨I(2); rule &I)
         apply (fact 0)
        using b-prop &E(1) ord=Eequiv:1 →E by blast
    next
      fix v'
      AOT-assume gv'-R1u'v': ‹[G]v' & [R1]u'v'›
      AOT-hence 0: ‹[R1]av'›
        using u'-eq-a by (meson rule=E &E(2))
      AOT-have 1: ‹[«?R»]av'›
        by (rule rule=E[rotated, OF R1-def]) (fact 0)
      AOT-have ‹(a ≠E u & v' ≠E v & [R]av') ∨
               (a =E a & v' =E b) ∨
               (a =E u & v' =E v)›
        using β→C(1)[OF 1] by simp
      moreover {
        AOT-assume 0: ‹a ≠E u & v' ≠E v & [R]av'›
        AOT-have ‹∃!v ([G]v & [R]u'v)›
          using A[THEN Ordinary.∀E, THEN →E, OF fu'].
        AOT-hence ‹∃!v ([G]v & [R]av)›
          using u'-eq-a rule=E by fast
        AOT-hence ‹∃v ([G]v & [R]av & ∀t ([G]t & [R]at → t =E v))›
          using equi:1[THEN ≡E(1)] by fast
        then AOT-obtain s where
          s-prop: ‹[G]s & [R]as & ∀t ([G]t & [R]at → t =E s)›
```

```
        using Ordinary.∃E[rotated] by meson
      AOT-have ‹v′ =E s›
        using s-prop[THEN &E(2), THEN Ordinary.∀E]
              gv′-R1u′v′[THEN &E(1)] 0[THEN &E(2)]
        by (metis &I vdash−properties:10)
      moreover AOT-have ‹v =E s›
        using s-prop[THEN &E(2), THEN Ordinary.∀E] gv Rav
        by (metis &I →E)
      ultimately AOT-have ‹v′ =E v›
        by (metis &I ord=Eequiv:2 ord=Eequiv:3 →E)
      moreover AOT-have ‹¬(v′ =E v)›
        using 0[THEN &E(1), THEN &E(2)]
        by (metis ≡E(1) thm−neg=E)
      ultimately AOT-have ‹v′ =E b›
        by (metis raa−cor:3)
    }
    moreover {
      AOT-assume ‹a =E u & v′ =E v›
      AOT-hence ‹v′ =E b›
        by (metis &E(1) not-a-eq-u reductio−aa:1)
    }
    ultimately AOT-show ‹v′ =E b›
      by (metis &E(2) ∨E(3) reductio−aa:1)
  qed
 }
 ultimately AOT-show ‹∃!v ([G]v & [R1]u′v)›
   by (metis raa−cor:1)
next
 fix v′
 AOT-assume gv′: ‹[G]v′›
 {
   AOT-assume not-v′-eq-v: ‹¬(v′ =E v)›
        and not-v′-eq-b: ‹¬(v′ =E b)›
   AOT-hence v′-noteq-v: ‹v′ ≠E v›
        and v′-noteq-b: ‹v′ ≠E b›
    by (metis ≡E(2) thm−neg=E)+
   AOT-have ‹∃!u ([F]u & [R]uv′)›
     using B[THEN Ordinary.∀E, THEN →E, OF gv′].
   AOT-hence ‹∃u ([F]u & [R]uv′ & ∀t ([F]t & [R]tv′ → t =E u))›
     using equi:1[THEN ≡E(1)] by simp
   then AOT-obtain u′ where
     u′-prop: ‹[F]u′ & [R]u′v′ & ∀t ([F]t & [R]tv′ → t =E u′)›
     using Ordinary.∃E[rotated] by meson
   AOT-hence fu′: ‹[F]u′› and Ru′v′: ‹[R]u′v′›
     using &E by blast+
   AOT-have not-u′-eq-u: ‹¬u′ =E u›
   proof (rule raa−cor:2)
     AOT-assume ‹u′ =E u›
     AOT-hence ‹u′ = u›
       by (metis =E−simple:2 →E)
     AOT-hence Ruv′: ‹[R]uv′›
       using rule=E Ru′v′ by fast
     AOT-have ‹v′ =E b›
       using b-unique[OF Ordinary.ψ, OF gv′, OF Ruv′].
     AOT-thus ‹v′ =E b & ¬v′ =E b›
       using not-v′-eq-b &I by blast
   qed
   AOT-hence u′-noteq-u: ‹u′ ≠E u›
     using ≡E(2) thm−neg=E by blast
   AOT-have ‹∀t ([F]t & [R]tv′ → t =E u′)›
     using u′-prop &E by blast
   AOT-hence ‹[F]t & [R]tv′ → t =E u′› for t
     using Ordinary.∀E by meson
```

```
  AOT-hence u'-unique: ‹t =E u'› if ‹[F]t› and ‹[R]tv'› for t
    by (metis &I that →E)

  AOT-have ‹[F]u' & [R1]u'v' & ∀ t ([F]t & [R1]tv' → t =E u')›
  proof (safe intro!: &I gv' R1xy Ru'v' u'-noteq-u Ordinary.GEN →I
                      thm−neg=E[THEN ≡E(2)] not-v'-eq-v fu')
    fix t
    AOT-assume 1: ‹[F]t & [R1]tv'›
    AOT-have ‹[R]tv'›
      using Rxy2[OF 1[THEN &E(2)], OF v'-noteq-v, OF v'-noteq-b].
    AOT-thus ‹t =E u'›
      using u'-unique 1[THEN &E(1)] by blast
  qed
  AOT-hence ‹∃ u ([F]u & [R1]uv' & ∀ t ([F]t & [R1]tv' → t =E u))›
    by (rule Ordinary.∃ I)
  AOT-hence ‹∃!u ([F]u & [R1]uv')›
    by (rule equi:1[THEN ≡E(2)])
}
moreover {
  AOT-assume 0: ‹v' =E v›
  AOT-hence u'-eq-u: ‹v' = v›
    using =E−simple:2 →E by blast
  AOT-have ‹∃!u ([F]u & [R1]uv')›
  proof (safe intro!: equi:1[THEN ≡E(2)] Ordinary.∃ I[where β=u]
                      &I Ordinary.GEN →I fu)
    AOT-show ‹[R1]uv'›
      by (rule rule=E[rotated, OF R1-def[symmetric]])
         (safe intro!: β←C(1) cqt:2 &I prod-denotesI Ordinary.ψ
                       ∨I(2) 0 ord=Eequiv:1[THEN →E])
  next
    fix u'
    AOT-assume ‹[F]u' & [R1]u'v'›
    AOT-hence 0: ‹[R1]u'v›
      using rule=E[rotated, OF u'-eq-u] &E(2) by fast
    AOT-have 1: ‹[«?R»]u'v›
      by (rule rule=E[rotated, OF R1-def]) (fact 0)
    AOT-have 2: ‹(u' ≠E u & v ≠E v & [R]u'v) ∨
                 (u' =E a & v =E b) ∨
                 (u' =E u & v =E v)›
      using β→C(1)[OF 1, simplified] by simp
    AOT-have ‹¬v ≠E v›
      using ≡E(4) modus−tollens:1 ord=Eequiv:1 Ordinary.ψ
            reductio−aa:2 thm−neg=E by blast
    AOT-hence ‹¬((u' ≠E u & v ≠E v & [R]u'v) ∨ (u' =E a & v =E b))›
      by (metis &E(1) &E(2) ∨E(3) not-v-eq-b raa−cor:3)
    AOT-hence ‹(u' =E u & v =E v)›
      using 2 by (metis ∨E(2))
    AOT-thus ‹u' =E u›
      using &E by blast
  qed
}
moreover {
  AOT-assume 0: ‹v' =E b›
  AOT-hence v'-eq-b: ‹v' = b›
    using =E−simple:2 →E by blast
  AOT-have ‹∃!u ([F]u & [R1]uv')›
  proof (safe intro!: equi:1[THEN ≡E(2)] ∃ I(2)[where β=a] &I
                      Ordinary.GEN →I b-prop[THEN &E(1)] Oa fa
                      b-prop[THEN &E(2), THEN &E(1), THEN &E(1)])
    AOT-show ‹[R1]av'›
      apply (rule rule=E[rotated, OF R1-def[symmetric]])
      apply (safe intro!: β←C(1) cqt:2 &I prod-denotesI)
      apply (rule ∨I(1); rule ∨I(2); rule &I)
```

```
          using Oa ord=Eequiv:1 →E apply blast
          using 0 by blast
      next
        fix u'
        AOT-assume fu'-R1u'v': ‹[F]u' & [R₁]u'v'›
        AOT-hence 0: ‹[R₁]u'b›
          using v'-eq-b by (meson rule=E &E(2))
        AOT-have 1: ‹[«?R»]u'b›
          by (rule rule=E[rotated, OF R₁-def]) (fact 0)
        AOT-have ‹(u' ≠E u & b ≠E v & [R]u'b) ∨
                  (u' =E a & b =E b) ∨
                  (u' =E u & b =E v)›
          using β→C(1)[OF 1, simplified] by simp
        moreover {
          AOT-assume 0: ‹u' ≠E u & b ≠E v & [R]u'b›
          AOT-have ‹∃!u ([F]u & [R]uv')›
            using B[THEN Ordinary.∀E, THEN →E, OF gv'].
          AOT-hence ‹∃!u ([F]u & [R]ub)›
            using v'-eq-b rule=E by fast
          AOT-hence ‹∃u ([F]u & [R]ub & ∀t ([F]t & [R]tb → t =E u))›
            using equi:1[THEN ≡E(1)] by fast
          then AOT-obtain s where
            s-prop: ‹[F]s & [R]sb & ∀t ([F]t & [R]tb → t =E s)›
            using Ordinary.∃E[rotated] by meson
          AOT-have ‹u' =E s›
            using s-prop[THEN &E(2), THEN Ordinary.∀E]
                  fu'-R1u'v'[THEN &E(1)] 0[THEN &E(2)]
            by (metis &I →E)
          moreover AOT-have ‹u =E s›
            using s-prop[THEN &E(2), THEN Ordinary.∀E] fu Rub
            by (metis &I →E)
          ultimately AOT-have ‹u' =E u›
            by (metis &I ord=Eequiv:2 ord=Eequiv:3 →E)
          moreover AOT-have ‹¬(u' =E u)›
            using 0[THEN &E(1), THEN &E(1)] by (metis ≡E(1) thm−neg=E)
          ultimately AOT-have ‹u' =E a›
            by (metis raa−cor:3)
        }
        moreover {
          AOT-assume ‹u' =E u & b =E v›
          AOT-hence ‹u' =E a›
            by (metis &E(2) not-b-eq-v reductio−aa:1)
        }
        ultimately AOT-show ‹u' =E a›
          by (metis &E(1) ∨E(3) reductio−aa:1)
      qed
    }
    ultimately AOT-show ‹∃!u ([F]u & [R₁]uv')›
      by (metis raa−cor:1)
  qed
  ultimately AOT-have ‹∃R R |: [F]⁻ᵘ 1−1⟷E [G]⁻ᵛ›
    using 1 by blast
 }
 ultimately AOT-have ‹∃R R |: [F]⁻ᵘ 1−1⟷E [G]⁻ᵛ›
  using R-prop by (metis reductio−aa:2)
 AOT-thus ‹[F]⁻ᵘ ≈E [G]⁻ᵛ›
  by (rule equi:3[THEN ≡df I])
qed


AOT-theorem P'−eq: ‹[F]⁻ᵘ ≈E [G]⁻ᵛ & [F]u & [G]v → F ≈E G›
proof(safe intro!: →I; frule &E(1); drule &E(2);
   frule &E(1); drule &E(2))
```

AOT-have ‹[λz [Π]z & z ≠$_E$ κ]↓› **for** Π κ **by** *cqt:2[lambda]*
note Π-*minus-κI* = *rule−id−df:2:b[2][*
  where τ=‹(λ(Π, κ). «[Π]$^{-κ}$»)›, *simplified, OF F−u, simplified, OF this]*
 and Π-*minus-κE* = *rule−id−df:2:a[2][*
 where τ=‹(λ(Π, κ). «[Π]$^{-κ}$»)›, *simplified, OF F−u, simplified, OF this]*
AOT-have Π-*minus-κ-den*: ‹[Π]$^{-κ}$↓› **for** Π κ
  by (*rule* Π-*minus-κI*) *cqt:2[lambda]+*

AOT-have Π-*minus-κE1*: ‹[Π]κ′›
   and Π-*minus-κE2*: ‹κ′ ≠$_E$ κ› **if** ‹[[Π]$^{-κ}$]κ′› **for** Π κ κ′
proof −
  AOT-have ‹[λz [Π]z & z ≠$_E$ κ]κ′›
   using Π-*minus-κE that* **by** *fast*
  AOT-hence ‹[Π]κ′ & κ′ ≠$_E$ κ›
   by (*rule β→C(1)*)
  AOT-thus ‹[Π]κ′› **and** ‹κ′ ≠$_E$ κ›
   using &*E* **by** *blast+*
qed
AOT-have Π-*minus-κI′*: ‹[[Π]$^{-κ}$]κ′› **if** ‹[Π]κ′› **and** ‹κ′ ≠$_E$ κ› **for** Π κ κ′
proof −
  AOT-have κ′-*den*: ‹κ′↓›
   by (*metis russell−axiom[exe,1].ψ-denotes-asm that(1)*)
  AOT-have ‹[λz [Π]z & z ≠$_E$ κ]κ′›
   by (*safe intro*!: *β←C(1) cqt:2 κ′-den &I that*)
  AOT-thus ‹[[Π]$^{-κ}$]κ′›
   using Π-*minus-κI* **by** *fast*
qed

AOT-assume *Gv*: ‹[G]v›
AOT-assume *Fu*: ‹[F]u›
AOT-assume ‹[F]$^{-u}$ ≈$_E$ [G]$^{-v}$›
AOT-hence ‹∃ R R |: [F]$^{-u}$ $_{1-1}$⟷$_E$ [G]$^{-v}$›
  using *equi:3[THEN* ≡$_{df}$*E]* **by** *blast*
then AOT-obtain *R* **where** *R-prop*: ‹R |: [F]$^{-u}$ $_{1-1}$⟷$_E$ [G]$^{-v}$›
  using ∃ *E[rotated]* **by** *blast*
AOT-hence *Fact1*: ‹∀ r([[F]$^{-u}$]r → ∃!s ([[G]$^{-v}$]s & [R]rs))›
    and *Fact1′*: ‹∀ s([[G]$^{-v}$]s → ∃!r ([[F]$^{-u}$]r & [R]rs))›
  using *equi:2[THEN* ≡$_{df}$*E]* &*E* **by** *blast+*
AOT-have ‹R |: [F]$^{-u}$ $_{1-1}$⟶$_{onto}$E [G]$^{-v}$›
  using *equi−rem−thm[unvarify F G, OF* Π-*minus-κ-den, OF* Π-*minus-κ-den,*
        THEN ≡*E(1), OF R-prop]*.
AOT-hence ‹R |: [F]$^{-u}$ $_{1-1}$⟶E [G]$^{-v}$ & R |: [F]$^{-u}$ ⟶$_{onto}$E [G]$^{-v}$›
  using *equi−rem:4[THEN* ≡$_{df}$*E]* **by** *blast*
AOT-hence *Fact2*:
  ‹∀ r∀ s∀ t(([[F]$^{-u}$]r & [[F]$^{-u}$]s & [[G]$^{-v}$]t) → ([R]rt & [R]st → r =$_E$ s))›
  using *equi−rem:2[THEN* ≡$_{df}$*E]* &*E* **by** *blast*

let *?R* = ‹«[λxy ([[F]$^{-u}$]x & [[G]$^{-v}$]y & [R]xy) ∨ (x =$_E$ u & y =$_E$ v)]»›
AOT-have *R-den*: ‹«?R»↓› **by** *cqt:2[lambda]*

AOT-show ‹F ≈$_E$ G›
proof(*safe intro*!: *equi:3[THEN* ≡$_{df}$*I]* ∃ *I(1)*[**where** τ=*?R*] *R-den*
        equi:2[THEN ≡$_{df}$*I]* &*I cqt:2 Ordinary.GEN* →*I*)
  fix *r*
  AOT-assume *Fr*: ‹[F]r›
  {
   AOT-assume *not-r-eq-u*: ‹¬(r =$_E$ u)›
   AOT-hence *r-noteq-u*: ‹r ≠$_E$ u›
    using ≡*E(2) thm−neg=E* **by** *blast*
   AOT-have ‹[[F]$^{-u}$]r›
    by(*rule* Π-*minus-κI*; *safe intro*!: *β←C(1) cqt:2* &*I Fr r-noteq-u*)
   AOT-hence ‹∃!s ([[G]$^{-v}$]s & [R]rs)›
    using *Fact1[THEN* ∀ *E(2)]* →*E Ordinary.ψ* **by** *blast*

```
    AOT-hence ‹∃ s ([[G]⁻ᵛ]s & [R]rs & ∀ t ([[G]⁻ᵛ]t & [R]rt → t =E s))›
      using equi:1[THEN ≡E(1)] by simp
    then AOT-obtain s where s-prop: ‹[[G]⁻ᵛ]s & [R]rs & ∀ t ([[G]⁻ᵛ]t & [R]rt → t =E s)›
      using Ordinary.∃ E[rotated] by meson
    AOT-hence G-minus-v-s: ‹[[G]⁻ᵛ]s› and Rrs: ‹[R]rs›
      using &E by blast+
    AOT-have s-unique: ‹t =E s› if ‹[[G]⁻ᵛ]t› and ‹[R]rt› for t
      using s-prop[THEN &E(2), THEN Ordinary.∀ E, THEN →E, OF &I, OF that].
    AOT-have Gs: ‹[G]s›
      using Π-minus-κE1[OF G-minus-v-s].
    AOT-have s-noteq-v: ‹s ≠E v›
      using Π-minus-κE2[OF G-minus-v-s].
    AOT-have ‹∃ s ([G]s & [«?R»]rs & (∀ t ([G]t & [«?R»]rt → t =E s)))›
    proof(safe intro!: Ordinary.∃ I[where β=s] &I Gs Ordinary.GEN →I)
      AOT-show ‹[«?R»]rs›
        by (auto intro!: β←C(1) cqt:2 &I ∨I(1) Π-minus-κI' Fr Gs
                         s-noteq-v Rrs r-noteq-u
                 simp: &I ex:1:a prod-denotesI rule-ui:3)
    next
      fix t
      AOT-assume 0: ‹[G]t & [«?R»]rt›
      AOT-hence ‹([[F]⁻ᵘ]r & [[G]⁻ᵛ]t & [R]rt) ∨ (r =E u & t =E v)›
        using β→C(1)[OF 0[THEN &E(2)], simplified] by blast
      AOT-hence 1: ‹[[F]⁻ᵘ]r & [[G]⁻ᵛ]t & [R]rt›
        using not-r-eq-u by (metis &E(1) ∨E(3) reductio-aa:1)
      AOT-show ‹t =E s› using s-unique 1 &E by blast
    qed
  }
  moreover {
    AOT-assume r-eq-u: ‹r =E u›
    AOT-have ‹∃ s ([G]s & [«?R»]rs & (∀ t ([G]t & [«?R»]rt → t =E s)))›
    proof(safe intro!: Ordinary.∃ I[where β=v] &I Gv Ordinary.GEN →I)
      AOT-show ‹[«?R»]rv›
        by (auto intro!: β←C(1) cqt:2 &I ∨I(2) Π-minus-κI' Fr r-eq-u
                         ord=Eequiv:1[THEN →E] Ordinary.ψ
                 simp: &I ex:1:a prod-denotesI rule-ui:3)
    next
      fix t
      AOT-assume 0: ‹[G]t & [«?R»]rt›
      AOT-hence ‹([[F]⁻ᵘ]r & [[G]⁻ᵛ]t & [R]rt) ∨ (r =E u & t =E v)›
        using β→C(1)[OF 0[THEN &E(2)], simplified] by blast
      AOT-hence ‹r =E u & t =E v›
        using r-eq-u Π-minus-κE2
        by (metis &E(1) ∨E(2) ≡E(1) reductio-aa:1 thm-neg=E)
      AOT-thus ‹t =E v› using &E by blast
    qed
  }
  ultimately AOT-show ‹∃!s ([G]s & [«?R»]rs)›
    using reductio-aa:2 equi:1[THEN ≡E(2)] by fast
next
  fix s
  AOT-assume Gs: ‹[G]s›

  {
    AOT-assume not-s-eq-v: ‹¬(s =E v)›
    AOT-hence s-noteq-v: ‹s ≠E v›
      using ≡E(2) thm-neg=E by blast
    AOT-have ‹[[G]⁻ᵛ]s›
      by (rule Π-minus-κI; auto intro!: β←C(1) cqt:2 &I Gs s-noteq-v)
    AOT-hence ‹∃!r ([[F]⁻ᵘ]r & [R]rs)›
      using Fact1'[THEN Ordinary.∀ E] →E by blast
    AOT-hence ‹∃ r ([[F]⁻ᵘ]r & [R]rs & ∀ t ([[F]⁻ᵘ]t & [R]ts → t =E r))›
      using equi:1[THEN ≡E(1)] by simp
```

```
      then AOT-obtain r where
        r-prop: ‹[[F]⁻ᵘ]r & [R]rs & ∀t ([[F]⁻ᵘ]t & [R]ts → t =E r)›
        using Ordinary.∃E[rotated] by meson
      AOT-hence F-minus-u-r: ‹[[F]⁻ᵘ]r› and Rrs: ‹[R]rs›
        using &E by blast+
      AOT-have r-unique: ‹t =E r› if ‹[[F]⁻ᵘ]t› and ‹[R]ts› for t
        using r-prop[THEN &E(2), THEN Ordinary.∀E,
                    THEN →E, OF &I, OF that].
      AOT-have Fr: ‹[F]r›
        using Π-minus-κE1[OF F-minus-u-r].
      AOT-have r-noteq-u: ‹r ≠E u›
        using Π-minus-κE2[OF F-minus-u-r].
      AOT-have ‹∃r ([F]r & [«?R»]rs & (∀t ([F]t & [«?R»]ts → t =E r)))›
      proof(safe intro!: Ordinary.∃I[where β=r] &I Fr Ordinary.GEN →I)
        AOT-show ‹[«?R»]rs›
          by (auto intro!: β←C(1) cqt:2 &I ∨I(1) Π-minus-κI' Fr
                          Gs s-noteq-v Rrs r-noteq-u
                 simp: &I ex:1:a prod-denotesI rule−ui:3)
      next
        fix t
        AOT-assume 0: ‹[F]t & [«?R»]ts›
        AOT-hence ‹([[F]⁻ᵘ]t & [[G]⁻ᵛ]s & [R]ts) ∨ (t =E u & s =E v)›
          using β→C(1)[OF 0[THEN &E(2)], simplified] by blast
        AOT-hence 1: ‹[[F]⁻ᵘ]t & [[G]⁻ᵛ]s & [R]ts›
          using not-s-eq-v by (metis &E(2) ∨E(3) reductio−aa:1)
        AOT-show ‹t =E r› using r-unique 1 &E by blast
      qed
    }
    moreover {
      AOT-assume s-eq-v: ‹s =E v›
      AOT-have ‹∃r ([F]r & [«?R»]rs & (∀t ([F]t & [«?R»]ts → t =E r)))›
      proof(safe intro!: Ordinary.∃I[where β=u] &I Fu Ordinary.GEN →I)
        AOT-show ‹[«?R»]us›
          by (auto intro!: β←C(1) cqt:2 &I prod-denotesI ∨I(2)
                          Π-minus-κI' Gs s-eq-v Ordinary.ψ
                          ord=Eequiv:1[THEN →E])
      next
        fix t
        AOT-assume 0: ‹[F]t & [«?R»]ts›
        AOT-hence 1: ‹([[F]⁻ᵘ]t & [[G]⁻ᵛ]s & [R]ts) ∨ (t =E u & s =E v)›
          using β→C(1)[OF 0[THEN &E(2)], simplified] by blast
        moreover AOT-have ‹¬([[F]⁻ᵘ]t & [[G]⁻ᵛ]s & [R]ts)›
        proof (rule raa−cor:2)
          AOT-assume ‹([[F]⁻ᵘ]t & [[G]⁻ᵛ]s & [R]ts)›
          AOT-hence ‹[[G]⁻ᵛ]s› using &E by blast
          AOT-thus ‹s =E v & ¬(s =E v)›
            by (metis Π-minus-κE2 ≡E(4) reductio−aa:1 s-eq-v thm−neg=E)
        qed
        ultimately AOT-have ‹t =E u & s =E v›
          by (metis ∨E(2))
        AOT-thus ‹t =E u› using &E by blast
      qed
    }
    ultimately AOT-show ‹∃!r ([F]r & [«?R»]rs)›
      using ≡E(2) equi:1 reductio−aa:2 by fast
  qed
qed


AOT-theorem approx−cont:1: ‹∃F∃G ◇(F ≈E G & ◇¬F ≈E G)›
proof −
  let ?P = ‹«[λx E!x & ¬𝓐E!x]»›
  AOT-have ‹◇q₀ & ◇¬q₀› by (metis q₀-prop)
```

```
AOT-hence 1: ‹◇∃x(E!x & ¬𝓐E!x) & ◇¬∃x(E!x & ¬𝓐E!x)›
  by (rule q₀-def[THEN =df E(2), rotated])
     (simp add: log−prop−prop:2)
AOT-have ϑ: ‹◇∃x [«?P»]x & ◇¬∃x [«?P»]x›
  apply (AOT-subst ‹[«?P»]x› ‹E!x & ¬𝓐E!x› for: x)
   apply (rule beta−C−meta[THEN →E]; cqt:2[lambda])
  by (fact 1)
show ?thesis
proof (rule ∃I(1))+
  AOT-have ‹◇[L]⁻ ≈E [«?P»] & ◇¬[L]⁻ ≈E [«?P»]›
  proof (rule &I; rule RM◇[THEN →E]; (rule →I)?)
    AOT-modally-strict {
      AOT-assume A: ‹¬∃x [«?P»]x›
      AOT-show ‹[L]⁻ ≈E [«?P»]›
      proof (safe intro!: empty−approx:1[unvarify F H, THEN →E]
                          rel−neg−T:3 &I)
        AOT-show ‹[«?P»]↓› by cqt:2[lambda]
      next
        AOT-show ‹¬∃u [L⁻]u›
        proof (rule raa−cor:2)
          AOT-assume ‹∃u [L⁻]u›
          then AOT-obtain u where ‹[L⁻]u›
            using Ordinary.∃E[rotated] by blast
          moreover AOT-have ‹¬[L⁻]u›
            using thm−noncont−e−e:2[THEN contingent−properties:2[THEN ≡df E],
                                    THEN &E(2)]
            by (metis qml:2[axiom-inst] rule−ui:3 →E)
          ultimately AOT-show ‹p & ¬p› for p
            by (metis  raa−cor:3)
        qed
      next
        AOT-show ‹¬∃v [«?P»]v›
        proof (rule raa−cor:2)
          AOT-assume ‹∃v [«?P»]v›
          then AOT-obtain u where ‹[«?P»]u›
            using Ordinary.∃E[rotated] by blast
          AOT-hence ‹[«?P»]u›
            using &E by blast
          AOT-hence ‹∃x [«?P»]x›
            by (rule ∃I)
          AOT-thus ‹∃x [«?P»]x & ¬∃x [«?P»]x›
            using A &I by blast
        qed
      qed
    }
  next
    AOT-show ‹◇¬∃x [«?P»]x›
      using ϑ &E by blast
  next
    AOT-modally-strict {
      AOT-assume A: ‹∃x [«?P»]x›
      AOT-have B: ‹¬[«?P»] ≈E [L]⁻›
      proof (safe intro!: empty−approx:2[unvarify F H, THEN →E]
                          rel−neg−T:3 &I)
        AOT-show ‹[«?P»]↓›
          by cqt:2[lambda]
      next
        AOT-obtain x where Px: ‹[«?P»]x›
          using A ∃E by blast
        AOT-hence ‹E!x & ¬𝓐E!x›
          by (rule β→C(1))
        AOT-hence 1: ‹◇E!x›
          by (metis T◇ &E(1) vdash−properties:10)
```

AOT-have ‹$[\lambda x\ \Diamond E!x]x$›
by (*auto intro*!: $\beta{\leftarrow}C(1)$ *cqt*:*2* *1*)
AOT-hence ‹$O!x$›
by (*rule AOT-ordinary*[*THEN* $=_{df}I(2)$, *rotated*]) *cqt*:*2*[*lambda*]
AOT-hence ‹$O!x$ & [«*?P*»]x›
using *Px* &*I* **by** *blast*
AOT-thus ‹$\exists\, u$ [«*?P*»]u›
by (*rule* $\exists\, I$)
next
AOT-show ‹$\neg\exists\, u\ [L^-]u$›
proof (*rule raa−cor*:*2*)
AOT-assume ‹$\exists\, u\ [L^-]u$›
then AOT-obtain u **where** ‹$[L^-]u$›
using *Ordinary*.$\exists\, E$[*rotated*] **by** *blast*
moreover AOT-have ‹$\neg[L^-]u$›
using *thm−noncont−e−e*:*2*[*THEN contingent−properties*:*2*[*THEN* $\equiv_{df}E$]]
by (*metis qml*:*2*[*axiom-inst*] *rule−ui*:*3* →*E* &*E*(*2*))
ultimately AOT-show ‹p & $\neg p$› **for** p
by (*metis raa−cor*:*3*)
qed
qed
AOT-show ‹$\neg[L]^- \approx_E$ [«*?P*»]›
proof (*rule raa−cor*:*2*)
AOT-assume ‹$[L]^- \approx_E$ [«*?P*»]›
AOT-hence ‹[«*?P*»] $\approx_E [L]^-$›
apply (*rule eq−part*:*2*[*unvarify F G*, *THEN* →*E*, *rotated 2*])
apply *cqt*:*2*[*lambda*]
by (*simp add*: *rel−neg−T*:*3*)
AOT-thus ‹[«*?P*»] $\approx_E [L]^-$ & $\neg$[«*?P*»] $\approx_E [L]^-$›
using *B* &*I* **by** *blast*
qed
}
next
AOT-show ‹$\Diamond\exists\, x$ [«*?P*»]x›
using ϑ &*E* **by** *blast*
qed
AOT-thus ‹$\Diamond([L]^- \approx_E$ [«*?P*»] & $\Diamond\neg[L]^- \approx_E$ [«*?P*»])›
using *S5Basic*:*11* ≡*E*(*2*) **by** *blast*
next
AOT-show ‹$[\lambda x\ [E!]x\ \&\ \neg\mathcal{A}[E!]x]\downarrow$›
by *cqt*:*2*
next
AOT-show ‹$[L]^-\downarrow$›
by (*simp add*: *rel−neg−T*:*3*)
qed
qed

AOT-theorem *approx−cont*:*2*:
‹$\exists\, F\exists\, G\ \Diamond([\lambda z\ \mathcal{A}[F]z] \approx_E G\ \&\ \Diamond\neg[\lambda z\ \mathcal{A}[F]z] \approx_E G)$›
proof −
let *?P* = ‹«$[\lambda x\ E!x\ \&\ \neg\mathcal{A}E!x]$»›
AOT-have ‹$\Diamond q_0\ \&\ \Diamond\neg q_0$› **by** (*metis* q_0*-prop*)
AOT-hence *1*: ‹$\Diamond\exists\, x(E!x\ \&\ \neg\mathcal{A}E!x)\ \&\ \Diamond\neg\exists\, x(E!x\ \&\ \neg\mathcal{A}E!x)$›
by (*rule* q_0*-def*[*THEN* $=_{df}E(2)$, *rotated*])
(*simp add*: *log−prop−prop*:*2*)
AOT-have ϑ: ‹$\Diamond\exists\, x$ [«*?P*»]x & $\Diamond\neg\exists\, x$ [«*?P*»]x›
apply (*AOT-subst* ‹[«*?P*»]x› ‹$E!x\ \&\ \neg\mathcal{A}E!x$› **for**: x)
apply (*rule beta−C−meta*[*THEN* →*E*]; *cqt*:*2*)
by (*fact 1*)
show *?thesis*
proof (*rule* $\exists\, I(1)$)+
AOT-have ‹$\Diamond[\lambda z\ \mathcal{A}[L^-]z] \approx_E$ [«*?P*»] & $\Diamond\neg[\lambda z\ \mathcal{A}[L^-]z] \approx_E$ [«*?P*»]›

```
proof (rule &I; rule RM◇[THEN →E]; (rule →I)?)
  AOT-modally-strict {
    AOT-assume A: ‹¬∃x [«?P»]x›
    AOT-show ‹[λz 𝓐[L⁻]z] ≈E [«?P»]›
    proof (safe intro!: empty−approx:1[unvarify F H, THEN →E]
                        rel−neg−T:3 &I)
      AOT-show ‹[«?P»]↓› by cqt:2
    next
      AOT-show ‹¬∃u [λz 𝓐[L⁻]z]u›
      proof (rule raa−cor:2)
        AOT-assume ‹∃u [λz 𝓐[L⁻]z]u›
        then AOT-obtain u where ‹[λz 𝓐[L⁻]z]u›
          using Ordinary.∃E[rotated] by blast
        AOT-hence ‹𝓐[L⁻]u›
          using β→C(1) &E by blast
        moreover AOT-have ‹□¬[L⁻]u›
          using thm−noncont−e−e:2[THEN contingent−properties:2[THEN ≡df E]]
          by (metis RN qml:2[axiom-inst] rule−ui:3 →E &E(2))
        ultimately AOT-show ‹p & ¬p› for p
          by (metis Act−Sub:3 KBasic2:1 ≡E(1) raa−cor:3 →E)
      qed
    next
      AOT-show ‹¬∃v [«?P»]v›
      proof (rule raa−cor:2)
        AOT-assume ‹∃v [«?P»]v›
        then AOT-obtain u where ‹[«?P»]u›
          using Ordinary.∃E[rotated] by blast
        AOT-hence ‹[«?P»]u›
          using &E by blast
        AOT-hence ‹∃x [«?P»]x›
          by (rule ∃I)
        AOT-thus ‹∃x [«?P»]x & ¬∃x [«?P»]x›
          using A &I by blast
      qed
    next
      AOT-show ‹[λz 𝓐[L⁻]z]↓› by cqt:2
    qed
  }
next
  AOT-show ‹◇¬∃x [«?P»]x› using ϑ &E by blast
next
  AOT-modally-strict {
    AOT-assume A: ‹∃x [«?P»]x›
    AOT-have B: ‹¬[«?P»] ≈E [λz 𝓐[L⁻]z]›
    proof (safe intro!: empty−approx:2[unvarify F H, THEN →E]
                        rel−neg−T:3 &I)
      AOT-show ‹[«?P»]↓› by cqt:2
    next
      AOT-obtain x where Px: ‹[«?P»]x›
        using A ∃E by blast
      AOT-hence ‹E!x & ¬𝓐E!x›
        by (rule β→C(1))
      AOT-hence ‹◇E!x›
        by (metis T◇ &E(1) →E)
      AOT-hence ‹[λx ◇E!x]x›
        by (auto intro!: β←C(1) cqt:2)
      AOT-hence ‹O!x›
        by (rule AOT-ordinary[THEN =df I(2), rotated]) cqt:2
      AOT-hence ‹O!x & [«?P»]x›
        using Px &I by blast
      AOT-thus ‹∃u [«?P»]u›
        by (rule ∃I)
    next
```

```
          AOT-show ‹¬∃ u [λz 𝓐[L⁻]z]u›
          proof (rule raa−cor:2)
            AOT-assume ‹∃ u [λz 𝓐[L⁻]z]u›
            then AOT-obtain u where ‹[λz 𝓐[L⁻]z]u›
              using Ordinary.∃ E[rotated] by blast
            AOT-hence ‹𝓐[L⁻]u›
              using β→C(1) &E by blast
            moreover AOT-have ‹□¬[L⁻]u›
              using thm−noncont−e−e:2[THEN contingent−properties:2[THEN ≡df E]]
              by (metis RN qml:2[axiom-inst] rule−ui:3 →E &E(2))
            ultimately AOT-show ‹p & ¬p› for p
              by (metis Act−Sub:3 KBasic2:1 ≡E(1) raa−cor:3 →E)
          qed
        next
          AOT-show ‹[λz 𝓐[L⁻]z]↓› by cqt:2
        qed
        AOT-show ‹¬[λz 𝓐[L⁻]z] ≈E [«?P»]›
        proof (rule raa−cor:2)
          AOT-assume ‹[λz 𝓐[L⁻]z] ≈E [«?P»]›
          AOT-hence ‹[«?P»] ≈E [λz 𝓐[L⁻]z]›
            by (rule eq−part:2[unvarify F G, THEN →E, rotated 2])
               cqt:2+
          AOT-thus ‹[«?P»] ≈E [λz 𝓐[L⁻]z] & ¬[«?P»] ≈E [λz 𝓐[L⁻]z]›
            using B &I by blast
        qed
      }
    next
      AOT-show ‹◇∃ x [«?P»]x›
        using ϑ &E by blast
    qed
    AOT-thus ‹◇([λz 𝓐[L⁻]z] ≈E [«?P»] & ◇¬[λz 𝓐[L⁻]z] ≈E [«?P»])›
      using S5Basic:11 ≡E(2) by blast
  next
    AOT-show ‹[λx [E!]x & ¬𝓐[E!]x]↓› by cqt:2
  next
    AOT-show ‹[L]⁻↓›
      by (simp add: rel−neg−T:3)
  qed
qed

notepad
begin
```

We already have defined being equivalent on the ordinary objects in the Extended Relation Comprehension theory.

```
  AOT-have ‹F ≡E G ≡df F↓ & G↓ & ∀ u ([F]u ≡ [G]u)› for F G
    using eqE by blast
end

AOT-theorem apE−eqE:1: ‹F ≡E G → F ≈E G›
proof(rule →I)
  AOT-assume 0: ‹F ≡E G›
  AOT-have ‹∃ R R |: F 1−1⟷E G›
  proof (safe intro!: ∃ I(1)[where τ=«(=E)»] equi:2[THEN ≡df I] &I
                  =E[denotes] cqt:2[const-var][axiom-inst] Ordinary.GEN
                  →I equi:1[THEN ≡E(2)])
    fix u
    AOT-assume Fu: ‹[F]u›
    AOT-hence Gu: ‹[G]u›
      using ≡df E[OF eqE, OF 0, THEN &E(2),
                THEN Ordinary.∀ E[where α=u], THEN ≡E(1)]
          Ordinary.ψ Fu by blast
    AOT-show ‹∃ v ([G]v & u =E v & ∀ v' ([G]v' & u =E v' → v' =E v))›
```

```
    by (safe intro!: Ordinary.∃ I[where β=u] &I GEN →I Ordinary.ψ Gu
                  ord=Eequiv:1[THEN →E, OF Ordinary.ψ]
                  ord=Eequiv:2[THEN →E] dest!: &E(2))
  next
   fix v
   AOT-assume Gv: ‹[G]v›
   AOT-hence Fv: ‹[F]v›
    using ≡df E[OF eqE, OF 0, THEN &E(2),
              THEN Ordinary.∀ E[where α=v], THEN ≡E(2)]
         Ordinary.ψ Gv by blast
   AOT-show ‹∃ u ([F]u & u =E v & ∀ v' ([F]v' & v' =E v → v' =E u))›
    by (safe intro!: Ordinary.∃ I[where β=v] &I GEN →I Ordinary.ψ Fv
                  ord=Eequiv:1[THEN →E, OF Ordinary.ψ]
                  ord=Eequiv:2[THEN →E] dest!: &E(2))
  qed
  AOT-thus ‹F ≈E G›
   by (rule equi:3[THEN ≡df I])
qed

AOT-theorem apE−eqE:2: ‹(F ≈E G & G ≡E H) → F ≈E H›
proof(rule →I)
  AOT-assume ‹F ≈E G & G ≡E H›
  AOT-hence ‹F ≈E G› and ‹G ≈E H›
   using apE−eqE:1[THEN →E] &E by blast+
  AOT-thus ‹F ≈E H›
   by (metis Adjunction eq−part:3 vdash−properties:10)
qed


AOT-act-theorem eq−part−act:1: ‹[λz 𝓐[F]z] ≡E F›
proof (safe intro!: eqE[THEN ≡df I] &I cqt:2 Ordinary.GEN →I)
  fix u
  AOT-have ‹[λz 𝓐[F]z]u ≡ 𝓐[F]u›
   by (rule beta−C−meta[THEN →E]) cqt:2[lambda]
  also AOT-have ‹... ≡ [F]u›
   using act−conj−act:4 logic−actual[act-axiom-inst, THEN →E] by blast
  finally AOT-show ‹[λz 𝓐[F]z]u ≡ [F]u›.
qed

AOT-act-theorem eq−part−act:2: ‹[λz 𝓐[F]z] ≈E F›
  by (safe intro!: apE−eqE:1[unvarify F, THEN →E] eq−part−act:1) cqt:2


AOT-theorem actuallyF:1: ‹𝓐(F ≈E [λz 𝓐[F]z])›
proof −
  AOT-have 1: ‹𝓐([F]x ≡ 𝓐[F]x)› for x
   by (meson Act−Basic:5 act−conj−act:4 ≡E(2) Commutativity of ≡)
  AOT-have ‹𝓐([F]x ≡ [λz 𝓐[F]z]x)› for x
   apply (AOT-subst ‹[λz 𝓐[F]z]x› ‹𝓐[F]x›)
    apply (rule beta−C−meta[THEN →E])
    apply cqt:2[lambda]
   by (fact 1)
  AOT-hence ‹O!x → 𝓐([F]x ≡ [λz 𝓐[F]z]x)› for x
   by (metis →I)
  AOT-hence ‹∀ u 𝓐([F]u ≡ [λz 𝓐[F]z]u)›
   using ∀ I by fast
  AOT-hence 1: ‹𝓐∀ u ([F]u ≡ [λz 𝓐[F]z]u)›
   by (metis Ordinary.res−var−bound−reas[2] →E)
  AOT-modally-strict {
   AOT-have ‹[λz 𝓐[F]z]↓› by cqt:2
  } note 2 = this
  AOT-have ‹𝓐(F ≡E [λz 𝓐[F]z])›
   apply (AOT-subst ‹F ≡E [λz 𝓐[F]z]› ‹∀ u ([F]u ≡ [λz 𝓐[F]z]u)›)
```

using *eqE*[*THEN* ≡*Df*, *THEN* ≡*S*(*1*), *OF* &*I*,
OF cqt:*2*[*const-var*][*axiom-inst*], *OF 2*]
by (*auto simp*: *1*)
moreover AOT-have ‹$\boldsymbol{\mathcal{A}}(F \equiv_E [\lambda z\ \boldsymbol{\mathcal{A}}[F]z] \to F \approx_E [\lambda z\ \boldsymbol{\mathcal{A}}[F]z])$›
using *apE−eqE*:*1*[*unvarify G*, *THEN RA*[*2*], *OF 2*] **by** *metis*
ultimately AOT-show ‹$\boldsymbol{\mathcal{A}}F \approx_E [\lambda z\ \boldsymbol{\mathcal{A}}[F]z]$›
by (*metis act−cond* →*E*)
qed

AOT-theorem *actuallyF*:*2*: ‹*Rigid*($[\lambda z\ \boldsymbol{\mathcal{A}}[F]z]$)›
proof(*safe intro*!: *GEN* →*I df−rigid−rel*:*1*[*THEN* $\equiv_{df}$*I*] &*I*)
AOT-show ‹$[\lambda z\ \boldsymbol{\mathcal{A}}[F]z]\downarrow$› **by** *cqt*:*2*
next
AOT-show ‹$\Box\forall x\ ([\lambda z\ \boldsymbol{\mathcal{A}}[F]z]x \to \Box[\lambda z\ \boldsymbol{\mathcal{A}}[F]z]x)$›
proof(*rule RN*; *rule GEN*; *rule* →*I*)
AOT-modally-strict {
fix *x*
AOT-assume ‹$[\lambda z\ \boldsymbol{\mathcal{A}}[F]z]x$›
AOT-hence ‹$\boldsymbol{\mathcal{A}}[F]x$›
by (*rule* β→*C*(*1*))
AOT-hence *1*: ‹$\Box\boldsymbol{\mathcal{A}}[F]x$› **by** (*metis Act−Basic*:*6* ≡*E*(*1*))
AOT-show ‹$\Box[\lambda z\ \boldsymbol{\mathcal{A}}[F]z]x$›
apply (*AOT-subst* ‹$[\lambda z\ \boldsymbol{\mathcal{A}}[F]z]x$› ‹$\boldsymbol{\mathcal{A}}[F]x$›)
apply (*rule beta−C−meta*[*THEN* →*E*])
apply *cqt*:*2*[*lambda*]
by (*fact 1*)
}
qed
qed

AOT-theorem *approx−nec*:*1*: ‹$Rigid(F) \to F \approx_E [\lambda z\ \boldsymbol{\mathcal{A}}[F]z]$›
proof(*rule* →*I*)
AOT-assume ‹*Rigid*([*F*])›
AOT-hence *A*: ‹$\Box\forall x\ ([F]x \to \Box[F]x)$›
using *df−rigid−rel*:*1*[*THEN* $\equiv_{df}$*E*, *THEN* &*E*(*2*)] **by** *blast*
AOT-hence *0*: ‹$\forall x\ \Box([F]x \to \Box[F]x)$›
using *CBF*[*THEN* →*E*] **by** *blast*
AOT-hence *1*: ‹$\forall x\ ([F]x \to \Box[F]x)$›
using *A qml*:*2*[*axiom-inst*, *THEN* →*E*] **by** *blast*
AOT-have *act-F-den*: ‹$[\lambda z\ \boldsymbol{\mathcal{A}}[F]z]\downarrow$›
by *cqt*:*2*
AOT-show ‹$F \approx_E [\lambda z\ \boldsymbol{\mathcal{A}}[F]z]$›
proof (*safe intro*!: *apE−eqE*:*1*[*unvarify G*, *THEN* →*E*] *eqE*[*THEN* $\equiv_{df}$*I*] &*I*
cqt:*2 act-F-den Ordinary*.*GEN* →*I* ≡*I*)
fix *u*
AOT-assume ‹[*F*]*u*›
AOT-hence ‹$\Box[F]u$›
using *1*[*THEN* ∀*E*(*2*), *THEN* →*E*] **by** *blast*
AOT-hence *act-F-u*: ‹$\boldsymbol{\mathcal{A}}[F]u$›
by (*metis nec−imp−act* →*E*)
AOT-show ‹$[\lambda z\ \boldsymbol{\mathcal{A}}[F]z]u$›
by (*auto intro*!: β←*C*(*1*) *cqt*:*2 act-F-u*)
next
fix *u*
AOT-assume ‹$[\lambda z\ \boldsymbol{\mathcal{A}}[F]z]u$›
AOT-hence ‹$\boldsymbol{\mathcal{A}}[F]u$›
by (*rule* β→*C*(*1*))
AOT-thus ‹[*F*]*u*›
using *0*[*THEN* ∀*E*(*2*)]
by (*metis* ≡*E*(*1*) *sc−eq−fur*:*2* →*E*)
qed
qed

AOT-theorem *approx−nec*:*2*:
‹$F \approx_E G \equiv \forall H$ ([λz $\boldsymbol{\mathcal{A}}$[H]z] $\approx_E F \equiv$ [λz $\boldsymbol{\mathcal{A}}$[H]z] $\approx_E G$)›
proof(*rule* ≡*I*; *rule* →*I*)
  AOT-assume *0*: ‹$F \approx_E G$›
  AOT-assume *0*: ‹$F \approx_E G$›
  AOT-hence ‹∀ H (H $\approx_E$ F ≡ H $\approx_E$ G)›
    using *eq−part*:*4*[*THEN* ≡*E*(*1*), *OF 0*] **by** *blast*
  AOT-have ‹[λz $\boldsymbol{\mathcal{A}}$[H]z] $\approx_E$ F ≡ [λz $\boldsymbol{\mathcal{A}}$[H]z] $\approx_E$ G› **for** *H*
    by (*rule* ∀ *E*(*1*)[*OF eq−part*:*4*[*THEN* ≡*E*(*1*), *OF 0*]]) *cqt*:*2*
  AOT-thus ‹∀ H ([λz $\boldsymbol{\mathcal{A}}$[H]z] $\approx_E$ F ≡ [λz $\boldsymbol{\mathcal{A}}$[H]z] $\approx_E$ G)›
    by (*rule GEN*)
next
  AOT-assume *0*: ‹∀ H ([λz $\boldsymbol{\mathcal{A}}$[H]z] $\approx_E$ F ≡ [λz $\boldsymbol{\mathcal{A}}$[H]z] $\approx_E$ G)›
  AOT-obtain *H* **where** ‹*Rigidifies*(*H*,*F*)›
    using *rigid−der*:*3* ∃ *E* **by** *metis*
  AOT-hence *H*: ‹*Rigid*(*H*) & ∀ x ([H]x ≡ [F]x)›
    using *df−rigid−rel*:*2*[*THEN* $\equiv_{df}$*E*] **by** *blast*
  AOT-have *H-rigid*: ‹□∀ x ([H]x → □[H]x)›
    using *H*[*THEN* &*E*(*1*), *THEN df−rigid−rel*:*1*[*THEN* $\equiv_{df}$*E*], *THEN* &*E*(*2*)].
  AOT-hence ‹∀ x □([H]x → □[H]x)›
    using *CBF vdash−properties*:*10* **by** *blast*
  AOT-hence ‹□([H]x → □[H]x)› **for** *x* **using** ∀ *E*(*2*) **by** *blast*
  AOT-hence *rigid*: ‹[H]x ≡ $\boldsymbol{\mathcal{A}}$[H]x› **for** *x*
     by (*metis* ≡*E*(*6*) *oth−class−taut*:*3*:*a sc−eq−fur*:*2* →*E*)
  AOT-have ‹$H \equiv_E F$›
  proof (*safe intro*!: *eqE*[*THEN* $\equiv_{df}$*I*] &*I cqt*:*2 Ordinary.GEN* →*I*)
    AOT-show ‹[H]u ≡ [F]u› **for** *u* **using** *H*[*THEN* &*E*(*2*)] ∀ *E*(*2*) **by** *fast*
  qed
  AOT-hence ‹$H \approx_E F$›
    by (*rule apE−eqE*:*2*[*THEN* →*E*, *OF* &*I*, *rotated*])
       (*simp add*: *eq−part*:*1*)
  AOT-hence *F-approx-H*: ‹$F \approx_E H$›
    by (*metis eq−part*:*2* →*E*)
  moreover AOT-have *H-eq-act-H*: ‹$H \equiv_E$ [λz $\boldsymbol{\mathcal{A}}$[H]z]›
  proof (*safe intro*!: *eqE*[*THEN* $\equiv_{df}$*I*] &*I cqt*:*2 Ordinary.GEN* →*I*)
    AOT-show ‹[H]u ≡ [λz $\boldsymbol{\mathcal{A}}$[H]z]u› **for** *u*
      apply (*AOT-subst* ‹[λz $\boldsymbol{\mathcal{A}}$[H]z]u› ‹$\boldsymbol{\mathcal{A}}$[H]u›)
       apply (*rule beta−C−meta*[*THEN* →*E*])
       apply *cqt*:*2*[*lambda*]
      using *rigid* **by** *blast*
  qed
  AOT-have *a*: ‹$F \approx_E$ [λz $\boldsymbol{\mathcal{A}}$[H]z]›
    apply (*rule apE−eqE*:*2*[*unvarify H*, *THEN* →*E*])
     apply *cqt*:*2*[*lambda*]
    using *F-approx-H H-eq-act-H* &*I* **by** *blast*
  AOT-hence ‹[λz $\boldsymbol{\mathcal{A}}$[H]z] $\approx_E$ F›
    apply (*rule eq−part*:*2*[*unvarify G*, *THEN* →*E*, *rotated*])
    by *cqt*:*2*[*lambda*]
  AOT-hence *b*: ‹[λz $\boldsymbol{\mathcal{A}}$[H]z] $\approx_E$ G›
    by (*rule 0*[*THEN* ∀ *E*(*1*), *THEN* ≡*E*(*1*), *rotated*]) *cqt*:*2*
  AOT-show ‹$F \approx_E G$›
    by (*rule eq−part*:*3*[*unvarify G*, *THEN* →*E*, *rotated*, *OF* &*I*, *OF a*, *OF b*])
       cqt:*2*
qed

AOT-theorem *approx−nec*:*3*:
  ‹(*Rigid*(*F*) & *Rigid*(*G*)) → □($F \approx_E G$ → □$F \approx_E G$)›
proof (*rule* →*I*)
  AOT-assume ‹*Rigid*(*F*) & *Rigid*(*G*)›
  AOT-hence ‹□∀ x([F]x → □[F]x)› **and** ‹□∀ x([G]x → □[G]x)›
    using *df−rigid−rel*:*1*[*THEN* $\equiv_{df}$*E*, *THEN* &*E*(*2*)] &*E* **by** *blast*+
  AOT-hence ‹□(□∀ x([F]x → □[F]x) & □∀ x([G]x → □[G]x))›

using *KBasic:3 4 &I ≡E(2) vdash−properties:10* **by** *meson*
moreover AOT-have $\langle\Box(\Box\forall x([F]x \to \Box[F]x)\ \&\ \Box\forall x([G]x \to \Box[G]x)) \to$
$\Box(F \approx_E G \to \Box F \approx_E G)\rangle$
proof(*rule RM*; *rule* $\to I$; *rule* $\to I$)
AOT-modally-strict {
AOT-assume $\langle\Box\forall x([F]x \to \Box[F]x)\ \&\ \Box\forall x([G]x \to \Box[G]x)\rangle$
AOT-hence $\langle\Box\forall x([F]x \to \Box[F]x)\rangle$ **and** $\langle\Box\forall x([G]x \to \Box[G]x)\rangle$
using *&E* **by** *blast+*
AOT-hence $\langle\forall x\Box([F]x \to \Box[F]x)\rangle$ **and** $\langle\forall x\Box([G]x \to \Box[G]x)\rangle$
using *CBF[THEN* $\to E$*]* **by** *blast+*
AOT-hence *F-nec*: $\langle\Box([F]x \to \Box[F]x)\rangle$
and *G-nec*: $\langle\Box([G]x \to \Box[G]x)\rangle$ **for** x
using $\forall E(2)$ **by** *blast+*
AOT-assume $\langle F \approx_E G\rangle$
AOT-hence $\langle\exists R\ R\ |{:}\ F\ {}_{1-1}\!\!\longleftrightarrow_E G\rangle$
by (*metis* $\equiv_{df}E$ *equi:3*)
then AOT-obtain R **where** $\langle R\ |{:}\ F\ {}_{1-1}\!\!\longleftrightarrow_E G\rangle$
using $\exists E$*[rotated]* **by** *blast*
AOT-hence *C1*: $\langle\forall u\ ([F]u \to \exists!v\ ([G]v\ \&\ [R]uv))\rangle$
and *C2*: $\langle\forall v\ ([G]v \to \exists!u\ ([F]u\ \&\ [R]uv))\rangle$
using *equi:2[THEN* $\equiv_{df}E$*] &E* **by** *blast+*
AOT-obtain R' **where** $\langle Rigidifies(R', R)\rangle$
using *rigid−der:3* $\exists E$*[rotated]* **by** *blast*
AOT-hence *1*: $\langle Rigid(R')\ \&\ \forall x_1...\forall x_n\ ([R']x_1...x_n \equiv [R]x_1...x_n)\rangle$
using *df−rigid−rel:2[THEN* $\equiv_{df}E$*]* **by** *blast*
AOT-hence $\langle\Box\forall x_1...\forall x_n\ ([R']x_1...x_n \to \Box[R']x_1...x_n)\rangle$
using *df−rigid−rel:1[THEN* $\equiv_{df}E$*] &E* **by** *blast*
AOT-hence $\langle\forall x_1...\forall x_n\ (\Diamond[R']x_1...x_n \to \Box[R']x_1...x_n)\rangle$
using $\equiv E(1)$ *rigid−rel−thms:1* **by** *blast*
AOT-hence *D*: $\langle\forall x_1\forall x_2\ (\Diamond[R']x_1x_2 \to \Box[R']x_1x_2)\rangle$
using *tuple-forall[THEN* $\equiv_{df}E$*]* **by** *blast*
AOT-have *E*: $\langle\forall x_1\forall x_2\ ([R']x_1x_2 \equiv [R]x_1x_2)\rangle$
using *tuple-forall[THEN* $\equiv_{df}E$*, OF 1[THEN &E(2)]]* **by** *blast*
AOT-have $\langle\forall u\ \Box([F]u \to \exists!v\ ([G]v\ \&\ [R']uv))\rangle$
and $\langle\forall v\ \Box([G]v \to \exists!u\ ([F]u\ \&\ [R']uv))\rangle$
proof (*safe intro*!: *Ordinary.GEN* $\to I$)
fix u
AOT-show $\langle\Box([F]u \to \exists!v\ ([G]v\ \&\ [R']uv))\rangle$
proof (*rule raa−cor:1*)
AOT-assume $\langle\neg\Box([F]u \to \exists!v\ ([G]v\ \&\ [R']uv))\rangle$
AOT-hence *1*: $\langle\Diamond\neg([F]u \to \exists!v\ ([G]v\ \&\ [R']uv))\rangle$
using *KBasic:11* $\equiv E(1)$ **by** *blast*
AOT-have $\langle\Diamond([F]u\ \&\ \neg\exists!v\ ([G]v\ \&\ [R']uv))\rangle$
apply (*AOT-subst* $\langle[F]u\ \&\ \neg\exists!v\ ([G]v\ \&\ [R']uv)\rangle$
$\langle\neg([F]u \to \exists!v\ ([G]v\ \&\ [R']uv))\rangle$)
apply (*meson* $\equiv E(6)$ *oth−class−taut:1:b oth−class−taut:3:a*)
by (*fact 1*)
AOT-hence *A*: $\langle\Diamond[F]u\ \&\ \Diamond\neg\exists!v\ ([G]v\ \&\ [R']uv)\rangle$
using *KBasic2:3* $\to E$ **by** *blast*
AOT-hence $\langle\Box[F]u\rangle$
using *F-nec &E(1)* $\equiv E(1)$ *sc−eq−box−box:1* $\to E$ **by** *blast*
AOT-hence $\langle[F]u\rangle$
by (*metis qml:2[axiom-inst]* $\to E$)
AOT-hence $\langle\exists!v\ ([G]v\ \&\ [R]uv)\rangle$
using *C1[THEN Ordinary.*$\forall E$*, THEN* $\to E$*]* **by** *blast*
AOT-hence $\langle\exists v\ ([G]v\ \&\ [R]uv\ \&\ \forall v'\ ([G]v'\ \&\ [R]uv' \to v' =_E v))\rangle$
using *equi:1[THEN* $\equiv E(1)$*]* **by** *auto*
then AOT-obtain a **where**
a-prop: $\langle O!a\ \&\ ([G]a\ \&\ [R]ua\ \&\ \forall v'\ ([G]v'\ \&\ [R]uv' \to v' =_E a))\rangle$
using $\exists E$*[rotated]* **by** *blast*
AOT-have $\langle\exists v\ \Box([G]v\ \&\ [R']uv\ \&\ \forall v'\ ([G]v'\ \&\ [R']uv' \to v' =_E v))\rangle$
proof(*safe intro*!: $\exists I(2)$[**where** β=a] *&I a-prop[THEN &E(1)]*
KBasic:3[THEN $\equiv E(2)$*]*)

```
      AOT-show ‹□[G]a›
        using a-prop[THEN &E(2), THEN &E(1), THEN &E(1)]
        by (metis G-nec qml:2[axiom-inst] →E)
    next
      AOT-show ‹□[R']ua›
        using D[THEN ∀E(2), THEN ∀E(2), THEN →E]
              E[THEN ∀E(2), THEN ∀E(2), THEN ≡E(2),
                OF a-prop[THEN &E(2), THEN &E(1), THEN &E(2)]]
        by (metis T◇ →E)
    next
      AOT-have ‹∀v' □([G]v' & [R']uv' → v' =E a)›
      proof (rule Ordinary.GEN; rule raa−cor:1)
        fix v'
        AOT-assume ‹¬□([G]v' & [R']uv' → v' =E a)›
        AOT-hence ‹◇¬([G]v' & [R']uv' → v' =E a)›
          by (metis KBasic:11 ≡E(1))
        AOT-hence ‹◇([G]v' & [R']uv' & ¬v' =E a)›
          by (AOT-subst ‹[G]v' & [R']uv' & ¬v' =E a›
                      ‹¬([G]v' & [R']uv' → v' =E a)›)
             (meson ≡E(6) oth−class−taut:1:b oth−class−taut:3:a)
        AOT-hence 1: ‹◇[G]v'› and 2: ‹◇[R']uv'› and 3: ‹◇¬v' =E a›
          using KBasic2:3[THEN →E, THEN &E(1)]
                KBasic2:3[THEN →E, THEN &E(2)] by blast+
        AOT-have Gv': ‹[G]v'› using G-nec 1
          by (meson B◇ KBasic:13 →E)
        AOT-have ‹□[R']uv'›
          using 2 D[THEN ∀E(2), THEN ∀E(2), THEN →E] by blast
        AOT-hence R'uv': ‹[R']uv'›
          by (metis B◇ T◇ →E)
        AOT-hence ‹[R]uv'›
          using E[THEN ∀E(2), THEN ∀E(2), THEN ≡E(1)] by blast
        AOT-hence ‹v' =E a›
          using a-prop[THEN &E(2), THEN &E(2), THEN Ordinary.∀E,
                      THEN →E, OF &I, OF Gv'] by blast
        AOT-hence ‹□(v' =E a)›
          by (metis id−nec3:1 ≡E(4) raa−cor:3)
        moreover AOT-have ‹¬□(v' =E a)›
          using 3 KBasic:11 ≡E(2) by blast
        ultimately AOT-show ‹□(v' =E a) & ¬□(v' =E a)›
          using &I by blast
      qed
      AOT-thus ‹□∀v'([G]v' & [R']uv' → v' =E a)›
        using Ordinary.res−var−bound−reas[BF] →E by fast
    qed
    AOT-hence ‹□∃v ([G]v & [R']uv & ∀v' ([G]v' & [R']uv' → v' =E v))›
      using Ordinary.res−var−bound−reas[Buridan] →E by fast
    AOT-hence ‹□∃!v ([G]v & [R']uv)›
      by (AOT-subst-thm equi:1)
    moreover AOT-have ‹¬□∃!v ([G]v & [R']uv)›
      using A[THEN &E(2)] KBasic:11[THEN ≡E(2)] by blast
    ultimately AOT-show ‹□∃!v ([G]v & [R']uv) & ¬□∃!v ([G]v & [R']uv)›
      by (rule &I)
  qed
next
  fix v
  AOT-show ‹□([G]v → ∃!u ([F]u & [R']uv))›
  proof (rule raa−cor:1)
    AOT-assume ‹¬□([G]v → ∃!u ([F]u & [R']uv))›
    AOT-hence 1: ‹◇¬([G]v → ∃!u ([F]u & [R']uv))›
      using KBasic:11 ≡E(1) by blast
    AOT-hence ‹◇([G]v & ¬∃!u ([F]u & [R']uv))›
      by (AOT-subst ‹[G]v & ¬∃!u ([F]u & [R']uv)›
                  ‹¬([G]v → ∃!u ([F]u & [R']uv))›)
```

```
    (meson ≡E(6) oth−class−taut:1:b oth−class−taut:3:a)
AOT-hence A: ‹◇[G]v & ◇¬∃!u ([F]u & [R']uv)›
  using KBasic2:3 →E by blast
AOT-hence ‹□[G]v›
  using G-nec &E(1) ≡E(1) sc−eq−box−box:1 →E by blast
AOT-hence ‹[G]v› by (metis qml:2[axiom-inst] →E)
AOT-hence ‹∃!u ([F]u & [R]uv)›
  using C2[THEN Ordinary.∀E, THEN →E] by blast
AOT-hence ‹∃u ([F]u & [R]uv & ∀u' ([F]u' & [R]u'v → u' =E u))›
  using equi:1[THEN ≡E(1)] by auto
then AOT-obtain a where
    a-prop: ‹O!a & ([F]a & [R]av & ∀u' ([F]u' & [R]u'v → u' =E a))›
  using ∃E[rotated] by blast
AOT-have ‹∃u □([F]u & [R']uv & ∀u' ([F]u' & [R']u'v → u' =E u))›
proof(safe intro!: ∃I(2)[where β=a] &I a-prop[THEN &E(1)]
                  KBasic:3[THEN ≡E(2)])
  AOT-show ‹□[F]a›
    using a-prop[THEN &E(2), THEN &E(1), THEN &E(1)]
    by (metis F-nec qml:2[axiom-inst] →E)
next
  AOT-show ‹□[R']av›
    using D[THEN ∀E(2), THEN ∀E(2), THEN →E]
          E[THEN ∀E(2), THEN ∀E(2), THEN ≡E(2),
            OF a-prop[THEN &E(2), THEN &E(1), THEN &E(2)]]
    by (metis T◇ →E)
next
  AOT-have ‹∀u' □([F]u' & [R']u'v → u' =E a)›
  proof (rule Ordinary.GEN; rule raa−cor:1)
    fix u'
    AOT-assume ‹¬□([F]u' & [R']u'v → u' =E a)›
    AOT-hence ‹◇¬([F]u' & [R']u'v → u' =E a)›
      by (metis KBasic:11 ≡E(1))
    AOT-hence ‹◇([F]u' & [R']u'v & ¬u' =E a)›
      by (AOT-subst ‹[F]u' & [R']u'v & ¬u' =E a›
                   ‹¬([F]u' & [R']u'v → u' =E a)›)
         (meson ≡E(6) oth−class−taut:1:b oth−class−taut:3:a)
    AOT-hence 1: ‹◇[F]u'› and 2: ‹◇[R']u'v› and 3: ‹◇¬u' =E a›
      using KBasic2:3[THEN →E, THEN &E(1)]
            KBasic2:3[THEN →E, THEN &E(2)] by blast+
    AOT-have Fu': ‹[F]u'› using F-nec 1
      by (meson B◇ KBasic:13 →E)
    AOT-have ‹□[R']u'v›
      using 2 D[THEN ∀E(2), THEN ∀E(2), THEN →E] by blast
    AOT-hence R'u'v: ‹[R']u'v›
      by (metis B◇ T◇ →E)
    AOT-hence ‹[R]u'v›
      using E[THEN ∀E(2), THEN ∀E(2), THEN ≡E(1)] by blast
    AOT-hence ‹u' =E a›
      using a-prop[THEN &E(2), THEN &E(2), THEN Ordinary.∀E,
                   THEN →E, OF &I, OF Fu'] by blast
    AOT-hence ‹□(u' =E a)›
      by (metis id−nec3:1 ≡E(4) raa−cor:3)
    moreover AOT-have ‹¬□(u' =E a)›
      using 3 KBasic:11 ≡E(2) by blast
    ultimately AOT-show ‹□(u' =E a) & ¬□(u' =E a)›
      using &I by blast
  qed
  AOT-thus ‹□∀u'([F]u' & [R']u'v → u' =E a)›
    using Ordinary.res−var−bound−reas[BF] →E by fast
qed
AOT-hence 1: ‹□∃u ([F]u & [R']uv & ∀u' ([F]u' & [R']u'v → u' =E u))›
  using Ordinary.res−var−bound−reas[Buridan] →E by fast
AOT-hence ‹□∃!u ([F]u & [R']uv)›
```

by (*AOT-subst-thm equi*:*1*)
moreover **AOT-have** ‹¬□∃!*u* ([*F*]*u* & [*R'*]*uv*)›
using *A*[*THEN* &*E*(*2*)] *KBasic*:*11*[*THEN* ≡*E*(*2*)] **by** *blast*
ultimately AOT-show ‹□∃!*u* ([*F*]*u* & [*R'*]*uv*) & ¬□∃!*u* ([*F*]*u* & [*R'*]*uv*)›
by (*rule* &*I*)
qed
qed
AOT-hence ‹□∀ *u* ([*F*]*u* → ∃!*v* ([*G*]*v* & [*R'*]*uv*))›
and ‹□∀ *v* ([*G*]*v* → ∃!*u* ([*F*]*u* & [*R'*]*uv*))›
using *Ordinary.res−var−bound−reas*[*BF*][*THEN* →*E*] **by** *auto*
moreover AOT-have ‹□[*R'*]↓› **and** ‹□[*F*]↓› **and** ‹□[*G*]↓›
by (*simp-all add*: *ex*:*2*:*a*)
ultimately AOT-have ‹□([*R'*]↓ & [*F*]↓ & [*G*]↓ & ∀ *u* ([*F*]*u* → ∃!*v* ([*G*]*v* & [*R'*]*uv*)) &
∀ *v* ([*G*]*v* → ∃!*u* ([*F*]*u* & [*R'*]*uv*)))›
using *KBasic*:*3* &*I* ≡*E*(*2*) **by** *meson*
AOT-hence ‹□*R'* |: *F* $_{1-1}$⟷$_E$ *G*›
by (*AOT-subst-def equi*:*2*)
AOT-hence ‹∃ *R* □*R* |: *F* $_{1-1}$⟷$_E$ *G*›
by (*rule* ∃ *I*(*2*))
AOT-hence ‹□∃ *R R* |: *F* $_{1-1}$⟷$_E$ *G*›
by (*metis Buridan* →*E*)
AOT-thus ‹□*F* ≈$_E$ *G*›
by (*AOT-subst-def equi*:*3*)
}
qed
ultimately AOT-show ‹□(*F* ≈$_E$ *G* → □*F* ≈$_E$ *G*)›
using →*E* **by** *blast*
qed

AOT-define *numbers* :: ‹τ ⇒ τ ⇒ φ› (‹*Numbers'*(-,-')›)
‹*Numbers*(*x*,*G*) ≡$_{df}$ *A*!*x* & *G*↓ & ∀ *F*(*x*[*F*] ≡ [λ*z* 𝓐[*F*]*z*] ≈$_E$ *G*)›

AOT-theorem *numbers*[*den*]:
‹Π↓ → (*Numbers*(κ, Π) ≡ *A*!κ & ∀ *F*(κ[*F*] ≡ [λ*z* 𝓐[*F*]*z*] ≈$_E$ Π))›
apply (*safe intro*!: *numbers*[*THEN* ≡$_{df}$*I*] &*I* ≡*I* →*I cqt*:*2*
dest!: *numbers*[*THEN* ≡$_{df}$*E*])
using &*E* **by** *blast+*

AOT-theorem *num−tran*:*1*:
‹*G* ≈$_E$ *H* → (*Numbers*(*x*, *G*) ≡ *Numbers*(*x*, *H*))›
proof (*safe intro*!: →*I* ≡*I*)
AOT-assume *0*: ‹*G* ≈$_E$ *H*›
AOT-assume ‹*Numbers*(*x*, *G*)›
AOT-hence *Ax*: ‹*A*!*x*› **and** ϑ: ‹∀ *F* (*x*[*F*] ≡ [λ*z* 𝓐[*F*]*z*] ≈$_E$ *G*)›
using *numbers*[*THEN* ≡$_{df}$*E*] &*E* **by** *blast+*
AOT-show ‹*Numbers*(*x*, *H*)›
proof(*safe intro*!: *numbers*[*THEN* ≡$_{df}$*I*] &*I Ax cqt*:*2 GEN*)
fix *F*
AOT-have ‹*x*[*F*] ≡ [λ*z* 𝓐[*F*]*z*] ≈$_E$ *G*›
using ϑ[*THEN* ∀ *E*(*2*)]**.**
also AOT-have ‹. . . ≡ [λ*z* 𝓐[*F*]*z*] ≈$_E$ *H*›
using *0 approx−nec*:*2*[*THEN* ≡*E*(*1*), *THEN* ∀ *E*(*2*)] **by** *metis*
finally AOT-show ‹*x*[*F*] ≡ [λ*z* 𝓐[*F*]*z*] ≈$_E$ *H*›**.**
qed
next
AOT-assume ‹*G* ≈$_E$ *H*›
AOT-hence *0*: ‹*H* ≈$_E$ *G*›
by (*metis eq−part*:*2* →*E*)
AOT-assume ‹*Numbers*(*x*, *H*)›
AOT-hence *Ax*: ‹*A*!*x*› **and** ϑ: ‹∀ *F* (*x*[*F*] ≡ [λ*z* 𝓐[*F*]*z*] ≈$_E$ *H*)›
using *numbers*[*THEN* ≡$_{df}$*E*] &*E* **by** *blast+*
AOT-show ‹*Numbers*(*x*, *G*)›

```
  proof(safe intro!: numbers[THEN ≡df I] &I Ax cqt:2 GEN)
    fix F
    AOT-have ‹x[F] ≡ [λz 𝓐[F]z] ≈E H›
      using ϑ[THEN ∀E(2)].
    also AOT-have ‹... ≡ [λz 𝓐[F]z] ≈E G›
      using 0 approx−nec:2[THEN ≡E(1), THEN ∀E(2)] by metis
    finally AOT-show ‹x[F] ≡ [λz 𝓐[F]z] ≈E G›.
  qed
qed

AOT-theorem num−tran:2:
  ‹(Numbers(x, G) & Numbers(x,H)) → G ≈E H›
proof (rule →I; frule &E(1); drule &E(2))
  AOT-assume ‹Numbers(x,G)›
  AOT-hence ‹∀F (x[F] ≡ [λz 𝓐[F]z] ≈E G)›
    using numbers[THEN ≡df E] &E by blast
  AOT-hence 1: ‹x[F] ≡ [λz 𝓐[F]z] ≈E G› for F
    using ∀E(2) by blast
  AOT-assume ‹Numbers(x,H)›
  AOT-hence ‹∀F (x[F] ≡ [λz 𝓐[F]z] ≈E H)›
    using numbers[THEN ≡df E] &E by blast
  AOT-hence ‹x[F] ≡ [λz 𝓐[F]z] ≈E H› for F
    using ∀E(2) by blast
  AOT-hence ‹[λz 𝓐[F]z] ≈E G ≡ [λz 𝓐[F]z] ≈E H› for F
    by (metis 1 ≡E(6))
  AOT-thus ‹G ≈E H›
    using approx−nec:2[THEN ≡E(2), OF GEN] by blast
qed

AOT-theorem num−tran:3:
  ‹G ≡E H → (Numbers(x, G) ≡ Numbers(x, H))›
  using apE−eqE:1 Hypothetical Syllogism num−tran:1 by blast

AOT-theorem pre−Hume:
  ‹(Numbers(x,G) & Numbers(y,H)) → (x = y ≡ G ≈E H)›
proof(safe intro!: →I ≡I; frule &E(1); drule &E(2))
  AOT-assume ‹Numbers(x, G)›
  moreover AOT-assume ‹x = y›
  ultimately AOT-have ‹Numbers(y, G)› by (rule rule=E)
  moreover AOT-assume ‹Numbers(y, H)›
  ultimately AOT-show ‹G ≈E H› using num−tran:2 →E &I by blast
next
  AOT-assume ‹Numbers(x, G)›
  AOT-hence Ax: ‹A!x› and xF: ‹∀F (x[F] ≡ [λz 𝓐[F]z] ≈E G)›
    using numbers[THEN ≡df E] &E by blast+
  AOT-assume ‹Numbers(y, H)›
  AOT-hence Ay: ‹A!y› and yF: ‹∀F (y[F] ≡ [λz 𝓐[F]z] ≈E H)›
    using numbers[THEN ≡df E] &E by blast+
  AOT-assume G-approx-H: ‹G ≈E H›
  AOT-show ‹x = y›
  proof(rule ab−obey:1[THEN →E, THEN →E, OF &I, OF Ax, OF Ay]; rule GEN)
    fix F
    AOT-have ‹x[F] ≡ [λz 𝓐[F]z] ≈E G›
      using xF[THEN ∀E(2)].
    also AOT-have ‹... ≡ [λz 𝓐[F]z] ≈E H›
      using approx−nec:2[THEN ≡E(1), OF G-approx-H, THEN ∀E(2)].
    also AOT-have ‹... ≡ y[F]›
      using yF[THEN ∀E(2), symmetric].
    finally AOT-show ‹x[F] ≡ y[F]›.
  qed
qed

AOT-theorem two−num−not:
```

‹∃ u∃ v(u ≠ v) → ∃ x∃ G∃ H(*Numbers*(x,G) & *Numbers*(x, H) & ¬G $\equiv_E$ H)›
proof (*rule* →*I*)
  AOT-have *eqE-den*: ‹[λx x $=_E$ y]↓› **for** y **by** *cqt*:*2*
  AOT-assume ‹∃ u∃ v(u ≠ v)›
  then AOT-obtain c **where** *Oc*: ‹O!c› **and** ‹∃ v (c ≠ v)›
    using &*E* ∃ *E*[*rotated*] **by** *blast*
  then AOT-obtain d **where** *Od*: ‹O!d› **and** *c-noteq-d*: ‹c ≠ d›
    using &*E* ∃ *E*[*rotated*] **by** *blast*
  AOT-hence *c-noteqE-d*: ‹c $\neq_E$ d›
    using =*E*−*simple*:*2*[*THEN* →*E*] =*E*−*simple*:*2* ≡*E*(*2*) *modus*−*tollens*:*1*
        =−*infix* $\equiv_{df}$*E* *thm*−*neg*=*E* **by** *fast*
  AOT-hence *not-c-eqE-d*: ‹¬c $=_E$ d›
    using ≡*E*(*1*) *thm*−*neg*=*E* **by** *blast*
  AOT-have ‹∃ x (A!x & ∀ F (x[F] ≡ [λz 𝓐[F]z] $\approx_E$ [λx x $=_E$ c]))›
    by (*simp add*: *A*−*objects*[*axiom-inst*])
  then AOT-obtain a **where** *a-prop*: ‹A!a & ∀ F (a[F] ≡ [λz 𝓐[F]z] $\approx_E$ [λx x $=_E$ c])›
    using ∃ *E*[*rotated*] **by** *blast*
  AOT-have ‹∃ x (A!x & ∀ F (x[F] ≡ [λz 𝓐[F]z] $\approx_E$ [λx x $=_E$ d]))›
    by (*simp add*: *A*−*objects vdash*−*properties*:*1*[*2*])
  then AOT-obtain b **where** *b-prop*: ‹A!b & ∀ F (b[F] ≡ [λz 𝓐[F]z] $\approx_E$ [λx x $=_E$ d])›
    using ∃ *E*[*rotated*] **by** *blast*
  AOT-have *num-a-eq-c*: ‹*Numbers*(a, [λx x $=_E$ c])›
    by (*safe intro*!: *numbers*[*THEN* $\equiv_{df}$*I*] &*I a-prop*[*THEN* &*E*(*1*)]
              a-prop[*THEN* &*E*(*2*)]) *cqt*:*2*
  moreover AOT-have *num-b-eq-d*: ‹*Numbers*(b, [λx x $=_E$ d])›
    by (*safe intro*!: *numbers*[*THEN* $\equiv_{df}$*I*] &*I b-prop*[*THEN* &*E*(*1*)]
              b-prop[*THEN* &*E*(*2*)]) *cqt*:*2*
  moreover AOT-have ‹[λx x $=_E$ c] $\approx_E$ [λx x $=_E$ d]›
  proof (*rule equi*:*3*[*THEN* $\equiv_{df}$*I*])
    let *?R* = ‹«[λxy (x $=_E$ c & y $=_E$ d)]»›
    AOT-have *Rcd*: ‹[«*?R*»]cd›
      by (*auto intro*!: β←*C*(*1*) *cqt*:*2* &*I prod-denotesI*
              ord=*Eequiv*:*1*[*THEN* →*E*] *Od Oc*)
    AOT-show ‹∃ R R |: [λx x $=_E$ c] $_{1-1}\longleftrightarrow_E$ [λx x $=_E$ d]›
    proof (*safe intro*!: ∃ *I*(*1*)[**where** τ=‹*?R*›] *equi*:*2*[*THEN* $\equiv_{df}$*I*] &*I*
              eqE-den Ordinary.*GEN* →*I*)
      AOT-show ‹«*?R*»↓› **by** *cqt*:*2*
    next
      fix u
      AOT-assume ‹[λx x $=_E$ c]u›
      AOT-hence ‹u $=_E$ c›
        by (*metis* β→*C*(*1*))
      AOT-hence *u-is-c*: ‹u = c›
        by (*metis* =*E*−*simple*:*2* →*E*)
      AOT-show ‹∃ !v ([λx x $=_E$ d]v & [«*?R*»]uv)›
      proof (*safe intro*!: *equi*:*1*[*THEN* ≡*E*(*2*)] ∃ *I*(*2*)[**where** β=d] &*I*
              Od Ordinary.*GEN* →*I*)
        AOT-show ‹[λx x $=_E$ d]d›
          by (*auto intro*!: β←*C*(*1*) *cqt*:*2* *ord*=*Eequiv*:*1*[*THEN* →*E*, *OF Od*])
      next
        AOT-show ‹[«*?R*»]ud›
          using *u-is-c*[*symmetric*] *Rcd rule*=*E* **by** *fast*
      next
        fix v
        AOT-assume ‹[λx x $=_E$ d]v & [«*?R*»]uv›
        AOT-thus ‹v $=_E$ d›
          by (*metis* β→*C*(*1*) &*E*(*1*))
      qed
    next
      fix v
      AOT-assume ‹[λx x $=_E$ d]v›
      AOT-hence ‹v $=_E$ d›
        by (*metis* β→*C*(*1*))

```
      AOT-hence v-is-d: ‹v = d›
        by (metis =E−simple:2 →E)
      AOT-show ‹∃!u ([λx x =E c]u & [«?R»]uv)›
      proof (safe intro!: equi:1[THEN ≡E(2)] ∃I(2)[where β=c] &I
                          Oc Ordinary.GEN →I)
        AOT-show ‹[λx x =E c]c›
          by (auto intro!: β←C(1) cqt:2 ord=Eequiv:1[THEN →E, OF Oc])
      next
        AOT-show ‹[«?R»]cv›
          using v-is-d[symmetric] Rcd rule=E by fast
      next
        fix u
        AOT-assume ‹[λx x =E c]u & [«?R»]uv›
        AOT-thus ‹u =E c›
          by (metis β→C(1) &E(1))
      qed
    next
      AOT-show ‹«?R»↓›
        by cqt:2
    qed
  qed
  ultimately AOT-have ‹a = b›
    using pre−Hume[unvarify G H, OF eqE-den, OF eqE-den, THEN →E,
                   OF &I, THEN ≡E(2)] by blast
  AOT-hence num-a-eq-d: ‹Numbers(a, [λx x =E d])›
    using num-b-eq-d rule=E id-sym by fast
  AOT-have not-equiv: ‹¬[λx x =E c] ≡E [λx x =E d]›
  proof (rule raa−cor:2)
    AOT-assume ‹[λx x =E c] ≡E [λx x =E d]›
    AOT-hence ‹[λx x =E c]c ≡ [λx x =E d]c›
      using eqE[THEN ≡df E, THEN &E(2), THEN ∀E(2), THEN →E] Oc by blast
    moreover AOT-have ‹[λx x =E c]c›
      by (auto intro!: β←C(1) cqt:2 ord=Eequiv:1[THEN →E, OF Oc])
    ultimately AOT-have ‹[λx x =E d]c›
      using ≡E(1) by blast
    AOT-hence ‹c =E d›
      by (rule β→C(1))
    AOT-thus ‹c =E d & ¬c =E d›
      using not-c-eqE-d &I by blast
  qed
  AOT-show ‹∃x ∃G ∃H (Numbers(x,G) & Numbers(x,H) & ¬G ≡E H)›
    apply (rule ∃I(2)[where β=a])
    apply (rule ∃I(1)[where τ=‹«[λx x =E c]»›])
     apply (rule ∃I(1)[where τ=‹«[λx x =E d]»›])
    by (safe intro!: eqE-den &I num-a-eq-c num-a-eq-d not-equiv)
qed

AOT-theorem num:1: ‹∃x Numbers(x,G)›
  by (AOT-subst ‹Numbers(x,G)› ‹[A!]x & ∀F (x[F] ≡ [λz 𝓐[F]z] ≈E G)› for: x)
     (auto simp: numbers[den][THEN →E, OF cqt:2[const-var][axiom-inst]]
                 A−objects[axiom-inst])

AOT-theorem num:2: ‹∃!x Numbers(x,G)›
  by (AOT-subst ‹Numbers(x,G)› ‹[A!]x & ∀F (x[F] ≡ [λz 𝓐[F]z] ≈E G)› for: x)
     (auto simp: numbers[den][THEN →E, OF cqt:2[const-var][axiom-inst]]
                 A−objects!)

AOT-theorem num−cont:1:
  ‹∃x∃G(Numbers(x, G) & ¬□Numbers(x, G))›
proof −
  AOT-have ‹∃F∃G ◇([λz 𝓐[F]z] ≈E G & ◇¬[λz 𝓐[F]z] ≈E G)›
    using approx−cont:2.
  then AOT-obtain F where ‹∃G ◇([λz 𝓐[F]z] ≈E G & ◇¬[λz 𝓐[F]z] ≈E G)›
```

using $\exists\, E[rotated]$ **by** *blast*
then AOT-obtain G **where** ‹$\Diamond([\lambda z\ \boldsymbol{\mathcal{A}}[F]z] \approx_E G\ \&\ \Diamond\neg[\lambda z\ \boldsymbol{\mathcal{A}}[F]z] \approx_E G)$›
using $\exists\, E[rotated]$ **by** *blast*
AOT-hence ϑ: ‹$\Diamond[\lambda z\ \boldsymbol{\mathcal{A}}[F]z] \approx_E G$› **and** ζ: ‹$\Diamond\neg[\lambda z\ \boldsymbol{\mathcal{A}}[F]z] \approx_E G$›
using *KBasic2*:*3*[*THEN* $\rightarrow E$] &*E* *4*$\Diamond$[*THEN* $\rightarrow E$] **by** *blast+*
AOT-obtain a **where** ‹$Numbers(a,\ G)$›
using *num*:*1* $\exists\, E[rotated]$ **by** *blast*
moreover AOT-have ‹$\neg\Box Numbers(a,\ G)$›
proof (*rule raa−cor*:*2*)
AOT-assume ‹$\Box Numbers(a,\ G)$›
AOT-hence ‹$\Box([A!]a\ \&\ G{\downarrow}\ \&\ \forall\, F\ (a[F] \equiv [\lambda z\ \boldsymbol{\mathcal{A}}[F]z] \approx_E G))$›
by (*AOT-subst-def* (*reverse*) *numbers*)
AOT-hence ‹$\Box A!a$› **and** ‹$\Box\forall\, F\ (a[F] \equiv [\lambda z\ \boldsymbol{\mathcal{A}}[F]z] \approx_E G)$›
using *KBasic*:*3*[*THEN* $\equiv E(1)$] &*E* **by** *blast+*
AOT-hence ‹$\forall\, F\ \Box(a[F] \equiv [\lambda z\ \boldsymbol{\mathcal{A}}[F]z] \approx_E G)$›
using *CBF*[*THEN* $\rightarrow E$] **by** *blast*
AOT-hence ‹$\Box(a[F] \equiv [\lambda z\ \boldsymbol{\mathcal{A}}[F]z] \approx_E G)$›
using $\forall\, E(2)$ **by** *blast*
AOT-hence A: ‹$\Box(a[F] \rightarrow [\lambda z\ \boldsymbol{\mathcal{A}}[F]z] \approx_E G)$›
and B: ‹$\Box([\lambda z\ \boldsymbol{\mathcal{A}}[F]z] \approx_E G \rightarrow a[F])$›
using *KBasic*:*4*[*THEN* $\equiv E(1)$] &*E* **by** *blast+*
AOT-have ‹$\Box(\neg[\lambda z\ \boldsymbol{\mathcal{A}}[F]z] \approx_E G \rightarrow \neg a[F])$›
apply (*AOT-subst* ‹$\neg[\lambda z\ \boldsymbol{\mathcal{A}}[F]z] \approx_E G \rightarrow \neg a[F]$› ‹$a[F] \rightarrow [\lambda z\ \boldsymbol{\mathcal{A}}[F]z] \approx_E G$›)
using $\equiv I$ *useful−tautologies*:*4* *useful−tautologies*:*5* **apply** *presburger*
by (*fact A*)
AOT-hence ‹$\Diamond\neg a[F]$›
by (*metis KBasic*:*13* ζ $\rightarrow E$)
AOT-hence ‹$\neg a[F]$›
by (*metis KBasic*:*11* *en−eq*:*2*[*1*] $\equiv E(2)$ $\equiv E(4)$)
AOT-hence ‹$\neg\Diamond a[F]$›
by (*metis en−eq*:*3*[*1*] $\equiv E(4)$)
moreover AOT-have ‹$\Diamond a[F]$›
by (*meson B* ϑ *KBasic*:*13* $\rightarrow E$)
ultimately AOT-show ‹$\Diamond a[F]\ \&\ \neg\Diamond a[F]$›
using &*I* **by** *blast*
qed

ultimately AOT-have ‹$Numbers(a,\ G)\ \&\ \neg\Box Numbers(a,\ G)$›
using &*I* **by** *blast*
AOT-hence ‹$\exists\, G\ (Numbers(a,\ G)\ \&\ \neg\Box Numbers(a,\ G))$›
by (*rule* $\exists\, I$)
AOT-thus ‹$\exists\, x \exists\, G\ (Numbers(x,\ G)\ \&\ \neg\Box Numbers(x,\ G))$›
by (*rule* $\exists\, I$)
qed

AOT-theorem *num−cont*:*2*:
‹$Rigid(G) \rightarrow \Box\forall\, x(Numbers(x,G) \rightarrow \Box Numbers(x,G))$›
proof(*rule* $\rightarrow I$)
AOT-assume ‹$Rigid(G)$›
AOT-hence ‹$\Box\forall\, z([G]z \rightarrow \Box[G]z)$›
using *df−rigid−rel*:*1*[*THEN* $\equiv_{df} E$, *THEN* &*E*(*2*)] **by** *blast*
AOT-hence ‹$\Box\Box\forall\, z([G]z \rightarrow \Box[G]z)$› **by** (*metis S5Basic*:*6* $\equiv E(1)$)
moreover AOT-have ‹$\Box\Box\forall\, z([G]z \rightarrow \Box[G]z) \rightarrow \Box\forall\, x(Numbers(x,G) \rightarrow \Box Numbers(x,G))$›
proof(*rule RM*; *safe intro*!: $\rightarrow I$ *GEN*)
AOT-modally-strict {
AOT-have *act-den*: ‹$[\lambda z\ \boldsymbol{\mathcal{A}}[F]z]{\downarrow}$› **for** F **by** *cqt*:*2*[*lambda*]
fix x
AOT-assume *G-nec*: ‹$\Box\forall\, z([G]z \rightarrow \Box[G]z)$›
AOT-hence *G-rigid*: ‹$Rigid(G)$›
using *df−rigid−rel*:*1*[*THEN* $\equiv_{df} I$, *OF* &*I*] *cqt*:*2*
by *blast*
AOT-assume ‹$Numbers(x,\ G)$›
AOT-hence ‹$[A!]x\ \&\ G{\downarrow}\ \&\ \forall\, F\ (x[F] \equiv [\lambda z\ \boldsymbol{\mathcal{A}}[F]z] \approx_E G)$›

```
        using numbers[THEN ≡df E] by blast
      AOT-hence Ax: ‹[A!]x› and ‹∀ F (x[F] ≡ [λz 𝓐[F]z] ≈E G)›
        using &E by blast+
      AOT-hence ‹x[F] ≡ [λz 𝓐[F]z] ≈E G› for F
        using ∀ E(2) by blast
      moreover AOT-have ‹□([λz 𝓐[F]z] ≈E G → □[λz 𝓐[F]z] ≈E G)› for F
        using approx−nec:3[unvarify F, OF act-den, THEN →E, OF &I,
                          OF actuallyF:2, OF G-rigid].
      moreover AOT-have ‹□(x[F] → □x[F])› for F
        by (simp add: RN pre−en−eq:1[1])
      ultimately AOT-have ‹□(x[F] ≡ [λz 𝓐[F]z] ≈E G)› for F
        using sc−eq−box−box:5 →E qml:2[axiom-inst] &I by meson
      AOT-hence ‹∀ F □(x[F] ≡ [λz 𝓐[F]z] ≈E G)›
        by (rule ∀ I)
      AOT-hence 1: ‹□∀ F (x[F] ≡ [λz 𝓐[F]z] ≈E G)›
        using BF[THEN →E] by fast
      AOT-have ‹□G↓›
        by (simp add: ex:2:a)
      moreover AOT-have ‹□[A!]x›
        using Ax oa−facts:2 →E by blast
      ultimately AOT-have ‹□(A!x & G↓)›
        by (metis KBasic:3 &I ≡E(2))
      AOT-hence ‹□(A!x & G↓ & ∀ F (x[F] ≡ [λz 𝓐[F]z] ≈E G))›
        using 1 KBasic:3 &I ≡E(2) by fast
      AOT-thus ‹□Numbers(x, G)›
        by (AOT-subst-def numbers)
    }
  qed
  ultimately AOT-show ‹□∀ x(Numbers(x,G) → □Numbers(x,G))›
    using →E by blast
qed

AOT-theorem num−cont:3:
  ‹□∀ x(Numbers(x, [λz 𝓐[G]z]) → □Numbers(x, [λz 𝓐[G]z]))›
  by (rule num−cont:2[unvarify G, THEN →E];
      (cqt:2[lambda] | rule actuallyF:2))

AOT-theorem num−uniq: ‹ιx Numbers(x, G)↓›
  using ≡E(2) A−Exists:2 RA[2] num:2 by blast

AOT-define num :: ‹τ ⇒ κs› (‹#-› [100] 100)
  num−def:1: ‹#G =df ιx Numbers(x, G)›

AOT-theorem num−def:2: ‹#G↓›
  using num−def:1[THEN =df I(1)] num−uniq by simp

AOT-theorem num−can:1:
  ‹#G = ιx(A!x & ∀ F (x[F] ≡ [λz 𝓐[F]z] ≈E G))›
proof −
  AOT-have ‹□∀ x(Numbers(x,G) ≡ [A!]x & ∀ F (x[F] ≡ [λz 𝓐[F]z] ≈E G))›
    by (safe intro!: RN GEN numbers[den][THEN →E] cqt:2)
  AOT-hence ‹ιx Numbers(x, G) = ιx([A!]x & ∀ F (x[F] ≡ [λz 𝓐[F]z] ≈E G))›
    using num−uniq equiv−desc−eq:3[THEN →E, OF &I] by auto
  thus ?thesis
    by (rule =df I(1)[OF num−def:1, OF num−uniq])
qed

AOT-theorem num−can:2: ‹#G = ιx(A!x & ∀ F (x[F] ≡ F ≈E G))›
proof (rule id-trans[OF num−can:1]; rule equiv−desc−eq:2[THEN →E];
       safe intro!: &I A−descriptions GEN Act−Basic:5[THEN ≡E(2)]
                    logic−actual−nec:3[axiom-inst, THEN ≡E(2)])
  AOT-have act-den: ‹⊢□ [λz 𝓐[F]z]↓› for F
    by cqt:2
```

```
  AOT-have eq-part:3[terms]: ‹⊢□ F ≈E G & F ≈E H → G ≈E H› for F G H
    by (metis &I eq-part:2 eq-part:3 →I &E →E)
  fix x
  {
    fix F
    AOT-have ‹𝓐(F ≈E [λz 𝓐[F]z])›
      by (simp add: actuallyF:1)
    moreover AOT-have ‹𝓐((F ≈E [λz 𝓐[F]z]) → ([λz 𝓐[F]z] ≈E G ≡ F ≈E G))›
      by (auto intro!: RA[2] →I ≡I
              simp: eq-part:3[unvarify G, OF act-den, THEN →E, OF &I]
                    eq-part:3[terms][unvarify G, OF act-den, THEN →E, OF &I])
    ultimately AOT-have ‹𝓐([λz 𝓐[F]z] ≈E G ≡ F ≈E G)›
      using logic-actual-nec:2[axiom-inst, THEN ≡E(1), THEN →E] by blast

    AOT-hence ‹𝓐[λz 𝓐[F]z] ≈E G ≡ 𝓐F ≈E G›
      by (metis Act-Basic:5 ≡E(1))
    AOT-hence 0: ‹(𝓐x[F] ≡ 𝓐[λz 𝓐[F]z] ≈E G) ≡ (𝓐x[F] ≡ 𝓐F ≈E G)›
      by (auto intro!: ≡I →I elim: ≡E)
    AOT-have ‹𝓐(x[F] ≡ [λz 𝓐[F]z] ≈E G) ≡ (𝓐x[F] ≡ 𝓐[λz 𝓐[F]z] ≈E G)›
      by (simp add: Act-Basic:5)
    also AOT-have ‹... ≡ (𝓐x[F] ≡ 𝓐F ≈E G)› using 0.
    also AOT-have ‹... ≡ 𝓐((x[F] ≡ F ≈E G))›
      by (meson Act-Basic:5 ≡E(6) oth-class-taut:3:a)
    finally AOT-have 0: ‹𝓐(x[F] ≡ [λz 𝓐[F]z] ≈E G) ≡ 𝓐((x[F] ≡ F ≈E G))›.
  } note 0 = this
  AOT-have ‹𝓐∀F (x[F] ≡ [λz 𝓐[F]z] ≈E G) ≡ ∀F 𝓐(x[F] ≡ [λz 𝓐[F]z] ≈E G)›
    using logic-actual-nec:3 vdash-properties:1[2] by blast
  also AOT-have ‹... ≡ ∀F 𝓐((x[F] ≡ F ≈E G))›
    apply (safe intro!: ≡I →I GEN)
    using 0 ≡E(1) ≡E(2) rule-ui:3 by blast+
  also AOT-have ‹... ≡ 𝓐(∀F (x[F] ≡ F ≈E G))›
    using ≡E(6) logic-actual-nec:3[axiom-inst] oth-class-taut:3:a by fast
  finally AOT-have 0: ‹𝓐∀F (x[F] ≡ [λz 𝓐[F]z] ≈E G) ≡ 𝓐(∀F (x[F] ≡ F ≈E G))›.
  AOT-have ‹𝓐([A!]x & ∀F (x[F] ≡ [λz 𝓐[F]z] ≈E G)) ≡
          (𝓐A!x & 𝓐∀F (x[F] ≡ [λz 𝓐[F]z] ≈E G))›
    by (simp add: Act-Basic:2)
  also AOT-have ‹... ≡ 𝓐[A!]x & 𝓐(∀F (x[F] ≡ F ≈E G))›
    using 0 oth-class-taut:4:f →E by blast
  also AOT-have ‹... ≡ 𝓐(A!x & ∀F (x[F] ≡ F ≈E G))›
    using Act-Basic:2 ≡E(6) oth-class-taut:3:a by blast
  finally AOT-show ‹𝓐([A!]x & ∀F (x[F] ≡ [λz 𝓐[F]z] ≈E G)) ≡
                  𝓐([A!]x & ∀F (x[F] ≡ F ≈E G))›.
qed

AOT-define NaturalCardinal :: ‹τ ⇒ φ› (‹NaturalCardinal'(-')›)
  card: ‹NaturalCardinal(x) ≡df ∃G(x = #G)›

AOT-theorem natcard-nec: ‹NaturalCardinal(x) → □NaturalCardinal(x)›
proof(rule →I)
  AOT-assume ‹NaturalCardinal(x)›
  AOT-hence ‹∃G(x = #G)› using card[THEN ≡df E] by blast
  then AOT-obtain G where ‹x = #G› using ∃E[rotated] by blast
  AOT-hence ‹□x = #G› by (metis id-nec:2 →E)
  AOT-hence ‹∃G □x = #G› by (rule ∃I)
  AOT-hence ‹□∃G x = #G› by (metis Buridan →E)
  AOT-thus ‹□NaturalCardinal(x)›
    by (AOT-subst-def card)
qed

AOT-act-theorem hume:1: ‹Numbers(#G, G)›
  apply (rule =df I(1)[OF num-def:1])
  apply (simp add: num-uniq)
  using num-uniq vdash-properties:10 y-in:3 by blast
```

```
AOT-act-theorem hume:2: ‹#F = #G ≡ F ≈E G›
  by (safe intro!: pre−Hume[unvarify x y, OF num−def:2,
                              OF num−def:2, THEN →E] &I hume:1)

AOT-act-theorem hume:3: ‹#F = #G ≡ ∃ R (R |: F 1−1⟶ontoE G)›
  using equi−rem−thm
  apply (AOT-subst (reverse) ‹R |: F 1−1⟶ontoE G›
                             ‹R |: F 1−1⟷E G› for: R :: ‹<κ×κ>›)
  using equi:3 hume:2 ≡E(5) ≡Df by blast

AOT-act-theorem hume:4: ‹F ≡E G → #F = #G›
  by (metis apE−eqE:1 deduction−theorem hume:2 ≡E(2) →E)

AOT-theorem hume−strict:1:
  ‹∃ x (Numbers(x, F) & Numbers(x, G)) ≡ F ≈E G›
proof(safe intro!: ≡I →I)
  AOT-assume ‹∃ x (Numbers(x, F) & Numbers(x, G))›
  then AOT-obtain a where ‹Numbers(a, F) & Numbers(a, G)›
    using ∃ E[rotated] by blast
  AOT-thus ‹F ≈E G›
    using num−tran:2 →E by blast
next
  AOT-assume 0: ‹F ≈E G›
  moreover AOT-obtain b where num-b-F: ‹Numbers(b, F)›
    by (metis instantiation num:1)
  moreover AOT-have num-b-G: ‹Numbers(b, G)›
    using calculation num−tran:1[THEN →E, THEN ≡E(1)] by blast
  ultimately AOT-have ‹Numbers(b, F) & Numbers(b, G)›
    by (safe intro!: &I)
  AOT-thus ‹∃ x (Numbers(x, F) & Numbers(x, G))›
    by (rule ∃ I)
qed

AOT-theorem hume−strict:2:
  ‹∃ x∃ y (Numbers(x, F) &
         ∀ z(Numbers(z,F) → z = x) &
         Numbers(y, G) &
         ∀ z (Numbers(z, G) → z = y) &
         x = y) ≡
   F ≈E G›
proof(safe intro!: ≡I →I)
  AOT-assume ‹∃ x∃ y (Numbers(x, F) & ∀ z(Numbers(z,F) → z = x) &
                    Numbers(y, G) & ∀ z (Numbers(z, G) → z = y) & x = y)›
  then AOT-obtain x where
    ‹∃ y (Numbers(x, F) & ∀ z(Numbers(z,F) → z = x) & Numbers(y, G) &
         ∀ z (Numbers(z, G) → z = y) & x = y)›
    using ∃ E[rotated] by blast
  then AOT-obtain y where
    ‹Numbers(x, F) & ∀ z(Numbers(z,F) → z = x) & Numbers(y, G) &
     ∀ z (Numbers(z, G) → z = y) & x = y›
    using ∃ E[rotated] by blast
  AOT-hence ‹Numbers(x, F)› and ‹Numbers(y,G)› and ‹x = y›
    using &E by blast+
  AOT-hence ‹Numbers(y, F) & Numbers(y, G)›
    using &I rule=E by fast
  AOT-hence ‹∃ y (Numbers(y, F) & Numbers(y, G))›
    by (rule ∃ I)
  AOT-thus ‹F ≈E G›
    using hume−strict:1[THEN ≡E(1)] by blast
next
  AOT-assume ‹F ≈E G›
  AOT-hence ‹∃ x (Numbers(x, F) & Numbers(x, G))›
```

```
    using hume−strict:1[THEN ≡E(2)] by blast
  then AOT-obtain x where ‹Numbers(x, F) & Numbers(x, G)›
    using ∃E[rotated] by blast
  moreover AOT-have ‹∀z (Numbers(z, F) → z = x)›
               and ‹∀z (Numbers(z, G) → z = x)›
    using calculation
    by (auto intro!: GEN →I pre−Hume[THEN →E, OF &I, THEN ≡E(2),
                                   rotated 2, OF eq−part:1] dest: &E)
  ultimately AOT-have ‹Numbers(x, F) & ∀z(Numbers(z,F) → z = x) &
                    Numbers(x, G) & ∀z (Numbers(z, G) → z = x) & x = x›
    by (auto intro!: &I id−eq:1 dest: &E)
  AOT-thus ‹∃x∃y (Numbers(x, F) & ∀z(Numbers(z,F) → z = x) & Numbers(y, G) &
               ∀z (Numbers(z, G) → z = y) & x = y)›
    by (auto intro!: ∃I)
qed

AOT-theorem unotEu: ‹¬∃y[λx O!x & x ≠E x]y›
proof(rule raa−cor:2)
  AOT-assume ‹∃y[λx O!x & x ≠E x]y›
  then AOT-obtain y where ‹[λx O!x & x ≠E x]y›
    using ∃E[rotated] by blast
  AOT-hence 0: ‹O!y & y ≠E y›
    by (rule β→C(1))
  AOT-hence ‹¬(y =E y)›
    using &E(2) ≡E(1) thm−neg=E by blast
  moreover AOT-have ‹y =E y›
    by (metis 0[THEN &E(1)] ord=Eequiv:1 →E)
  ultimately AOT-show ‹p & ¬p› for p
    by (metis raa−cor:3)
qed

AOT-define zero :: ‹κs› (‹0›)
  zero:1: ‹0 =df #[λx O!x & x ≠E x]›

AOT-theorem zero:2: ‹0↓›
  by (rule =dfI(2)[OF zero:1]; rule num−def:2[unvarify G]; cqt:2)

AOT-theorem zero−card: ‹NaturalCardinal(0)›
  apply (rule =dfI(2)[OF zero:1])
   apply (rule num−def:2[unvarify G]; cqt:2)
  apply (rule card[THEN ≡dfI])
  apply (rule ∃I(1)[where τ=‹«[λx [O!]x & x ≠E x]»›])
   apply (rule rule=I:1; rule num−def:2[unvarify G]; cqt:2)
  by cqt:2

AOT-theorem eq−num:1:
  ‹𝓐Numbers(x, G) ≡ Numbers(x,[λz 𝓐[G]z])›
proof −
  AOT-have act-den: ‹⊢□ [λz 𝓐[F]z]↓› for F by cqt:2
  AOT-have ‹□(∃x(Numbers(x, G) & Numbers(x,[λz 𝓐[G]z])) ≡ G ≈E [λz 𝓐[G]z])›
    using hume−strict:1[unvarify G, OF act-den, THEN RN].
  AOT-hence ‹𝓐(∃x(Numbers(x, G) & Numbers(x,[λz 𝓐[G]z])) ≡ G ≈E [λz 𝓐[G]z])›
    using nec−imp−act[THEN →E] by fast
  AOT-hence ‹𝓐(∃x(Numbers(x, G) & Numbers(x,[λz 𝓐[G]z])))›
    using actuallyF:1 Act−Basic:5 ≡E(1) ≡E(2) by fast
  AOT-hence ‹∃x 𝓐((Numbers(x, G) & Numbers(x,[λz 𝓐[G]z])))›
    by (metis Act−Basic:10 intro−elim:3:a)
  then AOT-obtain a where ‹𝓐(Numbers(a, G) & Numbers(a,[λz 𝓐[G]z]))›
    using ∃E[rotated] by blast
  AOT-hence act-a-num-G: ‹𝓐Numbers(a, G)›
     and act-a-num-actG: ‹𝓐Numbers(a,[λz 𝓐[G]z])›
    using Act−Basic:2 &E ≡E(1) by blast+
  AOT-hence num-a-act-g: ‹Numbers(a, [λz 𝓐[G]z])›
```

using *num−cont*:*2*[*unvarify G*, *OF act-den*, *THEN* →*E*, *OF actuallyF*:*2*,
THEN CBF[*THEN* →*E*], *THEN* ∀ *E*(*2*)]
by (*metis* ≡*E*(*1*) *sc−eq−fur*:*2 vdash−properties*:*6*)
AOT-have *0*: ‹$\vdash_\Box$ *Numbers*(*x*, *G*) & *Numbers*(*y*, *G*) → *x* = *y*› **for** *y*
using *pre−Hume*[*THEN* →*E*, *THEN* ≡*E*(*2*), *rotated*, *OF eq−part*:*1*]
→*I* **by** *blast*
show *?thesis*
proof(*safe intro*!: ≡*I* →*I*)
AOT-assume ‹**𝓐***Numbers*(*x*, *G*)›
AOT-hence ‹**𝓐***x* = *a*›
using *0*[*THEN RA*[*2*], *THEN act−cond*[*THEN* →*E*], *THEN* →*E*,
OF Act−Basic:*2*[*THEN* ≡*E*(*2*)], *OF* &*I*]
act-a-num-G **by** *blast*
AOT-hence ‹*x* = *a*› **by** (*metis id−act*:*1* ≡*E*(*2*))
AOT-hence ‹*a* = *x*› **using** *id-sym* **by** *auto*
AOT-thus ‹*Numbers*(*x*, [λ*z* **𝓐**[*G*]*z*])›
using *rule*=*E num-a-act-g* **by** *fast*
next
AOT-assume ‹*Numbers*(*x*, [λ*z* **𝓐**[*G*]*z*])›
AOT-hence ‹*a* = *x*›
using *pre−Hume*[*unvarify G H*, *THEN* →*E*, *OF act-den*, *OF act-den*, *OF* &*I*,
OF num-a-act-g, *THEN* ≡*E*(*2*)]
eq−part:*1*[*unvarify F*, *OF act-den*] **by** *blast*
AOT-thus ‹**𝓐***Numbers*(*x*, *G*)›
using *act-a-num-G rule*=*E* **by** *fast*
qed
qed

AOT-theorem *eq−num*:*2*: ‹*Numbers*(*x*,[λ*z* **𝓐**[*G*]*z*]) ≡ *x* = #*G*›
proof −
AOT-have *0*: ‹$\vdash_\Box$ *x* = ɩ*x Numbers*(*x*, *G*) ≡ ∀ *y* (*Numbers*(*y*, [λ*z* **𝓐**[*G*]*z*]) ≡ *y* = *x*)› **for** *x*
by (*AOT-subst* (*reverse*) ‹*Numbers*(*x*, [λ*z* **𝓐**[*G*]*z*])› ‹**𝓐***Numbers*(*x*, *G*)› **for**: *x*)
(*auto simp*: *eq−num*:*1 descriptions*[*axiom-inst*])
AOT-have ‹#*G* = ɩ*x Numbers*(*x*, *G*) ≡ ∀ *y* (*Numbers*(*y*, [λ*z* **𝓐**[*G*]*z*]) ≡ *y* = #*G*)›
using *0*[*unvarify x*, *OF num−def*:*2*]**.**
moreover AOT-have ‹#*G* = ɩ*x Numbers*(*x*, *G*)›
using *num−def*:*1 num−uniq rule−id−df*:*1* **by** *blast*
ultimately AOT-have ‹∀ *y* (*Numbers*(*y*, [λ*z* **𝓐**[*G*]*z*]) ≡ *y* = #*G*)›
using ≡*E* **by** *blast*
thus *?thesis* **using** ∀ *E*(*2*) **by** *blast*
qed

AOT-theorem *eq−num*:*3*: ‹*Numbers*(#*G*, [λ*y* **𝓐**[*G*]*y*])›
proof −
AOT-have ‹#*G* = #*G*›
by (*simp add*: *rule*=*I*:*1 num−def*:*2*)
thus *?thesis*
using *eq−num*:*2*[*unvarify x*, *OF num−def*:*2*, *THEN* ≡*E*(*2*)] **by** *blast*
qed

AOT-theorem *eq−num*:*4*:
‹*A*!#*G* & ∀ *F* (#*G*[*F*] ≡ [λ*z* **𝓐**[*F*]*z*] $\approx_E$ [λ*z* **𝓐**[*G*]*z*])›
by (*auto intro*!: &*I eq−num*:*3*[*THEN numbers*[*THEN* $\equiv_{df}$*E*],
THEN &*E*(*1*), *THEN* &*E*(*1*)]
eq−num:*3*[*THEN numbers*[*THEN* $\equiv_{df}$*E*], *THEN* &*E*(*2*)])

AOT-theorem *eq−num*:*5*: ‹#*G*[*G*]›
by (*auto intro*!: *eq−num*:*4*[*THEN* &*E*(*2*), *THEN* ∀ *E*(*2*), *THEN* ≡*E*(*2*)]
eq−part:*1*[*unvarify F*] *simp*: *cqt*:*2*)

AOT-theorem *eq−num*:*6*: ‹*Numbers*(*x*, *G*) → *NaturalCardinal*(*x*)›
proof(*rule* →*I*)
AOT-have *act-den*: ‹$\vdash_\Box$ [λ*z* **𝓐**[*F*]*z*]↓› **for** *F*

by *cqt*:*2*
AOT-obtain *F* **where** ‹*Rigidifies*(*F*, *G*)›
by (*metis instantiation rigid−der*:*3*)
AOT-hence *ϑ*: ‹*Rigid*(*F*)› **and** ‹∀ *x*([*F*]*x* ≡ [*G*]*x*)›
using *df−rigid−rel*:*2*[*THEN* $\equiv_{df}$*E*, *THEN* &*E*(*2*)]
df−rigid−rel:*2*[*THEN* $\equiv_{df}$*E*, *THEN* &*E*(*1*)]
by *blast*+
AOT-hence ‹*F* $\equiv_E$ *G*›
by (*auto intro*!: *eqE*[*THEN* $\equiv_{df}$*I*] &*I cqt*:*2 GEN* →*I elim*: ∀ *E*(*2*))
moreover AOT-assume ‹*Numbers*(*x*, *G*)›
ultimately AOT-have ‹*Numbers*(*x*, *F*)›
using *num−tran*:*3*[*THEN* →*E*, *THEN* ≡*E*(*2*)] **by** *blast*
moreover AOT-have ‹*F* $\approx_E$ [*λz* **𝓐**[*F*]*z*]›
using *ϑ approx−nec*:*1* →*E* **by** *blast*
ultimately AOT-have ‹*Numbers*(*x*, [*λz* **𝓐**[*F*]*z*])›
using *num−tran*:*1*[*unvarify H*, *OF act-den*, *THEN* →*E*, *THEN* ≡*E*(*1*)] **by** *blast*
AOT-hence ‹*x* = #*F*›
using *eq−num*:*2*[*THEN* ≡*E*(*1*)] **by** *blast*
AOT-hence ‹∃ *F x* = #*F*›
by (*rule* ∃ *I*)
AOT-thus ‹*NaturalCardinal*(*x*)›
using *card*[*THEN* $\equiv_{df}$*I*] **by** *blast*
qed

AOT-theorem *eq−df−num*: ‹∃ *G* (*x* = #*G*) ≡ ∃ *G* (*Numbers*(*x*,*G*))›
proof(*safe intro*!: ≡*I* →*I*)
AOT-assume ‹∃ *G* (*x* = #*G*)›
then AOT-obtain *P* **where** ‹*x* = #*P*›
using ∃ *E*[*rotated*] **by** *blast*
AOT-hence ‹*Numbers*(*x*,[*λz* **𝓐**[*P*]*z*])›
using *eq−num*:*2*[*THEN* ≡*E*(*2*)] **by** *blast*
moreover AOT-have ‹[*λz* **𝓐**[*P*]*z*]↓› **by** *cqt*:*2*
ultimately AOT-show ‹∃ *G*(*Numbers*(*x*,*G*))› **by** (*rule* ∃ *I*)
next
AOT-assume ‹∃ *G* (*Numbers*(*x*,*G*))›
then AOT-obtain *Q* **where** ‹*Numbers*(*x*,*Q*)›
using ∃ *E*[*rotated*] **by** *blast*
AOT-hence ‹*NaturalCardinal*(*x*)›
using *eq−num*:*6*[*THEN* →*E*] **by** *blast*
AOT-thus ‹∃ *G* (*x* = #*G*)›
using *card*[*THEN* $\equiv_{df}$*E*] **by** *blast*
qed

AOT-theorem *card−en*: ‹*NaturalCardinal*(*x*) → ∀ *F*(*x*[*F*] ≡ *x* = #*F*)›
proof(*rule* →*I*; *rule GEN*)
AOT-have *act-den*: ‹$\vdash_\Box$ [*λz* **𝓐**[*F*]*z*]↓› **for** *F* **by** *cqt*:*2*
fix *F*
AOT-assume ‹*NaturalCardinal*(*x*)›
AOT-hence ‹∃ *F x* = #*F*›
using *card*[*THEN* $\equiv_{df}$*E*] **by** *blast*
then AOT-obtain *P* **where** *x-def*: ‹*x* = #*P*›
using ∃ *E*[*rotated*] **by** *blast*
AOT-hence *num-x-act-P*: ‹*Numbers*(*x*,[*λz* **𝓐**[*P*]*z*])›
using *eq−num*:*2*[*THEN* ≡*E*(*2*)] **by** *blast*
AOT-have ‹#*P*[*F*] ≡ [*λz* **𝓐**[*F*]*z*] $\approx_E$ [*λz* **𝓐**[*P*]*z*]›
using *eq−num*:*4*[*THEN* &*E*(*2*), *THEN* ∀ *E*(*2*)] **by** *blast*
AOT-hence ‹*x*[*F*] ≡ [*λz* **𝓐**[*F*]*z*] $\approx_E$ [*λz* **𝓐**[*P*]*z*]›
using *x-def*[*symmetric*] *rule*=*E* **by** *fast*
also AOT-have ‹. . . ≡ *Numbers*(*x*, [*λz* **𝓐**[*F*]*z*])›
using *num−tran*:*1*[*unvarify G H*, *OF act-den*, *OF act-den*]
using *num−tran*:*2*[*unvarify G H*, *OF act-den*, *OF act-den*]
by (*metis* &*I deduction−theorem* ≡*I* ≡*E*(*2*) *num-x-act-P*)
also AOT-have ‹. . . ≡ *x* = #*F*›

using *eq−num:2* **by** *blast*
finally AOT-show ‹$x[F] \equiv x = \#F$›.
qed

AOT-theorem *0F:1*: ‹$\neg\exists u\ [F]u \equiv Numbers(0, F)$›
proof −
AOT-have *unotEu-act-ord*: ‹$\neg\exists v[\lambda x\ O!x\ \&\ \mathcal{A}x \neq_E x]v$›
proof(*rule raa−cor:2*)
AOT-assume ‹$\exists v[\lambda x\ O!x\ \&\ \mathcal{A}x \neq_E x]v$›
then AOT-obtain y **where** ‹$[\lambda x\ O!x\ \&\ \mathcal{A}x \neq_E x]y$›
using $\exists E$[*rotated*] $\&E$ **by** *blast*
AOT-hence *0*: ‹$O!y\ \&\ \mathcal{A}y \neq_E y$›
by (*rule* $\beta{\rightarrow}C(1)$)
AOT-have ‹$\mathcal{A}\neg(y =_E y)$›
apply (*AOT-subst* ‹$\neg(y =_E y)$› ‹$y \neq_E y$›)
apply (*meson* $\equiv E(2)$ *Commutativity of* $\equiv$ *thm−neg=E*)
by (*fact 0*[*THEN* $\&E(2)$])
AOT-hence ‹$\neg(y =_E y)$›
by (*metis* $\neg\neg I$ *Act−Sub:1 id−act2:1* $\equiv E(4)$)
moreover AOT-have ‹$y =_E y$›
by (*metis 0*[*THEN* $\&E(1)$] *ord=Eequiv:1* $\rightarrow E$)
ultimately AOT-show ‹$p\ \&\ \neg p$› **for** p
by (*metis raa−cor:3*)
qed
AOT-have ‹$Numbers(0, [\lambda y\ \mathcal{A}[\lambda x\ O!x\ \&\ x \neq_E x]y])$›
apply (*rule* $=_{df}I(2)$[*OF zero:1*])
apply (*rule num−def:2*[*unvarify G*]; *cqt:2*)
apply (*rule eq−num:3*[*unvarify G*])
by *cqt:2*[*lambda*]
AOT-hence *numbers0*: ‹$Numbers(0, [\lambda x\ [O!]x\ \&\ \mathcal{A}x \neq_E x])$›
proof (*rule num−tran:3*[*unvarify x G H*, *THEN* $\rightarrow E$, *THEN* $\equiv E(1)$, *rotated 4*])
AOT-show ‹$[\lambda y\ \mathcal{A}[\lambda x\ O!x\ \&\ x \neq_E x]y] \equiv_E [\lambda x\ [O!]x\ \&\ \mathcal{A}x \neq_E x]$›
proof (*safe intro*!: *eqE*[*THEN* $\equiv_{df}I$] $\&I$ *Ordinary.GEN* $\rightarrow I$ *cqt:2*)
fix u
AOT-have ‹$[\lambda y\ \mathcal{A}[\lambda x\ O!x\ \&\ x \neq_E x]y]u \equiv \mathcal{A}[\lambda x\ O!x\ \&\ x \neq_E x]u$›
by (*rule beta−C−meta*[*THEN* $\rightarrow E$]; *cqt:2*[*lambda*])
also AOT-have ‹$\ldots \equiv \mathcal{A}(O!u\ \&\ u \neq_E u)$›
apply (*AOT-subst* ‹$[\lambda x\ O!x\ \&\ x \neq_E x]u$› ‹$O!u\ \&\ u \neq_E u$›)
apply (*rule beta−C−meta*[*THEN* $\rightarrow E$]; *cqt:2*[*lambda*])
by (*simp add*: *oth−class−taut:3:a*)
also AOT-have ‹$\ldots \equiv (\mathcal{A}O!u\ \&\ \mathcal{A}u \neq_E u)$›
by (*simp add*: *Act−Basic:2*)
also AOT-have ‹$\ldots \equiv (O!u\ \&\ \mathcal{A}u \neq_E u)$›
by (*metis Ordinary.*ψ $\&I$ $\&E(2)$ $\rightarrow I$ $\equiv I$ $\equiv E(1)$ *oa−facts:7*)
also AOT-have ‹$\ldots \equiv [\lambda x\ [O!]x\ \&\ \mathcal{A}x \neq_E x]u$›
by (*rule beta−C−meta*[*THEN* $\rightarrow E$, *symmetric*]; *cqt:2*[*lambda*])
finally AOT-show ‹$[\lambda y\ \mathcal{A}[\lambda x\ O!x\ \&\ x \neq_E x]y]u \equiv [\lambda x\ [O!]x\ \&\ \mathcal{A}x \neq_E x]u$›.
qed
qed(*fact zero:2* | *cqt:2*)+
show *?thesis*
proof(*safe intro*!: $\equiv I$ $\rightarrow I$)
AOT-assume ‹$\neg\exists u\ [F]u$›
moreover AOT-have ‹$\neg\exists v\ [\lambda x\ [O!]x\ \&\ \mathcal{A}x \neq_E x]v$›
using *unotEu-act-ord*.
ultimately AOT-have *0*: ‹$F \approx_E [\lambda x\ [O!]x\ \&\ \mathcal{A}x \neq_E x]$›
by (*rule empty−approx:1*[*unvarify H*, *THEN* $\rightarrow E$, *rotated*, *OF* $\&I$]) *cqt:2*
AOT-thus ‹$Numbers(0, F)$›
by (*rule num−tran:1*[*unvarify x H*, *THEN* $\rightarrow E$,
THEN $\equiv E(2)$, *rotated*, *rotated*])
(*fact zero:2 numbers0* | *cqt:2*[*lambda*])+
next
AOT-assume ‹$Numbers(0, F)$›
AOT-hence *1*: ‹$F \approx_E [\lambda x\ [O!]x\ \&\ \mathcal{A}x \neq_E x]$›

by (*rule num−tran*:*2*[*unvarify x H*, *THEN* →*E*, *rotated 2*, *OF* &*I*])
(*fact numbers0 zero*:*2* | *cqt*:*2*[*lambda*])+
AOT-show ‹¬∃ *u* [*F*]*u*›
proof(*rule raa−cor*:*2*)
AOT-have *0*: ‹[λ*x* [*O*!]*x* & 𝓐*x* ≠*E* *x*]↓› **by** *cqt*:*2*[*lambda*]
AOT-assume ‹∃ *u* [*F*]*u*›
AOT-hence ‹¬(*F* ≈*E* [λ*x* [*O*!]*x* & 𝓐*x* ≠*E* *x*])›
by (*rule empty−approx*:*2*[*unvarify H*, *OF 0*, *THEN* →*E*, *OF* &*I*])
(*rule unotEu-act-ord*)
AOT-thus ‹*F* ≈*E* [λ*x* [*O*!]*x* & 𝓐*x* ≠*E* *x*] & ¬(*F* ≈*E* [λ*x* [*O*!]*x* & 𝓐*x* ≠*E* *x*])›
using *1* &*I* **by** *blast*
qed
qed
qed

AOT-theorem *0F*:*2*: ‹¬∃ *u* 𝓐[*F*]*u* ≡ #*F* = *0*›
proof(*rule* ≡*I*; *rule* →*I*)
AOT-assume *0*: ‹¬∃ *u* 𝓐[*F*]*u*›
AOT-have ‹¬∃ *u* [λ*z* 𝓐[*F*]*z*]*u*›
proof(*rule raa−cor*:*2*)
AOT-assume ‹∃ *u* [λ*z* 𝓐[*F*]*z*]*u*›
then AOT-obtain *u* **where** ‹[λ*z* 𝓐[*F*]*z*]*u*›
using *Ordinary*.∃ *E*[*rotated*] **by** *blast*
AOT-hence ‹𝓐[*F*]*u*›
by (*metis betaC*:*1*:*a*)
AOT-hence ‹∃ *u* 𝓐[*F*]*u*›
by (*rule Ordinary*.∃ *I*)
AOT-thus ‹∃ *u* 𝓐[*F*]*u* & ¬∃ *u* 𝓐[*F*]*u*›
using *0* &*I* **by** *blast*
qed
AOT-hence ‹*Numbers*(*0*,[λ*z* 𝓐[*F*]*z*])›
by (*safe intro*!: *0F*:*1*[*unvarify F*, *THEN* ≡*E*(*1*)]) *cqt*:*2*
AOT-hence ‹*0* = #*F*›
by (*rule eq−num*:*2*[*unvarify x*, *OF zero*:*2*, *THEN* ≡*E*(*1*)])
AOT-thus ‹#*F* = *0*› **using** *id-sym* **by** *blast*
next
AOT-assume ‹#*F* = *0*›
AOT-hence ‹*0* = #*F*› **using** *id-sym* **by** *blast*
AOT-hence ‹*Numbers*(*0*,[λ*z* 𝓐[*F*]*z*])›
by (*rule eq−num*:*2*[*unvarify x*, *OF zero*:*2*, *THEN* ≡*E*(*2*)])
AOT-hence *0*: ‹¬∃ *u* [λ*z* 𝓐[*F*]*z*]*u*›
by (*safe intro*!: *0F*:*1*[*unvarify F*, *THEN* ≡*E*(*2*)]) *cqt*:*2*
AOT-show ‹¬∃ *u* 𝓐[*F*]*u*›
proof(*rule raa−cor*:*2*)
AOT-assume ‹∃ *u* 𝓐[*F*]*u*›
then AOT-obtain *u* **where** ‹𝓐[*F*]*u*›
using *Ordinary*.∃ *E*[*rotated*] **by** *meson*
AOT-hence ‹[λ*z* 𝓐[*F*]*z*]*u*›
by (*auto intro*!: β←*C cqt*:*2*)
AOT-hence ‹∃ *u* [λ*z* 𝓐[*F*]*z*]*u*›
using *Ordinary*.∃ *I* **by** *blast*
AOT-thus ‹∃ *u* [λ*z* 𝓐[*F*]*z*]*u* & ¬∃ *u* [λ*z* 𝓐[*F*]*z*]*u*›
using &*I 0* **by** *blast*
qed
qed

AOT-theorem *0F*:*3*: ‹□¬∃ *u* [*F*]*u* → #*F* = *0*›
proof(*rule* →*I*)
AOT-assume ‹□¬∃ *u* [*F*]*u*›
AOT-hence *0*: ‹¬◇∃ *u* [*F*]*u*›
using *KBasic2*:*1* ≡*E*(*1*) **by** *blast*
AOT-have ‹¬∃ *u* [λ*z* 𝓐[*F*]*z*]*u*›
proof(*rule raa−cor*:*2*)

```
    AOT-assume ‹∃ u [λz 𝓐[F]z]u›
    then AOT-obtain u where ‹[λz 𝓐[F]z]u›
      using Ordinary.∃ E[rotated] by blast
    AOT-hence ‹𝓐[F]u›
      by (metis betaC:1:a)
    AOT-hence ‹◇[F]u›
      by (metis Act−Sub:3 →E)
    AOT-hence ‹∃ u ◇[F]u›
      by (rule Ordinary.∃ I)
    AOT-hence ‹◇∃ u [F]u›
      using Ordinary.res−var−bound−reas[CBF◇][THEN →E] by blast
    AOT-thus ‹◇∃ u [F]u & ¬◇∃ u [F]u›
      using 0 &I by blast
  qed
  AOT-hence ‹Numbers(0,[λz 𝓐[F]z])›
    by (safe intro!: 0F:1[unvarify F, THEN ≡E(1)]) cqt:2
  AOT-hence ‹0 = #F›
    by (rule eq−num:2[unvarify x, OF zero:2, THEN ≡E(1)])
  AOT-thus ‹#F = 0› using id-sym by blast
qed

AOT-theorem 0F:4: ‹w ⊨ ¬∃ u [F]u ≡ #[F]_w = 0›
proof (rule rule−id−df:2:b[OF w−index, where τ₁τₙ=(-,-), simplified])
  AOT-show ‹[λx₁...xₙ w ⊨ [F]x₁...xₙ]↓›
    by (simp add: w−rel:3)
next
  AOT-show ‹w ⊨ ¬∃ u [F]u ≡ #[λx w ⊨ [F]x] = 0›
  proof (rule ≡I; rule →I)
    AOT-assume ‹w ⊨ ¬∃ u [F]u›
    AOT-hence 0: ‹¬w ⊨ ∃ u [F]u›
      using coherent:1[unvarify p, OF log−prop−prop:2, THEN ≡E(1)] by blast
    AOT-have ‹¬∃ u 𝓐[λx w ⊨ [F]x]u›
    proof(rule raa−cor:2)
      AOT-assume ‹∃ u 𝓐[λx w ⊨ [F]x]u›
      then AOT-obtain u where ‹𝓐[λx w ⊨ [F]x]u›
        using Ordinary.∃ E[rotated] by meson
      AOT-hence ‹𝓐w ⊨ [F]u›
        by (AOT-subst (reverse) ‹w ⊨ [F]u› ‹[λx w ⊨ [F]x]u›;
            safe intro!: beta−C−meta[THEN →E] w−rel:1[THEN →E])
           cqt:2
      AOT-hence 1: ‹w ⊨ [F]u›
        using rigid−truth−at:4[unvarify p, OF log−prop−prop:2, THEN ≡E(1)]
        by blast
      AOT-have ‹□([F]u → ∃ u [F]u)›
        using Ordinary.∃ I →I RN by simp
      AOT-hence ‹w ⊨ ([F]u → ∃ u [F]u)›
        using fund:2[unvarify p, OF log−prop−prop:2, THEN ≡E(1)]
              PossibleWorld.∀ E by fast
      AOT-hence ‹w ⊨ ∃ u [F]u›
        using 1 conj−dist−w:2[unvarify p q, OF log−prop−prop:2,
                              OF log−prop−prop:2, THEN ≡E(1),
                              THEN →E] by blast
      AOT-thus ‹w ⊨ ∃ u [F]u & ¬w ⊨ ∃ u [F]u›
        using 0 &I by blast
    qed
    AOT-thus ‹#[λx w ⊨ [F]x] = 0›
      by (safe intro!: 0F:2[unvarify F, THEN ≡E(1)] w−rel:1[THEN →E])
         cqt:2
  next
    AOT-assume ‹#[λx w ⊨ [F]x] = 0›
    AOT-hence 0: ‹¬∃ u 𝓐[λx w ⊨ [F]x]u›
      by (safe intro!: 0F:2[unvarify F, THEN ≡E(2)] w−rel:1[THEN →E])
         cqt:2
```

```
    AOT-have ‹¬w ⊨ ∃ u [F]u›
    proof (rule raa−cor:2)
      AOT-assume ‹w ⊨ ∃ u [F]u›
      AOT-hence ‹∃ x w ⊨ (O!x & [F]x)›
        using conj−dist−w:6[THEN ≡E(1)] by fast
      then AOT-obtain x where ‹w ⊨ (O!x & [F]x)›
        using ∃ E[rotated] by blast
      AOT-hence ‹w ⊨ O!x› and Fx-in-w: ‹w ⊨ [F]x›
        using conj−dist−w:1[unvarify p q] ≡E(1) log−prop−prop:2
              &E by blast+
      AOT-hence ‹◊O!x›
        using fund:1[unvarify p, OF log−prop−prop:2, THEN ≡E(2)]
              PossibleWorld.∃ I by simp
      AOT-hence ord-x: ‹O!x›
        using oa−facts:3[THEN →E] by blast
      AOT-have ‹𝓐w ⊨ [F]x›
        using rigid−truth−at:4[unvarify p, OF log−prop−prop:2, THEN ≡E(2)]
              Fx-in-w by blast
      AOT-hence ‹𝓐[λx w ⊨ [F]x]x›
        by (AOT-subst ‹[λx w ⊨ [F]x]x› ‹w ⊨ [F]x›;
            safe intro!: beta−C−meta[THEN →E] w−rel:1[THEN →E]) cqt:2
      AOT-hence ‹O!x & 𝓐[λx w ⊨ [F]x]x›
        using ord-x &I by blast
      AOT-hence ‹∃ x (O!x & 𝓐[λx w ⊨ [F]x]x)›
        using ∃ I by fast
      AOT-thus ‹∃ u (𝓐[λx w ⊨ [F]x]u) & ¬∃ u 𝓐[λx w ⊨ [F]x]u›
        using 0 &I by blast
    qed
    AOT-thus ‹w ⊨ ¬∃ u[F]u›
      using coherent:1[unvarify p, OF log−prop−prop:2, THEN ≡E(2)] by blast
  qed
qed

AOT-act-theorem zero=:1:
  ‹NaturalCardinal(x) → ∀ F (x[F] ≡ Numbers(x, F))›
proof(safe intro!: →I GEN)
  fix F
  AOT-assume ‹NaturalCardinal(x)›
  AOT-hence ‹∀ F (x[F] ≡ x = #F)›
    by (metis card−en →E)
  AOT-hence 1: ‹x[F] ≡ x = #F›
    using ∀ E(2) by blast
  AOT-have 2: ‹x[F] ≡ x = ιy(Numbers(y, F))›
    by (rule num−def:1[THEN =df E(1)])
       (auto simp: 1 num−uniq)
  AOT-have ‹x = ιy(Numbers(y, F)) → Numbers(x, F)›
    using y−in:1 by blast
  moreover AOT-have ‹Numbers(x, F) → x = ιy(Numbers(y, F))›
  proof(rule →I)
    AOT-assume 1: ‹Numbers(x, F)›
    moreover AOT-obtain z where z-prop: ‹∀ y (Numbers(y, F) → y = z)›
      using num:2[THEN uniqueness:1[THEN ≡df E]] ∃ E[rotated] &E by blast
    ultimately AOT-have ‹x = z›
      using ∀ E(2) →E by blast
    AOT-hence ‹∀ y (Numbers(y, F) → y = x)›
      using z-prop rule=E id-sym by fast
    AOT-thus ‹x = ιy(Numbers(y,F))›
      by (rule hintikka[THEN ≡E(2), OF &I, rotated])
         (fact 1)
  qed
  ultimately AOT-have ‹x = ιy(Numbers(y, F)) ≡ Numbers(x, F)›
    by (metis ≡I)
  AOT-thus ‹x[F] ≡ Numbers(x, F)›
```

using *2* **by** (*metis* ≡*E*(*5*))
qed

AOT-act-theorem *zero=:2*: ‹*0*[*F*] ≡ ¬∃ *u*[*F*]*u*›
proof –
AOT-have ‹*0*[*F*] ≡ *Numbers*(*0*, *F*)›
using *zero=:1*[*unvarify x*, *OF zero:2*, *THEN* →*E*,
OF zero−card, *THEN* ∀ *E*(*2*)].
also AOT-have ‹. . . ≡ ¬∃ *u*[*F*]*u*›
using *0F:1*[*symmetric*].
finally show *?thesis*.
qed

AOT-act-theorem *zero=:3*: ‹¬∃ *u*[*F*]*u* ≡ #*F* = *0*›
proof –
AOT-have ‹¬∃ *u*[*F*]*u* ≡ *0*[*F*]› **using** *zero=:2*[*symmetric*].
also AOT-have ‹. . . ≡ *0* = #*F*›
using *card−en*[*unvarify x*, *OF zero:2*, *THEN* →*E*,
OF zero−card, *THEN* ∀ *E*(*2*)].
also AOT-have ‹. . . ≡ #*F* = *0*›
by (*simp add*: *deduction−theorem id-sym* ≡*I*)
finally show *?thesis*.
qed

AOT-define *Hereditary* :: ‹τ ⇒ τ ⇒ φ› (‹*Hereditary*′(-,-′)›)
hered:1:
‹*Hereditary*(*F*, *R*) $\equiv_{df}$ *R*↓ & *F*↓ & ∀ *x*∀ *y*([*R*]*xy* → ([*F*]*x* → [*F*]*y*))›

AOT-theorem *hered:2*:
‹[λ*xy* ∀ *F*((∀ *z*([*R*]*xz* → [*F*]*z*) & *Hereditary*(*F*,*R*)) → [*F*]*y*)]↓›
by *cqt:2*[*lambda*]

AOT-define *StrongAncestral* :: ‹τ ⇒ Π› (‹-*›)
ances−df:
‹*R** $=_{df}$ [λ*xy* ∀ *F*((∀ *z*([*R*]*xz* → [*F*]*z*) & *Hereditary*(*F*,*R*)) → [*F*]*y*)]›

AOT-theorem *ances*:
‹[*R**]*xy* ≡ ∀ *F*((∀ *z*([*R*]*xz* → [*F*]*z*) & *Hereditary*(*F*,*R*)) → [*F*]*y*)›
apply (*rule* $=_{df}$*I*(*1*)[*OF ances−df*])
apply *cqt:2*[*lambda*]
apply (*rule beta−C−meta*[*THEN* →*E*, *OF hered:2*, *unvarify* $\nu_1\nu_n$,
where τ=‹(-,-)›, *simplified*])
by (*simp add*: &*I ex:1:a prod-denotesI rule−ui:3*)

AOT-theorem *anc−her:1*:
‹[*R*]*xy* → [*R**]*xy*›
proof (*safe intro*!: →*I ances*[*THEN* ≡*E*(*2*)] *GEN*)
fix *F*
AOT-assume ‹∀ *z* ([*R*]*xz* → [*F*]*z*) & *Hereditary*(*F*, *R*)›
AOT-hence ‹[*R*]*xy* → [*F*]*y*›
using ∀ *E*(*2*) &*E* **by** *blast*
moreover AOT-assume ‹[*R*]*xy*›
ultimately AOT-show ‹[*F*]*y*›
using →*E* **by** *blast*
qed

AOT-theorem *anc−her:2*:
‹([*R**]*xy* & ∀ *z*([*R*]*xz* → [*F*]*z*) & *Hereditary*(*F*,*R*)) → [*F*]*y*›
proof(*rule* →*I*; (*frule* &*E*(*1*); *drule* &*E*(*2*))+)
AOT-assume ‹[*R**]*xy*›
AOT-hence ‹(∀ *z*([*R*]*xz* → [*F*]*z*) & *Hereditary*(*F*,*R*)) → [*F*]*y*›
using *ances*[*THEN* ≡*E*(*1*)] ∀ *E*(*2*) **by** *blast*
moreover AOT-assume ‹∀ *z*([*R*]*xz* → [*F*]*z*)›

```
  moreover AOT-assume ‹Hereditary(F,R)›
  ultimately AOT-show ‹[F]y›
    using →E &I by blast
qed

AOT-theorem anc−her:3:
  ‹([F]x & [R*]xy & Hereditary(F, R)) → [F]y›
proof(rule →I; (frule &E(1); drule &E(2))+)
  AOT-assume 1: ‹[F]x›
  AOT-assume 2: ‹Hereditary(F, R)›
  AOT-hence 3: ‹∀ x ∀ y ([R]xy → ([F]x → [F]y))›
    using hered:1[THEN ≡df E] &E by blast
  AOT-have ‹∀ z ([R]xz → [F]z)›
  proof (rule GEN; rule →I)
    fix z
    AOT-assume ‹[R]xz›
    moreover AOT-have ‹[R]xz → ([F]x → [F]z)›
      using 3 ∀ E(2) by blast
    ultimately AOT-show ‹[F]z›
      using 1 →E by blast
  qed
  moreover AOT-assume ‹[R*]xy›
  ultimately AOT-show ‹[F]y›
    by (auto intro!: 2 anc−her:2[THEN →E] &I)
qed

AOT-theorem anc−her:4: ‹([R]xy & [R*]yz) → [R*]xz›
proof(rule →I; frule &E(1); drule &E(2))
  AOT-assume 0: ‹[R*]yz› and 1: ‹[R]xy›
  AOT-show ‹[R*]xz›
  proof(safe intro!: ances[THEN ≡E(2)] GEN &I →I;
                    frule &E(1); drule &E(2))
    fix F
    AOT-assume ‹∀ z ([R]xz → [F]z)›
    AOT-hence 1: ‹[F]y›
      using 1 ∀ E(2) →E by blast
    AOT-assume 2: ‹Hereditary(F,R)›
    AOT-show ‹[F]z›
      by (rule anc−her:3[THEN →E]; auto intro!: &I 1 2 0)
  qed
qed

AOT-theorem anc−her:5: ‹[R*]xy → ∃ z [R]zy›
proof (rule →I)
  AOT-have 0: ‹[λy ∃ x [R]xy]↓› by cqt:2
  AOT-assume 1: ‹[R*]xy›
  AOT-have ‹[λy∃ x [R]xy]y›
  proof(rule anc−her:2[unvarify F, OF 0, THEN →E];
        safe intro!: &I GEN →I hered:1[THEN ≡df I] cqt:2 0)
    AOT-show ‹[R*]xy› using 1.
  next
    fix z
    AOT-assume ‹[R]xz›
    AOT-hence ‹∃ x [R]xz› by (rule ∃ I)
    AOT-thus ‹[λy∃ x [R]xy]z›
      by (auto intro!: β←C(1) cqt:2)
  next
    fix x y
    AOT-assume ‹[R]xy›
    AOT-hence ‹∃ x [R]xy› by (rule ∃ I)
    AOT-thus ‹[λy ∃ x [R]xy]y›
      by (auto intro!: β←C(1) cqt:2)
  qed
```

AOT-thus ‹∃ *z* [*R*]*zy*›
  by (*rule* $\beta{\to}C(1)$)
qed

AOT-theorem *anc−her:6*: ‹([R^*]*xy* & [R^*]*yz*) → [R^*]*xz*›
proof (*rule* →*I*; *frule* &*E*(*1*); *drule* &*E*(*2*))
  AOT-assume ‹[R^*]*xy*›
  AOT-hence ϑ: ‹∀ *z* ([*R*]*xz* → [*F*]*z*) & *Hereditary*(*F*,*R*) → [*F*]*y*› **for** *F*
    using ∀ *E*(*2*) *ances*[*THEN* ≡*E*(*1*)] **by** *blast*
  AOT-assume ‹[R^*]*yz*›
  AOT-hence ξ: ‹∀ *z* ([*R*]*yz* → [*F*]*z*) & *Hereditary*(*F*,*R*) → [*F*]*z*› **for** *F*
    using ∀ *E*(*2*) *ances*[*THEN* ≡*E*(*1*)] **by** *blast*
  AOT-show ‹[R^*]*xz*›
  proof (*rule ances*[*THEN* ≡*E*(*2*)]; *safe intro*!: *GEN* →*I*)
    fix *F*
    AOT-assume ζ: ‹∀ *z* ([*R*]*xz* → [*F*]*z*) & *Hereditary*(*F*,*R*)›
    AOT-show ‹[*F*]*z*›
    proof (*rule* ξ[*THEN* →*E*, *OF* &*I*])
      AOT-show ‹*Hereditary*(*F*,*R*)›
        using ζ[*THEN* &*E*(*2*)]**.**
    next
      AOT-show ‹∀ *z* ([*R*]*yz* → [*F*]*z*)›
      proof(*rule GEN*; *rule* →*I*)
        fix *z*
        AOT-assume ‹[*R*]*yz*›
        moreover AOT-have ‹[*F*]*y*›
          using ϑ[*THEN* →*E*, *OF* ζ]**.**
        ultimately AOT-show ‹[*F*]*z*›
          using ζ[*THEN* &*E*(*2*), *THEN hered:1*[*THEN* $\equiv_{df}E$],
                  THEN &*E*(*2*), *THEN* ∀ *E*(*2*), *THEN* ∀ *E*(*2*),
                  THEN →*E*, *THEN* →*E*]
          by *blast*
      qed
    qed
  qed
qed

AOT-define *OneToOne* :: ‹τ ⇒ φ› (‹*1*−*1*′(-′)›)
  df−1−1:1: ‹*1*−*1*(*R*) $\equiv_{df}$ *R*↓ & ∀ *x*∀ *y*∀ *z*([*R*]*xz* & [*R*]*yz* → *x* = *y*)›

AOT-define *RigidOneToOne* :: ‹τ ⇒ φ› (‹$Rigid_{1-1}$′(-′)›)
  df−1−1:2: ‹$Rigid_{1-1}$(*R*) $\equiv_{df}$ *1*−*1*(*R*) & *Rigid*(*R*)›

AOT-theorem *df−1−1:3*: ‹$Rigid_{1-1}$(*R*) → □*1*−*1*(*R*)›
proof(*rule* →*I*)
  AOT-assume ‹$Rigid_{1-1}$(*R*)›
  AOT-hence ‹*1*−*1*(*R*)› **and** *RigidR*: ‹*Rigid*(*R*)›
    using *df−1−1:2*[*THEN* $\equiv_{df}E$] &*E* **by** *blast*+
  AOT-hence *1*: ‹[*R*]*xz* & [*R*]*yz* → *x* = *y*› **for** *x y z*
    using *df−1−1:1*[*THEN* $\equiv_{df}E$] &*E*(*2*) ∀ *E*(*2*) **by** *blast*
  AOT-have *1*: ‹[*R*]*xz* & [*R*]*yz* → □*x* = *y*› **for** *x y z*
    by (*AOT-subst* (*reverse*) ‹□*x* = *y*› ‹*x* = *y*›)
      (*auto simp*: *1 id−nec:2* ≡*I qml:2*[*axiom-inst*])
  AOT-have ‹□∀ x_1...∀ x_n ([*R*]x_1...x_n → □[*R*]x_1...x_n)›
    using *df−rigid−rel:1*[*THEN* $\equiv_{df}E$, *OF RigidR*] &*E* **by** *blast*
  AOT-hence ‹∀ x_1...∀ x_n □([*R*]x_1...x_n → □[*R*]x_1...x_n)›
    using *CBF*[*THEN* →*E*] **by** *fast*
  AOT-hence ‹∀ x_1∀ x_2 □([*R*]x_1x_2 → □[*R*]x_1x_2)›
    using *tuple-forall*[*THEN* $\equiv_{df}E$] **by** *blast*
  AOT-hence ‹□([*R*]*xy* → □[*R*]*xy*)› **for** *x y*
    using ∀ *E*(*2*) **by** *blast*
  AOT-hence ‹□(([*R*]*xz* → □[*R*]*xz*) & ([*R*]*yz* → □[*R*]*yz*))› **for** *x y z*
    by (*metis KBasic:3* &*I* ≡*E*(*3*) *raa−cor:3*)

```
  moreover AOT-have ‹□(([R]xz → □[R]xz) & ([R]yz → □[R]yz)) →
                   □(([R]xz & [R]yz) → □([R]xz & [R]yz))› for x y z
    by (rule RM) (metis →I KBasic:3 &I &E(1) &E(2) ≡E(2) →E)
  ultimately AOT-have 2: ‹□(([R]xz & [R]yz) → □([R]xz & [R]yz))› for x y z
    using →E by blast
  AOT-hence 3: ‹□([R]xz & [R]yz → x = y)› for x y z
    using sc−eq−box−box:6[THEN →E, THEN →E, OF 2, OF 1] by blast
  AOT-hence 4: ‹□∀ x∀ y∀ z([R]xz & [R]yz → x = y)›
    by (safe intro!: GEN BF[THEN →E] 3)
  AOT-thus ‹□1−1(R)›
    by (AOT-subst-thm df−1−1:1[THEN ≡Df, THEN ≡S(1),
                            OF cqt:2[const-var][axiom-inst]])
qed

AOT-theorem df−1−1:4: ‹∀ R(Rigid_{1−1}(R) → □Rigid_{1−1}(R))›
proof(rule GEN;rule →I)
AOT-modally-strict {
  fix R
      AOT-assume 0: ‹Rigid_{1−1}(R)›
      AOT-hence 1: ‹R↓›
        by (meson ≡_{df}E &E(1) df−1−1:1 df−1−1:2)
      AOT-hence 2: ‹□R↓›
        using exist−nec →E by blast
      AOT-have 4: ‹□1−1(R)›
        using df−1−1:3[unvarify R, OF 1, THEN →E, OF 0].
      AOT-have ‹Rigid(R)›
        using 0 ≡_{df}E[OF df−1−1:2] &E by blast
      AOT-hence ‹□∀ x_1...∀ x_n ([R]x_1...x_n → □[R]x_1...x_n)›
        using  df−rigid−rel:1[THEN ≡_{df}E] &E by blast
      AOT-hence ‹□□∀ x_1...∀ x_n ([R]x_1...x_n → □[R]x_1...x_n)›
        by (metis S5Basic:6 ≡E(1))
      AOT-hence ‹□Rigid(R)›
        apply (AOT-subst-def df−rigid−rel:1)
        using 2 KBasic:3 ≡S(2) ≡E(2) by blast
      AOT-thus ‹□Rigid_{1−1}(R)›
        apply (AOT-subst-def df−1−1:2)
        using 4 KBasic:3 ≡S(2) ≡E(2) by blast
}
qed

AOT-define InDomainOf :: ‹τ ⇒ τ ⇒ φ› (‹InDomainOf'(-,-')›)
  df−1−1:5: ‹InDomainOf(x, R) ≡_{df} ∃ y [R]xy›

AOT-register-rigid-restricted-type
  RigidOneToOneRelation: ‹Rigid_{1−1}(Π)›
proof
  AOT-modally-strict {
    AOT-show ‹∃ α Rigid_{1−1}(α)›
    proof (rule ∃ I(1)[where τ=‹«(=_E)»›])
      AOT-show ‹Rigid_{1−1}((=_E))›
      proof (safe intro!: df−1−1:2[THEN ≡_{df}I] &I df−1−1:1[THEN ≡_{df}I]
                       GEN →I df−rigid−rel:1[THEN ≡_{df}I] =E[denotes])
        fix x y z
        AOT-assume ‹x =_E z & y =_E z›
        AOT-thus ‹x = y›
          by (metis rule=E &E(1) Conjunction Simplification(2)
                  =E−simple:2 id-sym →E)
      next
        AOT-have ‹∀ x∀ y □(x =_E y → □x =_E y)›
        proof(rule GEN; rule GEN)
          AOT-show ‹□(x =_E y → □x =_E y)› for x y
            by (meson RN deduction−theorem id−nec3:1 ≡E(1))
        qed
```

AOT-hence ‹$\forall x_1 ... \forall x_n \; \Box([(=_E)]x_1...x_n \rightarrow \Box[(=_E)]x_1...x_n)$›
  by (*rule tuple-forall*[*THEN* $\equiv_{df}I$])
AOT-thus ‹$\Box\forall x_1 ... \forall x_n \; ([(=_E)]x_1...x_n \rightarrow \Box[(=_E)]x_1...x_n)$›
  using *BF*[*THEN* $\rightarrow E$] **by** *fast*
qed
qed(*fact* $=E$[*denotes*])
}
next
AOT-modally-strict {
AOT-show ‹$Rigid_{1-1}(\Pi) \rightarrow \Pi\downarrow$› **for** Π
proof(*rule* $\rightarrow I$)
AOT-assume ‹$Rigid_{1-1}(\Pi)$›
AOT-hence ‹*1−1*(Π)›
  using *df−1−1*:*2*[*THEN* $\equiv_{df}E$] &*E* **by** *blast*
AOT-thus ‹$\Pi\downarrow$›
  using *df−1−1*:*1*[*THEN* $\equiv_{df}E$] &*E* **by** *blast*
qed
}
next
AOT-modally-strict {
AOT-show ‹$\forall F(Rigid_{1-1}(F) \rightarrow \Box Rigid_{1-1}(F))$›
  by (*safe intro*!: *GEN df−1−1*:*4*[*THEN* $\forall E(2)$])
}
qed
AOT-register-variable-names
RigidOneToOneRelation: $\mathcal{R}$ $\mathcal{S}$

AOT-define *IdentityRestrictedToDomain* :: ‹$\tau \Rightarrow \Pi$› (‹′(=-′)›)
id−d−R: ‹$(=_{\mathcal{R}}) =_{df} [\lambda xy \; \exists z \; ([\mathcal{R}]xz \; \& \; [\mathcal{R}]yz)]$›

syntax *-AOT-id-d-R-infix* :: ‹$\tau \Rightarrow \tau \Rightarrow \tau \Rightarrow \varphi$› (‹(- =-/ -)› [*50*, *51*, *51*] *50*)
translations
-AOT-id-d-R-infix κ Π κ' ==
CONST AOT-exe (*CONST IdentityRestrictedToDomain* Π) (κ,κ')

AOT-theorem *id−R−thm*:*1*: ‹$x =_{\mathcal{R}} y \equiv \exists z \; ([\mathcal{R}]xz \; \& \; [\mathcal{R}]yz)$›
proof −
AOT-have *0*: ‹$[\lambda xy \; \exists z \; ([\mathcal{R}]xz \; \& \; [\mathcal{R}]yz)]\downarrow$› **by** *cqt*:*2*
show *?thesis*
apply (*rule* $=_{df}I(1)$[*OF id−d−R*])
apply (*fact 0*)
apply (*rule beta−C−meta*[*THEN* $\rightarrow E$, *OF 0*, *unvarify* $\nu_1\nu_n$,
  where τ=‹(-,-)›, *simplified*])
by (*simp add*: &*I ex*:*1*:*a prod-denotesI rule−ui*:*3*)
qed

AOT-theorem *id−R−thm*:*2*:
‹$x =_{\mathcal{R}} y \rightarrow (InDomainOf(x, \mathcal{R}) \; \& \; InDomainOf(y, \mathcal{R}))$›
proof(*rule* $\rightarrow I$)
AOT-assume ‹$x =_{\mathcal{R}} y$›
AOT-hence ‹$\exists z \; ([\mathcal{R}]xz \; \& \; [\mathcal{R}]yz)$›
  using *id−R−thm*:*1*[*THEN* $\equiv E(1)$] **by** *simp*
then AOT-obtain *z* **where** *z-prop*: ‹$[\mathcal{R}]xz \; \& \; [\mathcal{R}]yz$›
  using $\exists E$[*rotated*] **by** *blast*
AOT-show ‹$InDomainOf(x, \mathcal{R}) \; \& \; InDomainOf(y, \mathcal{R})$›
proof (*safe intro*!: &*I df−1−1*:*5*[*THEN* $\equiv_{df}I$])
AOT-show ‹$\exists y \; [\mathcal{R}]xy$›
  using *z-prop*[*THEN* &*E*(*1*)] $\exists I$ **by** *fast*
next
AOT-show ‹$\exists z \; [\mathcal{R}]yz$›
  using *z-prop*[*THEN* &*E*(*2*)] $\exists I$ **by** *fast*
qed
qed

AOT-theorem *id−R−thm:3*: ‹$x =_{\mathcal{R}} y \rightarrow x = y$›
proof(*rule* →*I*)
  AOT-assume ‹$x =_{\mathcal{R}} y$›
  AOT-hence ‹$\exists z$ ([$\mathcal{R}$]xz & [$\mathcal{R}$]yz)›
    using *id−R−thm:1*[*THEN* ≡*E*(*1*)] **by** *simp*
  then AOT-obtain *z* **where** *z-prop*: ‹[$\mathcal{R}$]xz & [$\mathcal{R}$]yz›
    using ∃ *E*[*rotated*] **by** *blast*
  AOT-thus ‹$x = y$›
    using *df−1−1:3*[*THEN* →*E*, *OF RigidOneToOneRelation.ψ*,
      THEN qml:2[*axiom-inst*, *THEN* →*E*],
      THEN $\equiv_{df}$*E*[*OF df−1−1:1*], *THEN* &*E*(*2*),
      THEN ∀ *E*(*2*), *THEN* ∀ *E*(*2*),
      THEN ∀ *E*(*2*), *THEN* →*E*]
     by *blast*
qed

AOT-theorem *id−R−thm:4*:
  ‹(*InDomainOf*(x, $\mathcal{R}$) ∨ *InDomainOf*(y, $\mathcal{R}$)) → ($x =_{\mathcal{R}} y \equiv x = y$)›
proof (*rule* →*I*)
  AOT-assume ‹*InDomainOf*(x, $\mathcal{R}$) ∨ *InDomainOf*(y, $\mathcal{R}$)›
  moreover {
    AOT-assume ‹*InDomainOf*(x, $\mathcal{R}$)›
    AOT-hence ‹$\exists z$ [$\mathcal{R}$]xz›
      by (*metis* $\equiv_{df}$*E df−1−1:5*)
    then AOT-obtain *z* **where** *z-prop*: ‹[$\mathcal{R}$]xz›
      using ∃ *E*[*rotated*] **by** *blast*
    AOT-have ‹$x =_{\mathcal{R}} y \equiv x = y$›
    proof(*safe intro*!: ≡*I* →*I id−R−thm:3*[*THEN* →*E*])
      AOT-assume ‹$x = y$›
      AOT-hence ‹[$\mathcal{R}$]yz›
        using *z-prop rule=E* **by** *fast*
      AOT-hence ‹[$\mathcal{R}$]xz & [$\mathcal{R}$]yz›
        using *z-prop* &*I* **by** *blast*
      AOT-hence ‹$\exists z$ ([$\mathcal{R}$]xz & [$\mathcal{R}$]yz)›
        by (*rule* ∃ *I*)
      AOT-thus ‹$x =_{\mathcal{R}} y$›
        using *id−R−thm:1* ≡*E*(*2*) **by** *blast*
    qed
  }
  moreover {
    AOT-assume ‹*InDomainOf*(y, $\mathcal{R}$)›
    AOT-hence ‹$\exists z$ [$\mathcal{R}$]yz›
      by (*metis* $\equiv_{df}$*E df−1−1:5*)
    then AOT-obtain *z* **where** *z-prop*: ‹[$\mathcal{R}$]yz›
      using ∃ *E*[*rotated*] **by** *blast*
    AOT-have ‹$x =_{\mathcal{R}} y \equiv x = y$›
    proof(*safe intro*!: ≡*I* →*I id−R−thm:3*[*THEN* →*E*])
      AOT-assume ‹$x = y$›
      AOT-hence ‹[$\mathcal{R}$]xz›
        using *z-prop rule=E id-sym* **by** *fast*
      AOT-hence ‹[$\mathcal{R}$]xz & [$\mathcal{R}$]yz›
        using *z-prop* &*I* **by** *blast*
      AOT-hence ‹$\exists z$ ([$\mathcal{R}$]xz & [$\mathcal{R}$]yz)›
        by (*rule* ∃ *I*)
      AOT-thus ‹$x =_{\mathcal{R}} y$›
        using *id−R−thm:1* ≡*E*(*2*) **by** *blast*
    qed
  }
  ultimately AOT-show ‹$x =_{\mathcal{R}} y \equiv x = y$›
    by (*metis* ∨*E*(*2*) *raa−cor:1*)
qed

AOT-theorem *id−R−thm:5*: ‹*InDomainOf*(*x*, $\mathcal{R}$) → $x =_{\mathcal{R}} x$›
proof (*rule* →*I*)
  AOT-assume ‹*InDomainOf*(*x*, $\mathcal{R}$)›
  AOT-hence ‹∃ *z* [$\mathcal{R}$]*xz*›
    by (*metis* $\equiv_{df}$*E df−1−1:5*)
  then AOT-obtain *z* **where** *z-prop*: ‹[$\mathcal{R}$]*xz*›
    using ∃ *E*[*rotated*] **by** *blast*
  AOT-hence ‹[$\mathcal{R}$]*xz* & [$\mathcal{R}$]*xz*›
    using &*I* **by** *blast*
  AOT-hence ‹∃ *z* ([$\mathcal{R}$]*xz* & [$\mathcal{R}$]*xz*)›
    using ∃ *I* **by** *fast*
  AOT-thus ‹$x =_{\mathcal{R}} x$›
    using *id−R−thm:1* ≡*E*(*2*) **by** *blast*
qed

AOT-theorem *id−R−thm:6*: ‹$x =_{\mathcal{R}} y \to y =_{\mathcal{R}} x$›
proof(*rule* →*I*)
  AOT-assume *0*: ‹$x =_{\mathcal{R}} y$›
  AOT-hence *1*: ‹*InDomainOf*(*x*,$\mathcal{R}$) & *InDomainOf*(*y*,$\mathcal{R}$)›
    using *id−R−thm:2*[*THEN* →*E*] **by** *blast*
  AOT-hence ‹$x =_{\mathcal{R}} y \equiv x = y$›
    using *id−R−thm:4*[*THEN* →*E*, *OF* ∨*I*(*1*)] &*E* **by** *blast*
  AOT-hence ‹*x* = *y*›
    using *0* **by** (*metis* ≡*E*(*1*))
  AOT-hence ‹*y* = *x*›
    using *id-sym* **by** *blast*
  moreover AOT-have ‹$y =_{\mathcal{R}} x \equiv y = x$›
    using *id−R−thm:4*[*THEN* →*E*, *OF* ∨*I*(*2*)] *1* &*E* **by** *blast*
  ultimately AOT-show ‹$y =_{\mathcal{R}} x$›
    by (*metis* ≡*E*(*2*))
qed

AOT-theorem *id−R−thm:7*: ‹$x =_{\mathcal{R}} y$ & $y =_{\mathcal{R}} z \to x =_{\mathcal{R}} z$›
proof (*rule* →*I*; *frule* &*E*(*1*); *drule* &*E*(*2*))
  AOT-assume *0*: ‹$x =_{\mathcal{R}} y$›
  AOT-hence *1*: ‹*InDomainOf*(*x*,$\mathcal{R}$) & *InDomainOf*(*y*,$\mathcal{R}$)›
    using *id−R−thm:2*[*THEN* →*E*] **by** *blast*
  AOT-hence ‹$x =_{\mathcal{R}} y \equiv x = y$›
    using *id−R−thm:4*[*THEN* →*E*, *OF* ∨*I*(*1*)] &*E* **by** *blast*
  AOT-hence *x-eq-y*: ‹*x* = *y*›
    using *0* **by** (*metis* ≡*E*(*1*))
  AOT-assume *2*: ‹$y =_{\mathcal{R}} z$›
  AOT-hence *3*: ‹*InDomainOf*(*y*,$\mathcal{R}$) & *InDomainOf*(*z*,$\mathcal{R}$)›
    using *id−R−thm:2*[*THEN* →*E*] **by** *blast*
  AOT-hence ‹$y =_{\mathcal{R}} z \equiv y = z$›
    using *id−R−thm:4*[*THEN* →*E*, *OF* ∨*I*(*1*)] &*E* **by** *blast*
  AOT-hence ‹*y* = *z*›
    using *2* **by** (*metis* ≡*E*(*1*))
  AOT-hence *x-eq-z*: ‹*x* = *z*›
    using *x-eq-y id-trans* **by** *blast*
  AOT-have ‹*InDomainOf*(*x*,$\mathcal{R}$) & *InDomainOf*(*z*,$\mathcal{R}$)›
    using *1 3* &*I* &*E* **by** *meson*
  AOT-hence ‹$x =_{\mathcal{R}} z \equiv x = z$›
    using *id−R−thm:4*[*THEN* →*E*, *OF* ∨*I*(*1*)] &*E* **by** *blast*
  AOT-thus ‹$x =_{\mathcal{R}} z$›
    using *x-eq-z* ≡*E*(*2*) **by** *blast*
qed

AOT-define *WeakAncestral* :: ‹Π ⇒ Π› (‹-$^+$›)
  w−ances−df: ‹$[\mathcal{R}]^+ =_{df} [\lambda xy\ [\mathcal{R}]^* xy \lor x =_{\mathcal{R}} y]$›

AOT-theorem *w−ances−df*[*den1*]: ‹$[\lambda xy\ [\Pi]^* xy \lor x =_{\Pi} y]\downarrow$›
  by *cqt:2*

```
AOT-theorem w−ances−df[den2]: ‹[Π]⁺↓›
  using w−ances−df[den1] =df I(1)[OF w−ances−df] by blast

AOT-theorem w−ances: ‹[ℛ]⁺xy ≡ ([ℛ]*xy ∨ x =ℛ y)›
proof −
  AOT-have 0: ‹[λxy [ℛ*]xy ∨ x =ℛ y]↓›
    by cqt:2
  AOT-have 1: ‹«(AOT-term-of-var x,AOT-term-of-var y)»↓›
    by (simp add: &I ex:1:a prod-denotesI rule−ui:3)
  have 2: ‹«[λμ1...μn [ℛ*]μ1...μn ∨ [(=ℛ)]μ1...μn]xy» =
          «[λxy [ℛ*]xy ∨ [(=ℛ)]xy]xy»›
    by (simp add: cond-case-prod-eta)
  show ?thesis
    apply (rule =df I(1)[OF w−ances−df])
     apply (fact w−ances−df[den1])
    using beta−C−meta[THEN →E, OF 0, unvarify ν1νn,
                    where τ=‹(-,-)›, simplified, OF 1] 2 by simp
qed

AOT-theorem w−ances−her:1: ‹[ℛ]xy → [ℛ]⁺xy›
proof(rule →I)
  AOT-assume ‹[ℛ]xy›
  AOT-hence ‹[ℛ]*xy›
    using anc−her:1[THEN →E] by blast
  AOT-thus ‹[ℛ]⁺xy›
    using w−ances[THEN ≡E(2)] ∨I by blast
qed

AOT-theorem w−ances−her:2:
  ‹[F]x & [ℛ]⁺xy & Hereditary(F, ℛ) → [F]y›
proof(rule →I; (frule &E(1); drule &E(2))+)
  AOT-assume 0: ‹[F]x›
  AOT-assume 1: ‹Hereditary(F, ℛ)›
  AOT-assume ‹[ℛ]⁺xy›
  AOT-hence ‹[ℛ]*xy ∨ x =ℛ y›
    using w−ances[THEN ≡E(1)] by simp
  moreover {
    AOT-assume ‹[ℛ]*xy›
    AOT-hence ‹[F]y›
      using anc−her:3[THEN →E, OF &I, OF &I] 0 1 by blast
  }
  moreover {
    AOT-assume ‹x =ℛ y›
    AOT-hence ‹x = y›
      using id−R−thm:3[THEN →E] by blast
    AOT-hence ‹[F]y›
      using 0 rule=E by blast
  }
  ultimately AOT-show ‹[F]y›
    by (metis ∨E(3) raa−cor:1)
qed

AOT-theorem w−ances−her:3: ‹([ℛ]⁺xy & [ℛ]yz) → [ℛ]*xz›
proof(rule →I; frule &E(1); drule &E(2))
  AOT-assume ‹[ℛ]⁺xy›
  moreover AOT-assume Ryz: ‹[ℛ]yz›
  ultimately AOT-have ‹[ℛ]*xy ∨ x =ℛ y›
    using w−ances[THEN ≡E(1)] by metis
  moreover {
    AOT-assume R-star-xy: ‹[ℛ]*xy›
    AOT-have ‹[ℛ]*xz›
    proof (safe intro!: ances[THEN ≡E(2)] →I GEN)
      fix F
```

```
      AOT-assume 0: ‹∀ z ([ℛ]xz → [F]z) & Hereditary(F,ℛ)›
      AOT-hence ‹[F]y›
        using R-star-xy ances[THEN ≡E(1), OF R-star-xy,
                              THEN ∀ E(2), THEN →E] by blast
      AOT-thus ‹[F]z›
        using hered:1[THEN ≡df E, OF 0[THEN &E(2)], THEN &E(2)]
              ∀ E(2) →E Ryz by blast
    qed
  }
  moreover {
    AOT-assume ‹x =ℛ y›
    AOT-hence ‹x = y›
      using id−R−thm:3[THEN →E] by blast
    AOT-hence ‹[ℛ]xz›
      using Ryz rule=E id-sym by fast
    AOT-hence ‹[ℛ]*xz›
      by (metis anc−her:1[THEN →E])
  }
  ultimately AOT-show ‹[ℛ]*xz›
    by (metis ∨E(3) raa−cor:1)
qed

AOT-theorem w−ances−her:4: ‹([ℛ]*xy & [ℛ]yz) → [ℛ]+xz›
proof(rule →I; frule &E(1); drule &E(2))
  AOT-assume ‹[ℛ]*xy›
  AOT-hence ‹[ℛ]*xy ∨ x =ℛ y›
    using ∨I by blast
  AOT-hence ‹[ℛ]+xy›
    using w−ances[THEN ≡E(2)] by blast
  moreover AOT-assume ‹[ℛ]yz›
  ultimately AOT-have ‹[ℛ]*xz›
    using w−ances−her:3[THEN →E, OF &I] by simp
  AOT-hence ‹[ℛ]*xz ∨ x =ℛ z›
    using ∨I by blast
  AOT-thus ‹[ℛ]+xz›
    using w−ances[THEN ≡E(2)] by blast
qed

AOT-theorem w−ances−her:5: ‹([ℛ]xy & [ℛ]+yz) → [ℛ]*xz›
proof(rule →I; frule &E(1); drule &E(2))
  AOT-assume 0: ‹[ℛ]xy›
  AOT-assume ‹[ℛ]+yz›
  AOT-hence ‹[ℛ]*yz ∨ y =ℛ z›
    by (metis ≡E(1) w−ances)
  moreover {
    AOT-assume ‹[ℛ]*yz›
    AOT-hence ‹[ℛ]*xz›
      using 0 by (metis anc−her:4 Adjunction →E)
  }
  moreover {
    AOT-assume ‹y =ℛ z›
    AOT-hence ‹y = z›
      by (metis id−R−thm:3 →E)
    AOT-hence ‹[ℛ]xz›
      using 0 rule=E by fast
    AOT-hence ‹[ℛ]*xz›
      by (metis anc−her:1 →E)
  }
  ultimately AOT-show ‹[ℛ]*xz› by (metis ∨E(2) reductio−aa:1)
qed

AOT-theorem w−ances−her:6: ‹([ℛ]+xy & [ℛ]+yz) → [ℛ]+xz›
proof(rule →I; frule &E(1); drule &E(2))
```

```
  AOT-assume 0: ‹[ℛ]⁺xy›
  AOT-hence 1: ‹[ℛ]*xy ∨ x =ℛ y›
    by (metis ≡E(1) w−ances)
  AOT-assume 2: ‹[ℛ]⁺yz›
  {
    AOT-assume ‹x =ℛ y›
    AOT-hence ‹x = y›
      by (metis id−R−thm:3 →E)
    AOT-hence ‹[ℛ]⁺xz›
      using 2 rule=E id-sym by fast
  }
  moreover {
    AOT-assume ‹¬(x =ℛ y)›
    AOT-hence 3: ‹[ℛ]*xy›
      using 1 by (metis ∨E(3))
    AOT-have ‹[ℛ]*yz ∨ y =ℛ z›
      using 2 by (metis ≡E(1) w−ances)
    moreover {
      AOT-assume ‹[ℛ]*yz›
      AOT-hence ‹[ℛ]*xz›
        using 3 by (metis anc−her:6 Adjunction →E)
      AOT-hence ‹[ℛ]⁺xz›
        by (metis ∨I(1) ≡E(2) w−ances)
    }
    moreover {
      AOT-assume ‹y =ℛ z›
      AOT-hence ‹y = z›
        by (metis id−R−thm:3 →E)
      AOT-hence ‹[ℛ]⁺xz›
        using 0 rule=E id-sym by fast
    }
    ultimately AOT-have ‹[ℛ]⁺xz›
      by (metis ∨E(3) reductio−aa:1)
  }
  ultimately AOT-show ‹[ℛ]⁺xz›
    by (metis reductio−aa:1)
qed

AOT-theorem w−ances−her:7: ‹[ℛ]*xy → ∃z([ℛ]⁺xz & [ℛ]zy)›
proof(rule →I)
  AOT-assume 0: ‹[ℛ]*xy›
  AOT-have 1: ‹∀z ([ℛ]xz → [Π]z) & Hereditary(Π,ℛ) → [Π]y› if ‹Π↓› for Π
    using ances[THEN ≡E(1), THEN ∀E(1), OF 0] that by blast
  AOT-have ‹[λy ∃z([ℛ]⁺xz & [ℛ]zy)]y›
  proof (rule 1[THEN →E]; cqt:2[lambda]?;
         safe intro!: &I GEN →I hered:1[THEN ≡df I] cqt:2)
    fix z
    AOT-assume 0: ‹[ℛ]xz›
    AOT-hence ‹∃z [ℛ]xz› by (rule ∃I)
    AOT-hence ‹InDomainOf(x, ℛ)› by (metis ≡df I df−1−1:5)
    AOT-hence ‹x =ℛ x› by (metis id−R−thm:5 →E)
    AOT-hence ‹[ℛ]⁺xx› by (metis ∨I(2) ≡E(2) w−ances)
    AOT-hence ‹[ℛ]⁺xx & [ℛ]xz› using 0 &I by blast
    AOT-hence ‹∃y ([ℛ]⁺xy & [ℛ]yz)› by (rule ∃I)
    AOT-thus ‹[λy ∃z ([ℛ]⁺xz & [ℛ]zy)]z›
      by (auto intro!: β←C(1) cqt:2)
  next
    fix x′ y
    AOT-assume Rx′y: ‹[ℛ]x′y›
    AOT-assume ‹[λy ∃z ([ℛ]⁺xz & [ℛ]zy)]x′›
    AOT-hence ‹∃z ([ℛ]⁺xz & [ℛ]zx′)›
      using β→C(1) by blast
    then AOT-obtain c where c-prop: ‹[ℛ]⁺xc & [ℛ]cx′›
```

using $\exists E[rotated]$ **by** *blast*
AOT-hence ‹$[\mathcal{R}]^*xx'$›
by (*meson* $Rx'y$ *anc−her*:*1 anc−her*:*6 Adjunction* $\rightarrow E$ *w−ances−her*:*3*)
AOT-hence ‹$[\mathcal{R}]^*xx' \vee x =_{\mathcal{R}} x'$› **by** (*rule* $\vee I$)
AOT-hence ‹$[\mathcal{R}]^+xx'$› **by** (*metis* $\equiv E(2)$ *w−ances*)
AOT-hence ‹$[\mathcal{R}]^+xx'$ & $[\mathcal{R}]x'y$› **using** $Rx'y$ **by** (*metis* &I)
AOT-hence ‹$\exists z$ ($[\mathcal{R}]^+xz$ & $[\mathcal{R}]zy$)› **by** (*rule* $\exists I$)
AOT-thus ‹$[\lambda y\ \exists z$ ($[\mathcal{R}]^+xz$ & $[\mathcal{R}]zy$)$]y$›
by (*auto intro*!: $\beta\leftarrow C(1)$ *cqt*:*2*)
qed
AOT-thus ‹$\exists z$($[\mathcal{R}]^+xz$ & $[\mathcal{R}]zy$)›
using $\beta\rightarrow C(1)$ **by** *fast*
qed

AOT-theorem *1−1−R*:*1*: ‹($[\mathcal{R}]xy$ & $[\mathcal{R}]^*zy$) $\rightarrow$ $[\mathcal{R}]^+zx$›
proof(*rule* $\rightarrow I$; *frule* &$E(1)$; *drule* &$E(2)$)
AOT-assume ‹$[\mathcal{R}]^*zy$›
AOT-hence ‹$\exists x$ ($[\mathcal{R}]^+zx$ & $[\mathcal{R}]xy$)›
using *w−ances−her*:*7*[*THEN* $\rightarrow E$] **by** *simp*
then AOT-obtain *a* **where** *a-prop*: ‹$[\mathcal{R}]^+za$ & $[\mathcal{R}]ay$›
using $\exists E[rotated]$ **by** *blast*
moreover AOT-assume ‹$[\mathcal{R}]xy$›
ultimately AOT-have ‹$x = a$›
using *df−1−1*:*2*[*THEN* $\equiv_{df}E$, *OF RigidOneToOneRelation*.ψ, *THEN* &$E(1)$,
THEN $\equiv_{df}E$[*OF df−1−1*:*1*], *THEN* &$E(2)$, *THEN* $\forall E(2)$,
THEN $\forall E(2)$, *THEN* $\forall E(2)$, *THEN* $\rightarrow E$, *OF* &I]
&E **by** *blast*
AOT-thus ‹$[\mathcal{R}]^+zx$›
using *a-prop*[*THEN* &$E(1)$] *rule=E id-sym* **by** *fast*
qed

AOT-theorem *1−1−R*:*2*: ‹$[\mathcal{R}]xy \rightarrow (\neg[\mathcal{R}]^*xx \rightarrow \neg[\mathcal{R}]^*yy)$›
proof(*rule* $\rightarrow I$; *rule useful−tautologies*:*5*[*THEN* $\rightarrow E$]; *rule* $\rightarrow I$)
AOT-assume *0*: ‹$[\mathcal{R}]xy$›
moreover AOT-assume ‹$[\mathcal{R}]^*yy$›
ultimately AOT-have ‹$[\mathcal{R}]^+yx$›
using *1−1−R*:*1*[*THEN* $\rightarrow E$, *OF* &I] **by** *blast*
AOT-thus ‹$[\mathcal{R}]^*xx$›
using *0* **by** (*metis* &I $\rightarrow E$ *w−ances−her*:*5*)
qed

AOT-theorem *1−1−R*:*3*: ‹$\neg[\mathcal{R}]^*xx \rightarrow ([\mathcal{R}]^+xy \rightarrow \neg[\mathcal{R}]^*yy)$›
proof(*safe intro*!: $\rightarrow I$)
AOT-have *0*: ‹$[\lambda z\ \neg[\mathcal{R}]^*zz]\downarrow$› **by** *cqt*:*2*
AOT-assume *1*: ‹$\neg[\mathcal{R}]^*xx$›
AOT-assume *2*: ‹$[\mathcal{R}]^+xy$›
AOT-have ‹$[\lambda z\ \neg[\mathcal{R}]^*zz]y$›
proof(*rule w−ances−her*:*2*[*unvarify F*, *OF 0*, *THEN* $\rightarrow E$];
safe intro!: &I *hered*:*1*[*THEN* $\equiv_{df}I$] *cqt*:*2 GEN* $\rightarrow I$)
AOT-show ‹$[\lambda z\ \neg[\mathcal{R}]^*zz]x$›
by (*auto intro*!: $\beta\leftarrow C(1)$ *cqt*:*2 simp*: *1*)
next
AOT-show ‹$[\mathcal{R}]^+xy$› **by** (*fact 2*)
next
fix *x y*
AOT-assume ‹$[\lambda z\ \neg[\mathcal{R}^*]zz]x$›
AOT-hence ‹$\neg[\mathcal{R}]^*xx$› **by** (*rule* $\beta\rightarrow C(1)$)
moreover AOT-assume ‹$[\mathcal{R}]xy$›
ultimately AOT-have ‹$\neg[\mathcal{R}]^*yy$›
using *1−1−R*:*2*[*THEN* $\rightarrow E$, *THEN* $\rightarrow E$] **by** *blast*
AOT-thus ‹$[\lambda z\ \neg[\mathcal{R}^*]zz]y$›
by (*auto intro*!: $\beta\leftarrow C(1)$ *cqt*:*2*)
qed

```
  AOT-thus ‹¬[ℛ]*yy›
    using β→C(1) by blast
qed

AOT-theorem 1−1−R:4: ‹[ℛ]*xy → InDomainOf(x,ℛ)›
proof(rule →I; rule df−1−1:5[THEN ≡df I])
  AOT-assume 1: ‹[ℛ]*xy›
  AOT-have ‹[λz [ℛ*]xz → ∃ y [ℛ]xy]y›
  proof (safe intro!: anc−her:2[unvarify F, THEN →E];
         safe intro!: cqt:2 &I GEN →I hered:1[THEN ≡df I])
    AOT-show ‹[ℛ]*xy› by (fact 1)
  next
    fix z
    AOT-assume ‹[ℛ]xz›
    AOT-thus ‹[λz [ℛ*]xz → ∃ y [ℛ]xy]z›
      by (safe intro!: β←C(1) cqt:2)
         (meson →I existential:2[const-var])
  next
    fix x′ y
    AOT-assume Rx′y: ‹[ℛ]x′y›
    AOT-assume ‹[λz [ℛ*]xz → ∃ y [ℛ]xy]x′›
    AOT-hence 0: ‹[ℛ*]xx′ → ∃ y [ℛ]xy› by (rule β→C(1))
    AOT-have 1: ‹[ℛ*]xy → ∃ y [ℛ]xy›
    proof(rule →I)
      AOT-assume ‹[ℛ]*xy›
      AOT-hence ‹[ℛ]+xx′› by (metis Rx′y &I 1−1−R:1 →E)
      AOT-hence ‹[ℛ]*xx′ ∨ x =ℛ x′› by (metis ≡E(1) w−ances)
      moreover {
        AOT-assume ‹[ℛ]*xx′›
        AOT-hence ‹∃ y [ℛ]xy› using 0 by (metis →E)
      }
      moreover {
        AOT-assume ‹x =ℛ x′›
        AOT-hence ‹x = x′› by (metis id−R−thm:3 →E)
        AOT-hence ‹[ℛ]xy› using Rx′y rule=E id-sym by fast
        AOT-hence ‹∃ y [ℛ]xy› by (rule ∃ I)
      }
      ultimately AOT-show ‹∃ y [ℛ]xy›
        by (metis ∨E(3) reductio−aa:1)
    qed
    AOT-show ‹[λz [ℛ*]xz → ∃ y [ℛ]xy]y›
      by (auto intro!: β←C(1) cqt:2 1)
  qed
  AOT-hence ‹[ℛ*]xy → ∃ y [ℛ]xy› by (rule β→C(1))
  AOT-thus ‹∃ y [ℛ]xy› using 1 →E by blast
qed

AOT-theorem 1−1−R:5: ‹[ℛ]+xy → InDomainOf(x,ℛ)›
proof (rule →I)
  AOT-assume ‹[ℛ]+xy›
  AOT-hence ‹[ℛ]*xy ∨ x =ℛ y›
    by (metis ≡E(1) w−ances)
  moreover {
    AOT-assume ‹[ℛ]*xy›
    AOT-hence ‹InDomainOf(x,ℛ)›
      using 1−1−R:4 →E by blast
  }
  moreover {
    AOT-assume ‹x =ℛ y›
    AOT-hence ‹InDomainOf(x,ℛ)›
      by (metis Conjunction Simplification(1) id−R−thm:2 →E)
  }
  ultimately AOT-show ‹InDomainOf(x,ℛ)›
```

```
    by (metis ∨E(3) reductio−aa:1)
qed

AOT-theorem pre−ind:
  ‹([F]z & ∀x∀y(([ℛ]+zx & [ℛ]+zy) → ([ℛ]xy → ([F]x → [F]y)))) →
   ∀x ([ℛ]+zx → [F]x)›
proof(safe intro!: →I GEN)
  AOT-have den: ‹[λy [F]y & [ℛ]+zy]↓› by cqt:2
  fix x
  AOT-assume ϑ: ‹[F]z & ∀x∀y(([ℛ]+zx & [ℛ]+zy) → ([ℛ]xy → ([F]x → [F]y)))›
  AOT-assume 0: ‹[ℛ]+zx›

  AOT-have ‹[λy [F]y & [ℛ]+zy]x›
  proof (rule w−ances−her:2[unvarify F, OF den, THEN →E]; safe intro!: &I)
    AOT-show ‹[λy [F]y & [ℛ]+zy]z›
    proof (safe intro!: β←C(1) cqt:2 &I)
      AOT-show ‹[F]z› using ϑ &E by blast
    next
      AOT-show ‹[ℛ]+zz›
        by (rule w−ances[THEN ≡E(2), OF ∨I(2)])
           (meson 0 id−R−thm:5 1−1−R:5 →E)
    qed
  next
    AOT-show ‹[ℛ]+zx› by (fact 0)
  next
    AOT-show ‹Hereditary([λy [F]y & [ℛ]+zy],ℛ)›
    proof (safe intro!: hered:1[THEN ≡dfI] &I cqt:2 GEN →I)
      fix x' y
      AOT-assume 1: ‹[ℛ]x'y›
      AOT-assume ‹[λy [F]y & [ℛ]+zy]x'›
      AOT-hence 2: ‹[F]x' & [ℛ]+zx'› by (rule β→C(1))
      AOT-have ‹[ℛ]*zy› using 1 2[THEN &E(2)]
        by (metis Adjunction modus−tollens:1 reductio−aa:1 w−ances−her:3)
      AOT-hence 3: ‹[ℛ]+zy› by (metis ∨I(1) ≡E(2) w−ances)
      AOT-show ‹[λy [F]y & [ℛ]+zy]y›
      proof (safe intro!: β←C(1) cqt:2 &I 3)
        AOT-show ‹[F]y›
        proof (rule ϑ[THEN &E(2), THEN ∀E(2), THEN ∀E(2),
                     THEN →E, THEN →E, THEN →E])
          AOT-show ‹[ℛ]+zx' & [ℛ]+zy›
            using 2 3 &E &I by blast
        next
          AOT-show ‹[ℛ]x'y› by (fact 1)
        next
          AOT-show ‹[F]x'› using 2 &E by blast
        qed
      qed
    qed
  qed
  AOT-thus ‹[F]x› using β→C(1) &E(1) by fast
qed
```

The following is not part of PLM, but a theorem of AOT. It states that the predecessor relation coexists with numbering a property. We will use this fact to derive the predecessor axiom, which asserts that the predecessor relation denotes, from the fact that our models validate that numbering a property denotes.

```
AOT-theorem pred-coex:
  ‹[λxy ∃F∃u ([F]u & Numbers(y,F) & Numbers(x,[F]−u))]↓ ≡ ∀F ([λx Numbers(x,F)]↓)›
proof(safe intro!: ≡I →I GEN)
  fix F
  let ?P = ‹«[λxy ∃F∃u ([F]u & Numbers(y,F) & Numbers(x,[F]−u))]»›
  AOT-assume ‹[«?P»]↓›
  AOT-hence ‹□[«?P»]↓›
    using exist−nec →E by blast
```

moreover AOT-have
‹□[«?P»]↓ → □(∀x∀y(∀F([F]x ≡ [F]y) → (Numbers(x,F) ≡ Numbers(y,F))))›
proof(*rule RM*; *safe intro*!: →*I GEN*)
AOT-modally-strict {
fix *x y*
AOT-assume *pred-den*: ‹[«?P»]↓›
AOT-hence *pred-equiv*:
‹[«?P»]xy ≡ ∃F∃u ([F]u & Numbers(y,F) & Numbers(x,[F]$^{-u}$))› **for** *x y*
by (*safe intro*!: *beta−C−meta*[*unvarify* $\nu_1\nu_n$, **where** τ=‹(-,-)›, *THEN* →*E*,
rotated, *OF pred-den*, *simplified*]
tuple-denotes[*THEN* $\equiv_{df}$*I*] &*I cqt*:*2*)

We show as a subproof that any natural cardinal that is not zero has a predecessor.

AOT-have *CardinalPredecessor*:
‹∃y [«?P»]yx› **if** *card-x*: ‹*NaturalCardinal*(x)› **and** *x-nonzero*: ‹x ≠ 0› **for** *x*
proof −
AOT-have ‹∃G x = #G›
using *card*[*THEN* $\equiv_{df}$*E*, *OF card-x*]**.**
AOT-hence ‹∃G Numbers(x,G)›
using *eq−df−num*[*THEN* ≡*E*(*1*)] **by** *blast*
then AOT-obtain *G'* **where** *numxG'*: ‹Numbers(x,G')›
using ∃*E*[*rotated*] **by** *blast*
AOT-obtain *G* **where** ‹*Rigidifies*(G,G')›
using *rigid−der*:*3* ∃*E*[*rotated*] **by** *blast*

AOT-hence *H*: ‹*Rigid*(G) & ∀x ([G]x ≡ [G']x)›
using *df−rigid−rel*:*2*[*THEN* $\equiv_{df}$*E*] **by** *blast*
AOT-have *H-rigid*: ‹□∀x ([G]x → □[G]x)›
using *H*[*THEN* &*E*(*1*), *THEN df−rigid−rel*:*1*[*THEN* $\equiv_{df}$*E*], *THEN* &*E*(*2*)]**.**
AOT-hence ‹∀x □([G]x → □[G]x)›
using *CBF* →*E* **by** *blast*
AOT-hence *R*: ‹□([G]x → □[G]x)› **for** *x* **using** ∀*E*(*2*) **by** *blast*
AOT-hence *rigid*: ‹[G]x ≡ 𝓐[G]x› **for** *x*
by (*metis* ≡*E*(*6*) *oth−class−taut*:*3*:*a sc−eq−fur*:*2* →*E*)
AOT-have ‹G $\equiv_E$ G'›
proof (*safe intro*!: *eqE*[*THEN* $\equiv_{df}$*I*] &*I cqt*:*2 GEN* →*I*)
AOT-show ‹[G]x ≡ [G']x› **for** *x* **using** *H*[*THEN* &*E*(*2*)] ∀*E*(*2*) **by** *fast*
qed
AOT-hence ‹G $\approx_E$ G'›
by (*rule apE−eqE*:*2*[*THEN* →*E*, *OF* &*I*, *rotated*])
(*simp add*: *eq−part*:*1*)
AOT-hence *numxG*: ‹Numbers(x,G)›
using *num−tran*:*1*[*THEN* →*E*, *THEN* ≡*E*(*2*)] *numxG'* **by** *blast*

{
AOT-assume ‹¬∃y(y ≠ x & [«?P»]yx)›
AOT-hence ‹∀y ¬(y ≠ x & [«?P»]yx)›
using *cqt−further*:*4* →*E* **by** *blast*
AOT-hence ‹¬(y ≠ x & [«?P»]yx)› **for** *y*
using ∀*E*(*2*) **by** *blast*
AOT-hence *0*: ‹¬y ≠ x ∨ ¬[«?P»]yx› **for** *y*
using ¬¬*E intro−elim*:*3*:*c oth−class−taut*:*5*:*a* **by** *blast*
{
fix *y*
AOT-assume ‹[«?P»]yx›
AOT-hence ‹¬y ≠ x›
using *0* ¬¬*I con−dis−i−e*:*4*:*c* **by** *blast*
AOT-hence ‹y = x›
using =−*infix* $\equiv_{df}$*I raa−cor*:*4* **by** *blast*
} note *Pxy-imp-eq* = *this*
AOT-have ‹[«?P»]xx›
proof(*rule raa−cor*:*1*)
AOT-assume *notPxx*: ‹¬[«?P»]xx›

AOT-hence ‹¬∃ *F*∃ *u*([*F*]*u* & *Numbers*(*x*,*F*) & *Numbers*(*x*,[*F*]$^{-u}$))›
  using *pred-equiv intro−elim*:*3*:*c* **by** *blast*
AOT-hence ‹∀ *F* ¬∃ *u*([*F*]*u* & *Numbers*(*x*,*F*) & *Numbers*(*x*,[*F*]$^{-u}$))›
  using *cqt−further*:*4*[*THEN* →*E*] **by** *blast*
AOT-hence ‹¬∃ *u*([*F*]*u* & *Numbers*(*x*,*F*) & *Numbers*(*x*,[*F*]$^{-u}$))› **for** *F*
  using ∀ *E*(*2*) **by** *blast*
AOT-hence ‹∀ *y* ¬(*O*!*y* & ([*F*]*y* & *Numbers*(*x*,*F*) & *Numbers*(*x*,[*F*]$^{-y}$)))› **for** *F*
  using *cqt−further*:*4*[*THEN* →*E*] **by** *blast*
AOT-hence *0*: ‹¬(*O*!*u* & ([*F*]*u* & *Numbers*(*x*,*F*) & *Numbers*(*x*,[*F*]$^{-u}$)))› **for** *F u*
  using ∀ *E*(*2*) **by** *blast*
AOT-have ‹□¬∃ *u* [*G*]*u*›
proof(*rule raa−cor*:*1*)
  AOT-assume ‹¬□¬∃ *u* [*G*]*u*›
  AOT-hence ‹◇∃ *u* [*G*]*u*›
    using ≡$_{df}$*I conventions*:*5* **by** *blast*
  AOT-hence ‹∃ *u* ◇[*G*]*u*›
    by (*metis Ordinary.res−var−bound−reas*[*BF*◇][*THEN* →*E*])
  then AOT-obtain *u* **where** *posGu*: ‹◇[*G*]*u*›
    using *Ordinary*.∃ *E*[*rotated*] **by** *meson*
  AOT-hence *Gu*: ‹[*G*]*u*›
    by (*meson B*◇ *K*◇ →*E R*)
  AOT-have ‹¬([*G*]*u* & *Numbers*(*x*,*G*) & *Numbers*(*x*,[*G*]$^{-u}$))›
    using *0 Ordinary*.*ψ*
    by (*metis con−dis−i−e*:*1 raa−cor*:*1*)
  AOT-hence *notnumx*: ‹¬*Numbers*(*x*,[*G*]$^{-u}$)›
    using *Gu numxG con−dis−i−e*:*1 raa−cor*:*5* **by** *metis*
  AOT-obtain *y* **where** *numy*: ‹*Numbers*(*y*,[*G*]$^{-u}$)›
    using *num*:*1*[*unvarify G*, *OF F*−*u*[*den*]] ∃ *E*[*rotated*] **by** *blast*
  AOT-hence ‹[*G*]*u* & *Numbers*(*x*,*G*) & *Numbers*(*y*,[*G*]$^{-u}$)›
    using *Gu numxG* &*I* **by** *blast*
  AOT-hence ‹∃ *u* ([*G*]*u* & *Numbers*(*x*,*G*) & *Numbers*(*y*,[*G*]$^{-u}$))›
    by (*rule Ordinary*.∃ *I*)
  AOT-hence ‹∃ *G*∃ *u* ([*G*]*u* & *Numbers*(*x*,*G*) & *Numbers*(*y*,[*G*]$^{-u}$))›
    by (*rule* ∃ *I*)
  AOT-hence ‹[«*?P*»]*yx*›
    using *pred-equiv*[*THEN* ≡*E*(*2*)] **by** *blast*
  AOT-hence ‹*y* = *x*› **using** *Pxy-imp-eq* **by** *blast*
  AOT-hence ‹*Numbers*(*x*,[*G*]$^{-u}$)›
    using *numy rule*=*E* **by** *fast*
  AOT-thus ‹*p* & ¬*p*› **for** *p* **using** *notnumx reductio−aa*:*1* **by** *blast*
qed
AOT-hence ‹¬∃ *u* [*G*]*u*›
  using *qml*:*2*[*axiom-inst*, *THEN* →*E*] **by** *blast*
AOT-hence *num0G*: ‹*Numbers*(*0*, *G*)›
  using *0F*:*1*[*THEN* ≡*E*(*1*)] **by** *blast*
AOT-hence ‹*x* = *0*›
  using *pre−Hume*[*unvarify x*, *THEN* →*E*, *OF zero*:*2*, *OF* &*I*,
        THEN ≡*E*(*2*), *OF num0G*, *OF numxG*, *OF eq−part*:*1*]
    id-sym **by** *blast*
moreover AOT-have ‹¬*x* = *0*›
  using *x-nonzero*
  using =−*infix* ≡$_{df}$*E* **by** *blast*
ultimately AOT-show ‹*p* & ¬*p*› **for** *p* **using** *reductio−aa*:*1* **by** *blast*
qed
}
AOT-hence ‹[«*?P*»]*xx* ∨ ∃ *y* (*y* ≠ *x* & [«*?P*»]*yx*)›
  using *con−dis−i−e*:*3*:*a con−dis−i−e*:*3*:*b raa−cor*:*1* **by** *blast*
moreover {
  AOT-assume ‹[«*?P*»]*xx*›
  AOT-hence ‹∃ *y* [«*?P*»]*yx*›
    by (*rule* ∃ *I*)
}
moreover {

```
    AOT-assume ‹∃ y (y ≠ x & [«?P»]yx)›
    then AOT-obtain y where ‹y ≠ x & [«?P»]yx›
      using ∃ E[rotated] by blast
    AOT-hence ‹[«?P»]yx›
      using &E by blast
    AOT-hence ‹∃ y [«?P»]yx›
      by (rule ∃ I)
  }
  ultimately AOT-show ‹∃ y [«?P»]yx›
    using ∨E(1) →I by blast
qed
```

Given above lemma, we can show that if one of two indistinguishable objects numbers a property, the other one numbers this property as well.

```
AOT-assume indist: ‹∀ F([F]x ≡ [F]y)›
AOT-assume numxF: ‹Numbers(x,F)›
AOT-hence 0: ‹NaturalCardinal(x)›
  by (metis eq−num:6 vdash−properties:10)
```

We show by case distinction that x equals y. As first case we consider x to be non-zero.

```
{
  AOT-assume ‹¬(x = 0)›
  AOT-hence ‹x ≠ 0›
    by (metis =−infix ≡df I)
  AOT-hence ‹∃ y [«?P»]yx›
    using CardinalPredecessor 0 by blast
  then AOT-obtain z where Pxz: ‹[«?P»]zx›
    using ∃ E[rotated] by blast
  AOT-hence ‹[λy [«?P»]zy]x›
    by (safe intro!: β←C cqt:2)
  AOT-hence ‹[λy [«?P»]zy]y›
    by (safe intro!: indist[THEN ∀ E(1), THEN ≡E(1)] cqt:2)
  AOT-hence Pyz: ‹[«?P»]zy›
    using β→C(1) by blast
  AOT-hence ‹∃ F∃ u ([F]u & Numbers(y,F) & Numbers(z,[F]^-u))›
    using Pyz pred-equiv[THEN ≡E(1)] by blast
  then AOT-obtain F1 where ‹∃ u ([F1]u & Numbers(y,F1) & Numbers(z,[F1]^-u))›
    using ∃ E[rotated] by blast
  then AOT-obtain u where u-prop: ‹[F1]u & Numbers(y,F1) & Numbers(z,[F1]^-u)›
    using Ordinary.∃ E[rotated] by meson
  AOT-have ‹∃ F∃ u ([F]u & Numbers(x,F) & Numbers(z,[F]^-u))›
    using Pxz pred-equiv[THEN ≡E(1)] by blast
  then AOT-obtain F2 where ‹∃ u ([F2]u & Numbers(x,F2) & Numbers(z,[F2]^-u))›
    using ∃ E[rotated] by blast
  then AOT-obtain v where v-prop: ‹[F2]v & Numbers(x,F2) & Numbers(z,[F2]^-v)›
    using Ordinary.∃ E[rotated] by meson
  AOT-have ‹[F2]^-v ≈E [F1]^-u›
    using hume−strict:1[unvarify F G, THEN ≡E(1), OF F−u[den],
                        OF F−u[den], OF ∃ I(2)[where β=z], OF &I]
        v-prop u-prop &E by blast
  AOT-hence ‹F2 ≈E F1›
    using P′−eq[THEN →E, OF &I, OF &I]
        u-prop v-prop &E by meson
  AOT-hence ‹x = y›
    using pre−Hume[THEN →E, THEN ≡E(2), OF &I]
        v-prop u-prop &E by blast
}
```

The second case handles x being equal to zero.

```
moreover {
  fix u
  AOT-assume x-is-zero: ‹x = 0›
  moreover AOT-have ‹Numbers(0,[λz z =E u]^-u)›
```

```
    proof (safe intro!: 0F:1[unvarify F, THEN ≡E(1)] cqt:2 raa−cor:2
                    F−u[den][unvarify F])
      AOT-assume ‹∃ v [[λz z =E u]⁻ᵘ]v›
      then AOT-obtain v where ‹[[λz z =E u]⁻ᵘ]v›
        using Ordinary.∃ E[rotated] by meson
      AOT-hence ‹[λz z =E u]v & v ≠E u›
        by (auto intro: F−u[THEN =df E(1), where τ1τn=(-,-), simplified]
                intro!: cqt:2 F−u[equiv][unvarify F, THEN ≡E(1)]
                       F−u[den][unvarify F])
      AOT-thus ‹p & ¬p› for p
        using β→C thm−neg=E[THEN ≡E(1)] &E &I
             raa−cor:3 by fast
    qed
    ultimately AOT-have 0: ‹Numbers(x,[λz z =E u]⁻ᵘ)›
      using rule=E id-sym by fast
    AOT-have ‹∃ y Numbers(y,[λz z =E u])›
      by (safe intro!: num:1[unvarify G] cqt:2)
    then AOT-obtain z where ‹Numbers(z,[λz z =E u])›
      using ∃ E by metis
    moreover AOT-have ‹[λz z=E u]u›
      by (safe intro!: β←C cqt:2 ord=Eequiv:1[THEN →E] Ordinary.ψ)
    ultimately AOT-have
      1: ‹[λz z=E u]u & Numbers(z,[λz z=E u]) & Numbers(x,[λz z=E u]⁻ᵘ)›
      using 0 &I by auto
    AOT-hence ‹∃ v([λz z=E u]v & Numbers(z,[λz z =E u]) & Numbers(x,[λz z=E u]⁻ᵛ))›
      by (rule Ordinary.∃ I)
    AOT-hence ‹∃ F∃ u([F]u & Numbers(z,[F]) & Numbers(x,[F]⁻ᵘ))›
      by (rule ∃ I; cqt:2)
    AOT-hence Px1: ‹[«?P»]xz›
      using beta−C−cor:2[THEN →E, OF pred-den,
             THEN tuple-forall[THEN ≡df E], THEN ∀ E(2),
             THEN ∀ E(2), THEN ≡E(2)] by simp
    AOT-hence ‹[λy [«?P»]yz]x›
      by (safe intro!: β←C cqt:2)
    AOT-hence ‹[λy [«?P»]yz]y›
      by (safe intro!: indist[THEN ∀ E(1), THEN ≡E(1)] cqt:2)
    AOT-hence Py1: ‹[«?P»]yz›
      using β→C by blast
    AOT-hence ‹∃ F∃ u([F]u & Numbers(z,[F]) & Numbers(y,[F]⁻ᵘ))›
      using β→C by fast
    then AOT-obtain G where ‹∃ u([G]u & Numbers(z,[G]) & Numbers(y,[G]⁻ᵘ))›
      using ∃ E[rotated] by blast
    then AOT-obtain v where 2: ‹[G]v & Numbers(z,[G]) & Numbers(y,[G]⁻ᵛ)›
      using Ordinary.∃ E[rotated] by meson
    with 1 2 AOT-have ‹[λz z =E u] ≈E G›
      by (auto intro!: hume−strict:1[unvarify F, THEN ≡E(1), rotated,
                       OF ∃ I(2)[where β=z], OF &I] cqt:2
              dest: &E)
    AOT-hence 3: ‹[λz z =E u]⁻ᵘ ≈E [G]⁻ᵛ›
      using 1 2
      by (safe-step intro!: eqP′[unvarify F, THEN →E])
         (auto dest: &E intro!: cqt:2 &I)
    with 1 2 AOT-have ‹x = y›
      by (auto intro!: pre−Hume[unvarify G H, THEN →E,
                            THEN ≡E(2), rotated 3, OF 3]
                   F−u[den][unvarify F] cqt:2 &I
              dest: &E)
  }
  ultimately AOT-have ‹x = y›
    using ∨E(1) →I reductio−aa:1 by blast
```

Now since x numbers F, so does y.

```
  AOT-hence ‹Numbers(y,F)›
```

```
      using numxF rule=E by fast
  } note 0 = this
```

The only thing left is to generalize this result to a biconditional.

```
  AOT-modally-strict {
    fix x y
    AOT-assume ‹[«?P»]↓›
    moreover AOT-assume ‹∀ F([F]x ≡ [F]y)›
    moreover AOT-have ‹∀ F([F]y ≡ [F]x)›
      by (metis cqt−basic:11 intro−elim:3:a calculation(2))
    ultimately AOT-show ‹Numbers(x,F) ≡ Numbers(y,F)›
      using 0 ≡I →I by auto
  }
 qed
 ultimately AOT-show ‹[λx Numbers(x,F)]↓›
  using kirchner−thm:1[THEN ≡E(2)] →E by fast
next
```

The converse can be shown by coexistence.

```
 AOT-assume ‹∀ F [λx Numbers(x,F)]↓›
 AOT-hence ‹[λx Numbers(x,F)]↓› for F
  using ∀ E(2) by blast
 AOT-hence ‹□[λx Numbers(x,F)]↓› for F
  using exist−nec[THEN →E] by blast
 AOT-hence ‹∀ F □[λx Numbers(x,F)]↓›
  by (rule GEN)
 AOT-hence ‹□∀ F [λx Numbers(x,F)]↓›
  using BF[THEN →E] by fast
 moreover AOT-have
  ‹□∀ F [λx Numbers(x,F)]↓ →
   □∀ x ∀ y (∃ F ∃ u ([F]u & [λz Numbers(z,F)]y & [λz Numbers(z,[F]^-u)]x) ≡
          ∃ F ∃ u ([F]u & Numbers(y,F) & Numbers(x,[F]^-u)))›
 proof(rule RM; safe intro!: →I GEN)
  AOT-modally-strict {
    fix x y
    AOT-assume 0: ‹∀ F [λx Numbers(x,F)]↓›
    AOT-show ‹∃ F ∃ u ([F]u & [λz Numbers(z,F)]y & [λz Numbers(z,[F]^-u)]x) ≡
          ∃ F ∃ u ([F]u & Numbers(y,F) & Numbers(x,[F]^-u))›
    proof(safe intro!: ≡I →I)
      AOT-assume ‹∃ F ∃ u ([F]u & [λz Numbers(z,F)]y & [λz Numbers(z,[F]^-u)]x)›
      then AOT-obtain F where
        ‹∃ u ([F]u & [λz Numbers(z,F)]y & [λz Numbers(z,[F]^-u)]x)›
        using ∃ E[rotated] by blast
      then AOT-obtain u where ‹[F]u & [λz Numbers(z,F)]y & [λz Numbers(z,[F]^-u)]x›
        using Ordinary.∃ E[rotated] by meson
      AOT-hence ‹[F]u & Numbers(y,F) & Numbers(x,[F]^-u)›
        by (auto intro!: &I dest: &E β→C)
      AOT-thus ‹∃ F ∃ u ([F]u & Numbers(y,F) & Numbers(x,[F]^-u))›
        using ∃ I Ordinary.∃ I by fast
    next
      AOT-assume ‹∃ F ∃ u ([F]u & Numbers(y,F) & Numbers(x,[F]^-u))›
      then AOT-obtain F where ‹∃ u ([F]u & Numbers(y,F) & Numbers(x,[F]^-u))›
        using ∃ E[rotated] by blast
      then AOT-obtain u where ‹[F]u & Numbers(y,F) & Numbers(x,[F]^-u)›
        using Ordinary.∃ E[rotated] by meson
      AOT-hence ‹[F]u & [λz Numbers(z,F)]y & [λz Numbers(z,[F]^-u)]x›
        by (auto intro!: &I β←C 0[THEN ∀ E(1)] F−u[den]
              dest: &E intro: cqt:2)
      AOT-hence ‹∃ u([F]u & [λz Numbers(z,F)]y & [λz Numbers(z,[F]^-u)]x)›
        by (rule Ordinary.∃ I)
      AOT-thus ‹∃ F∃ u([F]u & [λz Numbers(z,F)]y & [λz Numbers(z,[F]^-u)]x)›
        by (rule ∃ I)
    qed
```

    }
  qed
  ultimately AOT-have
    ‹□∀ *x* ∀ *y* (∃ *F* ∃ *u* ([*F*]*u* & [λ*z Numbers*(*z*,*F*)]*y* & [λ*z Numbers*(*z*,[*F*]$^{-u}$)]*x*) ≡
        ∃ *F* ∃ *u* ([*F*]*u* & *Numbers*(*y*,*F*) & *Numbers*(*x*,[*F*]$^{-u}$)))›
    using →*E* **by** *blast*
  AOT-thus ‹[λ*xy* ∃ *F* ∃ *u* ([*F*]*u* & *Numbers*(*y*,*F*) & *Numbers*(*x*,[*F*]$^{-u}$))]↓›
    by (*rule safe*−*ext*[*2*][*axiom-inst*, *THEN* →*E*, *OF* &*I*, *rotated*]) *cqt*:*2*
qed

The following is not part of PLM, but a consequence of extended relation comprehension and can be used to *derive* the predecessor axiom.

AOT-theorem *numbers-prop-den*: ‹[λ*x Numbers*(*x*,*G*)]↓›
proof (*rule safe*−*ext*[*axiom-inst*, *THEN* →*E*, *OF* &*I*])
  AOT-show ‹[λ*x A*!*x* & [λ*x* ∀ *F* (*x*[*F*] ≡ [λ*z* ***𝒜***[*F*]*z*] $\approx_E$ *G*)]*x*]↓›
    by *cqt*:*2*
next
  AOT-have *0*: ‹⊢$_\Box$ [λ*x* ∀ *F* (*x*[*F*] ≡ [λ*z* ***𝒜***[*F*]*z*] $\approx_E$ *G*)]↓›
  proof(*safe intro*!: *Comprehension-3*[*THEN* →*E*] →*I RN GEN*)
    AOT-modally-strict {
      fix *F H*
      AOT-assume ‹□*H* $\equiv_E$ *F*›
      AOT-hence ‹□∀ *u* ([*H*]*u* ≡ [*F*]*u*)›
        by (*AOT-subst* (*reverse*) ‹∀ *u* ([*H*]*u* ≡ [*F*]*u*)› ‹*H* $\equiv_E$ *F*›)
          (*safe intro*!: *eqE*[*THEN* ≡*Df*, *THEN* ≡*S*(*1*), *OF* &*I*] *cqt*:*2*)
      AOT-hence ‹∀ *u* □([*H*]*u* ≡ [*F*]*u*)›
        by (*metis Ordinary*.*res*−*var*−*bound*−*reas*[*CBF*] →*E*)
      AOT-hence ‹□([*H*]*u* ≡ [*F*]*u*)› **for** *u*
        using *Ordinary*.∀ *E* **by** *fast*
      AOT-hence ‹***𝒜***([*H*]*u* ≡ [*F*]*u*)› **for** *u*
        by (*metis nec*−*imp*−*act* →*E*)
      AOT-hence ‹***𝒜***([*F*]*u* ≡ [*H*]*u*)› **for** *u*
        by (*metis Act*−*Basic*:*5 Commutativity of* ≡ *intro*−*elim*:*3*:*b*)
      AOT-hence ‹[λ*z* ***𝒜***[*F*]*z*] $\equiv_E$ [λ*z* ***𝒜***[*H*]*z*]›
        by (*safe intro*!: *eqE*[*THEN* $\equiv_{df}$*I*] &*I cqt*:*2 Ordinary*.*GEN*;
          AOT-subst ‹[λ*z* ***𝒜***[*F*]*z*]*u*› ‹***𝒜***[*F*]*u*› **for**: *u F*)
          (*auto intro*!: *beta*−*C*−*meta*[*THEN* →*E*] *cqt*:*2*
            Act−*Basic*:*5*[*THEN* ≡*E*(*1*)])
      AOT-hence ‹[λ*z* ***𝒜***[*F*]*z*] $\approx_E$ [λ*z* ***𝒜***[*H*]*z*]›
        by (*safe intro*!: *apE*−*eqE*:*1*[*unvarify F G*, *THEN* →*E*] *cqt*:*2*)
      AOT-thus ‹[λ*z* ***𝒜***[*F*]*z*] $\approx_E$ *G* ≡ [λ*z* ***𝒜***[*H*]*z*] $\approx_E$ *G*›
        using ≡*I eq*−*part*:*2*[*terms*] *eq*−*part*:*3*[*terms*] →*E* →*I*
        by *metis*
    }
  qed
  AOT-show ‹□∀ *x* (*A*!*x* & [λ*x* ∀ *F* (*x*[*F*] ≡ [λ*z* ***𝒜***[*F*]*z*] $\approx_E$ *G*)]*x* ≡ *Numbers*(*x*,*G*))›
  proof (*safe intro*!: *RN GEN*)
    AOT-modally-strict {
      fix *x*
      AOT-show ‹*A*!*x* & [λ*x* ∀ *F* (*x*[*F*] ≡ [λ*z* ***𝒜***[*F*]*z*] $\approx_E$ *G*)]*x* ≡ *Numbers*(*x*,*G*)›
        by (*AOT-subst-def numbers*; *AOT-subst-thm beta*−*C*−*meta*[*THEN* →*E*, *OF 0*])
          (*auto intro*!: *beta*−*C*−*meta*[*THEN* →*E*, *OF 0*] ≡*I* →*I* &*I cqt*:*2*
            dest: &*E*)
    }
  qed
qed

The two theorems above allow us to derive the predecessor axiom of PLM as theorem.

AOT-theorem *pred*: ‹[λ*xy* ∃ *F*∃ *u* ([*F*]*u* & *Numbers*(*y*,*F*) & *Numbers*(*x*,[*F*]$^{-u}$))]↓›
  using *pred-coex numbers-prop-den*[∀ *I G*] ≡*E* **by** *blast*

AOT-define *Predecessor* :: ‹Π› (‹ℙ›)
  pred−*thm*:*1*:

‹$\mathbb{P} =_{df}$ [λxy $\exists F \exists u$ ([F]u & Numbers(y,F) & Numbers(x,[F]$^{-u}$))]›

AOT-theorem *pred−thm:2*: ‹$\mathbb{P}\downarrow$›
using *pred pred−thm:1 rule−id−df:2:b[zero]* **by** *blast*

AOT-theorem *pred−thm:3*:
‹[$\mathbb{P}$]xy ≡ $\exists F \exists u$ ([F]u & Numbers(y,F) & Numbers(x,[F]$^{-u}$))›
by (*auto intro*!: *beta−C−meta*[*unvarify* $\nu_1\nu_n$, **where** τ=‹(-,-)›, *THEN* →*E*, *rotated*, *OF pred*, *simplified*]
tuple-denotes[*THEN* $\equiv_{df}$*I*] &*I cqt*:*2 pred*
intro: $=_{df}$*I*(*2*)[*OF pred−thm*:*1*])

AOT-theorem *pred−1−1:1*: ‹[$\mathbb{P}$]xy → □[$\mathbb{P}$]xy›
proof(*rule* →*I*)
AOT-assume ‹[$\mathbb{P}$]xy›
AOT-hence ‹$\exists F \exists u$ ([F]u & Numbers(y,F) & Numbers(x,[F]$^{-u}$))›
using ≡*E*(*1*) *pred−thm:3* **by** *fast*
then AOT-obtain *F* **where** ‹$\exists u$ ([F]u & Numbers(y,F) & Numbers(x,[F]$^{-u}$))›
using ∃ *E*[*rotated*] **by** *blast*
then AOT-obtain *u* **where** *props*: ‹[F]u & Numbers(y,F) & Numbers(x,[F]$^{-u}$)›
using *Ordinary*.∃ *E*[*rotated*] **by** *meson*
AOT-obtain *G* **where** *Ridigifies-G-F*: ‹*Rigidifies*(*G*, *F*)›
by (*metis instantiation rigid−der*:*3*)
AOT-hence ξ: ‹□∀ x([G]x → □[G]x)› **and** ζ: ‹∀ x([G]x ≡ [F]x)›
using *df−rigid−rel*:*2*[*THEN* $\equiv_{df}$*E*, *THEN* &*E*(*1*),
THEN $\equiv_{df}$*E*[*OF df−rigid−rel*:*1*], *THEN* &*E*(*2*)]
df−rigid−rel:*2*[*THEN* $\equiv_{df}$*E*, *THEN* &*E*(*2*)] **by** *blast*+

AOT-have *rigid-num-nec*: ‹Numbers(x,F) & Rigidifies(G,F) → □Numbers(x,G)›
for *x G F*
proof(*rule* →*I*; *frule* &*E*(*1*); *drule* &*E*(*2*))
fix *G F x*
AOT-assume *Numbers-xF*: ‹Numbers(x,F)›
AOT-assume ‹Rigidifies(G,F)›
AOT-hence ξ: ‹Rigid(G)› **and** ζ: ‹∀ x([G]x ≡ [F]x)›
using *df−rigid−rel*:*2*[*THEN* $\equiv_{df}$*E*] &*E* **by** *blast*+
AOT-thus ‹□Numbers(x,G)›
proof (*safe intro*!:
num−cont:*2*[*THEN* →*E*, *OF* ξ, *THEN qml*:*2*[*axiom-inst*, *THEN* →*E*],
THEN ∀ *E*(*2*), *THEN* →*E*]
num−tran:*3*[*THEN* →*E*, *THEN* ≡*E*(*1*), *rotated*, *OF Numbers-xF*]
eqE[*THEN* $\equiv_{df}$*I*]
&*I cqt*:*2*[*const-var*][*axiom-inst*] *Ordinary.GEN* →*I*)
AOT-show ‹[F]u ≡ [G]u› **for** *u*
using ζ[*THEN* ∀ *E*(*2*)] **by** (*metis* ≡*E*(*6*) *oth−class−taut*:*3*:*a*)
qed
qed
AOT-have ‹□Numbers(y,G)›
using *rigid-num-nec*[*THEN* →*E*, *OF* &*I*, *OF props*[*THEN* &*E*(*1*), *THEN* &*E*(*2*)],
OF Ridigifies-G-F].
moreover {
AOT-have ‹Rigidifies([G]$^{-u}$, [F]$^{-u}$)›
proof (*safe intro*!: *df−rigid−rel*:*1*[*THEN* $\equiv_{df}$*I*] *df−rigid−rel*:*2*[*THEN* $\equiv_{df}$*I*]
&*I F−u*[*den*] *GEN* ≡*I* →*I*)
AOT-have ‹□∀ x([G]x → □[G]x) → □∀ x([[G]$^{-u}$]x → □[[G]$^{-u}$]x)›
proof (*rule RM*; *safe intro*!: →*I GEN*)
AOT-modally-strict {
fix *x*
AOT-assume *0*: ‹∀ x([G]x → □[G]x)›
AOT-assume *1*: ‹[[G]$^{-u}$]x›
AOT-have ‹[λx [G]x & x $\neq_E$ u]x›
apply (*rule F−u*[*THEN* $=_{df}$*E*(*1*), **where** $\tau_1\tau_n$=(-,-), *simplified*])
apply *cqt*:*2*[*lambda*]

by (*fact 1*)
AOT-hence ‹[G]x & x ≠$_E$ u›
by (*rule β→C(1)*)
AOT-hence *2*: ‹□[G]x› **and** *3*: ‹□x ≠$_E$ u›
using *&E 0[THEN ∀E(2), THEN →E] id−nec4:1 ≡E(1)* **by** *blast+*
AOT-show ‹□[[G]$^{-u}$]x›
apply (*AOT-subst* ‹[[G]$^{-u}$]x› ‹[G]x & x ≠$_E$ u›)
apply (*rule F−u[THEN =$_{df}$I(1)*, **where** *τ$_1$τ$_n$*=(-,-), *simplified*])
apply *cqt:2[lambda]*
apply (*rule beta−C−meta[THEN →E]*)
apply *cqt:2[lambda]*
using *2 3 KBasic:3 ≡S(2) ≡E(2)* **by** *blast*
}
qed
AOT-thus ‹□∀x([[G]$^{-u}$]x → □[[G]$^{-u}$]x)› **using** *ξ →E* **by** *blast*
next
fix *x*
AOT-assume ‹[[G]$^{-u}$]x›
AOT-hence ‹[λx [G]x & x ≠$_E$ u]x›
by (*auto intro: F−u[THEN =$_{df}$E(1)*, **where** *τ$_1$τ$_n$*=(-,-), *simplified*]
intro!: *cqt:2*)
AOT-hence ‹[G]x & x ≠$_E$ u›
by (*rule β→C(1)*)
AOT-hence ‹[F]x & x ≠$_E$ u›
using *ζ &I &E(1) &E(2) ≡E(1) rule−ui:3* **by** *blast*
AOT-hence ‹[λx [F]x & x ≠$_E$ u]x›
by (*auto intro*!: *β←C(1) cqt:2*)
AOT-thus ‹[[F]$^{-u}$]x›
by (*auto intro: F−u[THEN =$_{df}$I(1)*, **where** *τ$_1$τ$_n$*=(-,-), *simplified*]
intro!: *cqt:2*)
next
fix *x*
AOT-assume ‹[[F]$^{-u}$]x›
AOT-hence ‹[λx [F]x & x ≠$_E$ u]x›
by (*auto intro: F−u[THEN =$_{df}$E(1)*, **where** *τ$_1$τ$_n$*=(-,-), *simplified*]
intro!: *cqt:2*)
AOT-hence ‹[F]x & x ≠$_E$ u›
by (*rule β→C(1)*)
AOT-hence ‹[G]x & x ≠$_E$ u›
using *ζ &I &E(1) &E(2) ≡E(2) rule−ui:3* **by** *blast*
AOT-hence ‹[λx [G]x & x ≠$_E$ u]x›
by (*auto intro*!: *β←C(1) cqt:2*)
AOT-thus ‹[[G]$^{-u}$]x›
by (*auto intro: F−u[THEN =$_{df}$I(1)*, **where** *τ$_1$τ$_n$*=(-,-), *simplified*]
intro!: *cqt:2*)
qed
AOT-hence ‹□Numbers(x,[G]$^{-u}$)›
using *rigid-num-nec[unvarify F G, OF F−u[den], OF F−u[den], THEN →E,*
OF &I, OF props[THEN &E(2)]] **by** *blast*
}
moreover AOT-have ‹□[G]u›
using *props[THEN &E(1), THEN &E(1), THEN ζ[THEN ∀E(2), THEN ≡E(2)]]*
ξ[THEN qml:2[axiom-inst, THEN →E], THEN ∀E(2), THEN →E]
by *blast*
ultimately AOT-have ‹□([G]u & Numbers(y,G) & Numbers(x,[G]$^{-u}$))›
by (*metis KBasic:3 &I ≡E(2)*)
AOT-hence ‹∃u (□([G]u & Numbers(y,G) & Numbers(x,[G]$^{-u}$)))›
by (*rule Ordinary.∃I*)
AOT-hence ‹□∃u ([G]u & Numbers(y,G) & Numbers(x,[G]$^{-u}$))›
using *Ordinary.res−var−bound−reas[Buridan] →E* **by** *fast*
AOT-hence ‹∃F □∃u ([F]u & Numbers(y,F) & Numbers(x,[F]$^{-u}$))›
by (*rule ∃I*)
AOT-hence *0*: ‹□∃F∃u ([F]u & Numbers(y,F) & Numbers(x,[F]$^{-u}$))›

```
    using Buridan vdash-properties:10 by fast
  AOT-show ‹□[ℙ]xy›
    by (AOT-subst ‹[ℙ]xy› ‹∃F∃u ([F]u & Numbers(y,F) & Numbers(x,[F]^-u))›;
        simp add: pred-thm:3 0)
qed

AOT-theorem pred-1-1:2: ‹Rigid(ℙ)›
  by (safe intro!: df-rigid-rel:1[THEN ≡df I] pred-thm:2 &I
                   RN tuple-forall[THEN ≡df I];
      safe intro!: GEN pred-1-1:1)

AOT-theorem pred-1-1:3: ‹1-1(ℙ)›
proof (safe intro!: df-1-1:1[THEN ≡df I] pred-thm:2 &I GEN →I;
       frule &E(1); drule &E(2))
  fix x y z
  AOT-assume ‹[ℙ]xz›
  AOT-hence ‹∃F∃u ([F]u & Numbers(z,F) & Numbers(x,[F]^-u))›
    using pred-thm:3[THEN ≡E(1)] by blast
  then AOT-obtain F where ‹∃u ([F]u & Numbers(z,F) & Numbers(x,[F]^-u))›
    using ∃E[rotated] by blast
  then AOT-obtain u where u-prop: ‹[F]u & Numbers(z,F) & Numbers(x,[F]^-u)›
    using Ordinary.∃E[rotated] by meson
  AOT-assume ‹[ℙ]yz›
  AOT-hence ‹∃F∃u ([F]u & Numbers(z,F) & Numbers(y,[F]^-u))›
    using pred-thm:3[THEN ≡E(1)] by blast
  then AOT-obtain G where ‹∃u ([G]u & Numbers(z,G) & Numbers(y,[G]^-u))›
    using ∃E[rotated] by blast
  then AOT-obtain v where v-prop: ‹[G]v & Numbers(z,G) & Numbers(y,[G]^-v)›
    using Ordinary.∃E[rotated] by meson
  AOT-show ‹x = y›
  proof (rule pre-Hume[unvarify G H, OF F-u[den], OF F-u[den],
                        THEN →E, OF &I, THEN ≡E(2)])
    AOT-show ‹Numbers(x, [F]^-u)›
      using u-prop &E by blast
  next
    AOT-show ‹Numbers(y, [G]^-v)›
      using v-prop &E by blast
  next
    AOT-have ‹F ≈E G›
      using u-prop[THEN &E(1), THEN &E(2)]
      using v-prop[THEN &E(1), THEN &E(2)]
      using num-tran:2[THEN →E, OF &I] by blast
    AOT-thus ‹[F]^-u ≈E [G]^-v›
      using u-prop[THEN &E(1), THEN &E(1)]
      using v-prop[THEN &E(1), THEN &E(1)]
      using eqP'[THEN →E, OF &I, OF &I]
      by blast
  qed
qed

AOT-theorem pred-1-1:4: ‹Rigid1-1(ℙ)›
  by (meson ≡df I &I df-1-1:2 pred-1-1:2 pred-1-1:3)

AOT-theorem assume-anc:1:
  ‹[ℙ]* = [λxy ∀F((∀z([ℙ]xz → [F]z) & Hereditary(F,ℙ)) → [F]y)]›
  apply (rule =df I(1)[OF ances-df])
   apply cqt:2[lambda]
  apply (rule =I(1))
  by cqt:2[lambda]

AOT-theorem assume-anc:2: ‹ℙ*↓›
  using t=t-proper:1 assume-anc:1 vdash-properties:10 by blast
```

AOT-theorem *assume−anc*:*3*:
‹$[\mathbb{P}^*]xy \equiv \forall F((\forall z([\mathbb{P}]xz \to [F]z)$ & $\forall x'\forall y'([\mathbb{P}]x'y' \to ([F]x' \to [F]y'))) \to [F]y)$›
proof −
AOT-have *prod-den*: ‹$\vdash_\Box$ «(*AOT-term-of-var* x_1,*AOT-term-of-var* x_2)»↓›
for x_1 x_2 :: ‹κ *AOT-var*›
by (*simp add*: &*I ex*:*1*:*a prod-denotesI rule−ui*:*3*)
AOT-have *den*: ‹$[\lambda xy\ \forall F((\forall z([\mathbb{P}]xz \to [F]z)$ & *Hereditary*$(F,\mathbb{P})) \to [F]y)]\downarrow$›
by *cqt*:*2*[*lambda*]
AOT-have *1*: ‹$[\mathbb{P}^*]xy \equiv \forall F((\forall z([\mathbb{P}]xz \to [F]z)$ & *Hereditary*$(F,\mathbb{P})) \to [F]y)$›
apply (*rule rule*=*E*[*rotated*, *OF assume−anc*:*1*[*symmetric*]])
by (*rule beta−C−meta*[*unvarify* $\nu_1\nu_n$, *OF prod-den*, *THEN* →*E*,
simplified, *OF den*, *simplified*])
show *?thesis*
apply (*AOT-subst* (*reverse*) ‹$\forall x'\forall y'$ $([\mathbb{P}]x'y' \to ([F]x' \to [F]y'))$›
‹*Hereditary*$(F,\mathbb{P})$› **for**: F :: ‹<κ>›)
using *hered*:*1*[*THEN* ≡*Df*, *THEN* ≡*S*(*1*), *OF* &*I*, *OF pred−thm*:*2*,
OF cqt:*2*[*const-var*][*axiom-inst*]] **apply** *blast*
by (*fact 1*)
qed

AOT-theorem *no−pred−0*:*1*: ‹$\neg\exists x\ [\mathbb{P}]x\ 0$›
proof(*rule raa−cor*:*2*)
AOT-assume ‹$\exists x\ [\mathbb{P}]x\ 0$›
then AOT-obtain *a* **where** ‹$[\mathbb{P}]a\ 0$›
using ∃*E*[*rotated*] **by** *blast*
AOT-hence ‹$\exists F\exists u\ ([F]u$ & *Numbers*$(0, F)$ & *Numbers*$(a, [F]^{-u}))$›
using *pred−thm*:*3*[*unvarify y*, *OF zero*:*2*, *THEN* ≡*E*(*1*)] **by** *blast*
then AOT-obtain *F* **where** ‹$\exists u\ ([F]u$ & *Numbers*$(0, F)$ & *Numbers*$(a, [F]^{-u}))$›
using ∃*E*[*rotated*] **by** *blast*
then AOT-obtain *u* **where** ‹$[F]u$ & *Numbers*$(0, F)$ & *Numbers*$(a, [F]^{-u})$›
using *Ordinary*.∃*E*[*rotated*] **by** *meson*
AOT-hence ‹$[F]u$› **and** *num0-F*: ‹*Numbers*$(0, F)$›
using &*E* &*I* **by** *blast+*
AOT-hence ‹$\exists u\ [F]u$›
using *Ordinary*.∃*I* **by** *fast*
moreover AOT-have ‹$\neg\exists u\ [F]u$›
using *num0-F* ≡*E*(*2*) *0F*:*1* **by** *blast*
ultimately AOT-show ‹p & $\neg p$› **for** p
by (*metis raa−cor*:*3*)
qed

AOT-theorem *no−pred−0*:*2*: ‹$\neg\exists x\ [\mathbb{P}^*]x\ 0$›
proof(*rule raa−cor*:*2*)
AOT-assume ‹$\exists x\ [\mathbb{P}^*]x\ 0$›
then AOT-obtain *a* **where** ‹$[\mathbb{P}^*]a\ 0$›
using ∃*E*[*rotated*] **by** *blast*
AOT-hence ‹$\exists z\ [\mathbb{P}]z\ 0$›
using *anc−her*:*5*[*unvarify R y*, *OF zero*:*2*,
OF pred−thm:*2*, *THEN* →*E*] **by** *auto*
AOT-thus ‹$\exists z\ [\mathbb{P}]z\ 0$ & $\neg\exists z\ [\mathbb{P}]z\ 0$›
by (*metis no−pred−0*:*1 raa−cor*:*3*)
qed

AOT-theorem *no−pred−0*:*3*: ‹$\neg[\mathbb{P}^*]0\ 0$›
by (*metis existential*:*1 no−pred−0*:*2 reductio−aa*:*1 zero*:*2*)

AOT-theorem *assume1*:*1*: ‹$(=_\mathbb{P}) = [\lambda xy\ \exists z\ ([\mathbb{P}]xz$ & $[\mathbb{P}]yz)]$›
apply (*rule* $=_{df}I(1)$[*OF id−d−R*])
apply *cqt*:*2*[*lambda*]
apply (*rule* =*I*(*1*))
by *cqt*:*2*[*lambda*]

AOT-theorem *assume1*:*2*: ‹$x =_\mathbb{P} y \equiv \exists z\ ([\mathbb{P}]xz$ & $[\mathbb{P}]yz)$›

proof (*rule rule=E*[*rotated*, *OF assume1*:*1*[*symmetric*]])
AOT-have *prod-den*: ‹$\vdash_\Box$ «(*AOT-term-of-var* x_1,*AOT-term-of-var* x_2)»↓›
for x_1 x_2 :: ‹κ *AOT-var*›
by (*simp add*: &*I ex*:*1*:*a prod-denotesI rule*−*ui*:*3*)
AOT-have *1*: ‹[λxy ∃ *z* ([ℙ]*xz* & [ℙ]*yz*)]↓›
by *cqt*:*2*
AOT-show ‹[λxy ∃ *z* ([ℙ]*xz* & [ℙ]*yz*)]*xy* ≡ ∃ *z* ([ℙ]*xz* & [ℙ]*yz*)›
using *beta*−*C*−*meta*[*THEN* →*E*, *OF 1*, *unvarify* $\nu_1\nu_n$,
OF prod-den, *simplified*] **by** *blast*
qed

AOT-theorem *assume1*:*3*: ‹$[\mathbb{P}]^+$ = [λxy $[\mathbb{P}]^*xy$ ∨ $x =_{\mathbb{P}} y$]›
apply (*rule* $=_{df}I(1)$[*OF w*−*ances*−*df*])
apply (*simp add*: *w*−*ances*−*df*[*den1*])
apply (*rule rule=E*[*rotated*, *OF assume1*:*1*[*symmetric*]])
apply (*rule* $=_{df}I(1)$[*OF id*−*d*−*R*])
apply *cqt*:*2*[*lambda*]
apply (*rule* =*I*(*1*))
by *cqt*:*2*[*lambda*]

AOT-theorem *assume1*:*4*: ‹$[\mathbb{P}]^+$↓›
using *w*−*ances*−*df*[*den2*].

AOT-theorem *assume1*:*5*: ‹$[\mathbb{P}]^+xy$ ≡ $[\mathbb{P}]^*xy$ ∨ $x =_{\mathbb{P}} y$›
proof −
AOT-have *0*: ‹[λxy $[\mathbb{P}]^*xy$ ∨ $x =_{\mathbb{P}} y$]↓› **by** *cqt*:*2*
AOT-have *prod-den*: ‹$\vdash_\Box$ «(*AOT-term-of-var* x_1, *AOT-term-of-var* x_2)»↓›
for x_1 x_2 :: ‹κ *AOT-var*›
by (*simp add*: &*I ex*:*1*:*a prod-denotesI rule*−*ui*:*3*)
show *?thesis*
apply (*rule rule=E*[*rotated*, *OF assume1*:*3*[*symmetric*]])
using *beta*−*C*−*meta*[*THEN* →*E*, *OF 0*, *unvarify* $\nu_1\nu_n$, *OF prod-den*, *simplified*]
by (*simp add*: *cond-case-prod-eta*)
qed

AOT-define *NaturalNumber* :: ‹τ› (‹ℕ›)
nnumber:*1*: ‹ℕ $=_{df}$ [λx $[\mathbb{P}]^+$ *0x*]›

AOT-theorem *nnumber*:*2*: ‹ℕ↓›
by (*rule* $=_{df}I(2)$[*OF nnumber*:*1*]; *cqt*:*2*[*lambda*])

AOT-theorem *nnumber*:*3*: ‹[ℕ]*x* ≡ $[\mathbb{P}]^+$ *0x*›
apply (*rule* $=_{df}I(2)$[*OF nnumber*:*1*])
apply *cqt*:*2*[*lambda*]
apply (*rule beta*−*C*−*meta*[*THEN* →*E*])
by *cqt*:*2*[*lambda*]

AOT-theorem *0*−*n*: ‹[ℕ]*0*›
proof (*safe intro*!: *nnumber*:*3*[*unvarify x*, *OF zero*:*2*, *THEN* ≡*E*(*2*)]
assume1:*5*[*unvarify x y*, *OF zero*:*2*, *OF zero*:*2*, *THEN* ≡*E*(*2*)]
∨*I*(*2*) *assume1*:*2*[*unvarify x y*, *OF zero*:*2*, *OF zero*:*2*, *THEN* ≡*E*(*2*)])
fix *u*
AOT-have *den*: ‹[λx *O*!*x* & $x =_E u$]↓› **by** *cqt*:*2*[*lambda*]
AOT-obtain *a* **where** *a-prop*: ‹*Numbers*(*a*, [λx *O*!*x* & $x =_E u$])›
using *num*:*1*[*unvarify G*, *OF den*] ∃ *E*[*rotated*] **by** *blast*
AOT-have ‹[ℙ]*0a*›
proof (*safe intro*!: *pred*−*thm*:*3*[*unvarify x*, *OF zero*:*2*, *THEN* ≡*E*(*2*)]
∃ *I*(*1*)[**where** τ=‹«[λx *O*!*x* & $x =_E u$]»›]
Ordinary.∃ *I*[**where** β=*u*] &*I den*
0F:*1*[*unvarify F*, *OF F*−*u*[*den*], *unvarify F*,
OF den, *THEN* ≡*E*(*1*)])
AOT-show ‹[λx [*O*!]*x* & $x =_E u$]*u*›
by (*auto intro*!: β←*C*(*1*) *cqt*:*2* &*I ord=Eequiv*:*1*[*THEN* →*E*]

```
                Ordinary.ψ)
  next
    AOT-show ‹Numbers(a,[λx [O!]x & x =E u])›
      using a-prop.
  next
    AOT-show ‹¬∃ v [[λx [O!]x & x =E u]⁻ᵘ]v›
    proof(rule raa−cor:2)
      AOT-assume ‹∃ v [[λx [O!]x & x =E u]⁻ᵘ]v›
      then AOT-obtain v where ‹[[λx [O!]x & x =E u]⁻ᵘ]v›
        using Ordinary.∃ E[rotated] &E by blast
      AOT-hence ‹[λz [λx [O!]x & x =E u]z & z ≠E u]v›
        apply (rule F−u[THEN =df E(1), where τ1τn=(-,-), simplified, rotated])
        by cqt:2[lambda]
      AOT-hence ‹[λx [O!]x & x =E u]v & v ≠E u›
        by (rule β→C(1))
      AOT-hence ‹v =E u› and ‹v ≠E u›
        using β→C(1) &E by blast+
      AOT-hence ‹v =E u & ¬(v =E u)›
        by (metis ≡E(4) reductio−aa:1 thm−neg=E)
      AOT-thus ‹p & ¬p› for p
        by (metis raa−cor:1)
    qed
  qed
  AOT-thus ‹∃ z ([ℙ]0z & [ℙ]0z)›
    by (safe intro!: &I ∃ I(2)[where β=a])
qed

AOT-theorem mod−col−num:1: ‹[ℕ]x → □[ℕ]x›
proof(rule →I)
  AOT-have nec0N: ‹[λx □[ℕ]x]0›
    by (auto intro!: β←C(1) cqt:2 simp: zero:2 RN 0−n)
  AOT-have 1: ‹[λx □[ℕ]x]0 &
    ∀ x∀ y ([[ℙ]⁺]0x & [[ℙ]⁺]0y → ([ℙ]xy → ([λx □[ℕ]x]x → [λx □[ℕ]x]y))) →
    ∀ x ([[ℙ]⁺]0x → [λx □[ℕ]x]x)›
    by (auto intro!: cqt:2
            intro: pre−ind[unconstrain ℛ, unvarify β, OF pred−thm:2,
                           THEN →E, OF pred−1−1:4, unvarify z, OF zero:2,
                           unvarify F])
  AOT-have ‹∀ x ([[ℙ]⁺]0x → [λx □[ℕ]x]x)›
  proof (rule 1[THEN →E]; safe intro!: &I GEN →I nec0N;
        frule &E(1); drule &E(2))
    fix x y
    AOT-assume ‹[ℙ]xy›
    AOT-hence 0: ‹□[ℙ]xy›
      by (metis pred−1−1:1 →E)
    AOT-assume ‹[λx □[ℕ]x]x›
    AOT-hence ‹□[ℕ]x›
      by (rule β→C(1))
    AOT-hence ‹□([ℙ]xy & [ℕ]x)›
      by (metis 0 KBasic:3 Adjunction ≡E(2) →E)
    moreover AOT-have ‹□([ℙ]xy & [ℕ]x) → □[ℕ]y›
    proof (rule RM; rule →I; frule &E(1); drule &E(2))
      AOT-modally-strict {
        AOT-assume 0: ‹[ℙ]xy›
        AOT-assume ‹[ℕ]x›
        AOT-hence 1: ‹[[ℙ]⁺]0x›
          by (metis ≡E(1) nnumber:3)
        AOT-show ‹[ℕ]y›
          apply (rule nnumber:3[THEN ≡E(2)])
          apply (rule assume1:5[unvarify x, OF zero:2, THEN ≡E(2)])
          apply (rule ∨I(1))
          apply (rule w−ances−her:3[unconstrain ℛ, unvarify β, OF pred−thm:2,
                                THEN →E, OF pred−1−1:4, unvarify x,
```

```
                              OF zero:2, THEN →E])
        apply (rule &I)
         apply (fact 1)
        by (fact 0)
      }
    qed
    ultimately AOT-have ‹□[ℕ]y›
      by (metis →E)
    AOT-thus ‹[λx □[ℕ]x]y›
      by (auto intro!: β←C(1) cqt:2)
  qed
  AOT-hence 0: ‹[[ℙ]+]0x → [λx □[ℕ]x]x›
    using ∀E(2) by blast
  AOT-assume ‹[ℕ]x›
  AOT-hence ‹[[ℙ]+]0x›
    by (metis ≡E(1) nnumber:3)
  AOT-hence ‹[λx □[ℕ]x]x›
    using 0[THEN →E] by blast
  AOT-thus ‹□[ℕ]x›
    by (rule β→C(1))
qed

AOT-theorem mod−col−num:2: ‹Rigid(ℕ)›
  by (safe intro!: df−rigid−rel:1[THEN ≡df I] &I RN GEN
                   mod−col−num:1 nnumber:2)

AOT-register-rigid-restricted-type
  Number: ‹[ℕ]κ›
proof
  AOT-modally-strict {
    AOT-show ‹∃x [ℕ]x›
      by (rule ∃I(1)[where τ=‹«0»›]; simp add: 0−n zero:2)
  }
next
  AOT-modally-strict {
    AOT-show ‹[ℕ]κ → κ↓› for κ
      by (simp add: →I cqt:5:a[1][axiom-inst, THEN →E, THEN &E(2)])
  }
next
  AOT-modally-strict {
    AOT-show ‹∀x([ℕ]x → □[ℕ]x)›
      by (simp add: GEN mod−col−num:1)
  }
qed
AOT-register-variable-names
  Number: m n k i j

AOT-theorem 0−pred: ‹¬∃n [ℙ]n 0›
proof (rule raa−cor:2)
  AOT-assume ‹∃n [ℙ]n 0›
  then AOT-obtain n where ‹[ℙ]n 0›
    using Number.∃E[rotated] by meson
  AOT-hence ‹∃x [ℙ]x 0›
    using &E ∃I by fast
  AOT-thus ‹∃x [ℙ]x 0 & ¬∃x [ℙ]x 0›
    using no−pred−0:1 &I by auto
qed

AOT-theorem no−same−succ:
  ‹∀n∀m∀k([ℙ]nk & [ℙ]mk → n = m)›
proof(safe intro!: Number.GEN →I)
  fix n m k
  AOT-assume ‹[ℙ]nk & [ℙ]mk›
```

```
  AOT-thus ‹n = m›
    by (safe intro!: cqt:2[const-var][axiom-inst] df−1−1:3[
          unvarify R, OF pred−thm:2,
          THEN →E, OF pred−1−1:4, THEN qml:2[axiom-inst, THEN →E],
          THEN ≡df E[OF df−1−1:1], THEN &E(2), THEN ∀E(1), THEN ∀E(1),
          THEN ∀E(1)[where τ=‹AOT-term-of-var (Number.Rep k)›], THEN →E])
qed

AOT-theorem induction:
  ‹∀F([F]0 & ∀n∀m([ℙ]nm → ([F]n → [F]m)) → ∀n[F]n)›
proof (safe intro!: GEN[where 'a=‹<κ>›] Number.GEN &I →I;
       frule &E(1); drule &E(2))
  fix F n
  AOT-assume F0: ‹[F]0›
  AOT-assume 0: ‹∀n∀m([ℙ]nm → ([F]n → [F]m))›
  {
    fix x y
    AOT-assume ‹[[ℙ]+]0x & [[ℙ]+]0y›
    AOT-hence ‹[ℕ]x› and ‹[ℕ]y›
      using &E ≡E(2) nnumber:3 by blast+
    moreover AOT-assume ‹[ℙ]xy›
    moreover AOT-assume ‹[F]x›
    ultimately AOT-have ‹[F]y›
      using 0[THEN ∀E(2), THEN →E, THEN ∀E(2), THEN →E,
            THEN →E, THEN →E] by blast
  } note 1 = this
  AOT-have 0: ‹[[ℙ]+]0n›
    by (metis ≡E(1) nnumber:3 Number.ψ)
  AOT-show ‹[F]n›
    apply (rule pre−ind[unconstrain R, unvarify β, THEN →E, OF pred−thm:2,
                        OF pred−1−1:4, unvarify z, OF zero:2, THEN →E,
                        THEN ∀E(2), THEN →E];
          safe intro!: 0 &I GEN →I F0)
    using 1 by blast
qed

AOT-theorem suc−num:1: ‹[ℙ]nx → [ℕ]x›
proof(rule →I)
  AOT-have ‹[[ℙ]+]0 n›
    by (meson Number.ψ ≡E(1) nnumber:3)
  moreover AOT-assume ‹[ℙ]nx›
  ultimately AOT-have ‹[[ℙ]*]0 x›
    using w−ances−her:3[unconstrain R, unvarify β, OF pred−thm:2, THEN →E,
                        OF pred−1−1:4, unvarify x, OF zero:2,
                        THEN →E, OF &I]
    by blast
  AOT-hence ‹[[ℙ]+]0 x›
    using assume1:5[unvarify x, OF zero:2, THEN ≡E(2), OF ∨I(1)]
    by blast
  AOT-thus ‹[ℕ]x›
    by (metis ≡E(2) nnumber:3)
qed

AOT-theorem suc−num:2: ‹[[ℙ]*]nx → [ℕ]x›
proof(rule →I)
  AOT-have ‹[[ℙ]+]0 n›
    using Number.ψ ≡E(1) nnumber:3 by blast
  AOT-assume ‹[[ℙ]*]n x›
  AOT-hence ‹∀F (∀z ([ℙ]nz → [F]z) & ∀x'∀y' ([ℙ]x'y' → ([F]x' → [F]y')) → [F]x)›
    using assume−anc:3[THEN ≡E(1)] by blast
  AOT-hence ϑ: ‹∀z ([ℙ]nz → [ℕ]z) & ∀x'∀y' ([ℙ]x'y' → ([ℕ]x' → [ℕ]y')) → [ℕ]x›
    using ∀E(1) nnumber:2 by blast
  AOT-show ‹[ℕ]x›
```

proof (*safe intro*!: ϑ[*THEN* →*E*] *GEN* →*I* &*I*)
  AOT-show ‹[ℕ]*z*› **if** ‹[ℙ]*nz*› **for** *z*
    using *Number.ψ suc−num*:*1 that* →*E* **by** *blast*
  next
  AOT-show ‹[ℕ]*y*› **if** ‹[ℙ]*xy*› **and** ‹[ℕ]*x*› **for** *x y*
    using *suc−num*:*1*[*unconstrain n*, *THEN* →*E*] *that* →*E* **by** *blast*
  qed
qed

AOT-theorem *suc−num*:*3*: ‹[ℙ]$^+$*nx* → [ℕ]*x*›
proof (*rule* →*I*)
  AOT-assume ‹[ℙ]$^+$*nx*›
  AOT-hence ‹[ℙ]**nx* ∨ *n* =$_ℙ$ *x*›
    by (*metis assume1*:*5* ≡*E*(*1*))
  moreover {
    AOT-assume ‹[ℙ]**nx*›
    AOT-hence ‹[ℕ]*x*›
      by (*metis suc−num*:*2* →*E*)
  }
  moreover {
    AOT-assume ‹*n* =$_ℙ$ *x*›
    AOT-hence ‹*n* = *x*›
      using *id−R−thm*:*3*[*unconstrain* $\mathcal{R}$, *unvarify β*, *OF pred−thm*:*2*,
            THEN →*E*, *OF pred−1−1*:*4*, *THEN* →*E*] **by** *blast*
    AOT-hence ‹[ℕ]*x*›
      by (*metis rule*=*E Number.ψ*)
  }
  ultimately AOT-show ‹[ℕ]*x*›
    by (*metis* ∨*E*(*3*) *reductio−aa*:*1*)
qed

AOT-theorem *pred−num*: ‹[ℙ]*xn* → [ℕ]*x*›
proof (*rule* →*I*)
  AOT-assume *0*: ‹[ℙ]*xn*›
  AOT-have ‹[[ℙ]$^+$]*0 n*›
    using *Number.ψ* ≡*E*(*1*) *nnumber*:*3* **by** *blast*
  AOT-hence ‹[[ℙ]*]*0 n* ∨ *0* =$_ℙ$ *n*›
    using *assume1*:*5*[*unvarify x*, *OF zero*:*2*] **by** (*metis* ≡*E*(*1*))
  moreover {
    AOT-assume ‹*0* =$_ℙ$ *n*›
    AOT-hence ‹∃ *z* ([ℙ]*0z* & [ℙ]*nz*)›
      using *assume1*:*2*[*unvarify x*, *OF zero*:*2*, *THEN* ≡*E*(*1*)] **by** *blast*
    then AOT-obtain *a* **where** ‹[ℙ]*0a* & [ℙ]*na*› **using** ∃ *E*[*rotated*] **by** *blast*
    AOT-hence ‹*0* = *n*›
      using *pred−1−1*:*3*[*THEN df−1−1*:*1*[*THEN* ≡$_{df}$*E*], *THEN* &*E*(*2*),
            THEN ∀ *E*(*1*), *OF zero*:*2*, *THEN* ∀ *E*(*2*),
            THEN ∀ *E*(*2*), *THEN* →*E*] **by** *blast*
    AOT-hence ‹[ℙ]*x 0*›
      using *0 rule*=*E id-sym* **by** *fast*
    AOT-hence ‹∃ *x* [ℙ]*x 0*›
      by (*rule* ∃ *I*)
    AOT-hence ‹∃ *x* [ℙ]*x 0* & ¬∃ *x* [ℙ]*x 0*›
      by (*metis no−pred−0*:*1 raa−cor*:*3*)
  }
  ultimately AOT-have ‹[[ℙ]*]*0n*›
    by (*metis* ∨*E*(*3*) *raa−cor*:*1*)
  AOT-hence ‹∃ *z* ([[ℙ]$^+$]*0z* & [ℙ]*zn*)›
    using *w−ances−her*:*7*[*unconstrain* $\mathcal{R}$, *unvarify β*, *OF pred−thm*:*2*,
            THEN →*E*, *OF pred−1−1*:*4*, *unvarify x*,
            OF zero:*2*, *THEN* →*E*] **by** *blast*
  then AOT-obtain *b* **where** *b-prop*: ‹[[ℙ]$^+$]*0b* & [ℙ]*bn*›
    using ∃ *E*[*rotated*] **by** *blast*
  AOT-hence ‹[ℕ]*b*›

```
    by (metis &E(1) ≡E(2) nnumber:3)
  moreover AOT-have ‹x = b›
    using pred−1−1:3[THEN df−1−1:1[THEN ≡_{df}E], THEN &E(2),
                     THEN ∀E(2), THEN ∀E(2), THEN ∀E(2), THEN →E,
                     OF &I, OF 0, OF b-prop[THEN &E(2)]].
  ultimately AOT-show ‹[ℕ]x›
    using rule=E id-sym by fast
qed

AOT-theorem nat−card: ‹[ℕ]x → NaturalCardinal(x)›
proof(rule →I)
  AOT-assume ‹[ℕ]x›
  AOT-hence ‹[[ℙ]^{+}]0x›
    by (metis ≡E(1) nnumber:3)
  AOT-hence ‹[[ℙ]^{*}]0x ∨ 0 =_ℙ x›
    using assume1:5[unvarify x, OF zero:2, THEN ≡E(1)] by blast
  moreover {
    AOT-assume ‹[[ℙ]^{*}]0x›
    then AOT-obtain a where ‹[ℙ]ax›
      using anc−her:5[unvarify R x, OF zero:2, OF pred−thm:2, THEN →E]
          ∃E[rotated] by blast
    AOT-hence ‹∃F∃u ([F]u & Numbers(x,F) & Numbers(a,[F]^{−u}))›
      using pred−thm:3[THEN ≡E(1)] by blast
    then AOT-obtain F where ‹∃u ([F]u & Numbers(x,F) & Numbers(a,[F]^{−u}))›
      using ∃E[rotated] by blast
    then AOT-obtain u where ‹[F]u & Numbers(x,F) & Numbers(a,[F]^{−u})›
      using Ordinary.∃E[rotated] by meson
    AOT-hence ‹NaturalCardinal(x)›
      using eq−num:6[THEN →E] &E by blast
  }
  moreover {
    AOT-assume ‹0 =_ℙ x›
    AOT-hence ‹0 = x›
      using id−R−thm:3[unconstrain ℛ, unvarify β, OF pred−thm:2,
                       THEN →E, OF pred−1−1:4, unvarify x,
                       OF zero:2, THEN →E] by blast
    AOT-hence ‹NaturalCardinal(x)›
      by (metis rule=E zero−card)
  }
  ultimately AOT-show ‹NaturalCardinal(x)›
    by (metis ∨E(2) raa−cor:1)
qed

AOT-theorem pred−func:1: ‹[ℙ]xy & [ℙ]xz → y = z›
proof (rule →I; frule &E(1); drule &E(2))
  AOT-assume ‹[ℙ]xy›
  AOT-hence ‹∃F∃u ([F]u & Numbers(y,F) & Numbers(x,[F]^{−u}))›
    using pred−thm:3[THEN ≡E(1)] by blast
  then AOT-obtain F where ‹∃u ([F]u & Numbers(y,F) & Numbers(x,[F]^{−u}))›
    using ∃E[rotated] by blast
  then AOT-obtain a where
          Oa: ‹O!a›
    and a-prop: ‹[F]a & Numbers(y,F) & Numbers(x,[F]^{−a})›
    using ∃E[rotated] &E by blast
  AOT-assume ‹[ℙ]xz›
  AOT-hence ‹∃F∃u ([F]u & Numbers(z,F) & Numbers(x,[F]^{−u}))›
    using pred−thm:3[THEN ≡E(1)] by blast
  then AOT-obtain G where ‹∃u ([G]u & Numbers(z,G) & Numbers(x,[G]^{−u}))›
    using ∃E[rotated] by blast
  then AOT-obtain b where Ob: ‹O!b›
                 and b-prop: ‹[G]b & Numbers(z,G) & Numbers(x,[G]^{−b})›
    using ∃E[rotated] &E by blast
  AOT-have ‹[F]^{−a} ≈_E [G]^{−b}›
```

```
    using num−tran:2[unvarify G H, OF F−u[den], OF F−u[den],
                     THEN →E, OF &I, OF a-prop[THEN &E(2)],
                     OF b-prop[THEN &E(2)]].
  AOT-hence ‹F ≈E G›
    using P′−eq[unconstrain u, THEN →E, OF Oa, unconstrain v, THEN →E,
                OF Ob, THEN →E, OF &I, OF &I]
          a-prop[THEN &E(1), THEN &E(1)]
          b-prop[THEN &E(1), THEN &E(1)] by blast
  AOT-thus ‹y = z›
    using pre−Hume[THEN →E, THEN ≡E(2), OF &I,
                   OF a-prop[THEN &E(1), THEN &E(2)],
                   OF b-prop[THEN &E(1), THEN &E(2)]]
    by blast
qed

AOT-theorem pred−func:2: ‹[ℙ]nm & [ℙ]nk → m = k›
  using pred−func:1.

AOT-theorem being-number-of-den: ‹[λx x = #G]↓›
proof (rule safe−ext[axiom-inst, THEN →E]; safe intro!: &I GEN RN)
  AOT-show ‹[λx Numbers(x,[λz 𝓐[G]z])]↓›
    by (rule numbers-prop-den[unvarify G]) cqt:2[lambda]
next
  AOT-modally-strict {
    AOT-show ‹Numbers(x,[λz 𝓐[G]z]) ≡ x = #G› for x
      using eq−num:2.
  }
qed

axiomatization ω-nat :: ‹ω ⇒ nat› where ω-nat: ‹surj ω-nat›
```

Unfortunately, since the axiom requires the type ω to have an infinite domain, **nitpick** can only find a potential model and no genuine model. However, since we could trivially choose ω as a copy of *nat*, we can still be assured that above axiom is consistent.

```
lemma ‹True› nitpick[satisfy, user-axioms, card nat=1, expect = potential] ..

AOT-axiom modal−axiom:
  ‹∃x([ℕ]x & x = #G) → ◇∃y([E!]y & ∀u (𝓐[G]u → u ≠E y))›
proof(rule AOT-model-axiomI) AOT-modally-strict {
```

The actual extension on the ordinary objects of a property is the set of ordinary urelements that exemplifies the property in the designated actual world.

```
  define act-ωext :: ‹<κ> ⇒ ω set› where
    ‹act-ωext ≡ λ Π . {x :: ω . [w0 ⊨ [Π]«ωκ x»]}›
```

Encoding a property with infinite actual extension on the ordinary objects denotes a property by extended relation comprehension.

```
  AOT-have enc-finite-act-ωext-den:
    ‹⊢□ [λx ∃F(¬«εo w. finite (act-ωext F)» & x[F])]↓›
  proof(safe intro!: Comprehension-1[THEN →E] RN GEN →I)
    AOT-modally-strict {
      fix F G
      AOT-assume ‹□G ≡E F›
      AOT-hence ‹𝓐G ≡E F›
        using nec−imp−act[THEN →E] by blast
      AOT-hence ‹𝓐(G↓ & F↓ & ∀u([G]u ≡ [F]u))›
        by (AOT-subst-def (reverse) eqE)
      hence ‹[w0 ⊨ [G]«ωκ x»] = [w0 ⊨ [F]«ωκ x»]› for x
        by (auto dest!: ∀E(1) →E
                 simp: AOT-model-denotes-κ-def AOT-sem-denotes AOT-sem-conj
                       AOT-model-ωκ-ordinary AOT-sem-act AOT-sem-equiv)
      AOT-thus ‹¬«εo w. finite (act-ωext (AOT-term-of-var F))» ≡
```

¬«$\varepsilon_\circ$ *w. finite* (*act-ωext* (*AOT-term-of-var G*))»›
by (*simp add*: *AOT-sem-not AOT-sem-equiv act-ωext-def*
AOT-model-proposition-choice-simp)
}
qed

By coexistence, encoding only properties with finite actual extension on the ordinary objects denotes.

AOT-have ‹[λx ∀F(x[F] → «$\varepsilon_\circ$ *w. finite* (*act-ωext F*)»)]↓›
proof(*rule safe−ext*[*axiom-inst*, *THEN* →*E*]; *safe intro*!: &*I RN GEN*)
AOT-show ‹[λx ¬[λx ∃F(¬«$\varepsilon_\circ$ *w. finite* (*act-ωext F*)» & x[F])]x]↓›
by *cqt*:*2*
next
AOT-modally-strict {
fix *x*
AOT-show ‹¬[λx ∃F (¬«$\varepsilon_\circ$ *w. finite* (*act-ωext F*)» & x[F])]x ≡
∀F(x[F] → «$\varepsilon_\circ$ *w. finite* (*act-ωext F*)»)›
by (*AOT-subst* ‹[λx ∃F (¬«$\varepsilon_\circ$ *w. finite* (*act-ωext F*)» & x[F])]x›
‹∃F (¬«$\varepsilon_\circ$ *w. finite* (*act-ωext F*)» & x[F])›;
(*rule beta−C−meta*[*THEN* →*E*])?)
(*auto simp*: *enc-finite-act-ωext-den AOT-sem-equiv AOT-sem-not*
AOT-sem-forall AOT-sem-imp AOT-sem-conj AOT-sem-exists)
}
qed

We show by induction that any property encoded by a natural number has a finite actual extension on the ordinary objects.

AOT-hence ‹[λx ∀F(x[F] → «$\varepsilon_\circ$ *w. finite* (*act-ωext F*)»)]n› **for** *n*
proof(*rule induction*[*THEN* ∀*E*(*1*), *THEN* →*E*, *THEN Number*.∀*E*];
safe intro!: &*I Number*.*GEN* β←*C zero*:*2* →*I cqt*:*2*
dest!: β→*C*)
AOT-show ‹∀F(*0*[F] → «$\varepsilon_\circ$ *w. finite* (*act-ωext F*)»)›
proof(*safe intro*!: *GEN* →*I*)
fix *F*
AOT-assume ‹*0*[F]›
AOT-actually {
AOT-hence ‹¬∃u [F]u›
using *zero*=:*2 intro−elim*:*3*:*a AOT-sem-enc-nec* **by** *blast*
AOT-hence ‹∀x ¬(O!x & [F]x)›
using *cqt−further*:*4 vdash−properties*:*10* **by** *blast*
hence ‹¬([w_0 ⊨ [F]«ωκ x»])› **for** *x*
by (*auto dest*!: ∀*E*(*1*)[**where** τ=‹ωκ x›]
simp: *AOT-sem-not AOT-sem-conj AOT-model-ωκ-ordinary*
russell−axiom[*exe*,*1*].ψ*-denotes-asm*)
}
AOT-thus ‹«$\varepsilon_\circ$ *w. finite* (*act-ωext* (*AOT-term-of-var F*))»›
by (*auto simp*: *AOT-model-proposition-choice-simp act-ωext-def*)
qed
next
fix *n m*
AOT-assume ‹[ℙ]nm›
AOT-hence ‹∃F∃u ([F]u & *Numbers*(m,F) & *Numbers*(n,[F]$^{-u}$))›
using *pred−thm*:*3*[*THEN* ≡*E*(*1*)] **by** *blast*
then AOT-obtain *G* **where** ‹∃u ([G]u & *Numbers*(m,G) & *Numbers*(n,[G]$^{-u}$))›
using ∃*E*[*rotated*] **by** *blast*
then AOT-obtain *u* **where** *0*: ‹[G]u & *Numbers*(m,G) & *Numbers*(n,[G]$^{-u}$)›
using *Ordinary*.∃*E*[*rotated*] **by** *meson*

AOT-assume *n-prop*: ‹∀F(n[F] → «$\varepsilon_\circ$ *w. finite* (*act-ωext F*)»)›
AOT-show ‹∀F(m[F] → «$\varepsilon_\circ$ *w. finite* (*act-ωext F*)»)›
proof(*safe intro*!: *GEN* →*I*)
fix *F*
AOT-assume ‹m[F]›
AOT-hence *1*: ‹[λx 𝓐[F]x] $\approx_E$ G›

```
  using 0[THEN &E(1), THEN &E(2), THEN numbers[THEN ≡df E],
        THEN &E(2), THEN ∀E(2), THEN ≡E(1)] by auto
AOT-show ‹«εo w. finite (act-ωext (AOT-term-of-var F))»›
proof(rule raa−cor:1)
  AOT-assume ‹¬«εo w. finite (act-ωext (AOT-term-of-var F))»›
  hence inf: ‹infinite (act-ωext (AOT-term-of-var F))›
    by (auto simp: AOT-sem-not AOT-model-proposition-choice-simp)
  then AOT-obtain v where act-F-v: ‹𝓐[F]v›
    unfolding AOT-sem-act act-ωext-def
    by (metis AOT-term-of-var-cases AOT-model-ωκ-ordinary
          AOT-model-denotes-κ-def Ordinary.Rep-cases κ.disc(7)
          mem-Collect-eq not-finite-existsD)
  AOT-hence ‹[λx 𝓐[F]x]v›
    by (safe intro!: β←C cqt:2)
  AOT-hence ‹[λx 𝓐[F]x]^−v ≈E [G]^−u›
    by (safe intro!: eqP′[unvarify F, THEN →E] &I cqt:2 1
              0[THEN &E(1), THEN &E(1)])
  moreover AOT-have ‹[λx 𝓐[F]x]^−v ≈E [λx 𝓐[λy [F]y & y ≠E v]x]›
  proof(safe intro!: apE−eqE:1[unvarify F G, THEN →E] cqt:2
              F−u[den][unvarify F] eqE[THEN ≡df I] &I
              Ordinary.GEN)
    fix u
    AOT-have ‹[λx [λx 𝓐[F]x]x & x ≠E v]u ≡ [λx 𝓐[F]x]u & u ≠E v›
      by (safe intro!: beta−C−meta[THEN →E] cqt:2)
    also AOT-have ‹[λx 𝓐[F]x]u & u ≠E v ≡ 𝓐[F]u & u ≠E v›
      by (AOT-subst ‹[λx 𝓐[F]x]u› ‹𝓐[F]u›)
        (safe intro!: beta−C−meta[THEN →E] cqt:2
              oth−class−taut:3:a)
    also AOT-have ‹𝓐[F]u & u ≠E v ≡ 𝓐([F]u & u ≠E v)›
      using id−act2:2 AOT-sem-conj AOT-sem-equiv AOT-sem-act by auto
    also AOT-have ‹𝓐([F]u & u ≠E v) ≡ 𝓐[λy [F]y & y ≠E v]u›
      by (AOT-subst ‹[λy [F]y & y ≠E v]u› ‹[F]u & u ≠E v›)
        (safe intro!: beta−C−meta[THEN →E] cqt:2
              oth−class−taut:3:a)
    also AOT-have ‹𝓐[λy [F]y & y ≠E v]u ≡ [λx 𝓐[λy [F]y & y ≠E v]x]u›
      by (safe intro!: beta−C−meta[THEN →E, symmetric] cqt:2)
    finally AOT-show ‹[[λx 𝓐[F]x]^−v]u ≡ [λx 𝓐[λy [F]y & y ≠E v]x]u›
      by (auto intro!: cqt:2
          intro: rule−id−df:2:b[OF F−u, where τ1τn=‹(-,-)›, simplified])
  qed
  ultimately AOT-have ‹[λx 𝓐[λy [F]y & y ≠E v]x] ≈E [G]^−u›
    using eq−part:2[terms] eq−part:3[terms] →E by blast
  AOT-hence ‹n[λy [F]y & y ≠E v]›
    by (safe intro!: 0[THEN &E(2), THEN numbers[THEN ≡df E],
        THEN &E(2), THEN ∀E(1), THEN ≡E(2)] cqt:2)
  hence finite: ‹finite (act-ωext «[λy [F]y & y ≠E v]»)›
    by (safe intro!: n-prop[THEN ∀E(1), THEN →E,
                simplified AOT-model-proposition-choice-simp]
              cqt:2)
  obtain y where y-def: ‹ωκ y = AOT-term-of-var (Ordinary.Rep v)›
    by (metis AOT-model-ordinary-ωκ Ordinary.restricted-var-condition)
  AOT-actually {
    fix x
    AOT-assume ‹[λy [F]y & y ≠E v]«ωκ x»›
    AOT-hence ‹[F]«ωκ x»›
      by (auto dest!: β→C &E(1))
  }
  moreover AOT-actually {
    AOT-have ‹[F]«ωκ y»›
      unfolding y-def using act-F-v AOT-sem-act by blast
  }
  moreover AOT-actually {
    fix x
```

```
        assume noteq: ‹x ≠ y›
        AOT-assume ‹[F]«ωκ x»›
        moreover AOT-have ωκ-x-den: ‹«ωκ x»↓›
          using AOT-sem-exe calculation by blast
        moreover {
          AOT-have ‹¬(«ωκ x» =E v)›
          proof(rule raa−cor:2)
            AOT-assume ‹«ωκ x» =E v›
            AOT-hence ‹«ωκ x» = v›
              using =E−simple:2[unvarify x, THEN →E, OF ωκ-x-den]
              by blast
            hence ‹ωκ x = ωκ y›
              unfolding y-def AOT-sem-eq
              by meson
            hence ‹x = y›
              by blast
            AOT-thus ‹p & ¬p› for p using noteq by blast
          qed
          AOT-hence ‹«ωκ x» ≠E v›
            by (safe intro!: thm−neg=E[unvarify x, THEN ≡E(2)] ωκ-x-den)
        }
        ultimately AOT-have ‹[λy [F]y & y ≠E v]«ωκ x»›
          by (auto intro!: β←C cqt:2 &I)
      }
      ultimately have ‹(insert y (act-ωext «[λy [F]y & y ≠E v]»)) =
                     (act-ωext (AOT-term-of-var F))›
        unfolding act-ωext-def
        by auto
      hence ‹finite (act-ωext (AOT-term-of-var F))›
        using finite finite.insertI by metis
      AOT-thus ‹p & ¬p› for p
        using inf by blast
    qed
  qed
qed
AOT-hence nat-enc-finite: ‹∀F(n[F] → «εo w. finite (act-ωext F)»)› for n
  using β→C(1) by blast
```

The main proof can now generate a witness, since we required the domain of ordinary objects to be infinite.

```
AOT-show ‹∃x ([ℕ]x & x = #G) → ◇∃y (E!y & ∀u (𝓐[G]u → u ≠E y))›
proof(safe intro!: →I)
  AOT-assume ‹∃x ([ℕ]x & x = #G)›
  then AOT-obtain n where ‹n = #G›
    using Number.∃E[rotated] by meson
  AOT-hence ‹Numbers(n,[λx 𝓐[G]x])›
    using eq−num:3 rule=E id-sym by fast
  AOT-hence ‹n[G]›
    by (auto intro!: numbers[THEN ≡df E, THEN &E(2),
                            THEN ∀E(2), THEN ≡E(2)]
                    eq−part:1[unvarify F] cqt:2)
  AOT-hence ‹«εo w. finite (act-ωext (AOT-term-of-var G))»›
    using nat-enc-finite[THEN ∀E(2), THEN →E] by blast
  hence finite: ‹finite (act-ωext (AOT-term-of-var G))›
    by (auto simp: AOT-model-proposition-choice-simp)
  AOT-have ‹∃u ¬𝓐[G]u›
  proof(rule raa−cor:1)
    AOT-assume ‹¬∃u ¬𝓐[G]u›
    AOT-hence ‹∀x ¬(O!x & ¬𝓐[G]x)›
      by (metis cqt−further:4 →E)
    AOT-hence ‹𝓐[G]x› if ‹O!x› for x
      using ∀E(2) AOT-sem-conj AOT-sem-not that by blast
    hence ‹[w0 ⊨ [G]«ωκ x»]› for x
```

by (*metis AOT-term-of-var-cases AOT-model-ωκ-ordinary*
AOT-model-denotes-κ-def AOT-sem-act κ.disc(*7*))
hence ‹(*act-ωext* (*AOT-term-of-var G*)) = *UNIV*›
unfolding *act-ωext-def* **by** *auto*
moreover have ‹*infinite* (*UNIV*::*ω set*)›
by (*metis ω-nat finite-imageI infinite-UNIV-char-0*)
ultimately have ‹*infinite* (*act-ωext* (*AOT-term-of-var G*))›
by *simp*
AOT-thus ‹p & ¬p› **for** p **using** *finite* **by** *blast*
qed
then AOT-obtain x **where** *x-prop*: ‹$O!x$ & ¬$\mathcal{A}[G]x$›
using ∃*E*[*rotated*] **by** *blast*
AOT-hence ‹$\Diamond E!x$›
by (*metis betaC*:*1*:*a con−dis−i−e*:*2*:*a AOT-sem-ordinary*)
moreover AOT-have ‹$\Box\forall u$ ($\mathcal{A}[G]u \to u \neq_E x$)›
proof(*safe intro*!: *RN GEN* →*I*)
AOT-modally-strict {
fix y
AOT-assume ‹$O!y$›
AOT-assume *0*: ‹$\mathcal{A}[G]y$›
AOT-show ‹$y \neq_E x$›
proof (*safe intro*!: *thm−neg=E*[*THEN* ≡*E*(*2*)] *raa−cor*:*2*)
AOT-assume ‹$y =_E x$›
AOT-hence ‹$y = x$›
by (*metis =E−simple*:*2 vdash−properties*:*10*)
hence ‹$y = x$›
by (*simp add*: *AOT-sem-eq AOT-term-of-var-inject*)
AOT-hence ‹¬$\mathcal{A}[G]y$›
using *x-prop* &*E AOT-sem-not AOT-sem-act* **by** *metis*
AOT-thus ‹$\mathcal{A}[G]y$ & ¬$\mathcal{A}[G]y$›
using *0* &*I* **by** *blast*
qed
}
qed
ultimately AOT-have ‹$\Diamond(\forall u$ ($\mathcal{A}[G]u \to u \neq_E x$) & $E!x$)›
using *KBasic*:*16*[*THEN* →*E*, *OF* &*I*] **by** *blast*
AOT-hence ‹$\Diamond(E!x$ & $\forall u$ ($\mathcal{A}[G]u \to u \neq_E x$))›
by (*AOT-subst* ‹$E!x$ & $\forall u$ ($\mathcal{A}[G]u \to u \neq_E x$)› ‹$\forall u$ ($\mathcal{A}[G]u \to u \neq_E x$) & $E!x$›)
(*auto simp*: *oth−class−taut*:*2*:*a*)
AOT-hence ‹$\exists y\ \Diamond(E!y$ & $\forall u$ ($\mathcal{A}[G]u \to u \neq_E y$))›
using ∃*I* **by** *fast*
AOT-thus ‹$\Diamond\exists y$ ($E!y$ & $\forall u$ ($\mathcal{A}[G]u \to u \neq_E y$))›
using *CBF*◇[*THEN* →*E*] **by** *fast*
qed
} qed

AOT-theorem *modal−lemma*:
‹$\Diamond\forall u(\mathcal{A}[G]u \to u \neq_E v) \to \forall u(\mathcal{A}[G]u \to u \neq_E v)$›
proof(*safe intro*!: →*I Ordinary.GEN*)
AOT-modally-strict {
fix u
AOT-assume *act-Gu*: ‹$\mathcal{A}[G]u$›
AOT-have ‹$\forall u$ ($\mathcal{A}[G]u \to u \neq_E v$) $\to u \neq_E v$›
proof(*rule* →*I*)
AOT-assume ‹$\forall u$ ($\mathcal{A}[G]u \to u \neq_E v$)›
AOT-hence ‹$\mathcal{A}[G]u \to u \neq_E v$›
using *Ordinary*.∀*E* **by** *fast*
AOT-thus ‹$u \neq_E v$›
using *act-Gu* →*E* **by** *blast*
qed
} note *0* = *this*
AOT-have ϑ: ‹$\Box(\forall u$ ($\mathcal{A}[G]u \to u \neq_E v$) $\to u \neq_E v$)› **if** ‹$\Box\mathcal{A}[G]u$› **for** u
proof −

AOT-have ‹□𝓐[G]u → □(∀u (𝓐[G]u → u $\neq_E$ v) → u $\neq_E$ v)›
apply (*rule RM*) **using** *0* &E →I **by** *blast*
thus *?thesis* **using** *that* →E **by** *blast*
qed
fix *u*
AOT-assume *1*: ‹◇∀u(𝓐[G]u → u $\neq_E$ v)›
AOT-assume ‹𝓐[G]u›
AOT-hence ‹□𝓐[G]u›
by (*metis Act−Basic:6* ≡E(*1*))
AOT-hence ‹□(∀u (𝓐[G]u → u $\neq_E$ v) → u $\neq_E$ v)›
using *Ordinary.ψ ϑ* **by** *blast*
AOT-hence ‹◇u $\neq_E$ v›
using *1 K*◇[*THEN* →E, *THEN* →E] **by** *blast*
AOT-thus ‹u $\neq_E$ v›
by (*metis id−nec4:2* ≡E(*1*))
qed

AOT-theorem *th−succ*: ‹∀n∃!m [ℙ]nm›
proof(*safe intro*!: *Number.GEN* →I *uniqueness*:*1*[*THEN* $\equiv_{df}$I])
fix *n*
AOT-have ‹*NaturalCardinal*(n)›
by (*metis nat−card Number.ψ* →E)
AOT-hence ‹∃G(n = #G)›
by (*metis* $\equiv_{df}$E *card*)
then AOT-obtain *G* **where** *n-num-G*: ‹n = #G›
using ∃E[*rotated*] **by** *blast*
AOT-hence ‹∃n (n = #G)›
by (*rule Number*.∃I)
AOT-hence ‹◇∃y ([E!]y & ∀u(𝓐[G]u → u $\neq_E$ y))›
using *modal−axiom*[*axiom-inst*, *THEN* →E] **by** *blast*
AOT-hence ‹∃y ◇([E!]y & ∀u(𝓐[G]u → u $\neq_E$ y))›
using *BF*◇[*THEN* →E] **by** *auto*
then AOT-obtain *y* **where** ‹◇([E!]y & ∀u(𝓐[G]u → u $\neq_E$ y))›
using ∃E[*rotated*] **by** *blast*
AOT-hence ‹◇E!y› **and** *2*: ‹◇∀u(𝓐[G]u → u $\neq_E$ y)›
using *KBasic2*:*3* &E →E **by** *blast+*
AOT-hence *Oy*: ‹O!y›
by (*auto intro*!: β←C(*1*) *cqt*:*2 intro*: *AOT-ordinary*[*THEN* $=_{df}$I(*2*)])
AOT-have *0*: ‹∀u(𝓐[G]u → u $\neq_E$ y)›
using *2 modal−lemma*[*unconstrain v*, *THEN* →E, *OF Oy*, *THEN* →E] **by** *simp*
AOT-have *1*: ‹[λx 𝓐[G]x ∨ x $=_E$ y]↓›
by *cqt*:*2*
AOT-obtain *b* **where** *b-prop*: ‹*Numbers*(b, [λx 𝓐[G]x ∨ x $=_E$ y])›
using *num*:*1*[*unvarify G*, *OF 1*] ∃E[*rotated*] **by** *blast*
AOT-have *Pnb*: ‹[ℙ]nb›
proof(*safe intro*!: *pred−thm*:*3*[*THEN* ≡E(*2*)]
∃I(*1*)[**where** τ=‹«[λx 𝓐[G]x ∨ x $=_E$ y]»›]
1 ∃I(*2*)[**where** β=y] &I *Oy b-prop*)
AOT-show ‹[λx 𝓐[G]x ∨ x $=_E$ y]y›
by (*auto intro*!: β←C(*1*) *cqt*:*2* ∨I(*2*)
ord=*Eequiv*:*1*[*THEN* →E, *OF Oy*])
next
AOT-have *equinum*: ‹[λx 𝓐[G]x ∨ x $=_E$ y]$^{-y}$ $\approx_E$ [λx 𝓐[G]x]›
proof(*rule apE−eqE*:*1*[*unvarify F G*, *THEN* →E];
(*cqt*:*2*[*lambda*] | *rule F−u*[*den*][*unvarify F*]; *cqt*:*2*[*lambda*])*?*)
AOT-show ‹[λx 𝓐[G]x ∨ x $=_E$ y]$^{-y}$ $\equiv_E$ [λx 𝓐[G]x]›
proof (*safe intro*!: *eqE*[*THEN* $\equiv_{df}$I] &I *F−u*[*den*][*unvarify F*]
Ordinary.GEN →I; *cqt*:*2?*)
fix *u*
AOT-have ‹[[λx 𝓐[G]x ∨ [($=_E$)]xy]$^{-y}$]u ≡ ([λx 𝓐[G]x ∨ x $=_E$ y]u) & u $\neq_E$ y›
apply (*rule F−u*[*THEN* $=_{df}$I(*1*)[**where** $\tau_1\tau_n$=‹(-,-)›], *simplified*]; *cqt*:*2?*)
by (*rule beta−C−cor*:*2*[*THEN* →E, *THEN* ∀E(*2*)]; *cqt*:*2*)
also AOT-have ‹. . . ≡ (𝓐[G]u ∨ u $=_E$ y) & u $\neq_E$ y›

    apply (*AOT-subst* ‹[λx 𝓐[G]x ∨ [(=E)]xy]u› ‹𝓐[G]u ∨ u =E y›)
     apply (*rule beta−C−cor:2[THEN →E, THEN ∀E(2)]; cqt:2*)
    using *oth−class−taut:3:a* **by** *blast*
  also AOT-have ‹... ≡ 𝓐[G]u›
  proof(*safe intro*!: ≡*I* →*I*)
    AOT-assume ‹(𝓐[G]u ∨ u =E y) & u ≠E y›
    AOT-thus ‹𝓐[G]u›
      by (*metis* &*E(1)* &*E(2)* ∨*E(3)* ≡*E(1)* *thm−neg=E*)
  next
    AOT-assume ‹𝓐[G]u›
    AOT-hence ‹u ≠E y› **and** ‹𝓐[G]u ∨ u =E y›
      using *0[THEN ∀E(2), THEN →E, OF Ordinary.ψ, THEN →E]*
            ∨*I* **by** *blast+*
    AOT-thus ‹(𝓐[G]u ∨ u =E y) & u ≠E y›
      using &*I* **by** *simp*
  qed
  also AOT-have ‹... ≡ [λx 𝓐[G]x]u›
    by (*rule beta−C−cor:2[THEN →E, THEN ∀E(2), symmetric]; cqt:2*)
  finally AOT-show ‹[[λx 𝓐[G]x ∨ [(=E)]xy]⁻ʸ]u ≡ [λx 𝓐[G]x]u›.
 qed
qed
AOT-have *2*: ‹[λx 𝓐[G]x]↓› **by** *cqt:2[lambda]*
AOT-show ‹*Numbers*(n,[λx 𝓐[G]x ∨ x =E y]⁻ʸ)›
  using *num−tran:1[unvarify G H, OF 2, OF F−u[den][unvarify F, OF 1],*
        THEN →E, OF equinum, THEN ≡E(2),
        OF eq−num:2[THEN ≡E(2), OF n-num-G]].
qed
AOT-show ‹∃α ([ℕ]α & [ℙ]nα & ∀β ([ℕ]β & [ℙ]nβ → β = α))›
proof(*safe intro*!: ∃*I(2)*[**where** *β=b*] &*I Pnb* →*I GEN*)
  AOT-show ‹[ℕ]b› **using** *suc−num:1[THEN →E, OF Pnb]*.
next
  fix *y*
  AOT-assume *0*: ‹[ℕ]y & [ℙ]ny›
  AOT-show ‹y = b›
    apply (*rule pred−func:1[THEN →E]*)
    using *0[THEN &E(2)] Pnb* &*I* **by** *blast*
qed
qed

AOT-define *Successor* :: ‹τ ⇒ κs› (‹-′′› [*100*] *100*)
  def−suc: ‹n′ =df ιm([ℙ]nm)›

Note: not explicitly in PLM

AOT-theorem *def−suc[den1]*: ‹ιm([ℙ]nm)↓›
  using *A−Exists:2 RA[2]* ≡*E(2) th−succ[THEN Number.∀E]* **by** *blast*

Note: not explicitly in PLM

AOT-theorem *def−suc[den2]*: **shows** ‹n′↓›
  by (*rule def−suc[THEN =dfI(1)]*)
    (*auto simp*: *def−suc[den1]*)

AOT-theorem *suc-eq-desc*: ‹n′ = ιm([ℙ]nm)›
  by (*rule def−suc[THEN =dfI(1)]*)
    (*auto simp*: *def−suc[den1] rule=I:1*)

AOT-theorem *suc−fact*: ‹n = m → n′ = m′›
proof (*rule* →*I*)
  AOT-assume *0*: ‹n = m›
  AOT-show ‹n′ = m′›
    apply (*rule rule=E[rotated, OF 0]*)
    by (*rule =I(1)[OF def−suc[den2]]*)

```
qed

AOT-theorem ind−gnd: ‹m = 0 ∨ ∃ n(m = n′)›
proof −
  AOT-have ‹[[ℙ]+]0m›
    using Number.ψ ≡E(1) nnumber:3 by blast
  AOT-hence ‹[[ℙ]*]0m ∨ 0 =ℙ m›
    using assume1:5[unvarify x, OF zero:2, THEN ≡E(1)] by blast
  moreover {
    AOT-assume ‹[[ℙ]*]0m›
    AOT-hence ‹∃ z ([[ℙ]+]0z & [ℙ]zm)›
      using w−ances−her:7[unconstrain ℛ, unvarify β x, OF zero:2,
                          OF pred−thm:2, THEN →E, OF pred−1−1:4,
                          THEN →E]
      by blast
    then AOT-obtain z where ϑ: ‹[[ℙ]+]0z› and ξ: ‹[ℙ]zm›
      using &E ∃ E[rotated] by blast
    AOT-have Nz: ‹[ℕ]z›
      using ϑ ≡E(2) nnumber:3 by blast
    moreover AOT-have ‹m = z′›
    proof (rule def−suc[THEN =df I(1)];
          safe intro!: def−suc[den1][unconstrain n, THEN →E, OF Nz]
                       nec−hintikka−scheme[THEN ≡E(2)] &I
                       GEN →I Act−Basic:2[THEN ≡E(2)])
      AOT-show ‹𝓐[ℕ]m› using Number.ψ
        by (meson mod−col−num:1 nec−imp−act →E)
    next
      AOT-show ‹𝓐[ℙ]zm› using ξ
        by (meson nec−imp−act pred−1−1:1 →E)
    next
      fix y
      AOT-assume ‹𝓐([ℕ]y & [ℙ]zy)›
      AOT-hence ‹𝓐[ℕ]y› and ‹𝓐[ℙ]zy›
        using Act−Basic:2 &E ≡E(1) by blast+
      AOT-hence 0: ‹[ℙ]zy›
        by (metis RN ≡E(1) pred−1−1:1 sc−eq−fur:2 →E)
      AOT-thus ‹y = m›
        using pred−func:1[THEN →E, OF &I] ξ by metis
    qed
    ultimately AOT-have ‹[ℕ]z & m = z′›
      by (rule &I)
    AOT-hence ‹∃ n m = n′›
      by (rule ∃ I)
    hence ?thesis
      by (rule ∨I)
  }
  moreover {
    AOT-assume ‹0 =ℙ m›
    AOT-hence ‹0 = m›
      using id−R−thm:3[unconstrain ℛ, unvarify β x, OF zero:2, OF pred−thm:2,
                       THEN →E, OF pred−1−1:4, THEN →E]
      by auto
    hence ?thesis using id-sym ∨I by blast
  }
  ultimately show ?thesis
    by (metis ∨E(2) raa−cor:1)
qed

AOT-theorem suc−thm: ‹[ℙ]n n′›
proof −
  AOT-obtain x where m-is-n: ‹x = n′›
    using free−thms:1[THEN ≡E(1), OF def−suc[den2]]
    using ∃ E by metis
```

```
  AOT-have ‹𝓐([ℕ]n' & [ℙ]n n')›
    apply (rule rule=E[rotated, OF suc-eq-desc[symmetric]])
    apply (rule actual−desc:4[THEN →E])
    by (simp add:  def−suc[den1])
  AOT-hence ‹𝓐[ℕ]n'› and ‹𝓐[ℙ]n n'›
    using Act−Basic:2 ≡E(1) &E by blast+
  AOT-hence ‹𝓐[ℙ]nx›
    using m-is-n[symmetric] rule=E by fast+
  AOT-hence ‹[ℙ]nx›
    by (metis RN ≡E(1) pred−1−1:1 sc−eq−fur:2 →E)
  thus ?thesis
    using m-is-n rule=E by fast
qed

AOT-define Numeral1 :: ‹κs› (‹1›)
  numerals:1: ‹1 =df 0'›

AOT-theorem prec−facts:1: ‹[ℙ]0 1›
  by (auto intro: numerals:1[THEN rule−id−df:2:b[zero],
                            OF def−suc[den2][unconstrain n, unvarify β,
                                             OF zero:2, THEN →E, OF 0−n]]
                  suc−thm[unconstrain n, unvarify β, OF zero:2,
                          THEN →E, OF 0−n])


AOT-define Finite :: ‹τ ⇒ φ› (‹Finite'(-')›)
  inf−card:1: ‹Finite(x) ≡df NaturalCardinal(x) & [ℕ]x›
AOT-define Infinite :: ‹τ ⇒ φ› (‹Infinite'(-')›)
  inf−card:2: ‹Infinite(x) ≡df NaturalCardinal(x) & ¬Finite(x)›

AOT-theorem inf−card−exist:1: ‹NaturalCardinal(#O!)›
  by (safe intro!: card[THEN ≡dfI] ∃I(1)[where τ=‹«O!»›] =I
                   num−def:2[unvarify G] oa−exist:1)

AOT-theorem inf−card−exist:2: ‹Infinite(#O!)›
proof (safe intro!: inf−card:2[THEN ≡dfI] &I inf−card−exist:1)
  AOT-show ‹¬Finite(#O!)›
  proof(rule raa−cor:2)
    AOT-assume ‹Finite(#O!)›
    AOT-hence 0: ‹[ℕ]#O!›
      using inf−card:1[THEN ≡dfE] &E(2) by blast
    AOT-have ‹Numbers(#O!, [λz 𝓐O!z])›
      using eq−num:3[unvarify G, OF oa−exist:1].
    AOT-hence ‹#O! = #O!›
      using eq−num:2[unvarify x G, THEN ≡E(1), OF oa−exist:1,
                     OF num−def:2[unvarify G], OF oa−exist:1]
      by blast
    AOT-hence ‹[ℕ]#O! & #O! = #O!›
      using 0 &I by blast
    AOT-hence ‹∃x ([ℕ]x & x = #O!)›
      using num−def:2[unvarify G, OF oa−exist:1] ∃I(1) by fast
    AOT-hence ‹◇∃y ([E!]y & ∀u (𝓐[O!]u → u ≠E y))›
      using modal−axiom[axiom-inst, unvarify G, THEN →E, OF oa−exist:1] by blast
    AOT-hence ‹∃y ◇([E!]y & ∀u (𝓐[O!]u → u ≠E y))›
      using BF◇[THEN →E] by blast
    then AOT-obtain b where ‹◇([E!]b & ∀u (𝓐[O!]u → u ≠E b))›
      using ∃E[rotated] by blast
    AOT-hence ‹◇[E!]b› and 2: ‹◇∀u (𝓐[O!]u → u ≠E b)›
      using KBasic2:3[THEN →E] &E by blast+
    AOT-hence ‹[λx ◇[E!]x]b›
      by (auto intro!: β←C(1) cqt:2)
```

moreover AOT-have ‹*O*! = [λ*x* ◇[*E*!]*x*]›
by (*rule rule−id−df*:*1*[*zero*][*OF oa*:*1*]) *cqt*:*2*
ultimately AOT-have *b-ord*: ‹*O*!*b*›
using *rule*=*E id-sym* **by** *fast*
AOT-hence ‹𝓐*O*!*b*›
by (*meson* ≡*E*(*1*) *oa−facts*:*7*)
moreover AOT-have *2*: ‹∀ *u* (𝓐[*O*!]*u* → *u* ≠*E* *b*)›
using *modal−lemma*[*unvarify G*, *unconstrain v*, *OF oa−exist*:*1*,
THEN →*E*, *OF b-ord*, *THEN* →*E*, *OF 2*].
ultimately AOT-have ‹*b* ≠*E* *b*›
using *Ordinary*.∀ *E*[*OF 2*, *unconstrain* α, *THEN* →*E*,
OF b-ord, *THEN* →*E*] **by** *blast*
AOT-hence ‹¬(*b* =*E* *b*)›
by (*metis* ≡*E*(*1*) *thm−neg*=*E*)
moreover AOT-have ‹*b* =*E* *b*›
using *ord*=*Eequiv*:*1*[*THEN* →*E*, *OF b-ord*].
ultimately AOT-show ‹*p* & ¬*p*› **for** *p*
by (*metis raa−cor*:*3*)
qed
qed

theory *AOT-misc*
imports *AOT-NaturalNumbers*
begin

14 Miscellaneous Theorems

AOT-theorem *PossiblyNumbersEmptyPropertyImpliesZero*:
‹◇*Numbers*(*x*,[λ*z O*!*z* & *z* ≠*E* *z*]) → *x* = *0*›
proof(*rule* →*I*)
AOT-have ‹*Rigid*([λ*z O*!*z* & *z* ≠*E* *z*])›
proof (*safe intro*!: *df−rigid−rel*:*1*[*THEN* ≡*df* *I*] &*I cqt*:*2*;
rule RN; *safe intro*!: *GEN* →*I*)
AOT-modally-strict {
fix *x*
AOT-assume ‹[λ*z O*!*z* & *z* ≠*E* *z*]*x*›
AOT-hence ‹*O*!*x* & *x* ≠*E* *x*› **by** (*rule* β→*C*)
moreover AOT-have ‹*x* =*E* *x*› **using** *calculation*[*THEN* &*E*(*1*)]
by (*metis ord*=*Eequiv*:*1 vdash−properties*:*10*)
ultimately AOT-have ‹*x* =*E* *x* & ¬*x* =*E* *x*›
by (*metis con−dis−i−e*:*1 con−dis−i−e*:*2*:*b intro−elim*:*3*:*a thm−neg*=*E*)
AOT-thus ‹□[λ*z O*!*z* & *z* ≠*E* *z*]*x*› **using** *raa−cor*:*1* **by** *blast*
}
qed
AOT-hence ‹□∀ *x* (*Numbers*(*x*,[λ*z O*!*z* & *z* ≠*E* *z*]) → □*Numbers*(*x*,[λ*z O*!*z* & *z* ≠*E* *z*]))›
by (*safe intro*!: *num−cont*:*2*[*unvarify G*, *THEN* →*E*] *cqt*:*2*)
AOT-hence ‹∀ *x* □(*Numbers*(*x*,[λ*z O*!*z* & *z* ≠*E* *z*]) → □*Numbers*(*x*,[λ*z O*!*z* & *z* ≠*E* *z*]))›
using *BFs*:*2*[*THEN* →*E*] **by** *blast*
AOT-hence ‹□(*Numbers*(*x*,[λ*z O*!*z* & *z* ≠*E* *z*]) → □*Numbers*(*x*,[λ*z O*!*z* & *z* ≠*E* *z*]))›
using ∀ *E*(*2*) **by** *auto*
moreover AOT-assume ‹◇*Numbers*(*x*,[λ*z O*!*z* & *z* ≠*E* *z*])›
ultimately AOT-have ‹𝓐*Numbers*(*x*,[λ*z O*!*z* & *z* ≠*E* *z*])›
using *sc−eq−box−box*:*1*[*THEN* ≡*E*(*1*), *THEN* →*E*, *THEN nec−imp−act*[*THEN* →*E*]]
by *blast*
AOT-hence ‹*Numbers*(*x*,[λ*z* 𝓐[λ*z O*!*z* & *z* ≠*E* *z*]*z*])›
by (*safe intro*!: *eq−num*:*1*[*unvarify G*, *THEN* ≡*E*(*1*)] *cqt*:*2*)
AOT-hence ‹*x* = #[λ*z O*!*z* & *z* ≠*E* *z*]›
by (*safe intro*!: *eq−num*:*2*[*unvarify G*, *THEN* ≡*E*(*1*)] *cqt*:*2*)
AOT-thus ‹*x* = *0*›
using *cqt*:*2*(*1*) *rule−id−df*:*2*:*b*[*zero*] *rule*=*E zero*:*1* **by** *blast*
qed

AOT-define *Numbers′* :: ‹τ ⇒ τ ⇒ φ› (‹*Numbers′′′*(-,-′)›)
‹*Numbers′*(x, G) $\equiv_{df}$ A!x & G↓ & ∀ F (x[F] ≡ F $\approx_E$ G)›
AOT-theorem *Numbers′equiv*: ‹*Numbers′*(x,G) ≡ A!x & ∀ F (x[F] ≡ F $\approx_E$ G)›
by (*AOT-subst-def Numbers′*)
(*auto intro*!: ≡I →I &I *cqt*:*2 dest*: &E)

AOT-theorem *Numbers′DistinctZeroes*:
‹∃ x∃ y (◇*Numbers′*(x,[λz O!z & z $\neq_E$ z]) & ◇*Numbers′*(y,[λz O!z & z $\neq_E$ z]) & x ≠ y)›
proof −
AOT-obtain w_1 **where** ‹∃ w w_1 ≠ w›
using *two−worlds−exist*:*4 PossibleWorld*.∃ E[*rotated*] **by** *fast*
then AOT-obtain w_2 **where** *distinct-worlds*: ‹w_1 ≠ w_2›
using *PossibleWorld*.∃ E[*rotated*] **by** *blast*
AOT-obtain x **where** *x-prop*:
‹A!x & ∀ F (x[F] ≡ w_1 ⊨ F $\approx_E$ [λz O!z & z $\neq_E$ z])›
using *A−objects*[*axiom-inst*] ∃ E[*rotated*] **by** *fast*
moreover AOT-obtain y **where** *y-prop*:
‹A!y & ∀ F (y[F] ≡ w_2 ⊨ F $\approx_E$ [λz O!z & z $\neq_E$ z])›
using *A−objects*[*axiom-inst*] ∃ E[*rotated*] **by** *fast*
moreover {
fix x w
AOT-assume *x-prop*: ‹A!x & ∀ F (x[F] ≡ w ⊨ F $\approx_E$ [λz O!z & z $\neq_E$ z])›
AOT-have ‹∀ F w ⊨ (x[F] ≡ F $\approx_E$ [λz O!z & z $\neq_E$ z])›
proof(*safe intro*!: *GEN conj−dist−w*:*4*[*unvarify p q*, *OF log−prop−prop*:*2*,
OF log−prop−prop:*2*,*THEN* ≡E(*2*)] ≡I →I)
fix F
AOT-assume ‹w ⊨ x[F]›
AOT-hence ‹◇x[F]›
using *fund*:*1*[*unvarify p*, *OF log−prop−prop*:*2*, *THEN* ≡E(*2*),
OF PossibleWorld.∃ I] **by** *blast*
AOT-hence ‹x[F]›
by (*metis en−eq*:*3*[*1*] *intro−elim*:*3*:*a*)
AOT-thus ‹w ⊨ (F $\approx_E$ [λz O!z & z $\neq_E$ z])›
using *x-prop*[*THEN* &E(*2*), *THEN* ∀ E(*2*), *THEN* ≡E(*1*)] **by** *blast*
next
fix F
AOT-assume ‹w ⊨ (F $\approx_E$ [λz O!z & z $\neq_E$ z])›
AOT-hence ‹x[F]›
using *x-prop*[*THEN* &E(*2*), *THEN* ∀ E(*2*), *THEN* ≡E(*2*)] **by** *blast*
AOT-hence ‹□x[F]›
using *pre−en−eq*:*1*[*1*][*THEN* →E] **by** *blast*
AOT-thus ‹w ⊨ x[F]›
using *fund*:*2*[*unvarify p*, *OF log−prop−prop*:*2*, *THEN* ≡E(*1*)]
PossibleWorld.∀ E **by** *fast*
qed
AOT-hence ‹w ⊨ ∀ F (x[F] ≡ F $\approx_E$ [λz O!z & z $\neq_E$ z])›
using *conj−dist−w*:*5*[*THEN* ≡E(*2*)] **by** *fast*
moreover {
AOT-have ‹□[λz O!z & z $\neq_E$ z]↓›
by (*safe intro*!: *RN cqt*:*2*)
AOT-hence ‹w ⊨ [λz O!z & z $\neq_E$ z]↓›
using *fund*:*2*[*unvarify p*, *OF log−prop−prop*:*2*, *THEN* ≡E(*1*),
THEN PossibleWorld.∀ E] **by** *blast*
}
moreover {
AOT-have ‹□A!x›
using *x-prop*[*THEN* &E(*1*)] **by** (*metis oa−facts*:*2* →E)
AOT-hence ‹w ⊨ A!x›
using *fund*:*2*[*unvarify p*, *OF log−prop−prop*:*2*,
THEN ≡E(*1*), *THEN PossibleWorld*.∀ E] **by** *blast*
}
ultimately AOT-have ‹w ⊨ (A!x & [λz O!z & z $\neq_E$ z]↓ &

```
                     ∀ F (x[F] ≡ F ≈E [λz O!z & z ≠E z]))›
    using conj−dist−w:1[unvarify p q, OF log−prop−prop:2,
          OF log−prop−prop:2, THEN ≡E(2), OF &I] by auto
  AOT-hence ‹∃ w w ⊨ (A!x & [λz O!z & z ≠E z]↓ &
                     ∀ F (x[F] ≡ F ≈E [λz O!z & z ≠E z]))›
    using PossibleWorld.∃ I by auto
  AOT-hence ‹◇(A!x & [λz O!z & z ≠E z]↓ & ∀ F (x[F] ≡ F ≈E [λz O!z & z ≠E z]))›
    using fund:1[unvarify p, OF log−prop−prop:2, THEN ≡E(2)] by blast
  AOT-hence ‹◇Numbers′(x,[λz O!z & z ≠E z])›
    by (AOT-subst-def Numbers′)
}
ultimately AOT-have ‹◇Numbers′(x,[λz O!z & z ≠E z])›
               and ‹◇Numbers′(y,[λz O!z & z ≠E z])›
  by auto
moreover AOT-have ‹x ≠ y›
proof (rule ab−obey:2[THEN →E])
  AOT-have ‹□¬∃ u [λz O!z & z ≠E z]u›
  proof (safe intro!: RN raa−cor:2)
    AOT-modally-strict {
      AOT-assume ‹∃ u [λz O!z & z ≠E z]u›
      then AOT-obtain u where ‹[λz O!z & z ≠E z]u›
        using Ordinary.∃ E[rotated] by blast
      AOT-hence ‹O!u & u ≠E u›
        by (rule β→C)
      AOT-hence ‹¬(u =E u)›
        by (metis con−dis−taut:2 intro−elim:3:d modus−tollens:1
                raa−cor:3 thm−neg=E)
      AOT-hence ‹u =E u & ¬u =E u›
        by (metis modus−tollens:1 ord=Eequiv:1 raa−cor:3 Ordinary.ψ)
      AOT-thus ‹p & ¬p› for p
        by (metis raa−cor:1)
    }
  qed
  AOT-hence nec-not-ex: ‹∀ w w ⊨ ¬∃ u [λz O!z & z ≠E z]u›
    using fund:2[unvarify p, OF log−prop−prop:2, THEN ≡E(1)] by blast
  AOT-have ‹□([λy p]x ≡ p)› for x p
    by (safe intro!: RN beta−C−meta[THEN →E] cqt:2)
  AOT-hence ‹∀ w w ⊨ ([λy p]x ≡ p)› for x p
    using fund:2[unvarify p, OF log−prop−prop:2, THEN ≡E(1)] by blast
  AOT-hence world-prop-beta: ‹∀ w (w ⊨ [λy p]x ≡ w ⊨ p)› for x p
    using conj−dist−w:4[unvarify p, OF log−prop−prop:2, THEN ≡E(1)]
        PossibleWorld.∀ E PossibleWorld.∀ I by meson

  AOT-have ‹∃ p (w1 ⊨ p & ¬w2 ⊨ p)›
  proof(rule raa−cor:1)
    AOT-assume 0: ‹¬∃ p (w1 ⊨ p & ¬w2 ⊨ p)›
    AOT-have 1: ‹w1 ⊨ p → w2 ⊨ p› for p
    proof(safe intro!: GEN →I)
      AOT-assume ‹w1 ⊨ p›
      AOT-thus ‹w2 ⊨ p›
        using 0 con−dis−i−e:1 ∃ I(2) raa−cor:4 by fast
    qed
    moreover AOT-have ‹w2 ⊨ p → w1 ⊨ p› for p
    proof(safe intro!: GEN →I)
      AOT-assume ‹w2 ⊨ p›
      AOT-hence ‹¬w2 ⊨ ¬p›
        using coherent:2 intro−elim:3:a by blast
      AOT-hence ‹¬w1 ⊨ ¬p›
        using 1[∀ I p, THEN ∀ E(1), OF log−prop−prop:2]
        by (metis modus−tollens:1)
      AOT-thus ‹w1 ⊨ p›
        using coherent:1 intro−elim:3:b reductio−aa:1 by blast
    qed
```

```
  ultimately AOT-have ‹w1 ⊨ p ≡ w2 ⊨ p› for p
    by (metis intro−elim:2)
  AOT-hence ‹w1 = w2›
    using sit−identity[unconstrain s, THEN →E,
        OF PossibleWorld.ψ[THEN world:1[THEN ≡df E], THEN &E(1)],
        unconstrain s′, THEN →E,
        OF PossibleWorld.ψ[THEN world:1[THEN ≡df E], THEN &E(1)],
        THEN ≡E(2)] GEN by fast
  AOT-thus ‹w1 = w2 & ¬w1 = w2›
    using =−infix ≡df E con−dis−i−e:1 distinct-worlds by blast
qed
then AOT-obtain p where 0: ‹w1 ⊨ p & ¬w2 ⊨ p›
  using ∃ E[rotated] by blast
AOT-have ‹y[λy p]›
proof (safe intro!: y-prop[THEN &E(2), THEN ∀ E(1), THEN ≡E(2)] cqt:2)
  AOT-show ‹w2 ⊨ [λy p] ≈E [λz O!z & z ≠E z]›
  proof (safe intro!:  cqt:2 empty−approx:1[unvarify F H, THEN RN,
            THEN fund:2[unvarify p, OF log−prop−prop:2, THEN ≡E(1)],
            THEN PossibleWorld.∀ E,
            THEN conj−dist−w:2[unvarify p q, OF log−prop−prop:2,
                          OF log−prop−prop:2, THEN ≡E(1)],
                          THEN →E]
            conj−dist−w:1[unvarify p q, OF log−prop−prop:2,
                       OF log−prop−prop:2, THEN ≡E(2)] &I)
    AOT-have ‹¬w2 ⊨ ∃ u [λy p]u›
    proof (rule raa−cor:2)
      AOT-assume ‹w2 ⊨ ∃ u [λy p]u›
      AOT-hence ‹∃ x w2 ⊨ (O!x & [λy p]x)›
        by (metis conj−dist−w:6 intro−elim:3:a)
      then AOT-obtain x where ‹w2 ⊨ (O!x & [λy p]x)›
        using ∃ E[rotated] by blast
      AOT-hence ‹w2 ⊨ [λy p]x›
        using conj−dist−w:1[unvarify p q, OF log−prop−prop:2,
            OF log−prop−prop:2, THEN ≡E(1), THEN &E(2)] by blast
      AOT-hence ‹w2 ⊨ p›
        using world-prop-beta[THEN PossibleWorld.∀ E, THEN ≡E(1)] by blast
      AOT-thus ‹w2 ⊨ p & ¬w2 ⊨ p›
        using 0[THEN &E(2)] &I by blast
    qed
    AOT-thus ‹w2 ⊨ ¬∃ u [λy p]u›
      by (safe intro!: coherent:1[unvarify p, OF log−prop−prop:2,
                            THEN ≡E(2)])
  next
    AOT-show ‹w2 ⊨ ¬∃ v [λz O!z & z ≠E z]v›
      using nec-not-ex[THEN PossibleWorld.∀ E] by blast
  qed
qed
moreover AOT-have ‹¬x[λy p]›
proof(rule raa−cor:2)
  AOT-assume ‹x[λy p]›
  AOT-hence w1 ⊨ [λy p] ≈E [λz O!z & z ≠E z]
    using x-prop[THEN &E(2), THEN ∀ E(1), THEN ≡E(1)]
        prop−prop2:2 by blast
  AOT-hence ¬w1 ⊨ ¬[λy p] ≈E [λz O!z & z ≠E z]
    using coherent:2[unvarify p, OF log−prop−prop:2, THEN ≡E(1)] by blast
  moreover AOT-have w1 ⊨ ¬([λy p] ≈E [λz O!z & z ≠E z])
  proof (safe intro!: cqt:2 empty−approx:2[unvarify F H, THEN RN,
              THEN fund:2[unvarify p, OF log−prop−prop:2, THEN ≡E(1)],
              THEN PossibleWorld.∀ E,
              THEN conj−dist−w:2[unvarify p q, OF log−prop−prop:2,
                  OF log−prop−prop:2, THEN ≡E(1)], THEN →E]
              conj−dist−w:1[unvarify p q, OF log−prop−prop:2,
                          OF log−prop−prop:2, THEN ≡E(2)] &I)
```

fix *u*
AOT-have ‹$w_1 \models O!u$›
  using *Ordinary.ψ[THEN RN,*
    THEN fund:2[unvarify p, OF log−prop−prop:2, THEN ≡E(1)],
    THEN PossibleWorld.∀ E] **by** *simp*
moreover AOT-have ‹$w_1 \models [\lambda y\ p]u$›
  by (*safe intro*!: *world-prop-beta[THEN PossibleWorld.∀ E, THEN ≡E(2)]*
    0[THEN &E(1)])
ultimately AOT-have ‹$w_1 \models (O!u\ \&\ [\lambda y\ p]u)$›
  using *conj−dist−w:1[unvarify p q, OF log−prop−prop:2,*
    OF log−prop−prop:2, THEN ≡E(2),
    OF &I] **by** *blast*
AOT-hence ‹$\exists x\ w_1 \models (O!x\ \&\ [\lambda y\ p]x)$›
  by (*rule ∃ I*)
AOT-thus ‹$w_1 \models \exists u\ [\lambda y\ p]u$›
  by (*metis conj−dist−w:6 intro−elim:3:b*)
next
  AOT-show ‹$w_1 \models \neg\exists v\ [\lambda z\ O!z\ \&\ z \neq_E z]v$›
    using *PossibleWorld.∀ E nec-not-ex* **by** *fastforce*
qed
ultimately AOT-show ‹$p\ \&\ \neg p$› **for** *p*
  using *raa−cor:3* **by** *blast*
qed
ultimately AOT-have ‹$y[\lambda y\ p]\ \&\ \neg x[\lambda y\ p]$›
  using *&I* **by** *blast*
AOT-hence ‹$\exists F\ (y[F]\ \&\ \neg x[F])$›
  by (*metis existential:1 prop−prop2:2*)
AOT-thus ‹$\exists F\ (x[F]\ \&\ \neg y[F]) \vee \exists F\ (y[F]\ \&\ \neg x[F])$›
  by (*rule ∨I*)
qed
ultimately AOT-have ‹$\Diamond Numbers'(x,[\lambda z\ O!z\ \&\ z \neq_E z])\ \&$
  $\Diamond Numbers'(y,[\lambda z\ O!z\ \&\ z \neq_E z])\ \&\ x \neq y$›
  using *&I* **by** *blast*
AOT-thus ‹$\exists x \exists y\ (\Diamond Numbers'(x,[\lambda z\ O!z\ \&\ z \neq_E z])\ \&$
  $\Diamond Numbers'(y,[\lambda z\ O!z\ \&\ z \neq_E z])\ \&\ x \neq y)$›
  using ∃ *I(2)*[**where** *β=x*] ∃ *I(2)*[**where** *β=y*] **by** *auto*
qed

AOT-theorem *restricted-identity*:
‹$x =_{\mathcal{R}} y \equiv (InDomainOf(x,\mathcal{R})\ \&\ InDomainOf(y,\mathcal{R})\ \&\ x = y)$›
by (*auto intro*!: *≡I →I &I*
  dest: *id−R−thm:2[THEN →E] &E*
    id−R−thm:3[THEN →E]
    id−R−thm:4[THEN →E, OF ∨I(1), THEN ≡E(2)])

AOT-theorem *induction′*: ‹$\forall F\ ([F]0\ \&\ \forall n([F]n \to [F]n') \to \forall n\ [F]n)$›
proof(*rule GEN*; *rule →I*)
fix *F n*
AOT-assume *A*: ‹$[F]0\ \&\ \forall n([F]n \to [F]n')$›
AOT-have ‹$\forall n \forall m([\mathbb{P}]nm \to ([F]n \to [F]m))$›
proof(*safe intro*!: *Number.GEN →I*)
  fix *n m*
  AOT-assume ‹$[\mathbb{P}]nm$›
  moreover AOT-have ‹$[\mathbb{P}]n\ n'$›
    using *suc−thm*.
  ultimately AOT-have *m-eq-suc-n*: ‹$m = n'$›
    using *pred−func:1[unvarify z, OF def−suc[den2], THEN →E, OF &I]*
    by *blast*
  AOT-assume ‹$[F]n$›
  AOT-hence ‹$[F]n'$›
    using *A[THEN &E(2), THEN Number.∀ E, THEN →E]* **by** *blast*
  AOT-thus ‹$[F]m$›
    using *m-eq-suc-n[symmetric] rule=E* **by** *fast*

```
  qed
  AOT-thus ‹∀ n[F]n›
    using induction[THEN ∀ E(2), THEN →E, OF &I, OF A[THEN &E(1)]]
    by simp
qed

AOT-define ExtensionOf :: ‹τ ⇒ Π ⇒ φ› (‹ExtensionOf′(-,-′)›)
  exten−property:1: ‹ExtensionOf(x,[G]) ≡df A!x & G↓ & ∀ F(x[F] ≡ ∀ z([F]z ≡ [G]z))›

AOT-define OrdinaryExtensionOf :: ‹τ ⇒ Π ⇒ φ› (‹OrdinaryExtensionOf′(-,-′)›)
  ‹OrdinaryExtensionOf(x,[G]) ≡df A!x & G↓ & ∀ F(x[F] ≡ ∀ z(O!z → ([F]z ≡ [G]z)))›

AOT-theorem BeingOrdinaryExtensionOfDenotes:
  ‹[λx OrdinaryExtensionOf(x,[G])]↓›
proof(rule safe−ext[axiom-inst, THEN →E, OF &I])
  AOT-show ‹[λx A!x & G↓ & [λx ∀ F(x[F] ≡ ∀ z(O!z → ([F]z ≡ [G]z)))]x]↓›
    by cqt:2
next
  AOT-show ‹□∀ x (A!x & G↓ & [λx ∀ F (x[F] ≡ ∀ z (O!z → ([F]z ≡ [G]z)))]x ≡
          OrdinaryExtensionOf(x,[G]))›
  proof(safe intro!: RN GEN)
    AOT-modally-strict {
      fix x
      AOT-modally-strict {
        AOT-have ‹[λx ∀ F (x[F] ≡ ∀ z (O!z → ([F]z ≡ [G]z)))]↓›
        proof (safe intro!: Comprehension-3[THEN →E] RN GEN
                      →I ≡I Ordinary.GEN)
          AOT-modally-strict {
            fix F H u
            AOT-assume ‹□H ≡E F›
            AOT-hence ‹∀ u([H]u ≡ [F]u)›
              using eqE[THEN ≡dfE, THEN &E(2)] qml:2[axiom-inst, THEN →E]
              by blast
            AOT-hence 0: ‹[H]u ≡ [F]u› using Ordinary.∀ E by fast
            {
              AOT-assume ‹∀ u([F]u ≡ [G]u)›
              AOT-hence 1: ‹[F]u ≡ [G]u› using Ordinary.∀ E by fast
              AOT-show ‹[G]u› if ‹[H]u› using 0 1 ≡E(1) that by blast
              AOT-show ‹[H]u› if ‹[G]u› using 0 1 ≡E(2) that by blast
            }
            {
              AOT-assume ‹∀ u([H]u ≡ [G]u)›
              AOT-hence 1: ‹[H]u ≡ [G]u› using Ordinary.∀ E by fast
              AOT-show ‹[G]u› if ‹[F]u› using 0 1 ≡E(1,2) that by blast
              AOT-show ‹[F]u› if ‹[G]u› using 0 1 ≡E(1,2) that by blast
            }
          }
        qed
      }
      AOT-thus ‹(A!x & G↓ & [λx ∀ F (x[F] ≡ ∀ z (O!z → ([F]z ≡ [G]z)))]x) ≡
              OrdinaryExtensionOf(x,[G])›
        apply (AOT-subst-def OrdinaryExtensionOf)
        apply (AOT-subst ‹[λx ∀ F (x[F] ≡ ∀ z (O!z → ([F]z ≡ [G]z)))]x›
                      ‹∀ F (x[F] ≡ ∀ z (O!z → ([F]z ≡ [G]z)))›)
        by (auto intro!: beta−C−meta[THEN →E] simp: oth−class−taut:3:a)
    }
  qed
qed
```

Fragments of PLM's theory of Concepts.

```
AOT-define FimpG :: ‹Π ⇒ Π ⇒ φ› (infixl ‹⇒› 50)
  F−imp−G: ‹[G] ⇒ [F] ≡df F↓ & G↓ & □∀ x ([G]x → [F]x)›
```

AOT-define *concept* :: ‹Π› (‹*C*!›)
concepts: ‹*C*! $=_{df}$ *A*!›

AOT-register-rigid-restricted-type
Concept: ‹*C*!κ›
proof
AOT-modally-strict {
AOT-have ‹∃ *x* *A*!*x*›
using *o*−*objects*−*exist*:*2* *qml*:*2*[*axiom-inst*] →*E* **by** *blast*
AOT-thus ‹∃ *x* *C*!*x*›
using *rule*−*id*−*df*:*1*[*zero*][*OF concepts*, *OF oa*−*exist*:*2*] *rule*=*E id-sym*
by *fast*
}
next
AOT-modally-strict {
AOT-show ‹*C*!κ → κ↓› **for** κ
using *cqt*:*5*:*a*[*axiom-inst*, *THEN* →*E*, *THEN* &*E*(*2*)] →*I*
by *blast*
}
next
AOT-modally-strict {
AOT-have ‹∀ *x*(*A*!*x* → □*A*!*x*)›
by (*simp add*: *oa*−*facts*:*2 GEN*)
AOT-thus ‹∀ *x*(*C*!*x* → □*C*!*x*)›
using *rule*−*id*−*df*:*1*[*zero*][*OF concepts*, *OF oa*−*exist*:*2*] *rule*=*E id-sym*
by *fast*
}
qed

AOT-register-variable-names
Concept: *c d e*

AOT-theorem *concept*−*comp*:*1*: ‹∃ *x*(*C*!*x* & ∀ *F*(*x*[*F*] ≡ φ{*F*}))›
using *concepts*[*THEN rule*−*id*−*df*:*1*[*zero*], *OF oa*−*exist*:*2*, *symmetric*]
A−*objects*[*axiom-inst*]
rule=*E* **by** *fast*

AOT-theorem *concept*−*comp*:*2*: ‹∃ !*x*(*C*!*x* & ∀ *F*(*x*[*F*] ≡ φ{*F*}))›
using *concepts*[*THEN rule*−*id*−*df*:*1*[*zero*], *OF oa*−*exist*:*2*, *symmetric*]
A−*objects*!
rule=*E* **by** *fast*

AOT-theorem *concept*−*comp*:*3*: ‹ι*x*(*C*!*x* & ∀ *F*(*x*[*F*] ≡ φ{*F*}))↓›
using *concept*−*comp*:*2 A*−*Exists*:*2*[*THEN* ≡*E*(*2*)] *RA*[*2*] **by** *blast*

AOT-theorem *concept*−*comp*:*4*:
‹ι*x*(*C*!*x* & ∀ *F*(*x*[*F*] ≡ φ{*F*})) = ι*x*(*A*!*x* & ∀ *F*(*x*[*F*] ≡ φ{*F*}))›
using =*I*(*1*)[*OF concept*−*comp*:*3*]
rule=*E*[*rotated*]
concepts[*THEN rule*−*id*−*df*:*1*[*zero*], *OF oa*−*exist*:*2*]
by *fast*

AOT-define *conceptInclusion* :: ‹τ ⇒ τ ⇒ φ› (**infixl** ‹⪯› *100*)
con:*1*: ‹*c* ⪯ *d* $\equiv_{df}$ ∀ *F*(*c*[*F*] → *d*[*F*])›

AOT-define *conceptOf* :: ‹τ ⇒ τ ⇒ φ› (‹*ConceptOf*′(-,-′)›)
concept−*of*−*G*: ‹*ConceptOf*(*c*,*G*) $\equiv_{df}$ *G*↓ & ∀ *F* (*c*[*F*] ≡ [*G*] ⇒ [*F*])›

AOT-theorem *ConceptOfOrdinaryProperty*: ‹([*H*] ⇒ *O*!) → [λ*x ConceptOf*(*x*,*H*)]↓›
proof(*rule* →*I*)
AOT-assume ‹[*H*] ⇒ *O*!›
AOT-hence ‹□∀ *x*([*H*]*x* → *O*!*x*)›

```
  using F−imp−G[THEN ≡df E] &E by blast
AOT-hence ‹□□∀x([H]x → O!x)›
  using S5Basic:6[THEN ≡E(1)] by blast
moreover AOT-have ‹□□∀x([H]x → O!x) →
                  □∀F∀G(□(G ≡E F) → ([H] ⇒ [F] ≡ [H] ⇒ [G]))›
proof(rule RM; safe intro!: →I GEN ≡I)
  AOT-modally-strict {
    fix F G
    AOT-assume 0: ‹□∀x([H]x → O!x)›
    AOT-assume ‹□G ≡E F›
    AOT-hence 1: ‹□∀u([G]u ≡ [F]u)›
      by (AOT-subst-thm eqE[THEN ≡Df, THEN ≡S(1), OF &I,
            OF cqt:2[const-var][axiom-inst],
            OF cqt:2[const-var][axiom-inst], symmetric])
    {
      AOT-assume ‹[H] ⇒ [F]›
      AOT-hence ‹□∀x([H]x → [F]x)›
        using F−imp−G[THEN ≡df E] &E by blast
      moreover AOT-modally-strict {
        AOT-assume ‹∀x([H]x → O!x)›
        moreover AOT-assume ‹∀u([G]u ≡ [F]u)›
        moreover AOT-assume ‹∀x([H]x → [F]x)›
        ultimately AOT-have ‹[H]x → [G]x› for x
          by (auto intro!: →I dest!: ∀E(2) dest: →E ≡E)
        AOT-hence ‹∀x([H]x → [G]x)›
          by (rule GEN)
      }
      ultimately AOT-have ‹□∀x([H]x → [G]x)›
        using RN[prem][where
            Γ={«∀x([H]x → O!x)», «∀u([G]u ≡ [F]u)», «∀x([H]x → [F]x)»}]
        using 0 1 by fast
      AOT-thus ‹[H] ⇒ [G]›
        by (AOT-subst-def F−imp−G)
          (safe intro!: cqt:2 &I)
    }
    {
      AOT-assume ‹[H] ⇒ [G]›
      AOT-hence ‹□∀x([H]x → [G]x)›
        using F−imp−G[THEN ≡df E] &E by blast
      moreover AOT-modally-strict {
        AOT-assume ‹∀x([H]x → O!x)›
        moreover AOT-assume ‹∀u([G]u ≡ [F]u)›
        moreover AOT-assume ‹∀x([H]x → [G]x)›
        ultimately AOT-have ‹[H]x → [F]x› for x
          by (auto intro!: →I dest!: ∀E(2) dest: →E ≡E)
        AOT-hence ‹∀x([H]x → [F]x)›
          by (rule GEN)
      }
      ultimately AOT-have ‹□∀x([H]x → [F]x)›
        using RN[prem][where
            Γ={«∀x([H]x → O!x)», «∀u([G]u ≡ [F]u)», «∀x([H]x → [G]x)»}]
        using 0 1 by fast
      AOT-thus ‹[H] ⇒ [F]›
        by (AOT-subst-def F−imp−G)
          (safe intro!: cqt:2 &I)
    }
  }
qed
ultimately AOT-have ‹□∀F∀G(□(G ≡E F) → ([H] ⇒ [F] ≡ [H] ⇒ [G]))›
  using →E by blast
AOT-hence 0: ‹[λx ∀F(x[F] ≡ ([H] ⇒ [F]))]↓›
  using Comprehension-3[THEN →E] by blast
AOT-show ‹[λx ConceptOf(x,H)]↓›
```

```
  proof (rule safe−ext[axiom-inst, THEN →E, OF &I])
    AOT-show ‹[λx C!x & [λx ∀F(x[F] ≡ ([H] ⇒ [F]))]x]↓› by cqt:2
  next
    AOT-show ‹□∀x (C!x & [λx ∀F (x[F] ≡ [H] ⇒ [F])]x ≡ ConceptOf(x,H))›
    proof (rule RN[prem][where Γ=‹{«[λx ∀F(x[F] ≡ ([H] ⇒ [F]))]↓»}›, simplified])
      AOT-modally-strict {
        AOT-assume 0: ‹[λx ∀F (x[F] ≡ [H] ⇒ [F])]↓›
        AOT-show ‹∀x (C!x & [λx ∀F (x[F] ≡ [H] ⇒ [F])]x ≡ ConceptOf(x,H))›
        proof(safe intro!: GEN ≡I →I &I)
          fix x
          AOT-assume ‹C!x & [λx ∀F (x[F] ≡ [H] ⇒ [F])]x›
          AOT-thus ‹ConceptOf(x,H)›
            by (AOT-subst-def concept−of−G)
               (auto intro!: &I cqt:2 dest: &E β→C)
        next
          fix x
          AOT-assume ‹ConceptOf(x,H)›
          AOT-hence ‹C!x & (H↓ & ∀F(x[F] ≡ [H] ⇒ [F]))›
            by (AOT-subst-def (reverse) concept−of−G)
          AOT-thus ‹C!x› and ‹[λx ∀F(x[F] ≡ [H] ⇒ [F])]x›
            by (auto intro!: β←C 0 cqt:2 dest: &E)
        qed
      }
    next
      AOT-show ‹□[λx ∀F(x[F] ≡ ([H] ⇒ [F]))]↓›
        using exist−nec[THEN →E] 0 by blast
    qed
  qed
qed

AOT-theorem con−exists:1: ‹∃c ConceptOf(c,G)›
proof −
  AOT-obtain c where ‹∀F (c[F] ≡ [G] ⇒ [F])›
    using concept−comp:1 Concept.∃E[rotated] by meson
  AOT-hence ‹ConceptOf(c,G)›
    by (auto intro!: concept−of−G[THEN ≡dfI] &I cqt:2 Concept.ψ)
  thus ?thesis by (rule Concept.∃I)
qed

AOT-theorem con−exists:2: ‹∃!c ConceptOf(c,G)›
proof −
  AOT-have ‹∃!c ∀F (c[F] ≡ [G] ⇒ [F])›
    using concept−comp:2 by simp
  moreover {
    AOT-modally-strict {
      fix x
      AOT-assume ‹∀F (x[F] ≡ [G] ⇒ [F])›
      moreover AOT-have ‹[G] ⇒ [G]›
        by (safe intro!: F−imp−G[THEN ≡dfI] &I cqt:2 RN GEN →I)
      ultimately AOT-have ‹x[G]›
        using ∀E(2) ≡E by blast
      AOT-hence ‹A!x›
        using encoders−are−abstract[THEN →E, OF ∃I(2)] by simp
      AOT-hence ‹C!x›
        using concepts[THEN rule−id−df:1[zero], OF oa−exist:2, symmetric]
              rule=E[rotated]
        by fast
    }
  }
  ultimately show ?thesis
    by (AOT-subst ‹ConceptOf(c,G)› ‹∀F (c[F] ≡ [G] ⇒ [F])› for: c;
          AOT-subst-def concept−of−G)
       (auto intro!: ≡I →I &I cqt:2 Concept.ψ dest: &E)
```

```
qed

AOT-theorem con−exists:3: ‹ιc ConceptOf(c,G)↓›
  by (safe intro!: A−Exists:2[THEN ≡E(2)] con−exists:2[THEN RA[2]])


AOT-define theConceptOfG :: ‹τ ⇒ κs› (‹c-›)
  concept−G: ‹cG =df ιc ConceptOf(c, G)›

AOT-theorem concept−G[den]: ‹cG↓›
  by (auto intro!: rule−id−df:1[OF concept−G]
                  t=t−proper:1[THEN →E]
                  con−exists:3)


AOT-theorem concept−G[concept]: ‹C!cG›
proof −
  AOT-have ‹𝒜(C!cG & ConceptOf(cG, G))›
    by (auto intro!: actual−desc:2[unvarify x, THEN →E]
                    rule−id−df:1[OF concept−G]
                    concept−G[den]
                    con−exists:3)
  AOT-hence ‹𝒜C!cG›
    by (metis Act−Basic:2 con−dis−i−e:2:a intro−elim:3:a)
  AOT-hence ‹𝒜A!cG›
    using rule−id−df:1[zero][OF concepts, OF oa−exist:2]
          rule=E by fast
  AOT-hence ‹A!cG›
    using oa−facts:8[unvarify x, THEN ≡E(2)] concept−G[den] by blast
  thus ?thesis
    using rule−id−df:1[zero][OF concepts, OF oa−exist:2, symmetric]
          rule=E by fast
qed

AOT-theorem conG−strict: ‹cG = ιc ∀F(c[F] ≡ [G] ⇒ [F])›
proof (rule id−eq:3[unvarify α β γ, THEN →E])
  AOT-have ‹□∀x (C!x & ConceptOf(x,G) ≡ C!x & ∀F (x[F] ≡ [G] ⇒ [F]))›
    by (auto intro!: concept−of−G[THEN ≡df I] RN GEN ≡I →I &I cqt:2
              dest: &E;
        auto dest: ∀E(2) ≡E(1,2) dest!: &E(2) concept−of−G[THEN ≡df E])
  AOT-thus ‹cG = ιc ConceptOf(c, G) & ιc ConceptOf(c, G) = ιc ∀F(c[F] ≡ [G] ⇒ [F])›
    by (auto intro!: &I rule−id−df:1[OF concept−G] con−exists:3
                    equiv−desc−eq:3[THEN →E])
qed(auto simp: concept−G[den] con−exists:3 concept−comp:3)


AOT-theorem conG−lemma:1: ‹∀F(cG[F] ≡ [G] ⇒ [F])›
proof(safe intro!: GEN ≡I →I)
  fix F
  AOT-have ‹𝒜∀F(cG[F] ≡ [G] ⇒ [F])›
    using actual−desc:4[THEN →E, OF concept−comp:3,
                        THEN Act−Basic:2[THEN ≡E(1)],
                        THEN &E(2)]
          conG−strict[symmetric] rule=E by fast
  AOT-hence ‹𝒜(cG[F] ≡ [G] ⇒ [F])›
    using logic−actual−nec:3[axiom-inst, THEN ≡E(1)] ∀E(2)
    by blast
  AOT-hence 0: ‹𝒜cG[F] ≡ 𝒜[G] ⇒ [F]›
    using Act−Basic:5[THEN ≡E(1)] by blast
  {
    AOT-assume ‹cG[F]›
    AOT-hence ‹𝒜cG[F]›
      by(safe intro!: en−eq:10[1][unvarify x1, THEN ≡E(2)]
```

concept−G[*den*])
AOT-hence ‹$\mathcal{A}$[*G*] ⇒ [*F*]›
using *0*[*THEN* ≡*E*(*1*)] **by** *blast*
AOT-hence ‹$\mathcal{A}$(*F*↓ & *G*↓ & □∀*x*([*G*]*x* → [*F*]*x*))›
by (*AOT-subst-def* (*reverse*) *F−imp−G*)
AOT-hence ‹$\mathcal{A}$□∀*x*([*G*]*x* → [*F*]*x*)›
using *Act−Basic*:*2*[*THEN* ≡*E*(*1*)] &*E* **by** *blast*
AOT-hence ‹□∀*x*([*G*]*x* → [*F*]*x*)›
using *qml−act*:*2*[*axiom-inst*, *THEN* ≡*E*(*2*)] **by** *simp*
AOT-thus ‹[*G*] ⇒ [*F*]›
by (*AOT-subst-def F−imp−G*; *auto intro*!: &*I cqt*:*2*)
}
{
AOT-assume ‹[*G*] ⇒ [*F*]›
AOT-hence ‹□∀*x*([*G*]*x* → [*F*]*x*)›
by (*safe dest*!: *F−imp−G*[*THEN* $\equiv_{df}$*E*] &*E*(*2*))
AOT-hence ‹$\mathcal{A}$□∀*x*([*G*]*x* → [*F*]*x*)›
using *qml−act*:*2*[*axiom-inst*, *THEN* ≡*E*(*1*)] **by** *simp*
AOT-hence ‹$\mathcal{A}$(*F*↓ & *G*↓ & □∀*x*([*G*]*x* → [*F*]*x*))›
by (*auto intro*!: *Act−Basic*:*2*[*THEN* ≡*E*(*2*)] &*I cqt*:*2*
intro: *RA*[*2*])
AOT-hence ‹$\mathcal{A}$([*G*] ⇒ [*F*])›
by (*AOT-subst-def F−imp−G*)
AOT-hence ‹$\mathcal{A}\mathbf{c}_G$[*F*]›
using *0*[*THEN* ≡*E*(*2*)] **by** *blast*
AOT-thus ‹$\mathbf{c}_G$[*F*]›
by(*safe intro*!: *en−eq*:*10*[*1*][*unvarify* x_1, *THEN* ≡*E*(*1*)]
concept−G[*den*])
}
qed

AOT-theorem *conH-enc-ord*:
‹([*H*] ⇒ *O*!) → □∀*F* ∀*G* (□*G* $\equiv_E$ *F* → ($\mathbf{c}_H$[*F*] ≡ $\mathbf{c}_H$[*G*]))›
proof(*rule* →*I*)
AOT-assume *0*: ‹[*H*] ⇒ *O*!›
AOT-have *0*: ‹□([*H*] ⇒ *O*!)›
apply (*AOT-subst-def F−imp−G*)
using *0*[*THEN* $\equiv_{df}$*E*[*OF F−imp−G*]]
by (*auto intro*!: *KBasic*:*3*[*THEN* ≡*E*(*2*)] &*I exist−nec*[*THEN* →*E*]
dest: &*E 4*[*THEN* →*E*])
moreover AOT-have ‹□([*H*] ⇒ *O*!) → □∀*F* ∀*G* (□*G* $\equiv_E$ *F* → ($\mathbf{c}_H$[*F*] ≡ $\mathbf{c}_H$[*G*]))›
proof(*rule RM*; *safe intro*!: →*I GEN*)
AOT-modally-strict {
fix *F G*
AOT-assume ‹[*H*] ⇒ *O*!›
AOT-hence *0*: ‹□∀*x* ([*H*]*x* → *O*!*x*)›
by (*safe dest*!: *F−imp−G*[*THEN* $\equiv_{df}$*E*] &*E*(*2*))
AOT-assume *1*: ‹□*G* $\equiv_E$ *F*›
AOT-assume ‹$\mathbf{c}_H$[*F*]›
AOT-hence ‹[*H*] ⇒ [*F*]›
using *conG−lemma*:*1*[*THEN* ∀*E*(*2*), *THEN* ≡*E*(*1*)] **by** *simp*
AOT-hence *2*: ‹□∀*x* ([*H*]*x* → [*F*]*x*)›
by (*safe dest*!: *F−imp−G*[*THEN* $\equiv_{df}$*E*] &*E*(*2*))
AOT-modally-strict {
AOT-assume *0*: ‹∀*x* ([*H*]*x* → *O*!*x*)›
AOT-assume *1*: ‹∀*x* ([*H*]*x* → [*F*]*x*)›
AOT-assume *2*: ‹*G* $\equiv_E$ *F*›
AOT-have ‹∀*x* ([*H*]*x* → [*G*]*x*)›
proof(*safe intro*!: *GEN* →*I*)
fix *x*
AOT-assume ‹[*H*]*x*›
AOT-hence ‹*O*!*x*› **and** ‹[*F*]*x*›
using *0 1* ∀*E*(*2*) →*E* **by** *blast*+

```
        AOT-thus ‹[G]x›
          using 2[THEN eqE[THEN ≡df E], THEN &E(2)]
                ∀ E(2) →E ≡E(2) calculation by blast
      qed
    }
    AOT-hence ‹□∀ x ([H]x → [G]x)›
      using RN[prem][where Γ=‹{«∀ x ([H]x → O!x)»,
                             «∀ x ([H]x → [F]x)»,
                             «G ≡E F»}›, simplified] 0 1 2 by fast
    AOT-hence ‹[H] ⇒ [G]›
      by (safe intro!: F−imp−G[THEN ≡df I] &I cqt:2)
    AOT-hence ‹cH[G]›
      using conG−lemma:1[THEN ∀ E(2), THEN ≡E(2)] by simp
  } note 0 = this
  AOT-modally-strict {
    fix F G
    AOT-assume ‹[H] ⇒ O!›
    moreover AOT-assume ‹□G ≡E F›
    moreover AOT-have ‹□F ≡E G›
      by (AOT-subst ‹F ≡E G› ‹G ≡E F›)
         (auto intro!: calculation(2)
                       eqE[THEN ≡df I]
                       ≡I →I &I cqt:2 Ordinary.GEN
                  dest!: eqE[THEN ≡df E] &E(2)
                  dest: ≡E(1,2) Ordinary.∀ E)
    ultimately AOT-show ‹(cH[F] ≡ cH[G])›
      using 0 ≡I →I by auto
  }
qed
ultimately AOT-show ‹□∀ F ∀ G (□G ≡E F → (cH[F] ≡ cH[G]))›
  using →E by blast
qed

AOT-theorem concept-inclusion-denotes-1:
 ‹([H] ⇒ O!) → [λx cH ⪯ x]↓›
proof(rule →I)
 AOT-assume 0: ‹[H] ⇒ O!›
 AOT-show ‹[λx cH ⪯ x]↓›
 proof(rule safe−ext[axiom-inst, THEN →E, OF &I])
   AOT-show ‹[λx C!x & ∀ F(cH[F] → x[F])]↓›
     by (safe intro!: conjunction-denotes[THEN →E, OF &I]
                      Comprehension-2′[THEN →E]
                      conH-enc-ord[THEN →E, OF 0]) cqt:2
 next
   AOT-show ‹□∀ x (C!x & ∀ F (cH[F] → x[F]) ≡ cH ⪯ x)›
     by (safe intro!: RN GEN; AOT-subst-def con:1)
        (auto intro!: ≡I →I &I concept−G[concept] dest: &E)
 qed
qed

AOT-theorem concept-inclusion-denotes-2:
 ‹([H] ⇒ O!) → [λx x ⪯ cH]↓›
proof(rule →I)
 AOT-assume 0: ‹[H] ⇒ O!›
 AOT-show ‹[λx x ⪯ cH]↓›
 proof(rule safe−ext[axiom-inst, THEN →E, OF &I])
   AOT-show ‹[λx C!x & ∀ F(x[F] → cH[F])]↓›
     by (safe intro!: conjunction-denotes[THEN →E, OF &I]
                      Comprehension-1′[THEN →E]
                      conH-enc-ord[THEN →E, OF 0]) cqt:2
 next
   AOT-show ‹□∀ x (C!x & ∀ F (x[F] → cH[F]) ≡ x ⪯ cH)›
     by (safe intro!: RN GEN; AOT-subst-def con:1)
```

(*auto intro*!: ≡*I* →*I* &*I concept*−*G*[*concept*] *dest*: &*E*)
qed
qed

AOT-define *ThickForm* :: ‹τ ⇒ τ ⇒ φ› (‹*FormOf*′(-,-′)›)
tform−*of*: ‹*FormOf*(*x*,*G*) $\equiv_{df}$ *A*!*x* & *G*↓ & ∀ *F*(*x*[*F*] ≡ [*G*] ⇒ [*F*])›

AOT-theorem *FormOfOrdinaryProperty*: ‹([*H*] ⇒ *O*!) → [λ*x FormOf*(*x*,*H*)]↓›
proof(*rule* →*I*)
AOT-assume *0*: ‹[*H*] ⇒ [*O*!]›
AOT-show ‹[λ*x FormOf*(*x*,*H*)]↓›
proof (*rule safe*−*ext*[*axiom-inst*, *THEN* →*E*, *OF* &*I*])
AOT-show ‹[λ*x ConceptOf*(*x*,*H*)]↓›
using *0 ConceptOfOrdinaryProperty*[*THEN* →*E*] **by** *blast*
AOT-show ‹□∀ *x* (*ConceptOf*(*x*,*H*) ≡ *FormOf*(*x*,*H*))›
proof(*safe intro*!: *RN GEN*)
AOT-modally-strict {
fix *x*
AOT-modally-strict {
AOT-have ‹*A*!*x* ≡ *A*!*x*›
by (*simp add*: *oth*−*class*−*taut*:*3*:*a*)
AOT-hence ‹*C*!*x* ≡ *A*!*x*›
using *rule*−*id*−*df*:*1*[*zero*][*OF concepts*, *OF oa*−*exist*:*2*]
rule=*E id-sym* **by** *fast*
}
AOT-thus ‹*ConceptOf*(*x*,*H*) ≡ *FormOf*(*x*,*H*)›
by (*AOT-subst-def tform*−*of*;
AOT-subst-def concept−*of*−*G*;
AOT-subst ‹*C*!*x*› ‹*A*!*x*›)
(*auto intro*!: ≡*I* →*I* &*I dest*: &*E*)
}
qed
qed
qed

AOT-theorem *equal-E-rigid-one-to-one*: ‹$Rigid_{1-1}$((=$_E$))›
proof (*safe intro*!: *df*−*1*−*1*:*2*[*THEN* $\equiv_{df}$*I*] &*I df*−*1*−*1*:*1*[*THEN* $\equiv_{df}$*I*]
GEN →*I df*−*rigid*−*rel*:*1*[*THEN* $\equiv_{df}$*I*] =*E*[*denotes*])
fix *x y z*
AOT-assume ‹*x* =$_E$ *z* & *y* =$_E$ *z*›
AOT-thus ‹*x* = *y*›
by (*metis rule*=*E* &*E*(*1*) *Conjunction Simplification*(*2*)
=*E*−*simple*:*2 id-sym* →*E*)
next
AOT-have ‹∀ *x*∀ *y* □(*x* =$_E$ *y* → □*x* =$_E$ *y*)›
proof(*rule GEN*; *rule GEN*)
AOT-show ‹□(*x* =$_E$ *y* → □*x* =$_E$ *y*)› **for** *x y*
by (*meson RN deduction*−*theorem id*−*nec3*:*1* ≡*E*(*1*))
qed
AOT-hence ‹∀ x_1...∀ x_n □([(=$_E$)]x_1...x_n → □[(=$_E$)]x_1...x_n)›
by (*rule tuple-forall*[*THEN* $\equiv_{df}$*I*])
AOT-thus ‹□∀ x_1...∀ x_n ([(=$_E$)]x_1...x_n → □[(=$_E$)]x_1...x_n)›
using *BF*[*THEN* →*E*] **by** *fast*
qed

AOT-theorem *equal-E-domain*: ‹*InDomainOf*(*x*,(=$_E$)) ≡ *O*!*x*›
proof(*safe intro*!: ≡*I* →*I*)
AOT-assume ‹*InDomainOf*(*x*,(=$_E$))›
AOT-hence ‹∃ *y x* =$_E$ *y*›
by (*metis* $\equiv_{df}$*E df*−*1*−*1*:*5*)
then AOT-obtain *y* **where** ‹*x* =$_E$ *y*›
using ∃ *E*[*rotated*] **by** *blast*
AOT-thus ‹*O*!*x*›

```
    using =E−simple:1[THEN ≡E(1)] &E by blast
next
  AOT-assume ‹O!x›
  AOT-hence ‹x =E x›
    by (metis ord=Eequiv:1[THEN →E])
  AOT-hence ‹∃ y x =E y›
    using ∃ I(2) by fast
  AOT-thus ‹InDomainOf(x,(=E))›
    by (metis ≡df I df−1−1:5)
qed

AOT-theorem shared-urelement-projection-identity:
  assumes ‹∀ y [λx (y[λz [R]zx])]↓›
  shows ‹∀ F([F]a ≡ [F]b) → [λz [R]za] = [λz [R]zb]›
proof(rule →I)
  AOT-assume 0: ‹∀ F([F]a ≡ [F]b)›
  {
    fix z
    AOT-have ‹[λx (z[λz [R]zx])]↓›
      using assms[THEN ∀ E(2)].
    AOT-hence 1: ‹∀ x ∀ y (∀ F ([F]x ≡ [F]y) → □(z[λz [R]zx] ≡ z[λz [R]zy]))›
      using kirchner−thm−cor:1[THEN →E]
      by blast
    AOT-have ‹□(z[λz [R]za] ≡ z[λz [R]zb])›
      using 1[THEN ∀ E(2), THEN ∀ E(2), THEN →E, OF 0] by blast
  }
  AOT-hence ‹∀ z □(z[λz [R]za] ≡ z[λz [R]zb])›
    by (rule GEN)
  AOT-hence ‹□∀ z(z[λz [R]za] ≡ z[λz [R]zb])›
    by (rule BF[THEN →E])
  AOT-thus ‹[λz [R]za] = [λz [R]zb]›
    by (AOT-subst-def identity:2)
       (auto intro!: &I cqt:2)
qed

AOT-theorem shared-urelement-exemplification-identity:
  assumes ‹∀ y [λx (y[λz [G]x])]↓›
  shows ‹∀ F([F]a ≡ [F]b) → ([G]a) = ([G]b)›
proof(rule →I)
  AOT-assume 0: ‹∀ F([F]a ≡ [F]b)›
  {
    fix z
    AOT-have ‹[λx (z[λz [G]x])]↓›
      using assms[THEN ∀ E(2)].
    AOT-hence 1: ‹∀ x ∀ y (∀ F ([F]x ≡ [F]y) → □(z[λz [G]x] ≡ z[λz [G]y]))›
      using kirchner−thm−cor:1[THEN →E]
      by blast
    AOT-have ‹□(z[λz [G]a] ≡ z[λz [G]b])›
      using 1[THEN ∀ E(2), THEN ∀ E(2), THEN →E, OF 0] by blast
  }
  AOT-hence ‹∀ z □(z[λz [G]a] ≡ z[λz [G]b])›
    by (rule GEN)
  AOT-hence ‹□∀ z(z[λz [G]a] ≡ z[λz [G]b])›
    by (rule BF[THEN →E])
  AOT-hence ‹[λz [G]a] = [λz [G]b]›
    by (AOT-subst-def identity:2)
       (auto intro!: &I cqt:2)
  AOT-thus ‹([G]a) = ([G]b)›
    by (safe intro!: identity:4[THEN ≡df I] &I log−prop−prop:2)
qed
```

The assumptions of the theorems above are derivable, if the additional introduction rules for the upcoming extension of *AOT-instance-of-cqt-2* $\varphi \Longrightarrow [\lambda\nu_1...\nu_n\ \varphi\{\nu_1...\nu_n\}]\downarrow \in \Lambda_\square$ are explicitly allowed (while they are currently not part of the abstraction layer).

notepad
begin
  AOT-modally-strict {
    AOT-have ‹$\forall R \forall y\ [\lambda x\ (y[\lambda z\ [R]zx])]\downarrow$›
      by (*safe intro*!: *GEN cqt*:*2 AOT-instance-of-cqt-2-intro-next*)
    AOT-have ‹$\forall G \forall y\ [\lambda x\ (y[\lambda z\ [G]x])]\downarrow$›
      by (*safe intro*!: *GEN cqt*:*2 AOT-instance-of-cqt-2-intro-next*)
  }
end

end