Metatheory of $\mathcal{Q}_0$

Javier Díaz
<javier.diaz.manzi@gmail.com>

May 14, 2024

Abstract

This entry is a formalization of the metatheory of $\mathcal{Q}_0$ in Isabelle/HOL. $\mathcal{Q}_0$ [2] is a classical higher-order logic equivalent to Church's Simple Theory of Types. In this entry we formalize Chapter 5 of [2], up to and including the proofs of soundness and consistency of $\mathcal{Q}_0$. These proof are, to the best of our knowledge, the first to be formalized in a proof assistant.

Contents

1 Utilities

theory *Utilities*
imports
Finite−Map−Extras.Finite-Map-Extras
begin

1.1 Utilities for lists

fun *foldr1* :: $('a \Rightarrow 'a \Rightarrow 'a) \Rightarrow 'a\ list \Rightarrow 'a$ **where**
foldr1 f $[x] = x$
| *foldr1 f* $(x \# xs) = f\ x$ (*foldr1 f xs*)
| *foldr1 f* $[] =$ *undefined f*

abbreviation *lset* **where** *lset* ≡ *List.set*

lemma *rev-induct2* [*consumes 1*, *case-names Nil snoc*]:
assumes *length xs* = *length ys*
and *P* [] []
and $\bigwedge x\ xs\ y\ ys.$ *length xs* = *length ys* $\Longrightarrow$ *P xs ys* $\Longrightarrow$ *P* (*xs* @ [*x*]) (*ys* @ [*y*])
shows *P xs ys*
using *assms* **proof** (*induction xs arbitrary*: *ys rule*: *rev-induct*)
case (*snoc x xs*)
then show *?case* **by** (*cases ys rule*: *rev-cases*) *simp-all*
qed *simp*

1.2 Utilities for finite maps

no-syntax
-fmaplet :: ['*a*, '*a*] ⇒ *fmaplet* (- /$$:=/ -)
-fmaplets :: ['*a*, '*a*] ⇒ *fmaplet* (- /[$$:=]/ -)

syntax
-fmaplet :: ['*a*, '*a*] ⇒ *fmaplet* (- /↣/ -)
-fmaplets :: ['*a*, '*a*] ⇒ *fmaplet* (- /[↣]/ -)

lemma *fmdom′-fmap-of-list* [*simp*]:
shows *fmdom′* (*fmap-of-list ps*) = *lset* (*map fst ps*)
by (*induction ps*) *force+*

lemma *fmran′-singleton* [*simp*]:
shows *fmran′* {*k* ↣ *v*} = {*v*}
proof −
have $v' \in$ *fmran′* {*k* ↣ *v*} $\Longrightarrow v' = v$ **for** v'
proof −
assume $v' \in$ *fmran′* {*k* ↣ *v*}
fix k'
have *fmdom′* {*k* ↣ *v*} = {*k*}
by *simp*
then show $v' = v$

```
    proof (cases k' = k)
      case True
      with ‹v' ∈ fmran' {k ↣ v}› show ?thesis
        using fmdom'I by fastforce
    next
      case False
      with ‹fmdom' {k ↣ v} = {k}› and ‹v' ∈ fmran' {k ↣ v}› show ?thesis
        using fmdom'I by fastforce
    qed
  qed
  moreover have v ∈ fmran' {k ↣ v}
    by (simp add: fmran'I)
  ultimately show ?thesis
    by blast
qed

lemma fmran'-fmupd [simp]:
  assumes m $$ x = None
  shows fmran' (m(x ↣ y)) = {y} ∪ fmran' m
using assms proof (intro subset-antisym subsetI)
  fix x'
  assume m $$ x = None and x' ∈ fmran' (m(x ↣ y))
  then show x' ∈ {y} ∪ fmran' m
    by (auto simp add: fmlookup-ran'-iff, metis option.inject)
next
  fix x'
  assume m $$ x = None and x' ∈ {y} ∪ fmran' m
  then show x' ∈ fmran' (m(x ↣ y))
    by (force simp add: fmlookup-ran'-iff)
qed

lemma fmran'-fmadd [simp]:
  assumes fmdom' m ∩ fmdom' m' = {}
  shows fmran' (m ++_f m') = fmran' m ∪ fmran' m'
using assms proof (intro subset-antisym subsetI)
  fix x
  assume fmdom' m ∩ fmdom' m' = {} and x ∈ fmran' (m ++_f m')
  then show x ∈ fmran' m ∪ fmran' m'
    by (auto simp add: fmlookup-ran'-iff) meson
next
  fix x
  assume fmdom' m ∩ fmdom' m' = {} and x ∈ fmran' m ∪ fmran' m'
  then show x ∈ fmran' (m ++_f m')
    using fmap-disj-comm and fmlookup-ran'-iff by fastforce
qed

lemma finite-fmran':
  shows finite (fmran' m)
  by (simp add: fmran'-alt-def)
```

lemma *fmap-of-zipped-list-range*:
  assumes *length ks = length vs*
  and *m = fmap-of-list (zip ks vs)*
  and *k ∈ fmdom′ m*
  shows *m $$! k ∈ lset vs*
  using *assms* **by** (*induction arbitrary*: *m rule*: *list-induct2*) *auto*

lemma *fmap-of-zip-nth* [*simp*]:
  assumes *length ks = length vs*
  and *distinct ks*
  and *i < length ks*
  shows *fmap-of-list (zip ks vs) $$! (ks ! i) = vs ! i*
  using *assms* **by** (*simp add*: *fmap-of-list.rep-eq map-of-zip-nth*)

lemma *fmap-of-zipped-list-fmran′* [*simp*]:
  assumes *distinct (map fst ps)*
  shows *fmran′ (fmap-of-list ps) = lset (map snd ps)*
using *assms* **proof** (*induction ps*)
  case *Nil*
  then show *?case*
    by *auto*
next
  case (*Cons p ps*)
  then show *?case*
  proof (*cases p ∈ lset ps*)
    case *True*
    then show *?thesis*
      using *Cons.prems* **by** *auto*
  next
    case *False*
    obtain *k* **and** *v* **where** *p = (k, v)*
      by *fastforce*
    with *Cons.prems* **have** *k ∉ fmdom′ (fmap-of-list ps)*
      by *auto*
    then have *fmap-of-list (p # ps) = {k ↣ v} ++$_f$ fmap-of-list ps*
      using ‹*p = (k, v)*› **and** *fmap-singleton-comm* **by** *fastforce*
    with *Cons.prems* **have** *fmran′ (fmap-of-list (p # ps)) = {v} ∪ fmran′ (fmap-of-list ps)*
      by (*simp add*: ‹*p = (k, v)*›)
    then have *fmran′ (fmap-of-list (p # ps)) = {v} ∪ lset (map snd ps)*
      using *Cons.IH* **and** *Cons.prems* **by** *force*
    then show *?thesis*
      by (*simp add*: ‹*p = (k, v)*›)
  qed
qed

lemma *fmap-of-list-nth* [*simp*]:
  assumes *distinct (map fst ps)*
    and *j < length ps*

```
  shows fmap-of-list ps $$ ((map fst ps) ! j) = Some (map snd ps ! j)
  using assms by (induction j) (simp-all add: fmap-of-list.rep-eq)

lemma fmap-of-list-nth-split [simp]:
  assumes distinct xs
    and j < length xs
    and length ys = length xs and length zs = length xs
  shows fmap-of-list (zip xs (take k ys @ drop k zs)) $$ (xs ! j) =
        (if j < k then Some (take k ys ! j) else Some (drop k zs ! (j − k)))
using assms proof (induction k arbitrary: xs ys zs j)
  case 0
  then show ?case
    by (simp add: fmap-of-list.rep-eq map-of-zip-nth)
next
  case (Suc k)
  then show ?case
  proof (cases xs)
    case Nil
    with Suc.prems(2) show ?thesis
      by auto
  next
    case (Cons x xs′)
    let ?ps = zip xs (take (Suc k) ys @ drop (Suc k) zs)
    from Cons and Suc.prems(3,4) obtain y and z and ys′ and zs′
      where ys = y # ys′ and zs = z # zs′
      by (metis length-0-conv neq-Nil-conv)
    let ?ps′ = zip xs′ (take k ys′ @ drop k zs′)
    from Cons have ∗: fmap-of-list ?ps = fmap-of-list ((x, y) # ?ps′)
      using ‹ys = y # ys′› and ‹zs = z # zs′› by fastforce
    also have ... = {x ↣ y} ++f fmap-of-list ?ps′
    proof −
      from ‹ys = y # ys′› and ‹zs = z # zs′› have fmap-of-list ?ps′ $$ x = None
        using Cons and Suc.prems(1,3,4) by (simp add: fmdom′-notD)
      then show ?thesis
        using fmap-singleton-comm by fastforce
    qed
    finally have fmap-of-list ?ps = {x ↣ y} ++f fmap-of-list ?ps′ .
    then show ?thesis
    proof (cases j = 0)
      case True
      with ‹ys = y # ys′› and Cons show ?thesis
        by simp
    next
      case False
      then have xs ! j = xs′ ! (j − 1)
        by (simp add: Cons)
      moreover from ‹ys = y # ys′› and ‹zs = z # zs′› have fmdom′ (fmap-of-list ?ps′) = lset xs′
        using Cons and Suc.prems(3,4) by force
      moreover from False and Suc.prems(2) and Cons have j − 1 < length xs′
```

```
      using le-simps(2) by auto
    ultimately have fmap-of-list ?ps $$ (xs ! j) = fmap-of-list ?ps' $$ (xs' ! (j - 1))
      using Cons and * and Suc.prems(1) by auto
    with Suc.IH and Suc.prems(1,3,4) and Cons have **: fmap-of-list ?ps $$ (xs ! j) =
      (if j - 1 < k then Some (take k ys' ! (j - 1)) else Some (drop k zs' ! ((j - 1) - k)))
      using ‹j - 1 < length xs'› and ‹ys = y # ys'› and ‹zs = z # zs'› by simp
    then show ?thesis
    proof (cases j - 1 < k)
      case True
      with False and ** show ?thesis
        using ‹ys = y # ys'› by auto
    next
      case False
      from Suc.prems(1) and Cons and ‹j - 1 < length xs'› and ‹xs ! j = xs' ! (j - 1)› have j >
0
        using nth-non-equal-first-eq by fastforce
      with False have j ≥ Suc k
        by simp
      moreover have fmap-of-list ?ps $$ (xs ! j) = Some (drop (Suc k) zs ! (j - Suc k))
        using ** and False and ‹zs = z # zs'› by fastforce
      ultimately show ?thesis
        by simp
    qed
  qed
 qed
qed

lemma fmadd-drop-cancellation [simp]:
 assumes m $$ k = Some v
 shows {k ↦ v} ++_f fmdrop k m = m
using assms proof (induction m)
 case fmempty
 then show ?case
   by simp
next
 case (fmupd k' v' m')
 then show ?case
 proof (cases k' = k)
   case True
   with fmupd.prems have v = v'
     by fastforce
   have fmdrop k' (m'(k' ↦ v')) = m'
     unfolding fmdrop-fmupd-same using fmdrop-idle'[OF fmdom'-notI[OF fmupd.hyps]] by (unfold
True)
   then have {k ↦ v} ++_f fmdrop k' (m'(k' ↦ v')) = {k ↦ v} ++_f m'
     by simp
   then show ?thesis
     using fmap-singleton-comm[OF fmupd.hyps] by (simp add: True ‹v = v'›)
 next
```

case *False*
with *fmupd.prems* **have** m' $\$\$$ k = *Some* v
by *force*
from *False* **have** $\{k \rightarrowtail v\}$ $++_f$ *fmdrop* k $(m'(k' \rightarrowtail v')) = \{k \rightarrowtail v\}$ $++_f$ $(fmdrop\ k\ m')(k' \rightarrowtail v')$
by (*simp add*: *fmdrop-fmupd*)
also have ... = $(\{k \rightarrowtail v\}$ $++_f$ *fmdrop* k $m')(k' \rightarrowtail v')$
by *fastforce*
also from *fmupd.prems* **and** *fmupd.IH*[*OF* ‹m' $\$\$$ k = *Some* v›] **have** ... = $m'(k' \rightarrowtail v')$
by *force*
finally show *?thesis* **.**
qed
qed

lemma *fmap-of-list-fmmap* [*simp*]:
shows *fmap-of-list* (*map2* ($\lambda v'$ A'. (v', f A')) *xs ys*) = *fmmap* f (*fmap-of-list* (*zip xs ys*))
unfolding *fmmap-of-list*
using *cond-case-prod-eta*
[**where** f = $\lambda v'$ A'.(v', f A') **and** g = *apsnd* f, *unfolded apsnd-conv*, *simplified*]
by (*rule arg-cong*)

end

2 Syntax

theory *Syntax*
imports
HOL−Library.Sublist
Utilities
begin

2.1 Type symbols

datatype *type* =
TInd (*i*)
| *TBool* (*o*)
| *TFun type type* (**infixr** → *101*)

primrec *type-size* :: *type* ⇒ *nat* **where**
type-size i = *1*
| *type-size o* = *1*
| *type-size* ($\alpha \rightarrow \beta$) = *Suc* (*type-size* α + *type-size* β)

primrec *subtypes* :: *type* ⇒ *type set* **where**
subtypes i = {}
| *subtypes o* = {}
| *subtypes* ($\alpha \rightarrow \beta$) = {α, β} ∪ *subtypes* α ∪ *subtypes* β

lemma *subtype-size-decrease*:
assumes $\alpha \in$ *subtypes* β

shows *type-size* α < *type-size* β
using *assms* **by** (*induction rule*: *type.induct*) *force+*

lemma *subtype-is-not-type*:
assumes $\alpha \in$ *subtypes* β
shows $\alpha \neq \beta$
using *assms* **and** *subtype-size-decrease* **by** *blast*

lemma *fun-type-atoms-in-subtypes*:
assumes $k <$ *length ts*
shows *ts* ! $k \in$ *subtypes* (*foldr* ($\rightarrow$) *ts* γ)
using *assms* **by** (*induction ts arbitrary*: *k*) (*cases k*, *use less-Suc-eq-0-disj* **in** ‹*fastforce+*›)

lemma *fun-type-atoms-neq-fun-type*:
assumes $k <$ *length ts*
shows *ts* ! $k \neq$ *foldr* ($\rightarrow$) *ts* γ
by (*fact fun-type-atoms-in-subtypes*[*OF assms*, *THEN subtype-is-not-type*])

2.2 Variables

Unfortunately, the Nominal package does not support multi-sort atoms yet; therefore, we need to implement this support from scratch.

type-synonym *var* = *nat* × *type*

abbreviation *var-name* :: *var* ⇒ *nat* **where**
var-name ≡ *fst*

abbreviation *var-type* :: *var* ⇒ *type* **where**
var-type ≡ *snd*

lemma *fresh-var-existence*:
assumes *finite* (*vs* :: *var set*)
obtains *x* **where** (*x*, α) $\notin$ *vs*
using *ex-new-if-finite*[*OF infinite-UNIV-nat*]
proof −
from *assms* **obtain** *x* **where** *x* $\notin$ *var-name* ‘ *vs*
using *ex-new-if-finite*[*OF infinite-UNIV-nat*] **by** *fastforce*
with *that* **show** *?thesis*
by *force*
qed

lemma *fresh-var-name-list-existence*:
assumes *finite* (*ns* :: *nat set*)
obtains *ns′* **where** *length ns′* = *n* **and** *distinct ns′* **and** *lset ns′* ∩ *ns* = {}
using *assms* **proof** (*induction n arbitrary*: *thesis*)
case *0*
then show *?case*
by *simp*
next

```
  case (Suc n)
  from assms obtain ns' where length ns' = n and distinct ns' and lset ns' ∩ ns = {}
    using Suc.IH by blast
  moreover from assms obtain n' where n' ∉ lset ns' ∪ ns
    using ex-new-if-finite[OF infinite-UNIV-nat] by blast
  ultimately
    have length (n' # ns') = Suc n and distinct (n' # ns') and lset (n' # ns') ∩ ns = {}
    by simp-all
  with Suc.prems(1) show ?case
    by blast
qed

lemma fresh-var-list-existence:
  fixes xs :: var list
  and ns :: nat set
  assumes finite ns
  obtains vs' :: var list
  where length vs' = length xs
  and distinct vs'
  and var-name ‘ lset vs' ∩ (ns ∪ var-name ‘ lset xs) = {}
  and map var-type vs' = map var-type xs
proof −
  from assms(1) have finite (ns ∪ var-name ‘ lset xs)
    by blast
  then obtain ns'
    where length ns' = length xs
    and distinct ns'
    and lset ns' ∩ (ns ∪ var-name ‘ lset xs) = {}
    by (rule fresh-var-name-list-existence)
  define vs'' where vs'' = zip ns' (map var-type xs)
  from vs''-def and ‹length ns' = length xs› have length vs'' = length xs
    by simp
  moreover from vs''-def and ‹distinct ns'› have distinct vs''
    by (simp add: distinct-zipI1)
  moreover have var-name ‘ lset vs'' ∩ (ns ∪ var-name ‘ lset xs) = {}
    unfolding vs''-def
    using ‹length ns' = length xs› and ‹lset ns' ∩ (ns ∪ var-name ‘ lset xs) = {}›
    by (metis length-map set-map map-fst-zip)
  moreover from vs''-def have map var-type vs'' = map var-type xs
    by (simp add: ‹length ns' = length xs›)
  ultimately show ?thesis
    by (fact that)
qed
```

2.3 Constants

```
type-synonym con = nat × type
```

2.4 Formulas

datatype *form* =
  FVar var
| *FCon con*
| *FApp form form* (**infixl** ▪ *200*)
| *FAbs var form*

syntax
  -FVar :: *nat* ⇒ *type* ⇒ *form* (-_ [*899*, *0*] *900*)
  -FCon :: *nat* ⇒ *type* ⇒ *form* (⦇-⦈_ [*899*, *0*] *900*)
  -FAbs :: *nat* ⇒ *type* ⇒ *form* ⇒ *form* ((*4*λ-_./ -) [*0*, *0*, *104*] *104*)
translations
  $x_\alpha \rightleftharpoons$ *CONST FVar* (*x*, *α*)
  ⦇*c*⦈$_\alpha \rightleftharpoons$ *CONST FCon* (*c*, *α*)
  λx_α. *A* ⇌ *CONST FAbs* (*x*, *α*) *A*

2.5 Generalized operators

Generalized application. We define ▪$^{\mathcal{Q}}_\star$ *A* [B_1, B_2, …, B_n] as *A* ▪ B_1 ▪ B_2 ▪ ⋯ ▪ B_n:

definition *generalized-app* :: *form* ⇒ *form list* ⇒ *form* (▪$^{\mathcal{Q}}_\star$ - - [*241*, *241*] *241*) **where**
  [*simp*]: ▪$^{\mathcal{Q}}_\star$ *A Bs* = *foldl* (▪) *A Bs*

Generalized abstraction. We define λ$^{\mathcal{Q}}_\star$ [x_1, …, x_n] *A* as λx_1. ⋯ λx_n. *A*:

definition *generalized-abs* :: *var list* ⇒ *form* ⇒ *form* (λ$^{\mathcal{Q}}_\star$ - - [*141*, *141*] *141*) **where**
  [*simp*]: λ$^{\mathcal{Q}}_\star$ *vs A* = *foldr* (λ(*x*, *α*) *B*. λx_α. *B*) *vs A*

fun *form-size* :: *form* ⇒ *nat* **where**
  form-size (x_α) = *1*
| *form-size* (⦇*c*⦈$_\alpha$) = *1*
| *form-size* (*A* ▪ *B*) = *Suc* (*form-size A* + *form-size B*)
| *form-size* (λx_α. *A*) = *Suc* (*form-size A*)

fun *form-depth* :: *form* ⇒ *nat* **where**
  form-depth (x_α) = *0*
| *form-depth* (⦇*c*⦈$_\alpha$) = *0*
| *form-depth* (*A* ▪ *B*) = *Suc* (*max* (*form-depth A*) (*form-depth B*))
| *form-depth* (λx_α. *A*) = *Suc* (*form-depth A*)

2.6 Subformulas

fun *subforms* :: *form* ⇒ *form set* **where**
  subforms (x_α) = {}
| *subforms* (⦇*c*⦈$_\alpha$) = {}
| *subforms* (*A* ▪ *B*) = {*A*, *B*}
| *subforms* (λx_α. *A*) = {*A*}

datatype *direction* = *Left* («) | *Right* (»)
type-synonym *position* = *direction list*

fun *positions* :: *form* ⇒ *position set* **where**
  positions (x_α) = {[]}
| *positions* ({|*c*|}$_\alpha$) = {[]}
| *positions* (*A* ∙ *B*) = {[]} ∪ {« # *p* | *p*. *p* ∈ *positions A*} ∪ {» # *p* | *p*. *p* ∈ *positions B*}
| *positions* (λx_α. *A*) = {[]} ∪ {« # *p* | *p*. *p* ∈ *positions A*}

lemma *empty-is-position* [*simp*]:
  shows [] ∈ *positions A*
  by (*cases A rule*: *positions.cases*) *simp-all*

fun *subform-at* :: *form* ⇒ *position* ⇀ *form* **where**
  subform-at A [] = *Some A*
| *subform-at* (*A* ∙ *B*) (« # *p*) = *subform-at A p*
| *subform-at* (*A* ∙ *B*) (» # *p*) = *subform-at B p*
| *subform-at* (λx_α. *A*) (« # *p*) = *subform-at A p*
| *subform-at* - - = *None*

fun *is-subform-at* :: *form* ⇒ *position* ⇒ *form* ⇒ *bool* ((- ⪯_/ -) [*51*,*0*,*51*] *50*) **where**
  is-subform-at A [] *A'* = (*A* = *A'*)
| *is-subform-at C* (« # *p*) (*A* ∙ *B*) = *is-subform-at C p A*
| *is-subform-at C* (» # *p*) (*A* ∙ *B*) = *is-subform-at C p B*
| *is-subform-at C* (« # *p*) (λx_α. *A*) = *is-subform-at C p A*
| *is-subform-at* - - - = *False*

lemma *is-subform-at-alt-def*:
  shows *A'* ⪯$_p$ *A* = (*case subform-at A p of Some B* ⇒ *B* = *A'* | *None* ⇒ *False*)
  by (*induction A' p A rule*: *is-subform-at.induct*) *auto*

lemma *superform-existence*:
  assumes *B* ⪯$_p$ @ [*d*] *C*
  obtains *A* **where** *B* ⪯$_{[d]}$ *A* **and** *A* ⪯$_p$ *C*
  using *assms* **by** (*induction B p C rule*: *is-subform-at.induct*) *auto*

lemma *subform-at-subforms-con*:
  assumes {|*c*|}$_\alpha$ ⪯$_p$ *C*
  shows ∄*A*. *A* ⪯$_p$ @ [*d*] *C*
  using *assms* **by** (*induction* {|*c*|}$_\alpha$ *p C rule*: *is-subform-at.induct*) *auto*

lemma *subform-at-subforms-var*:
  assumes x_α ⪯$_p$ *C*
  shows ∄*A*. *A* ⪯$_p$ @ [*d*] *C*
  using *assms* **by** (*induction* x_α *p C rule*: *is-subform-at.induct*) *auto*

lemma *subform-at-subforms-app*:
  assumes *A* ∙ *B* ⪯$_p$ *C*
  shows *A* ⪯$_p$ @ [«] *C* **and** *B* ⪯$_p$ @ [»] *C*
  using *assms* **by** (*induction A* ∙ *B p C rule*: *is-subform-at.induct*) *auto*

lemma *subform-at-subforms-abs*:
assumes $\lambda x_\alpha.\ A \preceq_p C$
shows $A \preceq_p @_{[\langle]} C$
using *assms* **by** (*induction* $\lambda x_\alpha.\ A\ p\ C$ *rule*: *is-subform-at.induct*) *auto*

lemma *is-subform-implies-in-positions*:
assumes $B \preceq_p A$
shows $p \in$ *positions* A
using *assms* **by** (*induction rule*: *is-subform-at.induct*) *simp-all*

lemma *subform-size-decrease*:
assumes $A \preceq_p B$ **and** $p \neq []$
shows *form-size* $A <$ *form-size* B
using *assms* **by** (*induction* $A\ p\ B$ *rule*: *is-subform-at.induct*) *force+*

lemma *strict-subform-is-not-form*:
assumes $p \neq []$ **and** $A' \preceq_p A$
shows $A' \neq A$
using *assms* **and** *subform-size-decrease* **by** *blast*

lemma *no-right-subform-of-abs*:
shows $\nexists B.\ B \preceq_{\rangle} \#\ p\ \lambda x_\alpha.\ A$
by *simp*

lemma *subforms-from-var*:
assumes $A \preceq_p x_\alpha$
shows $A = x_\alpha$ **and** $p = []$
using *assms* **by** (*auto elim*: *is-subform-at.elims*)

lemma *subforms-from-con*:
assumes $A \preceq_p \{\!|c|\!\}_\alpha$
shows $A = \{\!|c|\!\}_\alpha$ **and** $p = []$
using *assms* **by** (*auto elim*: *is-subform-at.elims*)

lemma *subforms-from-app*:
assumes $A \preceq_p B \cdot C$
shows
$(A = B \cdot C \wedge p = []) \vee$
$(A \neq B \cdot C \wedge$
$(\exists p' \in$ *positions* $B.\ p = \langle \#\ p' \wedge A \preceq_{p'} B) \vee (\exists p' \in$ *positions* $C.\ p = \rangle \#\ p' \wedge A \preceq_{p'} C))$
using *assms* **and** *strict-subform-is-not-form*
by (*auto simp add*: *is-subform-implies-in-positions elim*: *is-subform-at.elims*)

lemma *subforms-from-abs*:
assumes $A \preceq_p \lambda x_\alpha.\ B$
shows $(A = \lambda x_\alpha.\ B \wedge p = []) \vee (A \neq \lambda x_\alpha.\ B \wedge (\exists p' \in$ *positions* $B.\ p = \langle \#\ p' \wedge A \preceq_{p'} B))$
using *assms* **and** *strict-subform-is-not-form*
by (*auto simp add*: *is-subform-implies-in-positions elim*: *is-subform-at.elims*)

lemma *leftmost-subform-in-generalized-app*:
shows $B \preceq_{replicate\ (length\ As)\ \langle\ \cdot^{\mathcal{Q}}{}_{\star}\ B\ As}$
by (*induction As arbitrary*: *B*) (*simp-all*, *metis replicate-append-same subform-at-subforms-app*(*1*))

lemma *self-subform-is-at-top*:
assumes $A \preceq_p A$
shows $p = []$
using *assms* **and** *strict-subform-is-not-form* **by** *blast*

lemma *at-top-is-self-subform*:
assumes $A \preceq_{[]} B$
shows $A = B$
using *assms* **by** (*auto elim*: *is-subform-at.elims*)

lemma *is-subform-at-uniqueness*:
assumes $B \preceq_p A$ **and** $C \preceq_p A$
shows $B = C$
using *assms* **by** (*induction A arbitrary*: *p B C*) (*auto elim*: *is-subform-at.elims*)

lemma *is-subform-at-existence*:
assumes $p \in positions\ A$
obtains *B* **where** $B \preceq_p A$
using *assms* **by** (*induction A arbitrary*: *p*) (*auto elim*: *is-subform-at.elims*, *blast*+)

lemma *is-subform-at-transitivity*:
assumes $A \preceq_{p_1} B$ **and** $B \preceq_{p_2} C$
shows $A \preceq_{p_2\ @\ p_1} C$
using *assms* **by** (*induction B* p_2 *C arbitrary*: *A* p_1 *rule*: *is-subform-at.induct*) *simp-all*

lemma *subform-nesting*:
assumes *strict-prefix* p' *p*
and $B \preceq_{p'} A$
and $C \preceq_p A$
shows $C \preceq_{drop\ (length\ p')\ p} B$
proof −
from *assms*(*1*) **have** $p \neq []$
using *strict-prefix-simps*(*1*) **by** *blast*
with *assms*(*1*,*3*) **show** *?thesis*
proof (*induction p arbitrary*: *C rule*: *rev-induct*)
case *Nil*
then show *?case*
by *blast*
next
case (*snoc d p''*)
then show *?case*
proof (*cases* $p'' = p'$)
case *True*
obtain A' **where** $C \preceq_{[d]} A'$ **and** $A' \preceq_{p'} A$

```
      by (fact superform-existence[OF snoc.prems(2)[unfolded True]])
    from ‹A' ⪯_p' A› and assms(2) have A' = B
      by (rule is-subform-at-uniqueness)
    with ‹C ⪯_[d] A'› have C ⪯_[d] B
      by (simp only:)
    with True show ?thesis
      by auto
  next
    case False
    with snoc.prems(1) have strict-prefix p' p''
      using prefix-order.dual-order.strict-implies-order by fastforce
    then have p'' ≠ []
      by force
    moreover from snoc.prems(2) obtain A' where C ⪯_[d] A' and A' ⪯_p'' A
      using superform-existence by blast
    ultimately have A' ⪯_drop (length p') p'' B
      using snoc.IH and ‹strict-prefix p' p''› by blast
    with ‹C ⪯_[d] A'› and snoc.prems(1) show ?thesis
      using is-subform-at-transitivity and prefix-length-less by fastforce
  qed
 qed
qed

lemma loop-subform-impossibility:
 assumes B ⪯_p A
 and strict-prefix p' p
 shows ¬ B ⪯_p' A
 using assms and prefix-length-less and self-subform-is-at-top and subform-nesting by fastforce

lemma nested-subform-size-decreases:
 assumes strict-prefix p' p
 and B ⪯_p' A
 and C ⪯_p A
 shows form-size C < form-size B
proof −
 from assms(1) have p ≠ []
   by force
 have C ⪯_drop (length p') p B
   by (fact subform-nesting[OF assms])
 moreover have drop (length p') p ≠ []
   using prefix-length-less[OF assms(1)] by force
 ultimately show ?thesis
   using subform-size-decrease by simp
qed

definition is-subform :: form ⇒ form ⇒ bool (infix ⪯ 50) where
 [simp]: A ⪯ B = (∃ p. A ⪯_p B)
```

instantiation *form* :: *ord*
begin

definition
$A \leq B \longleftrightarrow A \preceq B$

definition
$A < B \longleftrightarrow A \preceq B \wedge A \neq B$

instance ..

end

instance *form* :: *preorder*
proof (*standard*, *unfold less-eq-form-def less-form-def*)
fix A
show $A \preceq A$
unfolding *is-subform-def* **using** *is-subform-at.simps*(*1*) **by** *blast*
next
fix A **and** B **and** C
assume $A \preceq B$ **and** $B \preceq C$
then show $A \preceq C$
unfolding *is-subform-def* **using** *is-subform-at-transitivity* **by** *blast*
next
fix A **and** B
show $A \preceq B \wedge A \neq B \longleftrightarrow A \preceq B \wedge \neg\, B \preceq A$
unfolding *is-subform-def*
by (*metis is-subform-at.simps*(*1*) *not-less-iff-gr-or-eq subform-size-decrease*)
qed

lemma *position-subform-existence-equivalence*:
shows $p \in$ *positions* $A \longleftrightarrow (\exists\, A'.\ A' \preceq_p A)$
by (*meson is-subform-at-existence is-subform-implies-in-positions*)

lemma *position-prefix-is-position*:
assumes $p \in$ *positions* A **and** *prefix* p' p
shows $p' \in$ *positions* A
using *assms* **proof** (*induction p rule*: *rev-induct*)
case *Nil*
then show *?case*
by *simp*
next
case (*snoc d p''*)
from *snoc.prems*(*1*) **have** $p'' \in$ *positions* A
by (*meson position-subform-existence-equivalence superform-existence*)
with *snoc.prems*(*1*,*2*) **show** *?case*
using *snoc.IH* **by** *fastforce*
qed

2.7 Free and bound variables

consts *vars* :: $'a \Rightarrow var\ set$

overloading
  vars-form $\equiv$ *vars* :: *form* $\Rightarrow$ *var set*
  vars-form-set $\equiv$ *vars* :: *form set* $\Rightarrow$ *var set*
begin

fun *vars-form* :: *form* $\Rightarrow$ *var set* **where**
  vars-form $(x_\alpha) = \{(x, \alpha)\}$
| *vars-form* $(\{\!|c|\!\}_\alpha) = \{\}$
| *vars-form* $(A \cdot B) =$ *vars-form* $A \cup$ *vars-form* B
| *vars-form* $(\lambda x_\alpha.\ A) =$ *vars-form* $A \cup \{(x, \alpha)\}$

fun *vars-form-set* :: *form set* $\Rightarrow$ *var set* **where**
  vars-form-set $S = (\bigcup A \in S.\ vars\ A)$

end

abbreviation *var-names* :: $'a \Rightarrow nat\ set$ **where**
  var-names $\mathcal{X} \equiv$ *var-name* ' (*vars* $\mathcal{X}$)

lemma *vars-form-finiteness*:
  fixes A :: *form*
  shows *finite* (*vars* A)
  by (*induction rule*: *vars-form.induct*) *simp-all*

lemma *vars-form-set-finiteness*:
  fixes S :: *form set*
  assumes *finite* S
  shows *finite* (*vars* S)
  using *assms* **unfolding** *vars-form-set.simps* **using** *vars-form-finiteness* **by** *blast*

lemma *form-var-names-finiteness*:
  fixes A :: *form*
  shows *finite* (*var-names* A)
  using *vars-form-finiteness* **by** *blast*

lemma *form-set-var-names-finiteness*:
  fixes S :: *form set*
  assumes *finite* S
  shows *finite* (*var-names* S)
  using *assms* **and** *vars-form-set-finiteness* **by** *blast*

consts *free-vars* :: $'a \Rightarrow var\ set$

overloading
  free-vars-form $\equiv$ *free-vars* :: *form* $\Rightarrow$ *var set*
  free-vars-form-set $\equiv$ *free-vars* :: *form set* $\Rightarrow$ *var set*

begin

fun *free-vars-form* :: *form* ⇒ *var set* **where**
  free-vars-form (x_α) = {(x, α)}
| *free-vars-form* ($\{\!|c|\!\}_\alpha$) = {}
| *free-vars-form* ($A \cdot B$) = *free-vars-form* A ∪ *free-vars-form* B
| *free-vars-form* ($\lambda x_\alpha.\ A$) = *free-vars-form* A − {(x, α)}

fun *free-vars-form-set* :: *form set* ⇒ *var set* **where**
  free-vars-form-set S = ($\bigcup A \in S.$ *free-vars* A)

end

abbreviation *free-var-names* :: $'a$ ⇒ *nat set* **where**
  free-var-names $\mathcal{X}$ ≡ *var-name* ‘ (*free-vars* $\mathcal{X}$)

lemma *free-vars-form-finiteness*:
  fixes A :: *form*
  shows *finite* (*free-vars* A)
  by (*induction rule*: *free-vars-form.induct*) *simp-all*

lemma *free-vars-of-generalized-app*:
  shows *free-vars* ($\cdot^{\mathcal{Q}}{}_\star$ A *Bs*) = *free-vars* A ∪ *free-vars* (*lset Bs*)
  by (*induction Bs arbitrary*: A) *auto*

lemma *free-vars-of-generalized-abs*:
  shows *free-vars* ($\lambda^{\mathcal{Q}}{}_\star$ *vs* A) = *free-vars* A − *lset vs*
  by (*induction vs arbitrary*: A) *auto*

lemma *free-vars-in-all-vars*:
  fixes A :: *form*
  shows *free-vars* A ⊆ *vars* A
proof (*induction* A)
  case (*FVar* v)
  then show *?case*
    using *surj-pair*[*of* v] **by** *force*
next
  case (*FCon* k)
  then show *?case*
    using *surj-pair*[*of* k] **by** *force*
next
  case (*FApp* A B)
  have *free-vars* ($A \cdot B$) = *free-vars* A ∪ *free-vars* B
    using *free-vars-form.simps*(*3*) **.**
  also from *FApp.IH* **have** ... ⊆ *vars* A ∪ *vars* B
    by *blast*
  also have ... = *vars* ($A \cdot B$)
    using *vars-form.simps*(*3*)[*symmetric*] **.**
  finally show *?case*

by (*simp only*:)
next
case (*FAbs v A*)
then show *?case*
using *surj-pair*[*of v*] **by** *force*
qed

lemma *free-vars-in-all-vars-set*:
fixes S :: *form set*
shows *free-vars* $S \subseteq$ *vars* S
using *free-vars-in-all-vars* **by** *fastforce*

lemma *singleton-form-set-vars*:
shows *vars* $\{FVar\ y\} = \{y\}$
using *surj-pair*[*of y*] **by** *force*

fun *bound-vars* **where**
bound-vars $(x_\alpha) = \{\}$
| *bound-vars* $(\{\!|c|\!\}_\alpha) = \{\}$
| *bound-vars* $(B \centerdot C)$ = *bound-vars* $B \cup$ *bound-vars* C
| *bound-vars* $(\lambda x_\alpha.\ B) = \{(x, \alpha)\} \cup$ *bound-vars* B

lemma *vars-is-free-and-bound-vars*:
shows *vars* A = *free-vars* $A \cup$ *bound-vars* A
by (*induction A*) *auto*

fun *binders-at* :: *form* $\Rightarrow$ *position* $\Rightarrow$ *var set* **where**
binders-at $(A \centerdot B)$ $(\langle\!\langle\ \#\ p) =$ *binders-at* $A\ p$
| *binders-at* $(A \centerdot B)$ $(\rangle\!\rangle\ \#\ p) =$ *binders-at* $B\ p$
| *binders-at* $(\lambda x_\alpha.\ A)$ $(\langle\!\langle\ \#\ p) = \{(x, \alpha)\} \cup$ *binders-at* $A\ p$
| *binders-at* $A\ [] = \{\}$
| *binders-at* $A\ p = \{\}$

lemma *binders-at-concat*:
assumes $A' \preceq_p A$
shows *binders-at* $A\ (p\ @\ p') =$ *binders-at* $A\ p \cup$ *binders-at* $A'\ p'$
using *assms* **by** (*induction p A rule*: *is-subform-at.induct*) *auto*

2.8 Free and bound occurrences

definition *occurs-at* :: *var* $\Rightarrow$ *position* $\Rightarrow$ *form* $\Rightarrow$ *bool* **where**
[*iff*]: *occurs-at* $v\ p\ B \longleftrightarrow (FVar\ v \preceq_p B)$

lemma *occurs-at-alt-def*:
shows *occurs-at* $v\ []\ (FVar\ v') \longleftrightarrow (v = v')$
and *occurs-at* $v\ p\ (\{\!|c|\!\}_\alpha) \longleftrightarrow$ *False*
and *occurs-at* $v\ (\langle\!\langle\ \#\ p)\ (A \centerdot B) \longleftrightarrow$ *occurs-at* $v\ p\ A$
and *occurs-at* $v\ (\rangle\!\rangle\ \#\ p)\ (A \centerdot B) \longleftrightarrow$ *occurs-at* $v\ p\ B$
and *occurs-at* $v\ (\langle\!\langle\ \#\ p)\ (\lambda x_\alpha.\ A) \longleftrightarrow$ *occurs-at* $v\ p\ A$

and *occurs-at v* (*d* # *p*) (*FVar v'*) ⟷ *False*
and *occurs-at v* (» # *p*) (λx_α. *A*) ⟷ *False*
and *occurs-at v* [] (*A* ▪ *B*) ⟷ *False*
and *occurs-at v* [] (λx_α. *A*) ⟷ *False*
by (*fastforce elim*: *is-subform-at.elims*)+

definition *occurs* :: *var* ⇒ *form* ⇒ *bool* **where**
[*iff*]: *occurs v B* ⟷ (∃ *p* ∈ *positions B*. *occurs-at v p B*)

lemma *occurs-in-vars*:
assumes *occurs v A*
shows *v* ∈ *vars A*
using *assms* **by** (*induction A*) *force*+

abbreviation *strict-prefixes* **where**
strict-prefixes xs ≡ [*ys* ← *prefixes xs*. *ys* ≠ *xs*]

definition *in-scope-of-abs* :: *var* ⇒ *position* ⇒ *form* ⇒ *bool* **where**
[*iff*]: *in-scope-of-abs v p B* ⟷ (
p ≠ [] ∧
(
∃ *p'* ∈ *lset* (*strict-prefixes p*).
case (*subform-at B p'*) *of*
Some (*FAbs v'* -) ⇒ *v* = *v'*
| - ⇒ *False*
)
)

lemma *in-scope-of-abs-alt-def*:
shows
in-scope-of-abs v p B
⟷
p ≠ [] ∧ (∃ *p'* ∈ *positions B*. ∃ *C*. *strict-prefix p' p* ∧ *FAbs v C* $\preceq_{p'}$ *B*)
proof
assume *in-scope-of-abs v p B*
then show *p* ≠ [] ∧ (∃ *p'* ∈ *positions B*. ∃ *C*. *strict-prefix p' p* ∧ *FAbs v C* $\preceq_{p'}$ *B*)
by (*induction rule*: *subform-at.induct*) *force*+
next
assume *p* ≠ [] ∧ (∃ *p'* ∈ *positions B*. ∃ *C*. *strict-prefix p' p* ∧ *FAbs v C* $\preceq_{p'}$ *B*)
then show *in-scope-of-abs v p B*
by (*induction rule*: *subform-at.induct*) *fastforce*+
qed

lemma *in-scope-of-abs-in-left-app*:
shows *in-scope-of-abs v* (« # *p*) (*A* ▪ *B*) ⟷ *in-scope-of-abs v p A*
by *force*

lemma *in-scope-of-abs-in-right-app*:
shows *in-scope-of-abs v* (» # *p*) (*A* ▪ *B*) ⟷ *in-scope-of-abs v p B*

by *force*

lemma *in-scope-of-abs-in-app*:
assumes *in-scope-of-abs v p* (*A* ▪ *B*)
obtains *p'* **where** (*p* = « # *p'* ∧ *in-scope-of-abs v p' A*) ∨ (*p* = » # *p'* ∧ *in-scope-of-abs v p' B*)
proof −
from *assms* **obtain** *d* **and** *p'* **where** *p* = *d* # *p'*
unfolding *in-scope-of-abs-def* **by** (*meson list.exhaust*)
then show *?thesis*
proof (*cases d*)
case *Left*
with *assms* **and** ‹*p* = *d* # *p'*› **show** *?thesis*
using *that* **and** *in-scope-of-abs-in-left-app* **by** *simp*
next
case *Right*
with *assms* **and** ‹*p* = *d* # *p'*› **show** *?thesis*
using *that* **and** *in-scope-of-abs-in-right-app* **by** *simp*
qed
qed

lemma *not-in-scope-of-abs-in-app*:
assumes
∀ *p'*.
(*p* = « # *p'* ⟶ ¬ *in-scope-of-abs v' p' A*)
∧
(*p* = » # *p'* ⟶ ¬ *in-scope-of-abs v' p' B*)
shows ¬ *in-scope-of-abs v' p* (*A* ▪ *B*)
using *assms* **and** *in-scope-of-abs-in-app* **by** *metis*

lemma *in-scope-of-abs-in-abs*:
shows *in-scope-of-abs v* (« # *p*) (*FAbs v' B*) ⟷ *v* = *v'* ∨ *in-scope-of-abs v p B*
proof
assume *in-scope-of-abs v* (« # *p*) (*FAbs v' B*)
then obtain *p'* **and** *C*
where *p'* ∈ *positions* (*FAbs v' B*)
and *strict-prefix p'* (« # *p*)
and *FAbs v C* $\preceq_{p'}$ *FAbs v' B*
unfolding *in-scope-of-abs-alt-def* **by** *blast*
then show *v* = *v'* ∨ *in-scope-of-abs v p B*
proof (*cases p'*)
case *Nil*
with ‹*FAbs v C* $\preceq_{p'}$ *FAbs v' B*› **have** *v* = *v'*
by *auto*
then show *?thesis*
by *simp*
next
case (*Cons d p''*)
with ‹*strict-prefix p'* (« # *p*)› **have** *d* = «
by *simp*

```
    from ‹FAbs v C ⪯_p' FAbs v' B› and Cons have p'' ∈ positions B
      by
        (cases (FAbs v C, p', FAbs v' B) rule: is-subform-at.cases)
        (simp-all add: is-subform-implies-in-positions)
    moreover from ‹FAbs v C ⪯_p' FAbs v' B› and Cons and ‹d = «› have FAbs v C ⪯_p'' B
      by (metis is-subform-at.simps(4) old.prod.exhaust)
    moreover from ‹strict-prefix p' (« # p)› and Cons have strict-prefix p'' p
      by auto
    ultimately have in-scope-of-abs v p B
      using in-scope-of-abs-alt-def by auto
    then show ?thesis
      by simp
  qed
next
  assume v = v' ∨ in-scope-of-abs v p B
  then show in-scope-of-abs v (« # p) (FAbs v' B)
    unfolding in-scope-of-abs-alt-def
    using position-subform-existence-equivalence and surj-pair[of v']
    by force
qed

lemma not-in-scope-of-abs-in-var:
  shows ¬ in-scope-of-abs v p (FVar v')
  unfolding in-scope-of-abs-def by (cases p) simp-all

lemma in-scope-of-abs-in-vars:
  assumes in-scope-of-abs v p A
  shows v ∈ vars A
using assms proof (induction A arbitrary: p)
  case (FVar v')
  then show ?case
    using not-in-scope-of-abs-in-var by blast
next
  case (FCon k)
  then show ?case
    using in-scope-of-abs-alt-def by (blast elim: is-subform-at.elims(2))
next
  case (FApp B C)
  from FApp.prems obtain d and p' where p = d # p'
    unfolding in-scope-of-abs-def by (meson neq-Nil-conv)
  then show ?case
  proof (cases d)
    case Left
    with FApp.prems and ‹p = d # p'› have in-scope-of-abs v p' B
      using in-scope-of-abs-in-left-app by blast
    then have v ∈ vars B
      by (fact FApp.IH(1))
    then show ?thesis
      by simp
```

```
  next
    case Right
    with FApp.prems and ‹p = d # p'› have in-scope-of-abs v p' C
      using in-scope-of-abs-in-right-app by blast
    then have v ∈ vars C
      by (fact FApp.IH(2))
    then show ?thesis
      by simp
  qed
next
  case (FAbs v' B)
  then show ?case
  proof (cases v = v')
    case True
    then show ?thesis
      using surj-pair[of v] by force
  next
    case False
    with FAbs.prems obtain p' and d where p = d # p'
      unfolding in-scope-of-abs-def by (meson neq-Nil-conv)
    then show ?thesis
    proof (cases d)
      case Left
      with FAbs.prems and False and ‹p = d # p'› have in-scope-of-abs v p' B
        using in-scope-of-abs-in-abs by blast
      then have v ∈ vars B
        by (fact FAbs.IH)
      then show ?thesis
        using surj-pair[of v'] by force
    next
      case Right
      with FAbs.prems and ‹p = d # p'› and False show ?thesis
        by (cases rule: is-subform-at.cases) auto
    qed
  qed
qed

lemma binders-at-alt-def:
  assumes p ∈ positions A
  shows binders-at A p = {v | v. in-scope-of-abs v p A}
  using assms and in-set-prefixes by (induction rule: binders-at.induct) auto

definition is-bound-at :: var ⇒ position ⇒ form ⇒ bool where
  [iff]: is-bound-at v p B ⟷ occurs-at v p B ∧ in-scope-of-abs v p B

lemma not-is-bound-at-in-var:
  shows ¬ is-bound-at v p (FVar v')
  by (fastforce elim: is-subform-at.elims(2))
```

lemma *not-is-bound-at-in-con*:
shows $\neg$ *is-bound-at v p* (*FCon k*)
by (*fastforce elim*: *is-subform-at.elims*(*2*))

lemma *is-bound-at-in-left-app*:
shows *is-bound-at v* (« # *p*) (*B* ▪ *C*) $\longleftrightarrow$ *is-bound-at v p B*
by *auto*

lemma *is-bound-at-in-right-app*:
shows *is-bound-at v* (» # *p*) (*B* ▪ *C*) $\longleftrightarrow$ *is-bound-at v p C*
by *auto*

lemma *is-bound-at-from-app*:
assumes *is-bound-at v p* (*B* ▪ *C*)
obtains *p'* **where** (*p* = « # *p'* $\wedge$ *is-bound-at v p' B*) $\vee$ (*p* = » # *p'* $\wedge$ *is-bound-at v p' C*)
proof −
from *assms* **obtain** *d* **and** *p'* **where** *p* = *d* # *p'*
using *subforms-from-app* **by** *blast*
then show *?thesis*
proof (*cases d*)
case *Left*
with *assms* **and** *that* **and** ‹*p* = *d* # *p'*› **show** *?thesis*
using *is-bound-at-in-left-app* **by** *simp*
next
case *Right*
with *assms* **and** *that* **and** ‹*p* = *d* # *p'*› **show** *?thesis*
using *is-bound-at-in-right-app* **by** *simp*
qed
qed

lemma *is-bound-at-from-abs*:
assumes *is-bound-at v* (« # *p*) (*FAbs v' B*)
shows *v* = *v'* $\vee$ *is-bound-at v p B*
using *assms* **by** (*fastforce elim*: *is-subform-at.elims*)

lemma *is-bound-at-from-absE*:
assumes *is-bound-at v p* (*FAbs v' B*)
obtains *p'* **where** *p* = « # *p'* **and** *v* = *v'* $\vee$ *is-bound-at v p' B*
proof −
obtain *x* **and** α **where** *v'* = (*x*, α)
by *fastforce*
with *assms* **obtain** *p'* **where** *p* = « # *p'*
using *subforms-from-abs* **by** *blast*
with *assms* **and** *that* **show** *?thesis*
using *is-bound-at-from-abs* **by** *simp*
qed

lemma *is-bound-at-to-abs*:
assumes (*v* = *v'* $\wedge$ *occurs-at v p B*) $\vee$ *is-bound-at v p B*

```
  shows is-bound-at v (« # p) (FAbs v' B)
unfolding is-bound-at-def proof
  from assms(1) show occurs-at v (« # p) (FAbs v' B)
    using surj-pair[of v'] by force
  from assms show in-scope-of-abs v (« # p) (FAbs v' B)
    using in-scope-of-abs-in-abs by auto
qed

lemma is-bound-at-in-bound-vars:
  assumes p ∈ positions A
  and is-bound-at v p A ∨ v ∈ binders-at A p
  shows v ∈ bound-vars A
using assms proof (induction A arbitrary: p)
  case (FApp B C)
  from FApp.prems(2) consider (a) is-bound-at v p (B • C) | (b) v ∈ binders-at (B • C) p
    by blast
  then show ?case
  proof cases
    case a
    then have p ≠ []
      using occurs-at-alt-def(8) by blast
    then obtain d and p' where p = d # p'
      by (meson list.exhaust)
    with ‹p ∈ positions (B • C)›
    consider (a1) p = « # p' and p' ∈ positions B | (a2) p = » # p' and p' ∈ positions C
      by force
    then show ?thesis
    proof cases
      case a1
      from a1(1) and ‹is-bound-at v p (B • C)› have is-bound-at v p' B
        using is-bound-at-in-left-app by blast
      with a1(2) have v ∈ bound-vars B
        using FApp.IH(1) by blast
      then show ?thesis
        by simp
    next
      case a2
      from a2(1) and ‹is-bound-at v p (B • C)› have is-bound-at v p' C
        using is-bound-at-in-right-app by blast
      with a2(2) have v ∈ bound-vars C
        using FApp.IH(2) by blast
      then show ?thesis
        by simp
    qed
  next
    case b
    then have p ≠ []
      by force
    then obtain d and p' where p = d # p'
```

```
      by (meson list.exhaust)
    with ‹p ∈ positions (B • C)›
    consider (b1) p = « # p' and p' ∈ positions B | (b2) p = » # p' and p' ∈ positions C
      by force
    then show ?thesis
    proof cases
      case b1
      from b1(1) and ‹v ∈ binders-at (B • C) p› have v ∈ binders-at B p'
        by force
      with b1(2) have v ∈ bound-vars B
        using FApp.IH(1) by blast
      then show ?thesis
        by simp
    next
      case b2
      from b2(1) and ‹v ∈ binders-at (B • C) p› have v ∈ binders-at C p'
        by force
      with b2(2) have v ∈ bound-vars C
        using FApp.IH(2) by blast
      then show ?thesis
        by simp
    qed
  qed
next
  case (FAbs v' B)
  from FAbs.prems(2) consider (a) is-bound-at v p (FAbs v' B) | (b) v ∈ binders-at (FAbs v' B) p
    by blast
  then show ?case
  proof cases
    case a
    then have p ≠ []
      using occurs-at-alt-def(9) by force
    with ‹p ∈ positions (FAbs v' B)› obtain p' where p = « # p' and p' ∈ positions B
      by (cases FAbs v' B rule: positions.cases) fastforce+
    from ‹p = « # p'› and ‹is-bound-at v p (FAbs v' B)› have v = v' ∨ is-bound-at v p' B
      using is-bound-at-from-abs by blast
    then consider (a1) v = v' | (a2) is-bound-at v p' B
      by blast
    then show ?thesis
    proof cases
      case a1
      then show ?thesis
        using surj-pair[of v'] by fastforce
    next
      case a2
      then have v ∈ bound-vars B
        using ‹p' ∈ positions B› and FAbs.IH by blast
      then show ?thesis
        using surj-pair[of v'] by fastforce
```

```
    qed
  next
    case b
    then have p ≠ []
      by force
    with FAbs.prems(1) obtain p' where p = « # p' and p' ∈ positions B
      by (cases FAbs v' B rule: positions.cases) fastforce+
    with b consider (b1) v = v' | (b2) v ∈ binders-at B p'
      by (cases FAbs v' B rule: positions.cases) fastforce+
    then show ?thesis
    proof cases
      case b1
      then show ?thesis
        using surj-pair[of v'] by fastforce
    next
      case b2
      then have v ∈ bound-vars B
        using ‹p' ∈ positions B› and FAbs.IH by blast
      then show ?thesis
        using surj-pair[of v'] by fastforce
    qed
  qed
qed fastforce+

lemma bound-vars-in-is-bound-at:
  assumes v ∈ bound-vars A
  obtains p where p ∈ positions A and is-bound-at v p A ∨ v ∈ binders-at A p
using assms proof (induction A arbitrary: thesis rule: bound-vars.induct)
  case (3 B C)
  from ‹v ∈ bound-vars (B • C)› consider (a) v ∈ bound-vars B | (b) v ∈ bound-vars C
    by fastforce
  then show ?case
  proof cases
    case a
    with 3.IH(1) obtain p where p ∈ positions B and is-bound-at v p B ∨ v ∈ binders-at B p
      by blast
    from ‹p ∈ positions B› have « # p ∈ positions (B • C)
      by simp
    from ‹is-bound-at v p B ∨ v ∈ binders-at B p›
    consider (a1) is-bound-at v p B | (a2) v ∈ binders-at B p
      by blast
    then show ?thesis
    proof cases
      case a1
      then have is-bound-at v (« # p) (B • C)
        using is-bound-at-in-left-app by blast
      then show ?thesis
        using 3.prems(1) and is-subform-implies-in-positions by blast
    next
```

```
    case a2
    then have v ∈ binders-at (B • C) («  # p)
      by simp
    then show ?thesis
      using 3.prems(1) and ‹« # p ∈ positions (B • C)› by blast
  qed
 next
  case b
  with 3.IH(2) obtain p where p ∈ positions C and is-bound-at v p C ∨ v ∈ binders-at C p
    by blast
  from ‹p ∈ positions C› have » # p ∈ positions (B • C)
    by simp
  from ‹is-bound-at v p C ∨ v ∈ binders-at C p›
  consider (b1) is-bound-at v p C | (b2) v ∈ binders-at C p
    by blast
  then show ?thesis
  proof cases
    case b1
    then have is-bound-at v (» # p) (B • C)
      using is-bound-at-in-right-app by blast
    then show ?thesis
      using 3.prems(1) and is-subform-implies-in-positions by blast
  next
    case b2
    then have v ∈ binders-at (B • C) (» # p)
      by simp
    then show ?thesis
      using 3.prems(1) and ‹» # p ∈ positions (B • C)› by blast
  qed
 qed
next
 case (4 x α B)
 from ‹v ∈ bound-vars (λx_α. B)› consider (a) v = (x, α) | (b) v ∈ bound-vars B
   by force
 then show ?case
 proof cases
   case a
   then have v ∈ binders-at (λx_α. B) [«]
     by simp
   then show ?thesis
     using 4.prems(1) and is-subform-implies-in-positions by fastforce
 next
   case b
   with 4.IH(1) obtain p where p ∈ positions B and is-bound-at v p B ∨ v ∈ binders-at B p
     by blast
   from ‹p ∈ positions B› have « # p ∈ positions (λx_α. B)
     by simp
   from ‹is-bound-at v p B ∨ v ∈ binders-at B p›
   consider (b1) is-bound-at v p B | (b2) v ∈ binders-at B p
```

```
    by blast
  then show ?thesis
  proof cases
    case b1
    then have is-bound-at v («  # p) (λx_α. B)
      using is-bound-at-to-abs by blast
    then show ?thesis
      using 4.prems(1) and ‹« # p ∈ positions (λx_α. B)› by blast
  next
    case b2
    then have v ∈ binders-at (λx_α. B) («  # p)
      by simp
    then show ?thesis
      using 4.prems(1) and ‹« # p ∈ positions (λx_α. B)› by blast
  qed
 qed
qed simp-all

lemma bound-vars-alt-def:
 shows bound-vars A = {v | v p. p ∈ positions A ∧ (is-bound-at v p A ∨ v ∈ binders-at A p)}
 using bound-vars-in-is-bound-at and is-bound-at-in-bound-vars
 by (intro subset-antisym subsetI CollectI, metis) blast

definition is-free-at :: var ⇒ position ⇒ form ⇒ bool where
 [iff]: is-free-at v p B ⟷ occurs-at v p B ∧ ¬ in-scope-of-abs v p B

lemma is-free-at-in-var:
 shows is-free-at v [] (FVar v') ⟷ v = v'
 by simp

lemma not-is-free-at-in-con:
 shows ¬ is-free-at v [] ({|c|}_α)
 by simp

lemma is-free-at-in-left-app:
 shows is-free-at v («  # p) (B • C) ⟷ is-free-at v p B
 by auto

lemma is-free-at-in-right-app:
 shows is-free-at v (»  # p) (B • C) ⟷ is-free-at v p C
 by auto

lemma is-free-at-from-app:
 assumes is-free-at v p (B • C)
 obtains p' where (p = «  # p' ∧ is-free-at v p' B) ∨ (p = »  # p' ∧ is-free-at v p' C)
proof −
 from assms obtain d and p' where p = d # p'
   using subforms-from-app by blast
 then show ?thesis
```

```
  proof (cases d)
    case Left
    with assms and that and ‹p = d # p'› show ?thesis
      using is-free-at-in-left-app by blast
  next
    case Right
    with assms and that and ‹p = d # p'› show ?thesis
      using is-free-at-in-right-app by blast
  qed
qed

lemma is-free-at-from-abs:
  assumes is-free-at v (« # p) (FAbs v' B)
  shows is-free-at v p B
  using assms by (fastforce elim: is-subform-at.elims)

lemma is-free-at-from-absE:
  assumes is-free-at v p (FAbs v' B)
  obtains p' where p = « # p' and is-free-at v p' B
proof −
  obtain x and α where v' = (x, α)
    by fastforce
  with assms obtain p' where p = « # p'
    using subforms-from-abs by blast
  with assms and that show ?thesis
    using is-free-at-from-abs by blast
qed

lemma is-free-at-to-abs:
  assumes is-free-at v p B and v ≠ v'
  shows is-free-at v (« # p) (FAbs v' B)
unfolding is-free-at-def proof
  from assms(1) show occurs-at v (« # p) (FAbs v' B)
    using surj-pair[of v'] by fastforce
  from assms show ¬ in-scope-of-abs v (« # p) (FAbs v' B)
    unfolding is-free-at-def using in-scope-of-abs-in-abs by presburger
qed

lemma is-free-at-in-free-vars:
  assumes p ∈ positions A and is-free-at v p A
  shows v ∈ free-vars A
using assms proof (induction A arbitrary: p)
  case (FApp B C)
  from ‹is-free-at v p (B • C)› have p ≠ []
    using occurs-at-alt-def(8) by blast
  then obtain d and p' where p = d # p'
    by (meson list.exhaust)
  with ‹p ∈ positions (B • C)›
  consider (a) p = « # p' and p' ∈ positions B | (b) p = » # p' and p' ∈ positions C
```

```
    by force
  then show ?case
  proof cases
    case a
    from a(1) and ‹is-free-at v p (B • C)› have is-free-at v p' B
      using is-free-at-in-left-app by blast
    with a(2) have v ∈ free-vars B
      using FApp.IH(1) by blast
    then show ?thesis
      by simp
  next
    case b
    from b(1) and ‹is-free-at v p (B • C)› have is-free-at v p' C
      using is-free-at-in-right-app by blast
    with b(2) have v ∈ free-vars C
      using FApp.IH(2) by blast
    then show ?thesis
      by simp
  qed
next
  case (FAbs v' B)
  from ‹is-free-at v p (FAbs v' B)› have p ≠ []
    using occurs-at-alt-def(9) by force
  with ‹p ∈ positions (FAbs v' B)› obtain p' where p = « # p' and p' ∈ positions B
    by (cases FAbs v' B rule: positions.cases) fastforce+
  moreover from ‹p = « # p'› and ‹is-free-at v p (FAbs v' B)› have is-free-at v p' B
    using is-free-at-from-abs by blast
  ultimately have v ∈ free-vars B
    using FAbs.IH by simp
  moreover from ‹p = « # p'› and ‹is-free-at v p (FAbs v' B)› have v ≠ v'
    using in-scope-of-abs-in-abs by blast
  ultimately show ?case
    using surj-pair[of v'] by force
qed fastforce+

lemma free-vars-in-is-free-at:
  assumes v ∈ free-vars A
  obtains p where p ∈ positions A and is-free-at v p A
using assms proof (induction A arbitrary: thesis rule: free-vars-form.induct)
  case (3 A B)
  from ‹v ∈ free-vars (A • B)› consider (a) v ∈ free-vars A | (b) v ∈ free-vars B
    by fastforce
  then show ?case
  proof cases
    case a
    with 3.IH(1) obtain p where p ∈ positions A and is-free-at v p A
      by blast
    from ‹p ∈ positions A› have « # p ∈ positions (A • B)
      by simp
```

```
    moreover from ‹is-free-at v p A› have is-free-at v (« # p) (A • B)
      using is-free-at-in-left-app by blast
    ultimately show ?thesis
      using 3.prems(1) by presburger
  next
    case b
    with 3.IH(2) obtain p where p ∈ positions B and is-free-at v p B
      by blast
    from ‹p ∈ positions B› have » # p ∈ positions (A • B)
      by simp
    moreover from ‹is-free-at v p B› have is-free-at v (» # p) (A • B)
      using is-free-at-in-right-app by blast
    ultimately show ?thesis
      using 3.prems(1) by presburger
  qed
next
  case (4 x α A)
  from ‹v ∈ free-vars (λx_α. A)› have v ∈ free-vars A − {(x, α)} and v ≠ (x, α)
    by simp-all
  then have v ∈ free-vars A
    by blast
  with 4.IH obtain p where p ∈ positions A and is-free-at v p A
    by blast
  from ‹p ∈ positions A› have « # p ∈ positions (λx_α. A)
    by simp
  moreover from ‹is-free-at v p A› and ‹v ≠ (x, α)› have is-free-at v (« # p) (λx_α. A)
    using is-free-at-to-abs by blast
  ultimately show ?case
    using 4.prems(1) by presburger
qed simp-all

lemma free-vars-alt-def:
  shows free-vars A = {v | v p. p ∈ positions A ∧ is-free-at v p A}
  using free-vars-in-is-free-at and is-free-at-in-free-vars
  by (intro subset-antisym subsetI CollectI, metis) blast
```

In the following definition, note that the variable immeditately preceded by λ counts as a bound variable:

```
definition is-bound :: var ⇒ form ⇒ bool where
  [iff]: is-bound v B ⟷ (∃ p ∈ positions B. is-bound-at v p B ∨ v ∈ binders-at B p)

lemma is-bound-in-app-homomorphism:
  shows is-bound v (A • B) ⟷ is-bound v A ∨ is-bound v B
proof
  assume is-bound v (A • B)
  then obtain p where p ∈ positions (A • B) and is-bound-at v p (A • B) ∨ v ∈ binders-at (A • B) p
    by auto
  then have p ≠ []
    by fastforce
```

with ‹$p \in$ *positions* $(A \cdot B)$› **obtain** p' **and** d **where** $p = d \# p'$
by *auto*
from ‹*is-bound-at* $v\ p\ (A \cdot B) \vee v \in$ *binders-at* $(A \cdot B)\ p$›
consider (a) *is-bound-at* $v\ p\ (A \cdot B) \mid (b)\ v \in$ *binders-at* $(A \cdot B)\ p$
by *blast*
then show *is-bound* $v\ A \vee$ *is-bound* $v\ B$
proof *cases*
case a
then show *?thesis*
proof (*cases* d)
case *Left*
then have $p' \in$ *positions* A
using ‹$p = d \# p'$› **and** ‹$p \in$ *positions* $(A \cdot B)$› **by** *fastforce*
moreover from ‹*is-bound-at* $v\ p\ (A \cdot B)$› **have** *occurs-at* $v\ p'\ A$
using *Left* **and** ‹$p = d \# p'$› **and** *is-subform-at.simps*(*2*) **by** *force*
moreover from ‹*is-bound-at* $v\ p\ (A \cdot B)$› **have** *in-scope-of-abs* $v\ p'\ A$
using *Left* **and** ‹$p = d \# p'$› **by** *fastforce*
ultimately show *?thesis*
by *auto*
next
case *Right*
then have $p' \in$ *positions* B
using ‹$p = d \# p'$› **and** ‹$p \in$ *positions* $(A \cdot B)$› **by** *fastforce*
moreover from ‹*is-bound-at* $v\ p\ (A \cdot B)$› **have** *occurs-at* $v\ p'\ B$
using *Right* **and** ‹$p = d \# p'$› **and** *is-subform-at.simps*(*3*) **by** *force*
moreover from ‹*is-bound-at* $v\ p\ (A \cdot B)$› **have** *in-scope-of-abs* $v\ p'\ B$
using *Right* **and** ‹$p = d \# p'$› **by** *fastforce*
ultimately show *?thesis*
by *auto*
qed
next
case b
then show *?thesis*
proof (*cases* d)
case *Left*
then have $p' \in$ *positions* A
using ‹$p = d \# p'$› **and** ‹$p \in$ *positions* $(A \cdot B)$› **by** *fastforce*
moreover from ‹$v \in$ *binders-at* $(A \cdot B)\ p$› **have** $v \in$ *binders-at* $A\ p'$
using *Left* **and** ‹$p = d \# p'$› **by** *force*
ultimately show *?thesis*
by *auto*
next
case *Right*
then have $p' \in$ *positions* B
using ‹$p = d \# p'$› **and** ‹$p \in$ *positions* $(A \cdot B)$› **by** *fastforce*
moreover from ‹$v \in$ *binders-at* $(A \cdot B)\ p$› **have** $v \in$ *binders-at* $B\ p'$
using *Right* **and** ‹$p = d \# p'$› **by** *force*
ultimately show *?thesis*
by *auto*

```
    qed
  qed
next
  assume is-bound v A ∨ is-bound v B
  then show is-bound v (A ▪ B)
  proof (rule disjE)
    assume is-bound v A
    then obtain p where p ∈ positions A and is-bound-at v p A ∨ v ∈ binders-at A p
      by auto
    from ‹p ∈ positions A› have « # p ∈ positions (A ▪ B)
      by auto
    from ‹is-bound-at v p A ∨ v ∈ binders-at A p›
    consider (a) is-bound-at v p A | (b) v ∈ binders-at A p
      by blast
    then show is-bound v (A ▪ B)
    proof cases
      case a
      then have occurs-at v (« # p) (A ▪ B)
        by auto
      moreover from a have is-bound-at v (« # p) (A ▪ B)
        by force
      ultimately show is-bound v (A ▪ B)
        using ‹« # p ∈ positions (A ▪ B)› by blast
    next
      case b
      then have v ∈ binders-at (A ▪ B) (« # p)
        by auto
      then show is-bound v (A ▪ B)
        using ‹« # p ∈ positions (A ▪ B)› by blast
    qed
  next
    assume is-bound v B
    then obtain p where p ∈ positions B and is-bound-at v p B ∨ v ∈ binders-at B p
      by auto
    from ‹p ∈ positions B› have » # p ∈ positions (A ▪ B)
      by auto
    from ‹is-bound-at v p B ∨ v ∈ binders-at B p›
    consider (a) is-bound-at v p B | (b) v ∈ binders-at B p
      by blast
    then show is-bound v (A ▪ B)
    proof cases
      case a
      then have occurs-at v (» # p) (A ▪ B)
        by auto
      moreover from a have is-bound-at v (» # p) (A ▪ B)
        by force
      ultimately show is-bound v (A ▪ B)
        using ‹» # p ∈ positions (A ▪ B)› by blast
    next
```

```
    case b
    then have v ∈ binders-at (A • B) (» # p)
      by auto
    then show is-bound v (A • B)
      using ‹» # p ∈ positions (A • B)› by blast
  qed
 qed
qed

lemma is-bound-in-abs-body:
 assumes is-bound v A
 shows is-bound v (λx_α. A)
using assms unfolding is-bound-def proof
 fix p
 assume p ∈ positions A and is-bound-at v p A ∨ v ∈ binders-at A p
 moreover from ‹p ∈ positions A› have « # p ∈ positions (λx_α. A)
   by simp
 ultimately consider (a) is-bound-at v p A | (b) v ∈ binders-at A p
   by blast
 then show ∃ p ∈ positions (λx_α. A). is-bound-at v p (λx_α. A) ∨ v ∈ binders-at (λx_α. A) p
 proof cases
   case a
   then have is-bound-at v (« # p) (λx_α. A)
     by force
   with ‹« # p ∈ positions (λx_α. A)› show ?thesis
     by blast
 next
   case b
   then have v ∈ binders-at (λx_α. A) (« # p)
     by simp
   with ‹« # p ∈ positions (λx_α. A)› show ?thesis
     by blast
 qed
qed

lemma absent-var-is-not-bound:
 assumes v ∉ vars A
 shows ¬ is-bound v A
 using assms and binders-at-alt-def and in-scope-of-abs-in-vars by blast

lemma bound-vars-alt-def2:
 shows bound-vars A = {v ∈ vars A. is-bound v A}
 unfolding bound-vars-alt-def using absent-var-is-not-bound by fastforce

definition is-free :: var ⇒ form ⇒ bool where
 [iff]: is-free v B ⟷ (∃ p ∈ positions B. is-free-at v p B)
```

2.9 Free variables for a formula in another formula

definition *is-free-for* :: *form* ⇒ *var* ⇒ *form* ⇒ *bool* **where**
[*iff*]: *is-free-for A v B* ⟷
(
∀ *v′* ∈ *free-vars A*.
∀ *p* ∈ *positions B*.
is-free-at v p B ⟶ ¬ *in-scope-of-abs v′ p B*
)

lemma *is-free-for-absent-var* [*intro*]:
assumes *v* ∉ *vars B*
shows *is-free-for A v B*
using *assms* **and** *occurs-def* **and** *is-free-at-def* **and** *occurs-in-vars* **by** *blast*

lemma *is-free-for-in-var* [*intro*]:
shows *is-free-for A v* (*x*$_\alpha$)
using *subforms-from-var*(*2*) **by** *force*

lemma *is-free-for-in-con* [*intro*]:
shows *is-free-for A v* (⦃*c*⦄$_\alpha$)
using *subforms-from-con*(*2*) **by** *force*

lemma *is-free-for-from-app*:
assumes *is-free-for A v* (*B* ▪ *C*)
shows *is-free-for A v B* **and** *is-free-for A v C*
proof −
{
fix *v′*
assume *v′* ∈ *free-vars A*
then have ∀ *p* ∈ *positions B*. *is-free-at v p B* ⟶ ¬ *in-scope-of-abs v′ p B*
proof (*intro ballI impI*)
fix *p*
assume *v′* ∈ *free-vars A* **and** *p* ∈ *positions B* **and** *is-free-at v p B*
from ‹*p* ∈ *positions B*› **have** « # *p* ∈ *positions* (*B* ▪ *C*)
by *simp*
moreover from ‹*is-free-at v p B*› **have** *is-free-at v* (« # *p*) (*B* ▪ *C*)
using *is-free-at-in-left-app* **by** *blast*
ultimately have ¬ *in-scope-of-abs v′* (« # *p*) (*B* ▪ *C*)
using *assms* **and** ‹*v′* ∈ *free-vars A*› **by** *blast*
then show ¬ *in-scope-of-abs v′ p B*
by *simp*
qed
}
then show *is-free-for A v B*
by *force*
next
{
fix *v′*
assume *v′* ∈ *free-vars A*

```
    then have ∀ p ∈ positions C. is-free-at v p C ⟶ ¬ in-scope-of-abs v' p C
    proof (intro ballI impI)
      fix p
      assume v' ∈ free-vars A and p ∈ positions C and is-free-at v p C
      from ‹p ∈ positions C› have » # p ∈ positions (B ▪ C)
        by simp
      moreover from ‹is-free-at v p C› have is-free-at v (» # p) (B ▪ C)
        using is-free-at-in-right-app by blast
      ultimately have ¬ in-scope-of-abs v' (» # p) (B ▪ C)
        using assms and ‹v' ∈ free-vars A› by blast
      then show ¬ in-scope-of-abs v' p C
        by simp
    qed
  }
  then show is-free-for A v C
    by force
qed

lemma is-free-for-to-app [intro]:
  assumes is-free-for A v B and is-free-for A v C
  shows is-free-for A v (B ▪ C)
unfolding is-free-for-def proof (intro ballI impI)
  fix v' and p
  assume v' ∈ free-vars A and p ∈ positions (B ▪ C) and is-free-at v p (B ▪ C)
  from ‹is-free-at v p (B ▪ C)› have p ≠ []
    using occurs-at-alt-def(8) by force
  then obtain d and p' where p = d # p'
    by (meson list.exhaust)
  with ‹p ∈ positions (B ▪ C)›
  consider (b) p = « # p' and p' ∈ positions B | (c) p = » # p' and p' ∈ positions C
    by force
  then show ¬ in-scope-of-abs v' p (B ▪ C)
  proof cases
    case b
    from b(1) and ‹is-free-at v p (B ▪ C)› have is-free-at v p' B
      using is-free-at-in-left-app by blast
    with assms(1) and ‹v' ∈ free-vars A› and ‹p' ∈ positions B› have ¬ in-scope-of-abs v' p' B
      by simp
    with b(1) show ?thesis
      using in-scope-of-abs-in-left-app by simp
  next
    case c
    from c(1) and ‹is-free-at v p (B ▪ C)› have is-free-at v p' C
      using is-free-at-in-right-app by blast
    with assms(2) and ‹v' ∈ free-vars A› and ‹p' ∈ positions C› have ¬ in-scope-of-abs v' p' C
      by simp
    with c(1) show ?thesis
      using in-scope-of-abs-in-right-app by simp
  qed
```

qed

lemma *is-free-for-in-app*:
shows *is-free-for A v* $(B \cdot C) \longleftrightarrow$ *is-free-for A v B* $\wedge$ *is-free-for A v C*
using *is-free-for-from-app* **and** *is-free-for-to-app* **by** *iprover*

lemma *is-free-for-to-abs* [*intro*]:
assumes *is-free-for A v B* **and** $(x, \alpha) \notin$ *free-vars A*
shows *is-free-for A v* $(\lambda x_\alpha.\ B)$
unfolding *is-free-for-def* **proof** (*intro ballI impI*)
fix v' **and** p
assume $v' \in$ *free-vars A* **and** $p \in$ *positions* $(\lambda x_\alpha.\ B)$ **and** *is-free-at v p* $(\lambda x_\alpha.\ B)$
from ‹*is-free-at v p* $(\lambda x_\alpha.\ B)$› **have** $p \neq []$
using *occurs-at-alt-def*(*9*) **by** *force*
with ‹$p \in$ *positions* $(\lambda x_\alpha.\ B)$› **obtain** p' **where** $p = « \#\ p'$ **and** $p' \in$ *positions B*
by *force*
then show $\neg$ *in-scope-of-abs v' p* $(\lambda x_\alpha.\ B)$
proof −
from ‹$p = « \#\ p'$› **and** ‹*is-free-at v p* $(\lambda x_\alpha.\ B)$› **have** *is-free-at v p' B*
using *is-free-at-from-abs* **by** *blast*
with *assms*(*1*) **and** ‹$v' \in$ *free-vars A*› **and** ‹$p' \in$ *positions B*› **have** $\neg$ *in-scope-of-abs v' p' B*
by *force*
moreover from ‹$v' \in$ *free-vars A*› **and** *assms*(*2*) **have** $v' \neq (x, \alpha)$
by *blast*
ultimately show *?thesis*
using ‹$p = « \#\ p'$› **and** *in-scope-of-abs-in-abs* **by** *auto*
qed
qed

lemma *is-free-for-from-abs*:
assumes *is-free-for A v* $(\lambda x_\alpha.\ B)$ **and** $v \neq (x, \alpha)$
shows *is-free-for A v B*
unfolding *is-free-for-def* **proof** (*intro ballI impI*)
fix v' **and** p
assume $v' \in$ *free-vars A* **and** $p \in$ *positions B* **and** *is-free-at v p B*
then show $\neg$ *in-scope-of-abs v' p B*
proof −
from ‹*is-free-at v p B*› **and** *assms*(*2*) **have** *is-free-at v* $(« \#\ p)$ $(\lambda x_\alpha.\ B)$
by (*rule is-free-at-to-abs*)
moreover from ‹$p \in$ *positions B*› **have** $« \#\ p \in$ *positions* $(\lambda x_\alpha.\ B)$
by *simp*
ultimately have $\neg$ *in-scope-of-abs v'* $(« \#\ p)$ $(\lambda x_\alpha.\ B)$
using *assms* **and** ‹$v' \in$ *free-vars A*› **by** *blast*
then show *?thesis*
using *in-scope-of-abs-in-abs* **by** *blast*
qed
qed

lemma *closed-is-free-for* [*intro*]:

assumes *free-vars A* = {}
shows *is-free-for A v B*
using *assms* **by** *force*

lemma *is-free-for-closed-form* [*intro*]:
assumes *free-vars B* = {}
shows *is-free-for A v B*
using *assms* **and** *is-free-at-in-free-vars* **by** *blast*

lemma *is-free-for-alt-def*:
shows
is-free-for A v B
⟷
(
∄ *p*.
(
p ∈ *positions B* ∧ *is-free-at v p B* ∧ *p* ≠ [] ∧
(∃ *v'* ∈ *free-vars A*. ∃ *p' C*. *strict-prefix p' p* ∧ *FAbs v' C* ⪯$_{p'}$ *B*)
)
)
unfolding *is-free-for-def*
using *in-scope-of-abs-alt-def* **and** *is-subform-implies-in-positions*
by *meson*

lemma *binding-var-not-free-for-in-abs*:
assumes *is-free x B* **and** *x* ≠ *w*
shows ¬ *is-free-for* (*FVar w*) *x* (*FAbs w B*)
proof (*rule ccontr*)
assume ¬ ¬ *is-free-for* (*FVar w*) *x* (*FAbs w B*)
then have
∀ *v'* ∈ *free-vars* (*FVar w*). ∀ *p* ∈ *positions* (*FAbs w B*). *is-free-at x p* (*FAbs w B*)
⟶ ¬ *in-scope-of-abs v' p* (*FAbs w B*)
by *force*
moreover have *free-vars* (*FVar w*) = {*w*}
using *surj-pair*[*of w*] **by** *force*
ultimately have
∀ *p* ∈ *positions* (*FAbs w B*). *is-free-at x p* (*FAbs w B*) ⟶ ¬ *in-scope-of-abs w p* (*FAbs w B*)
by *blast*
moreover from *assms*(*1*) **obtain** *p* **where** *is-free-at x p B*
by *fastforce*
from *this* **and** *assms*(*2*) **have** *is-free-at x* (« # *p*) (*FAbs w B*)
by (*rule is-free-at-to-abs*)
moreover from *this* **have** « # *p* ∈ *positions* (*FAbs w B*)
using *is-subform-implies-in-positions* **by** *force*
ultimately have ¬ *in-scope-of-abs w* (« # *p*) (*FAbs w B*)
by *blast*
moreover have *in-scope-of-abs w* (« # *p*) (*FAbs w B*)
using *in-scope-of-abs-in-abs* **by** *blast*
ultimately show *False*

by *contradiction*
qed

lemma *absent-var-is-free-for* [*intro*]:
assumes $x \notin vars\ A$
shows *is-free-for* $(FVar\ x)\ y\ A$
using *in-scope-of-abs-in-vars* **and** *assms* **and** *surj-pair*[*of* x] **by** *fastforce*

lemma *form-is-free-for-absent-var* [*intro*]:
assumes $x \notin vars\ A$
shows *is-free-for* $B\ x\ A$
using *assms* **and** *occurs-in-vars* **by** *fastforce*

lemma *form-with-free-binder-not-free-for*:
assumes $v \neq v'$ **and** $v' \in free\text{-}vars\ A$ **and** $v \in free\text{-}vars\ B$
shows $\neg$ *is-free-for* $A\ v\ (FAbs\ v'\ B)$
proof −
from *assms*(*3*) **obtain** p **where** $p \in positions\ B$ **and** *is-free-at* $v\ p\ B$
using *free-vars-in-is-free-at* **by** *blast*
then have $« \#\ p \in positions\ (FAbs\ v'\ B)$ **and** *is-free-at* $v\ (« \#\ p)\ (FAbs\ v'\ B)$
using *surj-pair*[*of* v'] **and** *is-free-at-to-abs*[*OF* ‹*is-free-at* $v\ p\ B$› *assms*(*1*)] **by** *force*+
moreover have *in-scope-of-abs* $v'\ (« \#\ p)\ (FAbs\ v'\ B)$
using *in-scope-of-abs-in-abs* **by** *blast*
ultimately show *?thesis*
using *assms*(*2*) **by** *blast*
qed

2.10 Replacement of subformulas

inductive
is-replacement-at :: $form \Rightarrow position \Rightarrow form \Rightarrow form \Rightarrow bool$
$((4\text{-}\langle\!|\text{-} \leftarrow \text{-}|\!\rangle \rhd \text{-})\ [1000,\ 0,\ 0,\ 0]\ 900)$
where
pos-found: $A\langle\!|p \leftarrow C|\!\rangle \rhd C'$ **if** $p = []$ **and** $C = C'$
| *replace-left-app*: $(G \cdot H)\langle\!|« \#\ p \leftarrow C|\!\rangle \rhd (G' \cdot H)$ **if** $p \in positions\ G$ **and** $G\langle\!|p \leftarrow C|\!\rangle \rhd G'$
| *replace-right-app*: $(G \cdot H)\langle\!|» \#\ p \leftarrow C|\!\rangle \rhd (G \cdot H')$ **if** $p \in positions\ H$ **and** $H\langle\!|p \leftarrow C|\!\rangle \rhd H'$
| *replace-abs*: $(\lambda x_\gamma.\ E)\langle\!|« \#\ p \leftarrow C|\!\rangle \rhd (\lambda x_\gamma.\ E')$ **if** $p \in positions\ E$ **and** $E\langle\!|p \leftarrow C|\!\rangle \rhd E'$

lemma *is-replacement-at-implies-in-positions*:
assumes $C\langle\!|p \leftarrow A|\!\rangle \rhd D$
shows $p \in positions\ C$
using *assms* **by** (*induction rule*: *is-replacement-at.induct*) *auto*

declare *is-replacement-at.intros* [*intro*!]

lemma *is-replacement-at-existence*:
assumes $p \in positions\ C$
obtains D **where** $C\langle\!|p \leftarrow A|\!\rangle \rhd D$
using *assms* **proof** (*induction C arbitrary*: *p thesis*)

case (*FApp* C_1 C_2)
from *FApp.prems*(*2*) **consider**
(*a*) $p = []$
| (*b*) $\exists\, p'.\ p = «\ \#\ p' \wedge p' \in$ *positions* C_1
| (*c*) $\exists\, p'.\ p = »\ \#\ p' \wedge p' \in$ *positions* C_2
by *fastforce*
then show *?case*
proof *cases*
case *a*
with *FApp.prems*(*1*) **show** *?thesis*
by *blast*
next
case *b*
with *FApp.prems*(*1*) **show** *?thesis*
using *FApp.IH*(*1*) **and** *replace-left-app* **by** *meson*
next
case *c*
with *FApp.prems*(*1*) **show** *?thesis*
using *FApp.IH*(*2*) **and** *replace-right-app* **by** *meson*
qed
next
case (*FAbs v C'*)
from *FAbs.prems*(*2*) **consider** (*a*) $p = []$ | (*b*) $\exists\, p'.\ p = «\ \#\ p' \wedge p' \in$ *positions* C'
using *surj-pair*[*of v*] **by** *fastforce*
then show *?case*
proof *cases*
case *a*
with *FAbs.prems*(*1*) **show** *?thesis*
by *blast*
next
case *b*
with *FAbs.prems*(*1*,*2*) **show** *?thesis*
using *FAbs.IH* **and** *surj-pair*[*of v*] **by** *blast*
qed
qed *force+*

lemma *is-replacement-at-minimal-change*:
assumes $C\langle p \leftarrow A\rangle \rhd D$
shows $A \preceq_p D$
and $\forall\, p' \in$ *positions* $D.\ \neg$ *prefix* $p'\ p \wedge \neg$ *prefix* $p\ p' \longrightarrow$ *subform-at* $D\ p' =$ *subform-at* $C\ p'$
using *assms* **by** (*induction rule*: *is-replacement-at.induct*) *auto*

lemma *is-replacement-at-binders*:
assumes $C\langle p \leftarrow A\rangle \rhd D$
shows *binders-at* $D\ p =$ *binders-at* $C\ p$
using *assms* **by** (*induction rule*: *is-replacement-at.induct*) *simp-all*

lemma *is-replacement-at-occurs*:
assumes $C\langle p \leftarrow A\rangle \rhd D$

```
  and ¬ prefix p' p and ¬ prefix p p'
  shows occurs-at v p' C ⟷ occurs-at v p' D
using assms proof (induction arbitrary: p' rule: is-replacement-at.induct)
  case pos-found
  then show ?case
    by simp
next
  case replace-left-app
  then show ?case
  proof (cases p')
    case (Cons d p'')
    with replace-left-app.prems(1,2) show ?thesis
      by (cases d) (use replace-left-app.IH in force)+
  qed force
next
  case replace-right-app
  then show ?case
  proof (cases p')
    case (Cons d p'')
    with replace-right-app.prems(1,2) show ?thesis
      by (cases d) (use replace-right-app.IH in force)+
  qed force
next
  case replace-abs
  then show ?case
  proof (cases p')
    case (Cons d p'')
    with replace-abs.prems(1,2) show ?thesis
      by (cases d) (use replace-abs.IH in force)+
  qed force
qed

lemma fresh-var-replacement-position-uniqueness:
  assumes v ∉ vars C
  and C⟨p ← FVar v⟩ ▷ G
  and occurs-at v p' G
  shows p' = p
proof (rule ccontr)
  assume p' ≠ p
  from assms(2) have occurs-at v p G
    by (simp add: is-replacement-at-minimal-change(1))
  moreover have *: occurs-at v p' C ⟷ occurs-at v p' G if ¬ prefix p' p and ¬ prefix p p'
    using assms(2) and that and is-replacement-at-occurs by blast
  ultimately show False
  proof (cases ¬ prefix p' p ∧ ¬ prefix p p')
    case True
    with assms(3) and * have occurs-at v p' C
      by simp
    then have v ∈ vars C
```

```
    using is-subform-implies-in-positions and occurs-in-vars by fastforce
  with assms(1) show ?thesis
    by contradiction
 next
  case False
  have FVar v ⪯p G
    by (fact is-replacement-at-minimal-change(1)[OF assms(2)])
  moreover from assms(3) have FVar v ⪯p' G
    by simp
  ultimately show ?thesis
    using ‹p' ≠ p› and False and loop-subform-impossibility
    by (blast dest: prefix-order.antisym-conv2)
 qed
qed

lemma is-replacement-at-new-positions:
 assumes C⟨p ← A⟩ ▷ D and prefix p p' and p' ∈ positions D
 obtains p'' where p' = p @ p'' and p'' ∈ positions A
 using assms by (induction arbitrary: thesis p' rule: is-replacement-at.induct, auto) blast+

lemma replacement-override:
 assumes C⟨p ← B⟩ ▷ D and C⟨p ← A⟩ ▷ F
 shows D⟨p ← A⟩ ▷ F
using assms proof (induction arbitrary: F rule: is-replacement-at.induct)
 case pos-found
 from pos-found.hyps(1) and pos-found.prems have A = F
   using is-replacement-at.simps by blast
 with pos-found.hyps(1) show ?case
   by blast
next
 case (replace-left-app p G C G' H)
 have p ∈ positions G'
   by (
       fact is-subform-implies-in-positions
         [OF is-replacement-at-minimal-change(1)[OF replace-left-app.hyps(2)]]
      )
 from replace-left-app.prems obtain F' where F = F' ▪ H and G⟨p ← A⟩ ▷ F'
   by (fastforce elim: is-replacement-at.cases)
 from ‹G⟨p ← A⟩ ▷ F'› have G'⟨p ← A⟩ ▷ F'
   by (fact replace-left-app.IH)
 with ‹p ∈ positions G'› show ?case
   unfolding ‹F = F' ▪ H› by blast
next
 case (replace-right-app p H C H' G)
 have p ∈ positions H'
   by
     (
       fact is-subform-implies-in-positions
         [OF is-replacement-at-minimal-change(1)[OF replace-right-app.hyps(2)]]
```

```
    )
  from replace-right-app.prems obtain F' where F = G ▪ F' and H⟪p ← A⟫ ▷ F'
    by (fastforce elim: is-replacement-at.cases)
  from ‹H⟪p ← A⟫ ▷ F'› have H'⟪p ← A⟫ ▷ F'
    by (fact replace-right-app.IH)
  with ‹p ∈ positions H'› show ?case
    unfolding ‹F = G ▪ F'› by blast
next
  case (replace-abs p E C E' x γ)
  have p ∈ positions E'
    by
      (
        fact is-subform-implies-in-positions
          [OF is-replacement-at-minimal-change(1)[OF replace-abs.hyps(2)]]
      )
  from replace-abs.prems obtain F' where F = λx_γ. F' and E⟪p ← A⟫ ▷ F'
    by (fastforce elim: is-replacement-at.cases)
  from ‹E⟪p ← A⟫ ▷ F'› have E'⟪p ← A⟫ ▷ F'
    by (fact replace-abs.IH)
  with ‹p ∈ positions E'› show ?case
    unfolding ‹F = λx_γ. F'› by blast
qed

lemma leftmost-subform-in-generalized-app-replacement:
  shows (•Q⋆ C As)⟪replicate (length As) « ← D⟫ ▷ (•Q⋆ D As)
  using is-replacement-at-implies-in-positions and replace-left-app
  by (induction As arbitrary: D rule: rev-induct) auto
```

2.11 Logical constants

```
abbreviation (input) 𝔵 where 𝔵 ≡ 0
abbreviation (input) 𝔶 where 𝔶 ≡ Suc 𝔵
abbreviation (input) 𝔷 where 𝔷 ≡ Suc 𝔶
abbreviation (input) 𝔣 where 𝔣 ≡ Suc 𝔷
abbreviation (input) 𝔤 where 𝔤 ≡ Suc 𝔣
abbreviation (input) 𝔥 where 𝔥 ≡ Suc 𝔤
abbreviation (input) 𝔠 where 𝔠 ≡ Suc 𝔥
abbreviation (input) 𝔠Q where 𝔠Q ≡ Suc 𝔠
abbreviation (input) 𝔠ι where 𝔠ι ≡ Suc 𝔠Q

definition Q-constant-of-type :: type ⇒ con where
  [simp]: Q-constant-of-type α = (𝔠Q, α→α→o)

definition iota-constant :: con where
  [simp]: iota-constant ≡ (𝔠ι, (i→o)→i)

definition Q :: type ⇒ form (Q-) where
  [simp]: Q_α = FCon (Q-constant-of-type α)
```

definition *iota* :: *form* (ι) **where**
[*simp*]: $\iota = FCon\ iota\text{-}constant$

definition *is-Q-constant-of-type* :: *con* $\Rightarrow$ *type* $\Rightarrow$ *bool* **where**
[*iff*]: *is-Q-constant-of-type p* $\alpha \longleftrightarrow p =$ *Q-constant-of-type* α

definition *is-iota-constant* :: *con* $\Rightarrow$ *bool* **where**
[*iff*]: *is-iota-constant p* $\longleftrightarrow p =$ *iota-constant*

definition *is-logical-constant* :: *con* $\Rightarrow$ *bool* **where**
[*iff*]: *is-logical-constant p* $\longleftrightarrow (\exists\beta.$ *is-Q-constant-of-type p* $\beta) \vee$ *is-iota-constant p*

definition *type-of-Q-constant* :: *con* $\Rightarrow$ *type* **where**
[*simp*]: *type-of-Q-constant p* $= (THE\ \alpha.$ *is-Q-constant-of-type p* $\alpha)$

lemma *constant-cases*[*case-names non-logical Q-constant* ι*-constant*, *cases type*: *con*]:
assumes $\neg$ *is-logical-constant p* $\Longrightarrow P$
and $\bigwedge\beta.$ *is-Q-constant-of-type p* $\beta \Longrightarrow P$
and *is-iota-constant p* $\Longrightarrow P$
shows P
using *assms* **by** *blast*

2.12 Definitions and abbreviations

definition *equality-of-type* :: *form* $\Rightarrow$ *type* $\Rightarrow$ *form* $\Rightarrow$ *form* ((- $=_-$/ -) [*103*, *0*, *103*] *102*) **where**
[*simp*]: $A =_\alpha B = Q_\alpha \cdot A \cdot B$

definition *equivalence* :: *form* $\Rightarrow$ *form* $\Rightarrow$ *form* (**infixl** $\equiv^{\mathfrak{Q}}$ *102*) **where**
[*simp*]: $A \equiv^{\mathfrak{Q}} B = A =_o B$ — more modular than the definition in [2]

definition *true* :: *form* (T_o) **where**
[*simp*]: $T_o = Q_o =_{o\to o\to o} Q_o$

definition *false* :: *form* (F_o) **where**
[*simp*]: $F_o = \lambda\mathfrak{x}_o.\ T_o =_{o\to o} \lambda\mathfrak{x}_o.\ \mathfrak{x}_o$

definition *PI* :: *type* $\Rightarrow$ *form* ($\prod$ _) **where**
[*simp*]: $\prod\alpha = Q_{\alpha\to o} \cdot (\lambda\mathfrak{x}_\alpha.\ T_o)$

definition *forall* :: *nat* $\Rightarrow$ *type* $\Rightarrow$ *form* $\Rightarrow$ *form* ((*4*$\forall$ -_./ -) [*0*, *0*, *141*] *141*) **where**
[*simp*]: $\forall x_\alpha.\ A = \prod\alpha \cdot (\lambda x_\alpha.\ A)$

Generalized universal quantification. We define $\forall^{\mathfrak{Q}}_\star\ [x_1, \ldots, x_n]\ A$ as $\forall x_1.\ \cdots\ \forall x_n.\ A$:

definition *generalized-forall* :: *var list* $\Rightarrow$ *form* $\Rightarrow$ *form* ($\forall^{\mathfrak{Q}}_\star$ - - [*141*, *141*] *141*) **where**
[*simp*]: $\forall^{\mathfrak{Q}}_\star$ *vs A* = *foldr* ($\lambda(x, \alpha)$ *B*. $\forall x_\alpha.$ *B*) *vs A*

lemma *innermost-subform-in-generalized-forall*:
assumes *vs* $\neq$ []
shows $A \preceq_{foldr\ (\lambda\text{-}\ p.\ [»,«]\ @\ p)\ vs\ []}\ \forall^{\mathfrak{Q}}_\star\ vs\ A$

using *assms* **by** (*induction vs*) *fastforce+*

lemma *innermost-replacement-in-generalized-forall*:
assumes $vs \neq []$
shows $(\forall^{\mathcal{Q}}_{\star}\ vs\ C)\langle\!|foldr\ (\lambda\text{-}.\ (@)\ [»,«])\ vs\ []\leftarrow B|\!\rangle \rhd (\forall^{\mathcal{Q}}_{\star}\ vs\ B)$
using *assms* **proof** (*induction vs*)
case *Nil*
then show *?case*
by *blast*
next
case (*Cons v vs*)
obtain x **and** α **where** $v = (x, \alpha)$
by *fastforce*
then show *?case*
proof (*cases vs* = []*)*
case *True*
with ‹$v = (x, \alpha)$› **show** *?thesis*
unfolding *True* **by** *force*
next
case *False*
then have *foldr* $(\lambda\text{-}.\ (@)\ [»,«])$ *vs* [] ∈ *positions* $(\forall^{\mathcal{Q}}_{\star}\ vs\ C)$
using *innermost-subform-in-generalized-forall* **and** *is-subform-implies-in-positions* **by** *blast*
moreover from *False* **have** $(\forall^{\mathcal{Q}}_{\star}\ vs\ C)\langle\!|foldr\ (\lambda\text{-}.\ (@)\ [»,«])\ vs\ []\leftarrow B|\!\rangle \rhd (\forall^{\mathcal{Q}}_{\star}\ vs\ B)$
by (*fact Cons.IH*)
ultimately have $(\lambda x_{\alpha}.\ \forall^{\mathcal{Q}}_{\star}\ vs\ C)\langle\!|« \#\ foldr\ (\lambda\text{-}.\ (@)\ [»,«])\ vs\ []\leftarrow B|\!\rangle \rhd (\lambda x_{\alpha}.\ \forall^{\mathcal{Q}}_{\star}\ vs\ B)$
by (*rule replace-abs*)
moreover have « # *foldr* $(\lambda\text{-}.\ (@)\ [»,«])$ *vs* [] ∈ *positions* $(\lambda x_{\alpha}.\ \forall^{\mathcal{Q}}_{\star}\ vs\ C)$
using ‹*foldr* $(\lambda\text{-}.\ (@)\ [»,«])$ *vs* [] ∈ *positions* $(\forall^{\mathcal{Q}}_{\star}\ vs\ C)$› **by** *simp*
ultimately have
$(\prod\alpha \cdot (\lambda x_{\alpha}.\ \forall^{\mathcal{Q}}_{\star}\ vs\ C))\langle\!|» \#\ « \#\ foldr\ (\lambda\text{-}.\ (@)\ [»,«])\ vs\ []\leftarrow B|\!\rangle \rhd (\prod\alpha \cdot (\lambda x_{\alpha}.\ \forall^{\mathcal{Q}}_{\star}\ vs\ B))$
by *blast*
then have $(\forall\, x_{\alpha}.\ \forall^{\mathcal{Q}}_{\star}\ vs\ C)\langle\!|[»,«] @ foldr\ (\lambda\text{-}.\ (@)\ [»,«])\ vs\ []\leftarrow B|\!\rangle \rhd (\forall\, x_{\alpha}.\ \forall^{\mathcal{Q}}_{\star}\ vs\ B)$
by *simp*
then show *?thesis*
unfolding ‹$v = (x, \alpha)$› **and** *generalized-forall-def* **and** *foldr.simps*(*2*) **and** *o-apply*
and *case-prod-conv* **.**
qed
qed

lemma *false-is-forall*:
shows $F_o = \forall\mathfrak{x}_o.\ \mathfrak{x}_o$
unfolding *false-def* **and** *forall-def* **and** *PI-def* **and** *equality-of-type-def* **..**

definition *conj-fun* :: *form* $(\wedge_{o\to o\to o})$ **where**
[*simp*]: $\wedge_{o\to o\to o} =$
$\lambda\mathfrak{x}_o.\ \lambda\mathfrak{y}_o.$
(
$(\lambda\mathfrak{g}_{o\to o\to o}.\ \mathfrak{g}_{o\to o\to o} \cdot T_o \cdot T_o) =_{(o\to o\to o)\to o} (\lambda\mathfrak{g}_{o\to o\to o}.\ \mathfrak{g}_{o\to o\to o} \cdot \mathfrak{x}_o \cdot \mathfrak{y}_o)$
)

definition *conj-op* :: *form* ⇒ *form* ⇒ *form* (**infixl** $\wedge^{\mathcal{Q}}$ *131*) **where**
[*simp*]: $A \wedge^{\mathcal{Q}} B = \wedge_{o\to o\to o} \cdot A \cdot B$

Generalized conjunction. We define $\wedge^{\mathcal{Q}}_{\star}\ [A_1, \ldots, A_n]$ as $A_1 \wedge^{\mathcal{Q}} (\cdots \wedge^{\mathcal{Q}} (A_{n-1} \wedge^{\mathcal{Q}} A_n) \cdots)$:

definition *generalized-conj-op* :: *form list* ⇒ *form* ($\wedge^{\mathcal{Q}}_{\star}$ - [*0*] *131*) **where**
[*simp*]: $\wedge^{\mathcal{Q}}_{\star}\ As = foldr1\ (\wedge^{\mathcal{Q}})\ As$

definition *imp-fun* :: *form* ($\supset_{o\to o\to o}$) **where** — ≡ used instead of =, see [2]
[*simp*]: $\supset_{o\to o\to o} = \lambda\mathfrak{x}_o.\ \lambda\mathfrak{y}_o.\ (\mathfrak{x}_o \equiv^{\mathcal{Q}} \mathfrak{x}_o \wedge^{\mathcal{Q}} \mathfrak{y}_o)$

definition *imp-op* :: *form* ⇒ *form* ⇒ *form* (**infixl** $\supset^{\mathcal{Q}}$ *111*) **where**
[*simp*]: $A \supset^{\mathcal{Q}} B = \supset_{o\to o\to o} \cdot A \cdot B$

Generalized implication. We define $[A_1, \ldots, A_n] \supset^{\mathcal{Q}}_{\star} B$ as $A_1 \supset^{\mathcal{Q}} (\cdots \supset^{\mathcal{Q}} (A_n \supset^{\mathcal{Q}} B) \cdots)$:

definition *generalized-imp-op* :: *form list* ⇒ *form* ⇒ *form* (**infixl** $\supset^{\mathcal{Q}}_{\star}$ *111*) **where**
[*simp*]: $As \supset^{\mathcal{Q}}_{\star} B = foldr\ (\supset^{\mathcal{Q}})\ As\ B$

Given the definition below, it is interesting to note that $\sim^{\mathcal{Q}} A$ and $F_o \equiv^{\mathcal{Q}} A$ are exactly the same formula, namely $Q_o \cdot F_o \cdot A$:

definition *neg* :: *form* ⇒ *form* ($\sim^{\mathcal{Q}}$ - [*141*] *141*) **where**
[*simp*]: $\sim^{\mathcal{Q}} A = Q_o \cdot F_o \cdot A$

definition *disj-fun* :: *form* ($\vee_{o\to o\to o}$) **where**
[*simp*]: $\vee_{o\to o\to o} = \lambda\mathfrak{x}_o.\ \lambda\mathfrak{y}_o.\ \sim^{\mathcal{Q}} (\sim^{\mathcal{Q}} \mathfrak{x}_o \wedge^{\mathcal{Q}} \sim^{\mathcal{Q}} \mathfrak{y}_o)$

definition *disj-op* :: *form* ⇒ *form* ⇒ *form* (**infixl** $\vee^{\mathcal{Q}}$ *126*) **where**
[*simp*]: $A \vee^{\mathcal{Q}} B = \vee_{o\to o\to o} \cdot A \cdot B$

definition *exists* :: *nat* ⇒ *type* ⇒ *form* ⇒ *form* ((*4*∃ -_:/ -) [*0*, *0*, *141*] *141*) **where**
[*simp*]: $\exists\, x_\alpha.\ A = \sim^{\mathcal{Q}} (\forall\, x_\alpha.\ \sim^{\mathcal{Q}} A)$

lemma *exists-fv*:
shows *free-vars* $(\exists\, x_\alpha.\ A) = free\text{-}vars\ A - \{(x, \alpha)\}$
by *simp*

definition *inequality-of-type* :: *form* ⇒ *type* ⇒ *form* ⇒ *form* ((- $\neq$_:/ -) [*103*, *0*, *103*] *102*) **where**
[*simp*]: $A \neq_\alpha B = \sim^{\mathcal{Q}} (A =_\alpha B)$

2.13 Well-formed formulas

inductive *is-wff-of-type* :: *type* ⇒ *form* ⇒ *bool* **where**
var-is-wff: *is-wff-of-type* α (x_α)
| *con-is-wff*: *is-wff-of-type* α $(\{\!|c|\!\}_\alpha)$
| *app-is-wff*: *is-wff-of-type* β $(A \cdot B)$ **if** *is-wff-of-type* $(\alpha\to\beta)$ A **and** *is-wff-of-type* α B
| *abs-is-wff*: *is-wff-of-type* $(\alpha\to\beta)$ $(\lambda x_\alpha.\ A)$ **if** *is-wff-of-type* β A

definition *wffs-of-type* :: *type* ⇒ *form set* (*wffs*_ [*0*]) **where**
$wffs_\alpha = \{f :: form.\ is\text{-}wff\text{-}of\text{-}type\ \alpha\ f\}$

abbreviation *wffs* :: *form set* **where**
  $wffs \equiv \bigcup \alpha.\ wffs_\alpha$

lemma *is-wff-of-type-wffs-of-type-eq* [*pred-set-conv*]:
  shows *is-wff-of-type* $\alpha = (\lambda f.\ f \in wffs_\alpha)$
  unfolding *wffs-of-type-def* **by** *simp*

lemmas *wffs-of-type-intros* [*intro*!] = *is-wff-of-type.intros*[*to-set*]
lemmas *wffs-of-type-induct* [*consumes 1*, *induct set*: *wffs-of-type*] = *is-wff-of-type.induct*[*to-set*]
lemmas *wffs-of-type-cases* [*consumes 1*, *cases set*: *wffs-of-type*] = *is-wff-of-type.cases*[*to-set*]
lemmas *wffs-of-type-simps* = *is-wff-of-type.simps*[*to-set*]

lemma *generalized-app-wff* [*intro*]:
  assumes *length As* = *length ts*
  and $\forall\, k < length\ As.\ As\ !\ k \in wffs_{ts\ !\ k}$
  and $B \in wffs_{foldr\ (\to)\ ts\ \beta}$
  shows $\cdot^{\mathcal{Q}}_\star\ B\ As \in wffs_\beta$
using *assms* **proof** (*induction As ts arbitrary*: *B rule*: *list-induct2*)
  case *Nil*
  then show *?case*
    by *simp*
next
  case (*Cons A As t ts*)
  from *Cons.prems*(*1*) **have** $A \in wffs_t$
    by *fastforce*
  moreover from *Cons.prems*(*2*) **have** $B \in wffs_{t \to foldr\ (\to)\ ts\ \beta}$
    by *auto*
  ultimately have $B \cdot A \in wffs_{foldr\ (\to)\ ts\ \beta}$
    by *blast*
  moreover have $\forall\, k < length\ As.\ (A\ \#\ As)\ !\ (Suc\ k) = As\ !\ k \wedge (t\ \#\ ts)\ !\ (Suc\ k) = ts\ !\ k$
    by *force*
  with *Cons.prems*(*1*) **have** $\forall\, k < length\ As.\ As\ !\ k \in wffs_{ts\ !\ k}$
    by *fastforce*
  ultimately have $\cdot^{\mathcal{Q}}_\star\ (B \cdot A)\ As \in wffs_\beta$
    using *Cons.IH* **by** (*simp only*:)
  moreover have $\cdot^{\mathcal{Q}}_\star\ B\ (A\ \#\ As) = \cdot^{\mathcal{Q}}_\star\ (B \cdot A)\ As$
    by *simp*
  ultimately show *?case*
    by (*simp only*:)
qed

lemma *generalized-abs-wff* [*intro*]:
  assumes $B \in wffs_\beta$
  shows $\lambda^{\mathcal{Q}}_\star\ vs\ B \in wffs_{foldr\ (\to)\ (map\ snd\ vs)\ \beta}$
using *assms* **proof** (*induction vs*)
  case *Nil*
  then show *?case*

by *simp*
next
case (*Cons v vs*)
let *?δ* = *foldr* (→) (*map snd vs*) *β*
obtain *x* **and** *α* **where** *v* = (*x*, *α*)
by *fastforce*
then have *FVar v* ∈ *wffs*$_\alpha$
by *auto*
from *Cons.prems* **have** $\lambda^{\mathcal{Q}}_\star$ *vs B* ∈ *wffs*$_{?\delta}$
by (*fact Cons.IH*)
with ‹*v* = (*x*, *α*)› **have** *FAbs v* ($\lambda^{\mathcal{Q}}_\star$ *vs B*) ∈ *wffs*$_{\alpha\to ?\delta}$
by *blast*
moreover from ‹*v* = (*x*, *α*)› **have** *foldr* (→) (*map snd* (*v* # *vs*)) *β* = *α*→*?δ*
by *simp*
moreover have $\lambda^{\mathcal{Q}}_\star$ (*v* # *vs*) *B* = *FAbs v* ($\lambda^{\mathcal{Q}}_\star$ *vs B*)
by *simp*
ultimately show *?case* **by** (*simp only*:)
qed

lemma *Q-wff* [*intro*]:
shows *Q*$_\alpha$ ∈ *wffs*$_{\alpha\to\alpha\to o}$
by *auto*

lemma *iota-wff* [*intro*]:
shows *ι* ∈ *wffs*$_{(i\to o)\to i}$
by *auto*

lemma *equality-wff* [*intro*]:
assumes *A* ∈ *wffs*$_\alpha$ **and** *B* ∈ *wffs*$_\alpha$
shows *A* =$_\alpha$ *B* ∈ *wffs*$_o$
using *assms* **by** *auto*

lemma *equivalence-wff* [*intro*]:
assumes *A* ∈ *wffs*$_o$ **and** *B* ∈ *wffs*$_o$
shows *A* ≡$^{\mathcal{Q}}$ *B* ∈ *wffs*$_o$
using *assms* **unfolding** *equivalence-def* **by** *blast*

lemma *true-wff* [*intro*]:
shows *T*$_o$ ∈ *wffs*$_o$
by *force*

lemma *false-wff* [*intro*]:
shows *F*$_o$ ∈ *wffs*$_o$
by *auto*

lemma *pi-wff* [*intro*]:
shows ∏ *α* ∈ *wffs*$_{(\alpha\to o)\to o}$
using *PI-def* **by** *fastforce*

lemma *forall-wff* [*intro*]:
assumes $A \in \mathit{wffs}_o$
shows $\forall\, x_\alpha.\ A \in \mathit{wffs}_o$
using *assms* **and** *pi-wff* **unfolding** *forall-def* **by** *blast*

lemma *generalized-forall-wff* [*intro*]:
assumes $B \in \mathit{wffs}_o$
shows $\forall^{\mathcal{Q}}_\star\ \mathit{vs}\ B \in \mathit{wffs}_o$
using *assms* **proof** (*induction vs*)
case (*Cons v vs*)
then show *?case*
using *surj-pair*[*of v*] **by** *force*
qed *simp*

lemma *conj-fun-wff* [*intro*]:
shows $\wedge_{o\to o\to o} \in \mathit{wffs}_{o\to o\to o}$
by *auto*

lemma *conj-op-wff* [*intro*]:
assumes $A \in \mathit{wffs}_o$ **and** $B \in \mathit{wffs}_o$
shows $A \wedge^{\mathcal{Q}} B \in \mathit{wffs}_o$
using *assms* **unfolding** *conj-op-def* **by** *blast*

lemma *imp-fun-wff* [*intro*]:
shows $\supset_{o\to o\to o} \in \mathit{wffs}_{o\to o\to o}$
by *auto*

lemma *imp-op-wff* [*intro*]:
assumes $A \in \mathit{wffs}_o$ **and** $B \in \mathit{wffs}_o$
shows $A \supset^{\mathcal{Q}} B \in \mathit{wffs}_o$
using *assms* **unfolding** *imp-op-def* **by** *blast*

lemma *neg-wff* [*intro*]:
assumes $A \in \mathit{wffs}_o$
shows $\sim^{\mathcal{Q}} A \in \mathit{wffs}_o$
using *assms* **by** *fastforce*

lemma *disj-fun-wff* [*intro*]:
shows $\vee_{o\to o\to o} \in \mathit{wffs}_{o\to o\to o}$
by *auto*

lemma *disj-op-wff* [*intro*]:
assumes $A \in \mathit{wffs}_o$ **and** $B \in \mathit{wffs}_o$
shows $A \vee^{\mathcal{Q}} B \in \mathit{wffs}_o$
using *assms* **by** *auto*

lemma *exists-wff* [*intro*]:
assumes $A \in \mathit{wffs}_o$
shows $\exists\, x_\alpha.\ A \in \mathit{wffs}_o$

using *assms* **by** *fastforce*

lemma *inequality-wff* [*intro*]:
assumes $A \in wffs_\alpha$ **and** $B \in wffs_\alpha$
shows $A \neq_\alpha B \in wffs_o$
using *assms* **by** *fastforce*

lemma *wffs-from-app*:
assumes $A \cdot B \in wffs_\beta$
obtains α **where** $A \in wffs_{\alpha\to\beta}$ **and** $B \in wffs_\alpha$
using *assms* **by** (*blast elim*: *wffs-of-type-cases*)

lemma *wffs-from-generalized-app*:
assumes $\cdot^{\mathcal{Q}}_\star\ B\ As \in wffs_\beta$
obtains *ts*
where *length ts* = *length As*
and $\forall\, k < length\ As.\ As\ !\ k \in wffs_{ts\ !\ k}$
and $B \in wffs_{foldr\ (\to)\ ts\ \beta}$
using *assms* **proof** (*induction As arbitrary*: *B thesis*)
case *Nil*
then show *?case*
by *simp*
next
case (*Cons A As*)
from *Cons.prems* **have** $\cdot^{\mathcal{Q}}_\star\ (B \cdot A)\ As \in wffs_\beta$
by *auto*
then obtain *ts*
where *length ts* = *length As*
and $\forall\, k < length\ As.\ As\ !\ k \in wffs_{ts\ !\ k}$
and $B \cdot A \in wffs_{foldr\ (\to)\ ts\ \beta}$
using *Cons.IH* **by** *blast*
moreover
from ‹$B \cdot A \in wffs_{foldr\ (\to)\ ts\ \beta}$› **obtain** *t* **where** $B \in wffs_{t\to foldr\ (\to)\ ts\ \beta}$ **and** $A \in wffs_t$
by (*elim wffs-from-app*)
moreover from ‹*length ts* = *length As*› **have** *length* (*t* # *ts*) = *length* (*A* # *As*)
by *simp*
moreover from ‹$A \in wffs_t$› **and** ‹$\forall\, k < length\ As.\ As\ !\ k \in wffs_{ts\ !\ k}$›
have $\forall\, k < length\ (A\ \#\ As).\ (A\ \#\ As)\ !\ k \in wffs_{(t\ \#\ ts)\ !\ k}$
by (*simp add*: *nth-Cons'*)
moreover from ‹$B \in wffs_{t\to foldr\ (\to)\ ts\ \beta}$› **have** $B \in wffs_{foldr\ (\to)\ (t\ \#\ ts)\ \beta}$
by *simp*
ultimately show *?case*
using *Cons.prems*(*1*) **by** *blast*
qed

lemma *wffs-from-abs*:
assumes $\lambda x_\alpha.\ A \in wffs_\gamma$
obtains β **where** $\gamma = \alpha\to\beta$ **and** $A \in wffs_\beta$

using *assms* **by** (*blast elim*: *wffs-of-type-cases*)

lemma *wffs-from-equality*:
assumes $A =_\alpha B \in wffs_o$
shows $A \in wffs_\alpha$ **and** $B \in wffs_\alpha$
using *assms* **by** (*fastforce elim*: *wffs-of-type-cases*)+

lemma *wffs-from-equivalence*:
assumes $A \equiv^{\mathcal{Q}} B \in wffs_o$
shows $A \in wffs_o$ **and** $B \in wffs_o$
using *assms* **unfolding** *equivalence-def* **by** (*fact wffs-from-equality*)+

lemma *wffs-from-forall*:
assumes $\forall\, x_\alpha.\ A \in wffs_o$
shows $A \in wffs_o$
using *assms* **unfolding** *forall-def* **and** *PI-def*
by (*fold equality-of-type-def*) (*drule wffs-from-equality*, *blast elim*: *wffs-from-abs*)

lemma *wffs-from-conj-fun*:
assumes $\wedge_{o\to o\to o} \cdot A \cdot B \in wffs_o$
shows $A \in wffs_o$ **and** $B \in wffs_o$
using *assms* **by** (*auto elim*: *wffs-from-app wffs-from-abs*)

lemma *wffs-from-conj-op*:
assumes $A \wedge^{\mathcal{Q}} B \in wffs_o$
shows $A \in wffs_o$ **and** $B \in wffs_o$
using *assms* **unfolding** *conj-op-def* **by** (*elim wffs-from-conj-fun*)+

lemma *wffs-from-imp-fun*:
assumes $\supset_{o\to o\to o} \cdot A \cdot B \in wffs_o$
shows $A \in wffs_o$ **and** $B \in wffs_o$
using *assms* **by** (*auto elim*: *wffs-from-app wffs-from-abs*)

lemma *wffs-from-imp-op*:
assumes $A \supset^{\mathcal{Q}} B \in wffs_o$
shows $A \in wffs_o$ **and** $B \in wffs_o$
using *assms* **unfolding** *imp-op-def* **by** (*elim wffs-from-imp-fun*)+

lemma *wffs-from-neg*:
assumes $\sim^{\mathcal{Q}} A \in wffs_o$
shows $A \in wffs_o$
using *assms* **unfolding** *neg-def* **by** (*fold equality-of-type-def*) (*drule wffs-from-equality*, *blast*)

lemma *wffs-from-disj-fun*:
assumes $\vee_{o\to o\to o} \cdot A \cdot B \in wffs_o$
shows $A \in wffs_o$ **and** $B \in wffs_o$
using *assms* **by** (*auto elim*: *wffs-from-app wffs-from-abs*)

lemma *wffs-from-disj-op*:

assumes $A \vee^{\mathfrak{Q}} B \in \mathit{wffs}_o$
shows $A \in \mathit{wffs}_o$ **and** $B \in \mathit{wffs}_o$
using *assms* **and** *wffs-from-disj-fun* **unfolding** *disj-op-def* **by** *blast+*

lemma *wffs-from-exists*:
assumes $\exists\, x_\alpha.\ A \in \mathit{wffs}_o$
shows $A \in \mathit{wffs}_o$
using *assms* **unfolding** *exists-def* **using** *wffs-from-neg* **and** *wffs-from-forall* **by** *blast*

lemma *wffs-from-inequality*:
assumes $A \neq_\alpha B \in \mathit{wffs}_o$
shows $A \in \mathit{wffs}_\alpha$ **and** $B \in \mathit{wffs}_\alpha$
using *assms* **unfolding** *inequality-of-type-def* **using** *wffs-from-equality* **and** *wffs-from-neg* **by** *meson+*

lemma *wff-has-unique-type*:
assumes $A \in \mathit{wffs}_\alpha$ **and** $A \in \mathit{wffs}_\beta$
shows $\alpha = \beta$
using *assms* **proof** (*induction arbitrary*: α β *rule*: *form.induct*)
case (*FVar v*)
obtain x **and** γ **where** $v = (x, \gamma)$
by *fastforce*
with *FVar.prems* **have** $\alpha = \gamma$ **and** $\beta = \gamma$
by (*blast elim*: *wffs-of-type-cases*)+
then show *?case* **..**
next
case (*FCon k*)
obtain x **and** γ **where** $k = (x, \gamma)$
by *fastforce*
with *FCon.prems* **have** $\alpha = \gamma$ **and** $\beta = \gamma$
by (*blast elim*: *wffs-of-type-cases*)+
then show *?case* **..**
next
case (*FApp A B*)
from *FApp.prems* **obtain** α' **and** β' **where** $A \in \mathit{wffs}_{\alpha' \to \alpha}$ **and** $A \in \mathit{wffs}_{\beta' \to \beta}$
by (*blast elim*: *wffs-from-app*)
with *FApp.IH*(*1*) **show** *?case*
by *blast*
next
case (*FAbs v A*)
obtain x **and** γ **where** $v = (x, \gamma)$
by *fastforce*
with *FAbs.prems* **obtain** α' **and** β'
where $\alpha = \gamma \to \alpha'$ **and** $\beta = \gamma \to \beta'$ **and** $A \in \mathit{wffs}_{\alpha'}$ **and** $A \in \mathit{wffs}_{\beta'}$
by (*blast elim*: *wffs-from-abs*)
with *FAbs.IH* **show** *?case*
by *simp*
qed

lemma *wffs-of-type-o-induct* [*consumes 1*, *case-names Var Con App*]:
assumes $A \in wffs_o$
and $\bigwedge x.\ \mathcal{P}\ (x_o)$
and $\bigwedge c.\ \mathcal{P}\ (\{\!|c|\!\}_o)$
and $\bigwedge A\ B\ \alpha.\ A \in wffs_{\alpha\to o} \Longrightarrow B \in wffs_\alpha \Longrightarrow \mathcal{P}\ (A \cdot B)$
shows $\mathcal{P}\ A$
using *assms* **by** (*cases rule*: *wffs-of-type-cases*) *simp-all*

lemma *diff-types-implies-diff-wffs*:
assumes $A \in wffs_\alpha$ **and** $B \in wffs_\beta$
and $\alpha \neq \beta$
shows $A \neq B$
using *assms* **and** *wff-has-unique-type* **by** *blast*

lemma *is-free-for-in-generalized-app* [*intro*]:
assumes *is-free-for A v B* **and** $\forall\, C \in lset\ Cs.$ *is-free-for A v C*
shows *is-free-for A v* ($\cdot^{\mathcal{Q}}{}_\star$ *B Cs*)
using *assms* **proof** (*induction Cs rule*: *rev-induct*)
case *Nil*
then show *?case*
by *simp*
next
case (*snoc C Cs*)
from *snoc.prems*(*2*) **have** *is-free-for A v C* **and** $\forall\, C \in lset\ Cs.$ *is-free-for A v C*
by *simp-all*
with *snoc.prems*(*1*) **have** *is-free-for A v* ($\cdot^{\mathcal{Q}}{}_\star$ *B Cs*)
using *snoc.IH* **by** *simp*
with ‹*is-free-for A v C*› **show** *?case*
using *is-free-for-to-app* **by** *simp*
qed

lemma *is-free-for-in-equality* [*intro*]:
assumes *is-free-for A v B* **and** *is-free-for A v C*
shows *is-free-for A v* ($B =_\alpha C$)
using *assms* **unfolding** *equality-of-type-def* **and** *Q-def* **and** *Q-constant-of-type-def*
by (*intro is-free-for-to-app is-free-for-in-con*)

lemma *is-free-for-in-equivalence* [*intro*]:
assumes *is-free-for A v B* **and** *is-free-for A v C*
shows *is-free-for A v* ($B \equiv^{\mathcal{Q}} C$)
using *assms* **unfolding** *equivalence-def* **by** (*rule is-free-for-in-equality*)

lemma *is-free-for-in-true* [*intro*]:
shows *is-free-for A v* (T_o)
by *force*

lemma *is-free-for-in-false* [*intro*]:
shows *is-free-for A v* (F_o)
unfolding *false-def* **by** (*intro is-free-for-in-equality is-free-for-closed-form*) *simp-all*

lemma *is-free-for-in-forall* [*intro*]:
assumes *is-free-for A v B* **and** $(x, \alpha) \notin$ *free-vars A*
shows *is-free-for A v* $(\forall\, x_\alpha.\ B)$
unfolding *forall-def* **and** *PI-def* **proof** (*fold equality-of-type-def*)
have *is-free-for A v* $(\lambda \mathfrak{x}_\alpha.\ T_o)$
using *is-free-for-to-abs*[*OF is-free-for-in-true assms*(*2*)] **by** *fastforce*
moreover have *is-free-for A v* $(\lambda x_\alpha.\ B)$
by (*fact is-free-for-to-abs*[*OF assms*])
ultimately show *is-free-for A v* $(\lambda \mathfrak{x}_\alpha.\ T_o =_{\alpha\to o} \lambda x_\alpha.\ B)$
by (*iprover intro*: *assms*(*1*) *is-free-for-in-equality is-free-for-in-true is-free-for-to-abs*)
qed

lemma *is-free-for-in-generalized-forall* [*intro*]:
assumes *is-free-for A v B* **and** *lset vs* $\cap$ *free-vars A* $= \{\}$
shows *is-free-for A v* $(\forall^{\mathfrak{Q}}_\star$ *vs B*$)$
using *assms* **proof** (*induction vs*)
case *Nil*
then show *?case*
by *simp*
next
case (*Cons v' vs*)
obtain *x* **and** α **where** $v' = (x, \alpha)$
by *fastforce*
from *Cons.prems*(*2*) **have** $v' \notin$ *free-vars A* **and** *lset vs* $\cap$ *free-vars A* $= \{\}$
by *simp-all*
from *Cons.prems*(*1*) **and** ‹*lset vs* $\cap$ *free-vars A* $= \{\}$› **have** *is-free-for A v* $(\forall^{\mathfrak{Q}}_\star$ *vs B*$)$
by (*fact Cons.IH*)
from *this* **and** ‹$v' \notin$ *free-vars A*›[*unfolded* ‹$v' = (x, \alpha)$›] **have** *is-free-for A v* $(\forall\, x_\alpha.\ \forall^{\mathfrak{Q}}_\star$ *vs B*$)$
by (*intro is-free-for-in-forall*)
with ‹$v' = (x, \alpha)$› **show** *?case*
by *simp*
qed

lemma *is-free-for-in-conj* [*intro*]:
assumes *is-free-for A v B* **and** *is-free-for A v C*
shows *is-free-for A v* $(B \wedge^{\mathfrak{Q}} C)$
proof −
have *free-vars* $\wedge_{o\to o\to o} = \{\}$
by *force*
then have *is-free-for A v* $(\wedge_{o\to o\to o})$
using *is-free-for-closed-form* **by** *fast*
with *assms* **have** *is-free-for A v* $(\wedge_{o\to o\to o} \cdot B \cdot C)$
by (*intro is-free-for-to-app*)
then show *?thesis*
by (*fold conj-op-def*)
qed

lemma *is-free-for-in-imp* [*intro*]:

assumes *is-free-for A v B* **and** *is-free-for A v C*
shows *is-free-for A v* $(B \supset^{\mathcal{Q}} C)$
proof −
have *free-vars* $\supset_{o\to o\to o} = \{\}$
by *force*
then have *is-free-for A v* $(\supset_{o\to o\to o})$
using *is-free-for-closed-form* **by** *fast*
with *assms* **have** *is-free-for A v* $(\supset_{o\to o\to o} \cdot B \cdot C)$
by (*intro is-free-for-to-app*)
then show *?thesis*
by (*fold imp-op-def*)
qed

lemma *is-free-for-in-neg* [*intro*]:
assumes *is-free-for A v B*
shows *is-free-for A v* $(\sim^{\mathcal{Q}} B)$
using *assms* **unfolding** *neg-def* **and** *Q-def* **and** *Q-constant-of-type-def*
by (*intro is-free-for-to-app is-free-for-in-false is-free-for-in-con*)

lemma *is-free-for-in-disj* [*intro*]:
assumes *is-free-for A v B* **and** *is-free-for A v C*
shows *is-free-for A v* $(B \vee^{\mathcal{Q}} C)$
proof −
have *free-vars* $\vee_{o\to o\to o} = \{\}$
by *force*
then have *is-free-for A v* $(\vee_{o\to o\to o})$
using *is-free-for-closed-form* **by** *fast*
with *assms* **have** *is-free-for A v* $(\vee_{o\to o\to o} \cdot B \cdot C)$
by (*intro is-free-for-to-app*)
then show *?thesis*
by (*fold disj-op-def*)
qed

lemma *replacement-preserves-typing*:
assumes $C\langle\!|p \leftarrow B|\!\rangle \rhd D$
and $A \preceq_p C$
and $A \in wffs_\alpha$ **and** $B \in wffs_\alpha$
shows $C \in wffs_\beta \longleftrightarrow D \in wffs_\beta$
using *assms* **proof** (*induction arbitrary*: β *rule*: *is-replacement-at.induct*)
case (*pos-found p C C′ A*)
then show *?case*
using *diff-types-implies-diff-wffs* **by** *auto*
qed (*metis is-subform-at.simps*(*2*,*3*,*4*) *wffs-from-app wffs-from-abs wffs-of-type-simps*)+

corollary *replacement-preserves-typing′*:
assumes $C\langle\!|p \leftarrow B|\!\rangle \rhd D$
and $A \preceq_p C$
and $A \in wffs_\alpha$ **and** $B \in wffs_\alpha$
and $C \in wffs_\beta$ **and** $D \in wffs_\gamma$

shows $\beta = \gamma$
using *assms* **and** *replacement-preserves-typing* **and** *wff-has-unique-type* **by** *simp*

Closed formulas and sentences:

definition *is-closed-wff-of-type* :: *form* ⇒ *type* ⇒ *bool* **where**
[*iff*]: *is-closed-wff-of-type A α* ⟷ $A \in wffs_\alpha \wedge$ *free-vars A* = {}

definition *is-sentence* :: *form* ⇒ *bool* **where**
[*iff*]: *is-sentence A* ⟷ *is-closed-wff-of-type A o*

2.14 Substitutions

type-synonym *substitution* = (*var*, *form*) *fmap*

definition *is-substitution* :: *substitution* ⇒ *bool* **where**
[*iff*]: *is-substitution ϑ* ⟷ (∀ (*x*, *α*) ∈ *fmdom′ ϑ*. *ϑ* $$! (*x*, *α*) ∈ $wffs_\alpha$)

fun *substitute* :: *substitution* ⇒ *form* ⇒ *form* (**S** - - [*51*, *51*]) **where**
S *ϑ* (x_α) = (*case ϑ* $$ (*x*, *α*) *of None* ⇒ x_α | *Some A* ⇒ *A*)
| **S** *ϑ* ($\{\!|c|\!\}_\alpha$) = $\{\!|c|\!\}_\alpha$
| **S** *ϑ* (*A* ∙ *B*) = (**S** *ϑ A*) ∙ (**S** *ϑ B*)
| **S** *ϑ* (λx_α. *A*) = (*if* (*x*, *α*) ∉ *fmdom′ ϑ then* λx_α. **S** *ϑ A else* λx_α. **S** (*fmdrop* (*x*, *α*) *ϑ*) *A*)

lemma *empty-substitution-neutrality*:
shows S {$$} *A* = *A*
by (*induction A*) *auto*

lemma *substitution-preserves-typing*:
assumes *is-substitution ϑ*
and $A \in wffs_\alpha$
shows S *ϑ A* ∈ $wffs_\alpha$
using *assms*(*2*) **and** *assms*(*1*)[*unfolded is-substitution-def*] **proof** (*induction arbitrary*: *ϑ*)
case (*var-is-wff α x*)
then show *?case*
by (*cases* (*x*, *α*) ∈ *fmdom′ ϑ*) (*use fmdom′-notI* **in** ‹*force*+›)
next
case (*abs-is-wff β A α x*)
then show *?case*
proof (*cases* (*x*, *α*) ∈ *fmdom′ ϑ*)
case *True*
then have S *ϑ* (λx_α. *A*) = λx_α. **S** (*fmdrop* (*x*, *α*) *ϑ*) *A*
by *simp*
moreover from *abs-is-wff.prems* **have** *is-substitution* (*fmdrop* (*x*, *α*) *ϑ*)
by *fastforce*
with *abs-is-wff.IH* **have S** (*fmdrop* (*x*, *α*) *ϑ*) *A* ∈ $wffs_\beta$
by *simp*
ultimately show *?thesis*
by *auto*
next

case *False*
then have **S** ϑ $(\lambda x_\alpha.\ A) = \lambda x_\alpha.$ **S** ϑ A
by *simp*
moreover from *abs-is-wff.IH* **have** **S** ϑ $A \in wffs_\beta$
using *abs-is-wff.prems* **by** *blast*
ultimately show *?thesis*
by *fastforce*
qed
qed *force+*

lemma *derived-substitution-simps*:
shows **S** ϑ $T_o = T_o$
and **S** ϑ $F_o = F_o$
and **S** ϑ $(\prod \alpha) = \prod \alpha$
and **S** ϑ $(\sim^{\mathfrak{Q}} B) = \sim^{\mathfrak{Q}}$ (**S** ϑ B)
and **S** ϑ $(B =_\alpha C) =$ (**S** ϑ B) $=_\alpha$ (**S** ϑ C)
and **S** ϑ $(B \wedge^{\mathfrak{Q}} C) =$ (**S** ϑ B) $\wedge^{\mathfrak{Q}}$ (**S** ϑ C)
and **S** ϑ $(B \vee^{\mathfrak{Q}} C) =$ (**S** ϑ B) $\vee^{\mathfrak{Q}}$ (**S** ϑ C)
and **S** ϑ $(B \supset^{\mathfrak{Q}} C) =$ (**S** ϑ B) $\supset^{\mathfrak{Q}}$ (**S** ϑ C)
and **S** ϑ $(B \equiv^{\mathfrak{Q}} C) =$ (**S** ϑ B) $\equiv^{\mathfrak{Q}}$ (**S** ϑ C)
and **S** ϑ $(B \neq_\alpha C) =$ (**S** ϑ B) $\neq_\alpha$ (**S** ϑ C)
and **S** ϑ $(\forall x_\alpha.\ B) =$ (*if* $(x, \alpha) \notin$ *fmdom'* ϑ *then* $\forall x_\alpha.$ **S** ϑ B *else* $\forall x_\alpha.$ **S** (*fmdrop* (x, α) ϑ) B)
and **S** ϑ $(\exists x_\alpha.\ B) =$ (*if* $(x, \alpha) \notin$ *fmdom'* ϑ *then* $\exists x_\alpha.$ **S** ϑ B *else* $\exists x_\alpha.$ **S** (*fmdrop* (x, α) ϑ) B)
by *auto*

lemma *generalized-app-substitution*:
shows **S** ϑ $(\cdot^{\mathfrak{Q}}{}_\star\ A\ Bs) = \cdot^{\mathfrak{Q}}{}_\star$ (**S** ϑ A) (*map* $(\lambda B.$ **S** ϑ $B)$ Bs)
by (*induction Bs arbitrary*: A) *simp-all*

lemma *generalized-abs-substitution*:
shows **S** ϑ $(\lambda^{\mathfrak{Q}}{}_\star\ vs\ A) = \lambda^{\mathfrak{Q}}{}_\star\ vs$ (**S** (*fmdrop-set* (*fmdom'* ϑ $\cap$ *lset vs*) ϑ) A)
proof (*induction vs arbitrary*: ϑ)
case *Nil*
then show *?case*
by *simp*
next
case (*Cons v vs*)
obtain x **and** α **where** $v = (x, \alpha)$
by *fastforce*
then show *?case*
proof (*cases* $v \notin$ *fmdom'* ϑ)
case *True*
then have $*$: *fmdom'* ϑ $\cap$ *lset* $(v \#\ vs)$ = *fmdom'* ϑ $\cap$ *lset vs*
by *simp*
from *True* **have** **S** ϑ $(\lambda^{\mathfrak{Q}}{}_\star\ (v \#\ vs)\ A) = \lambda x_\alpha.$ **S** ϑ $(\lambda^{\mathfrak{Q}}{}_\star\ vs\ A)$
using ‹$v = (x, \alpha)$› **by** *auto*
also have $\ldots = \lambda x_\alpha.\ \lambda^{\mathfrak{Q}}{}_\star\ vs$ (**S** (*fmdrop-set* (*fmdom'* ϑ $\cap$ *lset vs*) ϑ) A)
using *Cons.IH* **by** (*simp only*:)
also have $\ldots = \lambda^{\mathfrak{Q}}{}_\star\ (v \#\ vs)$ (**S** (*fmdrop-set* (*fmdom'* ϑ $\cap$ *lset* $(v \#\ vs)$) ϑ) A)

using ‹$v = (x, \alpha)$› **and** $*$ **by** *auto*
finally show *?thesis* **.**
next
case *False*
let $?\vartheta' = fmdrop\ v\ \vartheta$
have $*$: $fmdrop\text{-}set\ (fmdom'\ \vartheta \cap lset\ (v\ \#\ vs))\ \vartheta = fmdrop\text{-}set\ (fmdom'\ ?\vartheta' \cap lset\ vs)\ ?\vartheta'$
using *False* **by** *clarsimp* (*metis Int-Diff Int-commute fmdrop-set-insert insert-Diff-single*)
from *False* **have S** $\vartheta\ (\lambda^{\mathcal{Q}}_{\star}\ (v\ \#\ vs)\ A) = \lambda x_\alpha.$ **S** $?\vartheta'\ (\lambda^{\mathcal{Q}}_{\star}\ vs\ A)$
using ‹$v = (x, \alpha)$› **by** *auto*
also have $\ldots = \lambda x_\alpha.\ \lambda^{\mathcal{Q}}_{\star}\ vs$ (**S** $(fmdrop\text{-}set\ (fmdom'\ ?\vartheta' \cap lset\ vs)\ ?\vartheta')\ A$)
using *Cons.IH* **by** (*simp only*:)
also have $\ldots = \lambda^{\mathcal{Q}}_{\star}\ (v\ \#\ vs)$ (**S** $(fmdrop\text{-}set\ (fmdom'\ \vartheta \cap lset\ (v\ \#\ vs))\ \vartheta)\ A$)
using ‹$v = (x, \alpha)$› **and** $*$ **by** *auto*
finally show *?thesis* **.**
qed
qed

lemma *generalized-forall-substitution*:
shows S $\vartheta\ (\forall^{\mathcal{Q}}_{\star}\ vs\ A) = \forall^{\mathcal{Q}}_{\star}\ vs$ (**S** $(fmdrop\text{-}set\ (fmdom'\ \vartheta \cap lset\ vs)\ \vartheta)\ A$)
proof (*induction vs arbitrary*: ϑ)
case *Nil*
then show *?case*
by *simp*
next
case (*Cons v vs*)
obtain x **and** α **where** $v = (x, \alpha)$
by *fastforce*
then show *?case*
proof (*cases* $v \notin fmdom'\ \vartheta$)
case *True*
then have $*$: $fmdom'\ \vartheta \cap lset\ (v\ \#\ vs) = fmdom'\ \vartheta \cap lset\ vs$
by *simp*
from *True* **have S** $\vartheta\ (\forall^{\mathcal{Q}}_{\star}\ (v\ \#\ vs)\ A) = \forall\, x_\alpha.$ **S** $\vartheta\ (\forall^{\mathcal{Q}}_{\star}\ vs\ A)$
using ‹$v = (x, \alpha)$› **by** *auto*
also have $\ldots = \forall\, x_\alpha.\ \forall^{\mathcal{Q}}_{\star}\ vs$ (**S** $(fmdrop\text{-}set\ (fmdom'\ \vartheta \cap lset\ vs)\ \vartheta)\ A$)
using *Cons.IH* **by** (*simp only*:)
also have $\ldots = \forall^{\mathcal{Q}}_{\star}\ (v\ \#\ vs)$ (**S** $(fmdrop\text{-}set\ (fmdom'\ \vartheta \cap lset\ (v\ \#\ vs))\ \vartheta)\ A$)
using ‹$v = (x, \alpha)$› **and** $*$ **by** *auto*
finally show *?thesis* **.**
next
case *False*
let $?\vartheta' = fmdrop\ v\ \vartheta$
have $*$: $fmdrop\text{-}set\ (fmdom'\ \vartheta \cap lset\ (v\ \#\ vs))\ \vartheta = fmdrop\text{-}set\ (fmdom'\ ?\vartheta' \cap lset\ vs)\ ?\vartheta'$
using *False* **by** *clarsimp* (*metis Int-Diff Int-commute fmdrop-set-insert insert-Diff-single*)
from *False* **have S** $\vartheta\ (\forall^{\mathcal{Q}}_{\star}\ (v\ \#\ vs)\ A) = \forall\, x_\alpha.$ **S** $?\vartheta'\ (\forall^{\mathcal{Q}}_{\star}\ vs\ A)$
using ‹$v = (x, \alpha)$› **by** *auto*
also have $\ldots = \forall\, x_\alpha.\ \forall^{\mathcal{Q}}_{\star}\ vs$ (**S** $(fmdrop\text{-}set\ (fmdom'\ ?\vartheta' \cap lset\ vs)\ ?\vartheta')\ A$)
using *Cons.IH* **by** (*simp only*:)
also have $\ldots = \forall^{\mathcal{Q}}_{\star}\ (v\ \#\ vs)$ (**S** $(fmdrop\text{-}set\ (fmdom'\ \vartheta \cap lset\ (v\ \#\ vs))\ \vartheta)\ A$)

```
    using ‹v = (x, α)› and ∗ by auto
  finally show ?thesis .
 qed
qed

lemma singleton-substitution-simps:
 shows S {(x, α) ↣ A} (yβ) = (if (x, α) ≠ (y, β) then yβ else A)
 and S {(x, α) ↣ A} (⦇c⦈α) = ⦇c⦈α
 and S {(x, α) ↣ A} (B ▪ C) = (S {(x, α) ↣ A} B) ▪ (S {(x, α) ↣ A} C)
 and S {(x, α) ↣ A} (λyβ. B) = λyβ. (if (x, α) = (y, β) then B else S {(x, α) ↣ A} B)
 by (simp-all add: empty-substitution-neutrality fmdrop-fmupd-same)

lemma substitution-preserves-freeness:
 assumes y ∉ free-vars A and y ≠ z
 shows y ∉ free-vars S {x ↣ FVar z} A
using assms(1) proof (induction A rule: free-vars-form.induct)
 case (1 x′ α)
 with assms(2) show ?case
  using surj-pair[of z] by (cases x = (x′, α)) force+
next
 case (4 x′ α A)
 then show ?case
  using surj-pair[of z]
  by (cases x = (x′, α)) (use singleton-substitution-simps(4) in presburger, auto)
qed auto

lemma renaming-substitution-minimal-change:
 assumes y ∉ vars A and y ≠ z
 shows y ∉ vars (S {x ↣ FVar z} A)
using assms(1) proof (induction A rule: vars-form.induct)
 case (1 x′ α)
 with assms(2) show ?case
  using surj-pair[of z] by (cases x = (x′, α)) force+
next
 case (4 x′ α A)
 then show ?case
  using surj-pair[of z]
  by (cases x = (x′, α)) (use singleton-substitution-simps(4) in presburger, auto)
qed auto

lemma free-var-singleton-substitution-neutrality:
 assumes v ∉ free-vars A
 shows S {v ↣ B} A = A
 using assms
 by
  (induction A rule: free-vars-form.induct)
  (simp-all, metis empty-substitution-neutrality fmdrop-empty fmdrop-fmupd-same)

lemma identity-singleton-substitution-neutrality:
```

```
  shows S {v ↣ FVar v} A = A
  by
    (induction A rule: free-vars-form.induct)
    (simp-all add: empty-substitution-neutrality fmdrop-fmupd-same)

lemma free-var-in-renaming-substitution:
  assumes x ≠ y
  shows (x, α) ∉ free-vars (S {(x, α) ↣ y_α} B)
  using assms by (induction B rule: free-vars-form.induct) simp-all

lemma renaming-substitution-preserves-form-size:
  shows form-size (S {v ↣ FVar v'} A) = form-size A
proof (induction A rule: form-size.induct)
  case (1 x α)
  then show ?case
    using form-size.elims by auto
next
  case (4 x α A)
  then show ?case
    by (cases v = (x, α)) (use singleton-substitution-simps(4) in presburger, auto)
qed simp-all
```

The following lemma corresponds to X5100 in [2]:

```
lemma substitution-composability:
  assumes v' ∉ vars B
  shows S {v' ↣ A} S {v ↣ FVar v'} B = S {v ↣ A} B
using assms proof (induction B arbitrary: v')
  case (FAbs w C)
  then show ?case
  proof (cases v = w)
    case True
    from ‹v' ∉ vars (FAbs w C)› have v' ∉ free-vars (FAbs w C)
      using free-vars-in-all-vars by blast
    then have S {v' ↣ A} (FAbs w C) = FAbs w C
      by (rule free-var-singleton-substitution-neutrality)
    from ‹v = w› have v ∉ free-vars (FAbs w C)
      using surj-pair[of w] by fastforce
    then have S {v ↣ A} (FAbs w C) = FAbs w C
      by (fact free-var-singleton-substitution-neutrality)
    also from ‹S {v' ↣ A} (FAbs w C) = FAbs w C› have ... = S {v' ↣ A} (FAbs w C)
      by (simp only:)
    also from ‹v = w› have ... = S {v' ↣ A} S {v ↣ FVar v'} (FAbs w C)
      using free-var-singleton-substitution-neutrality[OF ‹v ∉ free-vars (FAbs w C)›] by (simp only:)
    finally show ?thesis ..
  next
    case False
    from FAbs.prems have v' ∉ vars C
      using surj-pair[of w] by fastforce
    then show ?thesis
```

```
    proof (cases v' = w)
      case True
      with FAbs.prems show ?thesis
        using vars-form.elims by auto
    next
      case False
      from ‹v ≠ w› have S {v ⟼ A} (FAbs w C) = FAbs w (S {v ⟼ A} C)
        using surj-pair[of w] by fastforce
      also from FAbs.IH have ... = FAbs w (S {v' ⟼ A} S {v ⟼ FVar v'} C)
        using ‹v' ∉ vars C› by simp
      also from ‹v' ≠ w› have ... = S {v' ⟼ A} (FAbs w (S {v ⟼ FVar v'} C))
        using surj-pair[of w] by fastforce
      also from ‹v ≠ w› have ... = S {v' ⟼ A} S {v ⟼ FVar v'} (FAbs w C)
        using surj-pair[of w] by fastforce
      finally show ?thesis ..
    qed
  qed
qed auto
```

The following lemma corresponds to X5101 in [2]:

```
lemma renaming-substitution-composability:
  assumes z ∉ free-vars A and is-free-for (FVar z) x A
  shows S {z ⟼ FVar y} S {x ⟼ FVar z} A = S {x ⟼ FVar y} A
using assms proof (induction A arbitrary: z)
  case (FVar v)
  then show ?case
    using surj-pair[of v] and surj-pair[of z] by fastforce
next
  case (FCon k)
  then show ?case
    using surj-pair[of k] by fastforce
next
  case (FApp B C)
  let ?ϑ_zy = {z ⟼ FVar y} and ?ϑ_xz = {x ⟼ FVar z} and ?ϑ_xy = {x ⟼ FVar y}
  from ‹is-free-for (FVar z) x (B • C)› have is-free-for (FVar z) x B and is-free-for (FVar z) x C
    using is-free-for-from-app by iprover+
  moreover from ‹z ∉ free-vars (B • C)› have z ∉ free-vars B and z ∉ free-vars C
    by simp-all
  ultimately have ∗: S ?ϑ_zy S ?ϑ_xz B = S ?ϑ_xy B and ∗∗: S ?ϑ_zy S ?ϑ_xz C = S ?ϑ_xy C
    using FApp.IH by simp-all
  have S ?ϑ_zy S ?ϑ_xz (B • C) = (S ?ϑ_zy S ?ϑ_xz B) • (S ?ϑ_zy S ?ϑ_xz C)
    by simp
  also from ∗ and ∗∗ have ... = (S ?ϑ_xy B) • (S ?ϑ_xy C)
    by (simp only:)
  also have ... = S ?ϑ_xy (B • C)
    by simp
  finally show ?case .
next
  case (FAbs w B)
```

let $?\vartheta_{zy} = \{z \rightarrowtail FVar\ y\}$ **and** $?\vartheta_{xz} = \{x \rightarrowtail FVar\ z\}$ **and** $?\vartheta_{xy} = \{x \rightarrowtail FVar\ y\}$
show *?case*
proof (*cases* $x = w$)
  case *True*
  then show *?thesis*
  proof (*cases* $z = w$)
    case *True*
    with ‹$x = w$› **have** $x \notin$ *free-vars* (*FAbs w B*) **and** $z \notin$ *free-vars* (*FAbs w B*)
      using *surj-pair*[*of w*] **by** *fastforce+*
    from ‹$x \notin$ *free-vars* (*FAbs w B*)› **have S** $?\vartheta_{xy}$ (*FAbs w B*) = *FAbs w B*
      by (*fact free-var-singleton-substitution-neutrality*)
    also from ‹$z \notin$ *free-vars* (*FAbs w B*)› **have** ... = **S** $?\vartheta_{zy}$ (*FAbs w B*)
      by (*fact free-var-singleton-substitution-neutrality*[*symmetric*])
    also from ‹$x \notin$ *free-vars* (*FAbs w B*)› **have** ... = **S** $?\vartheta_{zy}$ **S** $?\vartheta_{xz}$ (*FAbs w B*)
      using *free-var-singleton-substitution-neutrality* **by** *simp*
    finally show *?thesis* **..**
  next
    case *False*
    with ‹$x = w$› **have** $z \notin$ *free-vars B* **and** $x \notin$ *free-vars* (*FAbs w B*)
      using ‹$z \notin$ *free-vars* (*FAbs w B*)› **and** *surj-pair*[*of w*] **by** *fastforce+*
    from ‹$z \notin$ *free-vars B*› **have S** $?\vartheta_{zy}$ $B = B$
      by (*fact free-var-singleton-substitution-neutrality*)
    from ‹$x \notin$ *free-vars* (*FAbs w B*)› **have S** $?\vartheta_{xy}$ (*FAbs w B*) = *FAbs w B*
      by (*fact free-var-singleton-substitution-neutrality*)
    also from ‹**S** $?\vartheta_{zy}$ $B = B$› **have** ... = *FAbs w* (**S** $?\vartheta_{zy}$ B)
      by (*simp only*:)
    also from ‹$z \notin$ *free-vars* (*FAbs w B*)› **have** ... = **S** $?\vartheta_{zy}$ (*FAbs w B*)
      by (*simp add*: ‹*FAbs w B* = *FAbs w* (**S** $?\vartheta_{zy}$ B)› *free-var-singleton-substitution-neutrality*)
    also from ‹$x \notin$ *free-vars* (*FAbs w B*)› **have** ... = **S** $?\vartheta_{zy}$ **S** $?\vartheta_{xz}$ (*FAbs w B*)
      using *free-var-singleton-substitution-neutrality* **by** *simp*
    finally show *?thesis* **..**
  qed
next
  case *False*
  then show *?thesis*
  proof (*cases* $z = w$)
    case *True*
    have $x \notin$ *free-vars B*
    proof (*rule ccontr*)
      assume ¬ $x \notin$ *free-vars B*
      with ‹$x \neq w$› **have** $x \in$ *free-vars* (*FAbs w B*)
        using *surj-pair*[*of w*] **by** *fastforce*
      then obtain p **where** $p \in$ *positions* (*FAbs w B*) **and** *is-free-at x p* (*FAbs w B*)
        using *free-vars-in-is-free-at* **by** *blast*
      with ‹*is-free-for* (*FVar z*) *x* (*FAbs w B*)› **have** ¬ *in-scope-of-abs z p* (*FAbs w B*)
        by (*meson empty-is-position is-free-at-in-free-vars is-free-at-in-var is-free-for-def*)
      moreover obtain p' **where** $p = \langle\!\langle\ \#\ p'$
        using *is-free-at-from-absE*[*OF* ‹*is-free-at x p* (*FAbs w B*)›] **by** *blast*
      ultimately have $z \neq w$

```
        using in-scope-of-abs-in-abs by blast
      with ‹z = w› show False
        by contradiction
    qed
    then have ∗: S ?ϑ_xy B = S ?ϑ_xz B
      using free-var-singleton-substitution-neutrality by auto
    from ‹x ≠ w› have S ?ϑ_xy (FAbs w B) = FAbs w (S ?ϑ_xy B)
      using surj-pair[of w] by fastforce
    also from ∗ have ... = FAbs w (S ?ϑ_xz B)
      by (simp only:)
    also from FAbs.prems(1) have ... = S ?ϑ_zy (FAbs w (S ?ϑ_xz B))
      using ‹x ∉ free-vars B› and free-var-singleton-substitution-neutrality by auto
    also from ‹x ≠ w› have ... = S ?ϑ_zy S ?ϑ_xz (FAbs w B)
      using surj-pair[of w] by fastforce
    finally show ?thesis ..
  next
    case False
    obtain v_w and α where w = (v_w, α)
      by fastforce
    with ‹is-free-for (FVar z) x (FAbs w B)› and ‹x ≠ w› have is-free-for (FVar z) x B
      using is-free-for-from-abs by iprover
    moreover from ‹z ∉ free-vars (FAbs w B)› and ‹z ≠ w› and ‹w = (v_w, α)› have z ∉ free-vars
B
      by simp
    ultimately have ∗: S ?ϑ_zy S ?ϑ_xz B = S ?ϑ_xy B
      using FAbs.IH by simp
    from ‹x ≠ w› have S ?ϑ_xy (FAbs w B) = FAbs w (S ?ϑ_xy B)
      using ‹w = (v_w, α)› and free-var-singleton-substitution-neutrality by simp
    also from ∗ have ... = FAbs w (S ?ϑ_zy S ?ϑ_xz B)
      by (simp only:)
    also from ‹z ≠ w› have ... = S ?ϑ_zy (FAbs w (S ?ϑ_xz B))
      using ‹w = (v_w, α)› and free-var-singleton-substitution-neutrality by simp
    also from ‹x ≠ w› have ... = S ?ϑ_zy S ?ϑ_xz (FAbs w B)
      using ‹w = (v_w, α)› and free-var-singleton-substitution-neutrality by simp
    finally show ?thesis ..
  qed
 qed
qed

lemma absent-vars-substitution-preservation:
 assumes v ∉ vars A
 and ∀ v' ∈ fmdom' ϑ. v ∉ vars (ϑ $$! v')
 shows v ∉ vars (S ϑ A)
using assms proof (induction A arbitrary: ϑ)
 case (FVar v')
 then show ?case
   using surj-pair[of v'] by (cases v' ∈ fmdom' ϑ) (use fmlookup-dom'-iff in force)+
next
 case (FCon k)
```

```
  then show ?case
    using surj-pair[of k] by fastforce
next
  case FApp
  then show ?case
    by simp
next
  case (FAbs w B)
  from FAbs.prems(1) have v ∉ vars B
    using vars-form.elims by auto
  then show ?case
  proof (cases w ∈ fmdom' ϑ)
    case True
    from FAbs.prems(2) have ∀ v' ∈ fmdom' (fmdrop w ϑ). v ∉ vars ((fmdrop w ϑ) $$! v')
      by auto
    with ‹v ∉ vars B› have v ∉ vars (S (fmdrop w ϑ) B)
      by (fact FAbs.IH)
    with FAbs.prems(1) have v ∉ vars (FAbs w (S (fmdrop w ϑ) B))
      using surj-pair[of w] by fastforce
    moreover from True have S ϑ (FAbs w B) = FAbs w (S (fmdrop w ϑ) B)
      using surj-pair[of w] by fastforce
    ultimately show ?thesis
      by simp
  next
    case False
    then show ?thesis
      using FAbs.IH and FAbs.prems and surj-pair[of w] by fastforce
  qed
qed

lemma substitution-free-absorption:
  assumes ϑ $$ v = None and v ∉ free-vars B
  shows S ({v ↣ A} ++f ϑ) B = S ϑ B
using assms proof (induction B arbitrary: ϑ)
  case (FAbs w B)
  show ?case
  proof (cases v ≠ w)
    case True
    with FAbs.prems(2) have v ∉ free-vars B
      using surj-pair[of w] by fastforce
    then show ?thesis
    proof (cases w ∈ fmdom' ϑ)
      case True
      then have S ({v ↣ A} ++f ϑ) (FAbs w B) = FAbs w (S (fmdrop w ({v ↣ A} ++f ϑ)) B)
        using surj-pair[of w] by fastforce
      also from ‹v ≠ w› and True have ... = FAbs w (S ({v ↣ A} ++f fmdrop w ϑ) B)
        by (simp add: fmdrop-fmupd)
      also from FAbs.prems(1) and ‹v ∉ free-vars B› have ... = FAbs w (S (fmdrop w ϑ) B)
        using FAbs.IH by simp
```

```
    also from True have ... = S ϑ (FAbs w B)
      using surj-pair[of w] by fastforce
    finally show ?thesis .
  next
    case False
    with FAbs.prems(1) have S ({v ↣ A} ++f ϑ) (FAbs w B) = FAbs w (S ({v ↣ A} ++f ϑ) B)
      using ‹v ≠ w› and surj-pair[of w] by fastforce
    also from FAbs.prems(1) and ‹v ∉ free-vars B› have ... = FAbs w (S ϑ B)
      using FAbs.IH by simp
    also from False have ... = S ϑ (FAbs w B)
      using surj-pair[of w] by fastforce
    finally show ?thesis .
  qed
next
  case False
  then have fmdrop w ({v ↣ A} ++f ϑ) = fmdrop w ϑ
    by (simp add: fmdrop-fmupd-same)
  then show ?thesis
    using surj-pair[of w] by (metis (no-types, lifting) fmdrop-idle' substitute.simps(4))
qed
qed fastforce+

lemma substitution-absorption:
  assumes ϑ $$ v = None and v ∉ vars B
  shows S ({v ↣ A} ++f ϑ) B = S ϑ B
  using assms by (meson free-vars-in-all-vars in-mono substitution-free-absorption)

lemma is-free-for-with-renaming-substitution:
  assumes is-free-for A x B
  and y ∉ vars B
  and x ∉ fmdom' ϑ
  and ∀ v ∈ fmdom' ϑ. y ∉ vars (ϑ $$! v)
  and ∀ v ∈ fmdom' ϑ. is-free-for (ϑ $$! v) v B
  shows is-free-for A y (S ({x ↣ FVar y} ++f ϑ) B)
using assms proof (induction B arbitrary: ϑ)
  case (FVar w)
  then show ?case
  proof (cases w = x)
    case True
    with FVar.prems(3) have S ({x ↣ FVar y} ++f ϑ) (FVar w) = FVar y
      using surj-pair[of w] by fastforce
    then show ?thesis
      using self-subform-is-at-top by fastforce
  next
    case False
    then show ?thesis
    proof (cases w ∈ fmdom' ϑ)
      case True
      from False have S ({x ↣ FVar y} ++f ϑ) (FVar w) = S ϑ (FVar w)
```

```
      using substitution-absorption and surj-pair[of w] by force
    also from True have ... = ϑ $$! w
      using surj-pair[of w] by (metis fmdom'-notI option.case-eq-if substitute.simps(1))
    finally have S ({x ↣ FVar y} ++f ϑ) (FVar w) = ϑ $$! w .
    moreover from True and FVar.prems(4) have y ∉ vars (ϑ $$! w)
      by blast
    ultimately show ?thesis
      using form-is-free-for-absent-var by presburger
  next
    case False
    with FVar.prems(3) and ‹w ≠ x› have S ({x ↣ FVar y} ++f ϑ) (FVar w) = FVar w
      using surj-pair[of w] by fastforce
    with FVar.prems(2) show ?thesis
      using form-is-free-for-absent-var by presburger
  qed
 qed
next
 case (FCon k)
 then show ?case
  using surj-pair[of k] by fastforce
next
 case (FApp C D)
 from FApp.prems(2) have y ∉ vars C and y ∉ vars D
  by simp-all
 from FApp.prems(1) have is-free-for A x C and is-free-for A x D
  using is-free-for-from-app by iprover+
 have ∀ v ∈ fmdom' ϑ. is-free-for (ϑ $$! v) v C ∧ is-free-for (ϑ $$! v) v D
 proof (rule ballI)
  fix v
  assume v ∈ fmdom' ϑ
  with FApp.prems(5) have is-free-for (ϑ $$! v) v (C ▪ D)
   by blast
  then show is-free-for (ϑ $$! v) v C ∧ is-free-for (ϑ $$! v) v D
   using is-free-for-from-app by iprover+
 qed
 then have
  ∗: ∀ v ∈ fmdom' ϑ. is-free-for (ϑ $$! v) v C and ∗∗: ∀ v ∈ fmdom' ϑ. is-free-for (ϑ $$! v) v D
  by auto
 have S ({x ↣ FVar y} ++f ϑ) (C ▪ D) = (S ({x ↣ FVar y} ++f ϑ) C) ▪ (S ({x ↣ FVar y}
++f ϑ) D)
  by simp
 moreover have is-free-for A y (S ({x ↣ FVar y} ++f ϑ) C)
  by (rule FApp.IH(1)[OF ‹is-free-for A x C› ‹y ∉ vars C› FApp.prems(3,4) ∗])
 moreover have is-free-for A y (S ({x ↣ FVar y} ++f ϑ) D)
  by (rule FApp.IH(2)[OF ‹is-free-for A x D› ‹y ∉ vars D› FApp.prems(3,4) ∗∗])
 ultimately show ?case
  using is-free-for-in-app by simp
next
 case (FAbs w B)
```

obtain x_w **and** α_w **where** $w = (x_w, \alpha_w)$
by *fastforce*
from *FAbs.prems*(*2*) **have** $y \notin vars\ B$
using *vars-form.elims* **by** *auto*
then show *?case*
proof (*cases* $w = x$)
case *True*
from *True* **and** ‹$x \notin fmdom'\ \vartheta$› **have** $w \notin fmdom'\ \vartheta$ **and** $x \notin free\text{-}vars\ (FAbs\ w\ B)$
using ‹$w = (x_w, \alpha_w)$› **by** *fastforce+*
with *True* **have S** $(\{x \rightarrowtail FVar\ y\} +\!\!+_f \vartheta)\ (FAbs\ w\ B) = \mathbf{S}\ \vartheta\ (FAbs\ w\ B)$
using *substitution-free-absorption* **by** *blast*
also have $\ldots = FAbs\ w\ (\mathbf{S}\ \vartheta\ B)$
using ‹$w = (x_w, \alpha_w)$› ‹$w \notin fmdom'\ \vartheta$› *substitute.simps*(*4*) **by** *presburger*
finally have S $(\{x \rightarrowtail FVar\ y\} +\!\!+_f \vartheta)\ (FAbs\ w\ B) = FAbs\ w\ (\mathbf{S}\ \vartheta\ B)$ **.**
moreover from ‹$\mathbf{S}\ \vartheta\ (FAbs\ w\ B) = FAbs\ w\ (\mathbf{S}\ \vartheta\ B)$› **have** $y \notin vars\ (FAbs\ w\ (\mathbf{S}\ \vartheta\ B))$
using *absent-vars-substitution-preservation*[*OF FAbs.prems*(*2*,*4*)] **by** *simp*
ultimately show *?thesis*
using *is-free-for-absent-var* **by** (*simp only*:)
next
case *False*
obtain v_w **and** α_w **where** $w = (v_w, \alpha_w)$
by *fastforce*
from *FAbs.prems*(*1*) **and** ‹$w \neq x$› **and** ‹$w = (v_w, \alpha_w)$› **have** *is-free-for A x B*
using *is-free-for-from-abs* **by** *iprover*
then show *?thesis*
proof (*cases* $w \in fmdom'\ \vartheta$)
case *True*
then have S $(\{x \rightarrowtail FVar\ y\} +\!\!+_f \vartheta)\ (FAbs\ w\ B) = FAbs\ w\ (\mathbf{S}\ (fmdrop\ w\ (\{x \rightarrowtail FVar\ y\} +\!\!+_f \vartheta))\ B)$
using ‹$w = (v_w, \alpha_w)$› **by** (*simp add*: *fmdrop-idle'*)
also from ‹$w \neq x$› **and** *True* **have** $\ldots = FAbs\ w\ (\mathbf{S}\ (\{x \rightarrowtail FVar\ y\} +\!\!+_f fmdrop\ w\ \vartheta)\ B)$
by (*simp add*: *fmdrop-fmupd*)
finally
have $*$: **S** $(\{x \rightarrowtail FVar\ y\} +\!\!+_f \vartheta)\ (FAbs\ w\ B) = FAbs\ w\ (\mathbf{S}\ (\{x \rightarrowtail FVar\ y\} +\!\!+_f fmdrop\ w\ \vartheta)\ B)$ **.**
have $\forall v \in fmdom'\ (fmdrop\ w\ \vartheta).\ is\text{-}free\text{-}for\ (fmdrop\ w\ \vartheta\ \$\$!\ v)\ v\ B$
proof
fix v
assume $v \in fmdom'\ (fmdrop\ w\ \vartheta)$
with *FAbs.prems*(*5*) **have** *is-free-for* $(fmdrop\ w\ \vartheta\ \$\$!\ v)\ v\ (FAbs\ w\ B)$
by *auto*
moreover from ‹$v \in fmdom'\ (fmdrop\ w\ \vartheta)$› **have** $v \neq w$
by *auto*
ultimately show *is-free-for* $(fmdrop\ w\ \vartheta\ \$\$!\ v)\ v\ B$
unfolding ‹$w = (v_w, \alpha_w)$› **using** *is-free-for-from-abs* **by** *iprover*
qed
moreover from *FAbs.prems*(*3*) **have** $x \notin fmdom'\ (fmdrop\ w\ \vartheta)$
by *simp*
moreover from *FAbs.prems*(*4*) **have** $\forall v \in fmdom'\ (fmdrop\ w\ \vartheta).\ y \notin vars\ (fmdrop\ w\ \vartheta\ \$\$!\ v)$

```
      by simp
    ultimately have is-free-for A y (S ({x ↣ FVar y} ++f fmdrop w ϑ) B)
      using ‹is-free-for A x B› and ‹y ∉ vars B› and FAbs.IH by iprover
    then show ?thesis
    proof (cases x ∉ free-vars B)
      case True
      have y ∉ vars (S ({x ↣ FVar y} ++f ϑ) (FAbs w B))
      proof −
        have S ({x ↣ FVar y} ++f ϑ) (FAbs w B) = FAbs w (S ({x ↣ FVar y} ++f fmdrop w ϑ)
B)
          using ∗ .
        also from ‹x ∉ free-vars B› and FAbs.prems(3) have ... = FAbs w (S (fmdrop w ϑ) B)
          using substitution-free-absorption by (simp add: fmdom′-notD)
        finally have S ({x ↣ FVar y} ++f ϑ) (FAbs w B) = FAbs w (S (fmdrop w ϑ) B) .
        with FAbs.prems(2) and ‹w = (v_w, α_w)› and FAbs.prems(4) show ?thesis
          using absent-vars-substitution-preservation by auto
      qed
      then show ?thesis
        using is-free-for-absent-var by simp
    next
      case False
      have w ∉ free-vars A
      proof (rule ccontr)
        assume ¬ w ∉ free-vars A
        with False and ‹w ≠ x› have ¬ is-free-for A x (FAbs w B)
          using form-with-free-binder-not-free-for by simp
        with FAbs.prems(1) show False
          by contradiction
      qed
      with ‹is-free-for A y (S ({x ↣ FVar y} ++f fmdrop w ϑ) B)›
      have is-free-for A y (FAbs w (S ({x ↣ FVar y} ++f fmdrop w ϑ) B))
        unfolding ‹w = (v_w, α_w)› using is-free-for-to-abs by iprover
      with ∗ show ?thesis
        by (simp only:)
    qed
  next
    case False
    have ∀ v ∈ fmdom′ ϑ. is-free-for (ϑ $$! v) v B
    proof (rule ballI)
      fix v
      assume v ∈ fmdom′ ϑ
      with FAbs.prems(5) have is-free-for (ϑ $$! v) v (FAbs w B)
        by blast
      moreover from ‹v ∈ fmdom′ ϑ› and ‹w ∉ fmdom′ ϑ› have v ≠ w
        by blast
      ultimately show is-free-for (ϑ $$! v) v B
        unfolding ‹w = (v_w, α_w)› using is-free-for-from-abs by iprover
    qed
    with ‹is-free-for A x B› and ‹y ∉ vars B› and FAbs.prems(3,4)
```

```
      have is-free-for A y (S ({x ↣ FVar y} ++f ϑ) B)
        using FAbs.IH by iprover
      then show ?thesis
      proof (cases x ∉ free-vars B)
        case True
        have y ∉ vars (S ({x ↣ FVar y} ++f ϑ) (FAbs w B))
        proof −
          from False and ‹w = (vw, αw)› and ‹w ≠ x›
          have S ({x ↣ FVar y} ++f ϑ) (FAbs w B) = FAbs w (S ({x ↣ FVar y} ++f ϑ) B)
            by auto
          also from ‹x ∉ free-vars B› and FAbs.prems(3) have ... = FAbs w (S ϑ B)
            using substitution-free-absorption by (simp add: fmdom'-notD)
          finally have S ({x ↣ FVar y} ++f ϑ) (FAbs w B) = FAbs w (S ϑ B) .
          with FAbs.prems(2,4) and ‹w = (vw, αw)› show ?thesis
            using absent-vars-substitution-preservation by auto
        qed
        then show ?thesis
          using is-free-for-absent-var by simp
      next
        case False
        have w ∉ free-vars A
        proof (rule ccontr)
          assume ¬ w ∉ free-vars A
          with False and ‹w ≠ x› have ¬ is-free-for A x (FAbs w B)
            using form-with-free-binder-not-free-for by simp
          with FAbs.prems(1) show False
            by contradiction
        qed
        with ‹is-free-for A y (S ({x ↣ FVar y} ++f ϑ) B)›
        have is-free-for A y (FAbs w (S ({x ↣ FVar y} ++f ϑ) B))
          unfolding ‹w = (vw, αw)› using is-free-for-to-abs by iprover
        moreover from ‹w ∉ fmdom' ϑ› and ‹w ≠ x› and FAbs.prems(3)
        have S ({x ↣ FVar y} ++f ϑ) (FAbs w B) = FAbs w (S ({x ↣ FVar y} ++f ϑ) B)
          using surj-pair[of w] by fastforce
        ultimately show ?thesis
          by (simp only:)
      qed
    qed
  qed
qed
```

The following lemma allows us to fuse a singleton substitution and a simultaneous substitution, as long as the variable of the former does not occur anywhere in the latter:

```
lemma substitution-fusion:
  assumes is-substitution ϑ and is-substitution {v ↣ A}
  and ϑ $$ v = None and ∀ v' ∈ fmdom' ϑ. v ∉ vars (ϑ $$! v')
  shows S {v ↣ A} S ϑ B = S ({v ↣ A} ++f ϑ) B
using assms(1,3,4) proof (induction B arbitrary: ϑ)
  case (FVar v')
```

```
  then show ?case
  proof (cases v' ∉ fmdom' ϑ)
    case True
    then show ?thesis
      using surj-pair[of v'] by fastforce
  next
    case False
    then obtain A' where ϑ $$ v' = Some A'
      by (meson fmlookup-dom'-iff)
    with False and FVar.prems(3) have v ∉ vars A'
      by fastforce
    then have S {v ↣ A} A' = A'
      using free-var-singleton-substitution-neutrality and free-vars-in-all-vars by blast
    from ‹ϑ $$ v' = Some A'› have S {v ↣ A} S ϑ (FVar v') = S {v ↣ A} A'
      using surj-pair[of v'] by fastforce
    also from ‹S {v ↣ A} A' = A'› have ... = A'
      by (simp only:)
    also from ‹ϑ $$ v' = Some A'› and ‹ϑ $$ v = None› have ... = S ({v ↣ A} ++f ϑ) (FVar v')
      using surj-pair[of v'] by fastforce
    finally show ?thesis .
  qed
next
  case (FCon k)
  then show ?case
    using surj-pair[of k] by fastforce
next
  case (FApp C D)
  have S {v ↣ A} S ϑ (C • D) = S {v ↣ A} ((S ϑ C) • (S ϑ D))
    by auto
  also have ... = (S {v ↣ A} S ϑ C) • (S {v ↣ A} S ϑ D)
    by simp
  also from FApp.IH have ... = (S ({v ↣ A} ++f ϑ) C) • (S ({v ↣ A} ++f ϑ) D)
    using FApp.prems(1,2,3) by presburger
  also have ... = S ({v ↣ A} ++f ϑ) (C • D)
    by simp
  finally show ?case .
next
  case (FAbs w C)
  obtain v_w and α where w = (v_w, α)
    by fastforce
  then show ?case
  proof (cases v ≠ w)
    case True
    show ?thesis
    proof (cases w ∉ fmdom' ϑ)
      case True
      then have S {v ↣ A} S ϑ (FAbs w C) = S {v ↣ A} (FAbs w (S ϑ C))
        by (simp add: ‹w = (v_w, α)›)
      also from ‹v ≠ w› have ... = FAbs w (S {v ↣ A} S ϑ C)
```

```
      by (simp add: ‹w = (v_w, α)›)
    also from FAbs.IH have ... = FAbs w (S ({v ↣ A} ++_f ϑ) C)
      using FAbs.prems(1,2,3) by blast
    also from ‹v ≠ w› and True have ... = S ({v ↣ A} ++_f ϑ) (FAbs w C)
      by (simp add: ‹w = (v_w, α)›)
    finally show ?thesis .
  next
    case False
    then have S {v ↣ A} S ϑ (FAbs w C) = S {v ↣ A} (FAbs w (S (fmdrop w ϑ) C))
      by (simp add: ‹w = (v_w, α)›)
    also from ‹v ≠ w› have ... = FAbs w (S {v ↣ A} S (fmdrop w ϑ) C)
      by (simp add: ‹w = (v_w, α)›)
    also have ... = FAbs w (S ({v ↣ A} ++_f fmdrop w ϑ) C)
    proof −
      from ‹is-substitution ϑ› have is-substitution (fmdrop w ϑ)
        by fastforce
      moreover from ‹ϑ $$ v = None› have (fmdrop w ϑ) $$ v = None
        by force
      moreover from FAbs.prems(3) have ∀ v' ∈ fmdom' (fmdrop w ϑ). v ∉ vars ((fmdrop w ϑ) $$!
v')
        by force
      ultimately show ?thesis
        using FAbs.IH by blast
    qed
    also from ‹v ≠ w› have ... = S ({v ↣ A} ++_f ϑ) (FAbs w C)
      by (simp add: ‹w = (v_w, α)› fmdrop-idle')
    finally show ?thesis .
  qed
next
  case False
  then show ?thesis
  proof (cases w ∉ fmdom' ϑ)
    case True
    then have S {v ↣ A} S ϑ (FAbs w C) = S {v ↣ A} (FAbs w (S ϑ C))
      by (simp add: ‹w = (v_w, α)›)
    also from ‹¬ v ≠ w› have ... = FAbs w (S ϑ C)
      using ‹w = (v_w, α)› and singleton-substitution-simps(4) by presburger
    also from ‹¬ v ≠ w› and True have ... = FAbs w (S (fmdrop w ({v ↣ A} ++_f ϑ)) C)
      by (simp add: fmdrop-fmupd-same fmdrop-idle')
    also from ‹¬ v ≠ w› have ... = S ({v ↣ A} ++_f ϑ) (FAbs w C)
      by (simp add: ‹w = (v_w, α)›)
    finally show ?thesis .
  next
    case False
    then have S {v ↣ A} S ϑ (FAbs w C) = S {v ↣ A} (FAbs w (S (fmdrop w ϑ) C))
      by (simp add: ‹w = (v_w, α)›)
    also from ‹¬ v ≠ w› have ... = FAbs w (S (fmdrop w ϑ) C)
      using ‹ϑ $$ v = None› and False by (simp add: fmdom'-notI)
    also from ‹¬ v ≠ w› have ... = FAbs w (S (fmdrop w ({v ↣ A} ++_f ϑ)) C)
```

    by (*simp add*: *fmdrop-fmupd-same*)
  also from ‹¬ $v \neq w$› **and** *False* **and** ‹ϑ $$ v = *None*› **have** ... = **S** ({$v \rightarrowtail A$} $++_f$ ϑ) (*FAbs w C*)
    by (*simp add*: *fmdom'-notI*)
  finally show *?thesis* **.**
 qed
 qed
qed

lemma *updated-substitution-is-substitution*:
 assumes $v \notin$ *fmdom'* ϑ **and** *is-substitution* ($\vartheta(v \rightarrowtail A)$)
 shows *is-substitution* ϑ
unfolding *is-substitution-def* **proof** (*intro ballI*)
 fix v' :: *var*
 obtain x **and** α **where** $v' = (x, \alpha)$
  by *fastforce*
 assume $v' \in$ *fmdom'* ϑ
 with *assms*(*2*)[*unfolded is-substitution-def*] **have** $v' \in$ *fmdom'* ($\vartheta(v \rightarrowtail A)$)
  by *simp*
 with *assms*(*2*)[*unfolded is-substitution-def*] **have** $\vartheta(v \rightarrowtail A)$ $$! $(x, \alpha) \in wffs_\alpha$
  using ‹$v' = (x, \alpha)$› **by** *fastforce*
 with *assms*(*1*) **and** ‹$v' \in$ *fmdom'* ϑ› **and** ‹$v' = (x, \alpha)$› **have** ϑ $$! $(x, \alpha) \in wffs_\alpha$
  by (*metis fmupd-lookup*)
 then show *case* v' *of* $(x, \alpha) \Rightarrow \vartheta$ $$! $(x, \alpha) \in wffs_\alpha$
  by (*simp add*: ‹$v' = (x, \alpha)$›)
qed

definition *is-renaming-substitution* **where**
 [*iff*]: *is-renaming-substitution* $\vartheta \longleftrightarrow$ *is-substitution* $\vartheta \wedge$ *fmpred* (λ- A. ∃ v. A = *FVar* v) ϑ

The following lemma proves that $\mathsf{S}^{x^1_{\alpha_1} \ldots x^n_{\alpha_n}}_{y^1_{\alpha_1} \ldots y^n_{\alpha_n}} B = \mathsf{S}^{x^1_{\alpha_1}}_{y^1_{\alpha_1}} \cdots \mathsf{S}^{x^n_{\alpha_n}}_{y^n_{\alpha_n}} B$ provided that

- $x^1_{\alpha_1} \ \ldots \ x^n_{\alpha_n}$ are distinct variables
- $y^1_{\alpha_1} \ \ldots \ y^n_{\alpha_n}$ are distinct variables, distinct from $x^1_{\alpha_1} \ \ldots \ x^n_{\alpha_n}$ and from all variables in B (i.e., they are fresh variables)

In other words, simultaneously renaming distinct variables with fresh ones is equivalent to renaming each variable one at a time.

lemma *fresh-vars-substitution-unfolding*:
 fixes *ps* :: (*var* × *form*) *list*
 assumes ϑ = *fmap-of-list ps* **and** *is-renaming-substitution* ϑ
 and *distinct* (*map fst ps*) **and** *distinct* (*map snd ps*)
 and *vars* (*fmran'* ϑ) ∩ (*fmdom'* ϑ ∪ *vars* B) = {}
 shows S ϑ B = *foldr* (λ(x, y) C. **S** {$x \rightarrowtail y$} C) *ps* B
using *assms* **proof** (*induction ps arbitrary*: ϑ)
 case *Nil*
 then have ϑ = {$$}

```
    by simp
  then have S ϑ B = B
    using empty-substitution-neutrality by (simp only:)
  with Nil show ?case
    by simp
next
  case (Cons p ps)
  from Cons.prems(1,2) obtain x and y where ϑ $$ (fst p) = Some (FVar y) and p = (x, FVar y)
    using surj-pair[of p] by fastforce
  let ?ϑ' = fmap-of-list ps
  from Cons.prems(1) and ‹p = (x, FVar y)› have ϑ = fmupd x (FVar y) ?ϑ'
    by simp
  moreover from Cons.prems(3) and ‹p = (x, FVar y)› have x ∉ fmdom' ?ϑ'
    by simp
  ultimately have ϑ = {x ↦ FVar y} ++f ?ϑ'
    using fmap-singleton-comm by fastforce
  with Cons.prems(2) and ‹x ∉ fmdom' ?ϑ'› have is-renaming-substitution ?ϑ'
    unfolding is-renaming-substitution-def and ‹ϑ = fmupd x (FVar y) ?ϑ'›
    using updated-substitution-is-substitution by (metis fmdiff-fmupd fmdom'-notD fmpred-filter)
  from Cons.prems(2) and ‹ϑ = fmupd x (FVar y) ?ϑ'› have is-renaming-substitution {x ↦ FVar
y}
    by auto
  have
    foldr (λ(x, y) C. S {x ↦ y} C) (p # ps) B
    =
    S {x ↦ FVar y} (foldr (λ(x, y) C. S {x ↦ y} C) ps B)
    by (simp add: ‹p = (x, FVar y)›)
  also have ... = S {x ↦ FVar y} S ?ϑ' B
  proof −
    from Cons.prems(3,4) have distinct (map fst ps) and distinct (map snd ps)
      by fastforce+
    moreover have vars (fmran' ?ϑ') ∩ (fmdom' ?ϑ' ∪ vars B) = {}
    proof −
      have vars (fmran' ϑ) = vars ({FVar y} ∪ fmran' ?ϑ')
       using ‹ϑ = fmupd x (FVar y) ?ϑ'› and ‹x ∉ fmdom' ?ϑ'› by (metis fmdom'-notD fmran'-fmupd)
      then have vars (fmran' ϑ) = {y} ∪ vars (fmran' ?ϑ')
        using singleton-form-set-vars by auto
      moreover have fmdom' ϑ = {x} ∪ fmdom' ?ϑ'
        by (simp add: ‹ϑ = {x ↦ FVar y} ++f ?ϑ'›)
      ultimately show ?thesis
        using Cons.prems(5) by auto
    qed
    ultimately show ?thesis
      using Cons.IH and ‹is-renaming-substitution ?ϑ'› by simp
  qed
  also have ... = S ({x ↦ FVar y} ++f ?ϑ') B
  proof (rule substitution-fusion)
    show is-substitution ?ϑ'
      using ‹is-renaming-substitution ?ϑ'› by simp
```

```
    show is-substitution {x ↣ FVar y}
      using ‹is-renaming-substitution {x ↣ FVar y}› by simp
    show ?ϑ' $$ x = None
      using ‹x ∉ fmdom' ?ϑ'› by blast
    show ∀ v' ∈ fmdom' ?ϑ'. x ∉ vars (?ϑ' $$! v')
    proof −
      have x ∈ fmdom' ϑ
        using ‹ϑ = {x ↣ FVar y} ++f ?ϑ'› by simp
      then have x ∉ vars (fmran' ϑ)
        using Cons.prems(5) by blast
      moreover have {?ϑ' $$! v' | v'. v' ∈ fmdom' ?ϑ'} ⊆ fmran' ϑ
        unfolding ‹ϑ = ?ϑ'(x ↣ FVar y)› using ‹?ϑ' $$ x = None›
        by (auto simp add: fmlookup-of-list fmlookup-dom'-iff fmran'I weak-map-of-SomeI)
      ultimately show ?thesis
        by force
    qed
  qed
  also from ‹ϑ = {x ↣ FVar y} ++f ?ϑ'› have ... = S ϑ B
    by (simp only:)
  finally show ?case ..
qed

lemma free-vars-agreement-substitution-equality:
  assumes fmdom' ϑ = fmdom' ϑ'
  and ∀ v ∈ free-vars A ∩ fmdom' ϑ. ϑ $$! v = ϑ' $$! v
  shows S ϑ A = S ϑ' A
using assms proof (induction A arbitrary: ϑ ϑ')
  case (FVar v)
  have free-vars (FVar v) = {v}
    using surj-pair[of v] by fastforce
  with FVar have ϑ $$! v = ϑ' $$! v
    by force
  with FVar.prems(1) show ?case
    using surj-pair[of v] by (metis fmdom'-notD fmdom'-notI option.collapse substitute.simps(1))
next
  case FCon
  then show ?case
    by (metis prod.exhaust-sel substitute.simps(2))
next
  case (FApp B C)
  have S ϑ (B • C) = (S ϑ B) • (S ϑ C)
    by simp
  also have ... = (S ϑ' B) • (S ϑ' C)
  proof −
    have ∀ v ∈ free-vars B ∩ fmdom' ϑ. ϑ $$! v = ϑ' $$! v
    and ∀ v ∈ free-vars C ∩ fmdom' ϑ. ϑ $$! v = ϑ' $$! v
      using FApp.prems(2) by auto
    with FApp.IH(1,2) and FApp.prems(1) show ?thesis
      by blast
```

```
  qed
  finally show ?case
    by simp
next
  case (FAbs w B)
  from FAbs.prems(1,2) have ∗: ∀ v ∈ free-vars B − {w} ∩ fmdom′ ϑ. ϑ $$! v = ϑ′ $$! v
    using surj-pair[of w] by fastforce
  show ?case
  proof (cases w ∈ fmdom′ ϑ)
    case True
    then have S ϑ (FAbs w B) = FAbs w (S (fmdrop w ϑ) B)
      using surj-pair[of w] by fastforce
    also have ... = FAbs w (S (fmdrop w ϑ′) B)
    proof −
      from ∗ have ∀ v ∈ free-vars B ∩ fmdom′ (fmdrop w ϑ). (fmdrop w ϑ) $$! v = (fmdrop w ϑ′) $$!
v
        by simp
      moreover have fmdom′ (fmdrop w ϑ) = fmdom′ (fmdrop w ϑ′)
        by (simp add: FAbs.prems(1))
      ultimately show ?thesis
        using FAbs.IH by blast
    qed
    finally show ?thesis
      using FAbs.prems(1) and True and surj-pair[of w] by fastforce
  next
    case False
    then have S ϑ (FAbs w B) = FAbs w (S ϑ B)
      using surj-pair[of w] by fastforce
    also have ... = FAbs w (S ϑ′ B)
    proof −
      from ∗ have ∀ v ∈ free-vars B ∩ fmdom′ ϑ. ϑ $$! v = ϑ′ $$! v
        using False by blast
      with FAbs.prems(1) show ?thesis
        using FAbs.IH by blast
    qed
    finally show ?thesis
      using FAbs.prems(1) and False and surj-pair[of w] by fastforce
  qed
qed
```

The following lemma proves that $\S\ {}^{x_\alpha}_{A_\alpha}\ \S\ {}^{x^1_{\alpha_1}\ \ldots\ x^n_{\alpha_n}}_{A^1_{\alpha_1}\ \ldots\ A^n_{\alpha_n}} B = \S\ {}^{x_\alpha\quad x^1_{\alpha_1}\quad \ldots\quad x^n_{\alpha_n}}_{A_\alpha\ \S\ {}^{x_\alpha}_{A_\alpha} A^1_{\alpha_1}\ \ldots\ \S\ {}^{x_\alpha}_{A_\alpha} A^n_{\alpha_n}} B$ provided that x_α is distinct from $x^1_{\alpha_1}, \ldots, x^n_{\alpha_n}$ and $A^i_{\alpha_i}$ is free for $x^i_{\alpha_i}$ in B:

```
lemma substitution-consolidation:
  assumes v ∉ fmdom′ ϑ
  and ∀ v′ ∈ fmdom′ ϑ. is-free-for (ϑ $$! v′) v′ B
  shows S {v ⇀ A} S ϑ B = S ({v ⇀ A} ++f fmmap (λA′. S {v ⇀ A} A′) ϑ) B
using assms proof (induction B arbitrary: ϑ)
  case (FApp B C)
```

```
  have ∀ v' ∈ fmdom' ϑ. is-free-for (ϑ $$! v') v' B ∧ is-free-for (ϑ $$! v') v' C
  proof
    fix v'
    assume v' ∈ fmdom' ϑ
    with FApp.prems(2) have is-free-for (ϑ $$! v') v' (B ▪ C)
      by blast
    then show is-free-for (ϑ $$! v') v' B ∧ is-free-for (ϑ $$! v') v' C
      using is-free-for-from-app by iprover
  qed
  with FApp.IH and FApp.prems(1) show ?case
    by simp
next
  case (FAbs w B)
  let ?ϑ' = fmmap (λA'. S {v ↣ A} A') ϑ
  show ?case
  proof (cases w ∈ fmdom' ϑ)
    case True
    then have w ∈ fmdom' ?ϑ'
      by simp
    with True and FAbs.prems have v ≠ w
      by blast
    from True have S {v ↣ A} S ϑ (FAbs w B) = S {v ↣ A} (FAbs w (S (fmdrop w ϑ) B))
      using surj-pair[of w] by fastforce
    also from ‹v ≠ w› have ... = FAbs w (S {v ↣ A} S (fmdrop w ϑ) B)
      using surj-pair[of w] by fastforce
    also have ... = FAbs w (S (fmdrop w ({v ↣ A} ++f ?ϑ')) B)
    proof −
      obtain x_w and α_w where w = (x_w, α_w)
        by fastforce
      have ∀ v' ∈ fmdom' (fmdrop w ϑ). is-free-for ((fmdrop w ϑ) $$! v') v' B
      proof
        fix v'
        assume v' ∈ fmdom' (fmdrop w ϑ)
        with FAbs.prems(2) have is-free-for (ϑ $$! v') v' (FAbs w B)
          by auto
        with ‹w = (x_w, α_w)› and ‹v' ∈ fmdom' (fmdrop w ϑ)›
        have is-free-for (ϑ $$! v') v' (λx_w α_w. B) and v' ≠ (x_w, α_w)
          by auto
        then have is-free-for (ϑ $$! v') v' B
          using is-free-for-from-abs by presburger
        with ‹v' ≠ (x_w, α_w)› and ‹w = (x_w, α_w)› show is-free-for (fmdrop w ϑ $$! v') v' B
          by simp
      qed
      moreover have v ∉ fmdom' (fmdrop w ϑ)
        by (simp add: FAbs.prems(1))
      ultimately show ?thesis
        using FAbs.IH and ‹v ≠ w› by (simp add: fmdrop-fmupd)
    qed
    finally show ?thesis
```

```
    using ‹w ∈ fmdom' ?ϑ'› and surj-pair[of w] by fastforce
  next
    case False
    then have w ∉ fmdom' ?ϑ'
      by simp
    from FAbs.prems have v ∉ fmdom' ?ϑ'
      by simp
    from False have *: S {v ⇀ A} S ϑ (FAbs w B) = S {v ⇀ A} (FAbs w (S ϑ B))
      using surj-pair[of w] by fastforce
    then show ?thesis
    proof (cases v ≠ w)
      case True
      then have S {v ⇀ A} (FAbs w (S ϑ B)) = FAbs w (S {v ⇀ A} (S ϑ B))
        using surj-pair[of w] by fastforce
      also have ... = FAbs w (S ({v ⇀ A} ++f ?ϑ') B)
      proof −
        obtain x_w and α_w where w = (x_w, α_w)
          by fastforce
        have ∀ v' ∈ fmdom' ϑ. is-free-for (ϑ $$! v') v' B
        proof
          fix v'
          assume v' ∈ fmdom' ϑ
          with FAbs.prems(2) have is-free-for (ϑ $$! v') v' (FAbs w B)
            by auto
          with ‹w = (x_w, α_w)› and ‹v' ∈ fmdom' ϑ› and False
          have is-free-for (ϑ $$! v') v' (λx_w α_w. B) and v' ≠ (x_w, α_w)
            by fastforce+
          then have is-free-for (ϑ $$! v') v' B
            using is-free-for-from-abs by presburger
          with ‹v' ≠ (x_w, α_w)› and ‹w = (x_w, α_w)› show is-free-for (ϑ $$! v') v' B
            by simp
        qed
        with FAbs.IH show ?thesis
          using FAbs.prems(1) by blast
      qed
      finally show ?thesis
      proof −
        assume
          S {v ⇀ A} (FAbs w (S ϑ B)) = FAbs w (S ({v ⇀ A} ++f fmmap (substitute {v ⇀ A}) ϑ)
B)
        moreover have w ∉ fmdom' ({v ⇀ A} ++f fmmap (substitute {v ⇀ A}) ϑ)
          using False and True by auto
        ultimately show ?thesis
          using * and surj-pair[of w] by fastforce
      qed
    next
      case False
      then have v ∉ free-vars (FAbs w (S ϑ B))
        using surj-pair[of w] by fastforce
```

```
      then have **: S {v ↣ A} (FAbs w (S ϑ B)) = FAbs w (S ϑ B)
        using free-var-singleton-substitution-neutrality by blast
      also have ... = FAbs w (S ?ϑ' B)
      proof −
        {
          fix v'
          assume v' ∈ fmdom' ϑ
          with FAbs.prems(1) have v' ≠ v
            by blast
          assume v ∈ free-vars (ϑ $$! v') and v' ∈ free-vars B
          with ‹v' ≠ v› have ¬ is-free-for (ϑ $$! v') v' (FAbs v B)
            using form-with-free-binder-not-free-for by blast
          with FAbs.prems(2) and ‹v' ∈ fmdom' ϑ› and False have False
            by blast
        }
        then have ∀ v' ∈ fmdom' ϑ. v ∉ free-vars (ϑ $$! v') ∨ v' ∉ free-vars B
          by blast
        then have ∀ v' ∈ fmdom' ϑ. v' ∈ free-vars B ⟶ S {v ↣ A} (ϑ $$! v') = ϑ $$! v'
          using free-var-singleton-substitution-neutrality by blast
        then have ∀ v' ∈ free-vars B. ϑ $$! v' = ?ϑ' $$! v'
          by (metis fmdom'-map fmdom'-notD fmdom'-notI fmlookup-map option.map-sel)
        then have S ϑ B = S ?ϑ' B
          using free-vars-agreement-substitution-equality by (metis IntD1 fmdom'-map)
        then show ?thesis
          by simp
      qed
      also from False and FAbs.prems(1) have ... = FAbs w (S (fmdrop w ({v ↣ A} ++f ?ϑ')) B)
        by (simp add: fmdrop-fmupd-same fmdrop-idle')
      also from False have ... = S ({v ↣ A} ++f ?ϑ') (FAbs w B)
        using surj-pair[of w] by fastforce
      finally show ?thesis
        using * and ** by (simp only:)
    qed
  qed
qed force+

lemma vars-range-substitution:
  assumes is-substitution ϑ
  and v ∉ vars (fmran' ϑ)
  shows v ∉ vars (fmran' (fmdrop w ϑ))
using assms proof (induction ϑ)
  case fmempty
  then show ?case
    by simp
next
  case (fmupd v' A ϑ)
  from fmdom'-notI[OF fmupd.hyps] and fmupd.prems(1) have is-substitution ϑ
    by (rule updated-substitution-is-substitution)
  moreover from fmupd.prems(2) and fmupd.hyps have v ∉ vars (fmran' ϑ)
```

```
    by simp
  ultimately have v ∉ vars (fmran' (fmdrop w ϑ))
    by (rule fmupd.IH)
  with fmupd.hyps and fmupd.prems(2) show ?case
    by (simp add: fmdrop-fmupd)
qed

lemma excluded-var-from-substitution:
  assumes is-substitution ϑ
  and v ∉ fmdom' ϑ
  and v ∉ vars (fmran' ϑ)
  and v ∉ vars A
  shows v ∉ vars (S ϑ A)
using assms proof (induction A arbitrary: ϑ)
  case (FVar v')
  then show ?case
  proof (cases v' ∈ fmdom' ϑ)
    case True
    then have ϑ $$! v' ∈ fmran' ϑ
      by (simp add: fmlookup-dom'-iff fmran'I)
    with FVar(3) have v ∉ vars (ϑ $$! v')
      by simp
    with True show ?thesis
      using surj-pair[of v'] and fmdom'-notI by force
  next
    case False
    with FVar.prems(4) show ?thesis
      using surj-pair[of v'] by force
  qed
next
  case (FCon k)
  then show ?case
    using surj-pair[of k] by force
next
  case (FApp B C)
  then show ?case
    by auto
next
  case (FAbs w B)
  have v ∉ vars B and v ≠ w
    using surj-pair[of w] and FAbs.prems(4) by fastforce+
  then show ?case
  proof (cases w ∉ fmdom' ϑ)
    case True
    then have S ϑ (FAbs w B) = FAbs w (S ϑ B)
      using surj-pair[of w] by fastforce
    moreover from FAbs.IH have v ∉ vars (S ϑ B)
      using FAbs.prems(1−3) and ‹v ∉ vars B› by blast
    ultimately show ?thesis
```

```
    using ‹v ≠ w› and surj-pair[of w] by fastforce
  next
    case False
    then have S ϑ (FAbs w B) = FAbs w (S (fmdrop w ϑ) B)
      using surj-pair[of w] by fastforce
    moreover have v ∉ vars (S (fmdrop w ϑ) B)
    proof −
      from FAbs.prems(1) have is-substitution (fmdrop w ϑ)
        by fastforce
      moreover from FAbs.prems(2) have v ∉ fmdom′ (fmdrop w ϑ)
        by simp
      moreover from FAbs.prems(1,3) have v ∉ vars (fmran′ (fmdrop w ϑ))
        by (fact vars-range-substitution)
      ultimately show ?thesis
        using FAbs.IH and ‹v ∉ vars B› by simp
    qed
    ultimately show ?thesis
      using ‹v ≠ w› and surj-pair[of w] by fastforce
  qed
qed
```

2.15 Renaming of bound variables

```
fun rename-bound-var :: var ⇒ nat ⇒ form ⇒ form where
  rename-bound-var v y (x_α) = x_α
| rename-bound-var v y (⦇c⦈_α) = ⦇c⦈_α
| rename-bound-var v y (B ▪ C) = rename-bound-var v y B ▪ rename-bound-var v y C
| rename-bound-var v y (λx_α. B) =
    (
      if (x, α) = v then
        λy_α. S {(x, α) ⟼ y_α} (rename-bound-var v y B)
      else
        λx_α. (rename-bound-var v y B)
    )

lemma rename-bound-var-preserves-typing:
  assumes A ∈ wffs_α
  shows rename-bound-var (y, γ) z A ∈ wffs_α
using assms proof (induction A)
  case (abs-is-wff β A δ x)
  then show ?case
  proof (cases (x, δ) = (y, γ))
    case True
    from abs-is-wff.IH have S {(y, γ) ⟼ z_γ} (rename-bound-var (y, γ) z A) ∈ wffs_β
      using substitution-preserves-typing by (simp add: wffs-of-type-intros(1))
    then have λz_γ. S {(y, γ) ⟼ z_γ} (rename-bound-var (y, γ) z A) ∈ wffs_{γ→β}
      by blast
    with True show ?thesis
      by simp
```

```
  next
    case False
    from abs-is-wff.IH have λx_δ. rename-bound-var (y, γ) z A ∈ wffs_{δ→β}
      by blast
    with False show ?thesis
      by auto
  qed
qed auto

lemma old-bound-var-not-free-in-abs-after-renaming:
  assumes A ∈ wffs_α
  and z_γ ≠ y_γ
  and (z, γ) ∉ vars A
  shows (y, γ) ∉ free-vars (rename-bound-var (y, γ) z (λy_γ. A))
  using assms and free-var-in-renaming-substitution by (induction A) auto

lemma rename-bound-var-free-vars:
  assumes A ∈ wffs_α
  and z_γ ≠ y_γ
  and (z, γ) ∉ vars A
  shows (z, γ) ∉ free-vars (rename-bound-var (y, γ) z A)
  using assms by (induction A) auto

lemma old-bound-var-not-free-after-renaming:
  assumes A ∈ wffs_α
  and z_γ ≠ y_γ
  and (z, γ) ∉ vars A
  and (y, γ) ∉ free-vars A
  shows (y, γ) ∉ free-vars (rename-bound-var (y, γ) z A)
using assms proof induction
  case (abs-is-wff β A α x)
  then show ?case
  proof (cases (x, α) = (y, γ))
    case True
    with abs-is-wff.hyps and abs-is-wff.prems(2) show ?thesis
      using old-bound-var-not-free-in-abs-after-renaming by auto
  next
    case False
    with abs-is-wff.prems(2,3) and assms(2) show ?thesis
      using abs-is-wff.IH by force
  qed
qed fastforce+

lemma old-bound-var-not-ocurring-after-renaming:
  assumes A ∈ wffs_α
  and z_γ ≠ y_γ
  shows ¬ occurs-at (y, γ) p (S {(y, γ) ⇀ z_γ} (rename-bound-var (y, γ) z A))
using assms(1) proof (induction A arbitrary: p)
  case (var-is-wff α x)
```

```
  from assms(2) show ?case
    using subform-size-decrease by (cases (x, α) = (y, γ)) fastforce+
next
  case (con-is-wff α c)
  then show ?case
    using occurs-at-alt-def(2) by auto
next
  case (app-is-wff α β A B)
  then show ?case
  proof (cases p)
    case (Cons d p')
    then show ?thesis
      by (cases d) (use app-is-wff.IH in auto)
  qed simp
next
  case (abs-is-wff β A α x)
  then show ?case
  proof (cases p)
    case (Cons d p')
    then show ?thesis
    proof (cases d)
      case Left
      have ∗: ¬ occurs-at (y, γ) p (λx_α. S {(y, γ) ⇀ z_γ} (rename-bound-var (y, γ) z A))
        for x and α
        using Left and Cons and abs-is-wff.IH by simp
      then show ?thesis
      proof (cases (x, α) = (y, γ))
        case True
        with assms(2) have
          S {(y, γ) ⇀ z_γ} (rename-bound-var (y, γ) z (λx_α. A))
          =
          λz_γ. S {(y, γ) ⇀ z_γ} (rename-bound-var (y, γ) z A)
          using free-var-in-renaming-substitution and free-var-singleton-substitution-neutrality
          by simp
        moreover have ¬ occurs-at (y, γ) p (λz_γ. S {(y, γ) ⇀ z_γ} (rename-bound-var (y, γ) z A))
          using Left and Cons and ∗ by simp
        ultimately show ?thesis
          by simp
      next
        case False
        with assms(2) have
          S {(y, γ) ⇀ z_γ} (rename-bound-var (y, γ) z (λx_α. A))
          =
          λx_α. S {(y, γ) ⇀ z_γ} (rename-bound-var (y, γ) z A)
          by simp
        moreover have ¬ occurs-at (y, γ) p (λx_α. S {(y, γ) ⇀ z_γ} (rename-bound-var (y, γ) z A))
          using Left and Cons and ∗ by simp
        ultimately show ?thesis
          by simp
```

```
    qed
  qed (simp add: Cons)
 qed simp
qed
```

The following lemma states that the result of *rename-bound-var* does not contain bound occurrences of the renamed variable:

```
lemma rename-bound-var-not-bound-occurrences:
 assumes A ∈ wffs_α
 and z_γ ≠ y_γ
 and (z, γ) ∉ vars A
 and occurs-at (y, γ) p (rename-bound-var (y, γ) z A)
 shows ¬ in-scope-of-abs (z, γ) p (rename-bound-var (y, γ) z A)
using assms(1,3,4) proof (induction arbitrary: p)
 case (var-is-wff α x)
 then show ?case
  by (simp add: subforms-from-var(2))
next
 case (con-is-wff α c)
 then show ?case
  using occurs-at-alt-def(2) by auto
next
 case (app-is-wff α β B C)
 from app-is-wff.prems(1) have (z, γ) ∉ vars B and (z, γ) ∉ vars C
  by simp-all
 from app-is-wff.prems(2)
 have occurs-at (y, γ) p (rename-bound-var (y, γ) z B • rename-bound-var (y, γ) z C)
  by simp
 then consider
  (a) ∃p'. p = « # p' ∧ occurs-at (y, γ) p' (rename-bound-var (y, γ) z B)
| (b) ∃p'. p = » # p' ∧ occurs-at (y, γ) p' (rename-bound-var (y, γ) z C)
  using subforms-from-app by force
 then show ?case
 proof cases
  case a
  then obtain p' where p = « # p' and occurs-at (y, γ) p' (rename-bound-var (y, γ) z B)
   by blast
  then have ¬ in-scope-of-abs (z, γ) p' (rename-bound-var (y, γ) z B)
   using app-is-wff.IH(1)[OF ‹(z, γ) ∉ vars B›] by blast
  then have ¬ in-scope-of-abs (z, γ) p (rename-bound-var (y, γ) z (B • C)) for C
   using ‹p = « # p'› and in-scope-of-abs-in-left-app by simp
  then show ?thesis
   by blast
 next
  case b
  then obtain p' where p = » # p' and occurs-at (y, γ) p' (rename-bound-var (y, γ) z C)
   by blast
  then have ¬ in-scope-of-abs (z, γ) p' (rename-bound-var (y, γ) z C)
   using app-is-wff.IH(2)[OF ‹(z, γ) ∉ vars C›] by blast
```

```
    then have ¬ in-scope-of-abs (z, γ) p (rename-bound-var (y, γ) z (B ∙ C)) for B
      using ‹p = » # p'› and in-scope-of-abs-in-right-app by simp
    then show ?thesis
      by blast
  qed
next
  case (abs-is-wff β A α x)
  from abs-is-wff.prems(1) have (z, γ) ∉ vars A and (z, γ) ≠ (x, α)
    by fastforce+
  then show ?case
  proof (cases (y, γ) = (x, α))
    case True
    then have occurs-at (y, γ) p (λz_γ. S {(y, γ) ⇀ z_γ} (rename-bound-var (y, γ) z A))
      using abs-is-wff.prems(2) by simp
    moreover have ¬ occurs-at (y, γ) p (λz_γ. S {(y, γ) ⇀ z_γ} (rename-bound-var (y, γ) z A))
     using old-bound-var-not-ocurring-after-renaming[OF abs-is-wff.hyps assms(2)] and subforms-from-abs
      by fastforce
    ultimately show ?thesis
      by contradiction
  next
    case False
    then have ∗: rename-bound-var (y, γ) z (λx_α. A) = λx_α. rename-bound-var (y, γ) z A
      by auto
    with abs-is-wff.prems(2) have occurs-at (y, γ) p (λx_α. rename-bound-var (y, γ) z A)
      by auto
    then obtain p' where p = « # p' and occurs-at (y, γ) p' (rename-bound-var (y, γ) z A)
      using subforms-from-abs by fastforce
    then have ¬ in-scope-of-abs (z, γ) p' (rename-bound-var (y, γ) z A)
      using abs-is-wff.IH[OF ‹(z, γ) ∉ vars A›] by blast
    then have ¬ in-scope-of-abs (z, γ) (« # p') (λx_α. rename-bound-var (y, γ) z A)
      using ‹p = « # p'› and in-scope-of-abs-in-abs and ‹(z, γ) ≠ (x, α)› by auto
    then show ?thesis
      using ∗ and ‹p = « # p'› by simp
  qed
qed

lemma is-free-for-in-rename-bound-var:
  assumes A ∈ wffs_α
  and z_γ ≠ y_γ
  and (z, γ) ∉ vars A
  shows is-free-for (z_γ) (y, γ) (rename-bound-var (y, γ) z A)
proof (rule ccontr)
  assume ¬ is-free-for (z_γ) (y, γ) (rename-bound-var (y, γ) z A)
  then obtain p
    where is-free-at (y, γ) p (rename-bound-var (y, γ) z A)
    and in-scope-of-abs (z, γ) p (rename-bound-var (y, γ) z A)
    by force
  then show False
    using rename-bound-var-not-bound-occurrences[OF assms] by fastforce
```

qed

lemma *renaming-substitution-preserves-bound-vars*:
shows *bound-vars* (**S** $\{(y, \gamma) \rightarrowtail z_\gamma\}$ *A*) = *bound-vars A*
proof (*induction A*)
case (*FAbs v A*)
then show *?case*
using *singleton-substitution-simps*(*4*) **and** *surj-pair*[*of v*]
by (*cases v* = (*y*, γ)) (*presburger*, *force*)
qed *force*+

lemma *rename-bound-var-bound-vars*:
assumes $A \in wffs_\alpha$
and $z_\gamma \neq y_\gamma$
shows $(y, \gamma) \notin$ *bound-vars* (*rename-bound-var* (*y*, γ) *z A*)
using *assms* **and** *renaming-substitution-preserves-bound-vars* **by** (*induction A*) *auto*

lemma *old-var-not-free-not-occurring-after-rename*:
assumes $A \in wffs_\alpha$
and $z_\gamma \neq y_\gamma$
and $(y, \gamma) \notin$ *free-vars A*
and $(z, \gamma) \notin$ *vars A*
shows $(y, \gamma) \notin$ *vars* (*rename-bound-var* (*y*, γ) *z A*)
using *assms* **and** *rename-bound-var-bound-vars*[*OF assms*(*1*,*2*)]
and *old-bound-var-not-free-after-renaming* **and** *vars-is-free-and-bound-vars* **by** *blast*

end

3 Boolean Algebra

theory *Boolean-Algebra*
imports
ZFC-in-HOL.ZFC-Typeclasses
begin

This theory contains an embedding of two-valued boolean algebra into *V*.

hide-const (**open**) *List.set*

definition *bool-to-V* :: *bool* ⇒ *V* **where**
bool-to-V = (*SOME f*. *inj f*)

lemma *bool-to-V-injectivity* [*simp*]:
shows *inj bool-to-V*
unfolding *bool-to-V-def* **by** (*fact someI-ex*[*OF embeddable-class.ex-inj*])

definition *bool-from-V* :: *V* ⇒ *bool* **where**
[*simp*]: *bool-from-V* = *inv bool-to-V*

definition *top* :: *V* (**T**) **where**

[*simp*]: **T** = *bool-to-V True*

definition *bottom* :: *V* (**F**) **where**
[*simp*]: **F** = *bool-to-V False*

definition *two-valued-boolean-algebra-universe* :: *V* (𝔹) **where**
[*simp*]: 𝔹 = *set* {**T**, **F**}

definition *negation* :: *V* ⇒ *V* (∼ - [*141*] *141*) **where**
[*simp*]: ∼ *p* = *bool-to-V* (¬ *bool-from-V p*)

definition *conjunction* :: *V* ⇒ *V* ⇒ *V* (**infixr** ∧ *136*) **where**
[*simp*]: *p* ∧ *q* = *bool-to-V* (*bool-from-V p* ∧ *bool-from-V q*)

definition *disjunction* :: *V* ⇒ *V* ⇒ *V* (**infixr** ∨ *131*) **where**
[*simp*]: *p* ∨ *q* = ∼ (∼ *p* ∧ ∼ *q*)

definition *implication* :: *V* ⇒ *V* ⇒ *V* (**infixr** ⊃ *121*) **where**
[*simp*]: *p* ⊃ *q* = ∼ *p* ∨ *q*

definition *iff* :: *V* ⇒ *V* ⇒ *V* (**infixl** ≡ *150*) **where**
[*simp*]: *p* ≡ *q* = (*p* ⊃ *q*) ∧ (*q* ⊃ *p*)

lemma *boolean-algebra-simps* [*simp*]:
assumes *p* ∈ *elts* 𝔹 **and** *q* ∈ *elts* 𝔹 **and** *r* ∈ *elts* 𝔹
shows ∼ ∼ *p* = *p*
and ((∼ *p*) ≡ (∼ *q*)) = (*p* ≡ *q*)
and ∼ (*p* ≡ *q*) = (*p* ≡ (∼ *q*))
and (*p* ∨ ∼ *p*) = **T**
and (∼ *p* ∨ *p*) = **T**
and (*p* ≡ *p*) = **T**
and (∼ *p*) ≠ *p*
and *p* ≠ (∼ *p*)
and (**T** ≡ *p*) = *p*
and (*p* ≡ **T**) = *p*
and (**F** ≡ *p*) = (∼ *p*)
and (*p* ≡ **F**) = (∼ *p*)
and (**T** ⊃ *p*) = *p*
and (**F** ⊃ *p*) = **T**
and (*p* ⊃ **T**) = **T**
and (*p* ⊃ *p*) = **T**
and (*p* ⊃ **F**) = (∼ *p*)
and (*p* ⊃ ∼ *p*) = (∼ *p*)
and (*p* ∧ **T**) = *p*
and (**T** ∧ *p*) = *p*
and (*p* ∧ **F**) = **F**
and (**F** ∧ *p*) = **F**
and (*p* ∧ *p*) = *p*
and (*p* ∧ (*p* ∧ *q*)) = (*p* ∧ *q*)

and $(p \wedge \sim p) = \mathbf{F}$
and $(\sim p \wedge p) = \mathbf{F}$
and $(p \vee \mathbf{T}) = \mathbf{T}$
and $(\mathbf{T} \vee p) = \mathbf{T}$
and $(p \vee \mathbf{F}) = p$
and $(\mathbf{F} \vee p) = p$
and $(p \vee p) = p$
and $(p \vee (p \vee q)) = (p \vee q)$
and $p \wedge q = q \wedge p$
and $p \wedge (q \wedge r) = q \wedge (p \wedge r)$
and $p \vee q = q \vee p$
and $p \vee (q \vee r) = q \vee (p \vee r)$
and $(p \vee q) \vee r = p \vee (q \vee r)$
and $p \wedge (q \vee r) = p \wedge q \vee p \wedge r$
and $(p \vee q) \wedge r = p \wedge r \vee q \wedge r$
and $p \vee (q \wedge r) = (p \vee q) \wedge (p \vee r)$
and $(p \wedge q) \vee r = (p \vee r) \wedge (q \vee r)$
and $(p \supset (q \wedge r)) = ((p \supset q) \wedge (p \supset r))$
and $((p \wedge q) \supset r) = (p \supset (q \supset r))$
and $((p \vee q) \supset r) = ((p \supset r) \wedge (q \supset r))$
and $((p \supset q) \vee r) = (p \supset q \vee r)$
and $(q \vee (p \supset r)) = (p \supset q \vee r)$
and $\sim (p \vee q) = \sim p \wedge \sim q$
and $\sim (p \wedge q) = \sim p \vee \sim q$
and $\sim (p \supset q) = p \wedge \sim q$
and $\sim p \vee q = (p \supset q)$
and $p \vee \sim q = (q \supset p)$
and $(p \supset q) = (\sim p) \vee q$
and $p \vee q = \sim p \supset q$
and $(p \equiv q) = (p \supset q) \wedge (q \supset p)$
and $(p \supset q) \wedge (\sim p \supset q) = q$
and $p = \mathbf{T} \Longrightarrow \neg\, (p = \mathbf{F})$
and $p = \mathbf{F} \Longrightarrow \neg\, (p = \mathbf{T})$
and $p = \mathbf{T} \vee p = \mathbf{F}$
using *assms* **by** (*auto simp add*: *inj-eq*)

lemma *tv-cases* [*consumes 1*, *case-names top bottom*, *cases type*: V]:
assumes $p \in$ *elts* $\mathbb{B}$
and $p = \mathbf{T} \Longrightarrow P$
and $p = \mathbf{F} \Longrightarrow P$
shows P
using *assms* **by** *auto*

end

4 Propositional Well-Formed Formulas

theory *Propositional-Wff*
imports

Syntax
Boolean-Algebra
begin

4.1 Syntax

inductive-set *pwffs* :: *form set* **where**
T-pwff: $T_o \in pwffs$
| *F-pwff*: $F_o \in pwffs$
| *var-pwff*: $p_o \in pwffs$
| *neg-pwff*: $\sim^{\mathfrak{Q}} A \in pwffs$ **if** $A \in pwffs$
| *conj-pwff*: $A \wedge^{\mathfrak{Q}} B \in pwffs$ **if** $A \in pwffs$ **and** $B \in pwffs$
| *disj-pwff*: $A \vee^{\mathfrak{Q}} B \in pwffs$ **if** $A \in pwffs$ **and** $B \in pwffs$
| *imp-pwff*: $A \supset^{\mathfrak{Q}} B \in pwffs$ **if** $A \in pwffs$ **and** $B \in pwffs$
| *eqv-pwff*: $A \equiv^{\mathfrak{Q}} B \in pwffs$ **if** $A \in pwffs$ **and** $B \in pwffs$

lemmas [*intro*!] = *pwffs.intros*

lemma *pwffs-distinctnesses* [*induct-simp*]:
shows $T_o \neq F_o$
and $T_o \neq p_o$
and $T_o \neq \sim^{\mathfrak{Q}} A$
and $T_o \neq A \wedge^{\mathfrak{Q}} B$
and $T_o \neq A \vee^{\mathfrak{Q}} B$
and $T_o \neq A \supset^{\mathfrak{Q}} B$
and $T_o \neq A \equiv^{\mathfrak{Q}} B$
and $F_o \neq p_o$
and $F_o \neq \sim^{\mathfrak{Q}} A$
and $F_o \neq A \wedge^{\mathfrak{Q}} B$
and $F_o \neq A \vee^{\mathfrak{Q}} B$
and $F_o \neq A \supset^{\mathfrak{Q}} B$
and $F_o \neq A \equiv^{\mathfrak{Q}} B$
and $p_o \neq \sim^{\mathfrak{Q}} A$
and $p_o \neq A \wedge^{\mathfrak{Q}} B$
and $p_o \neq A \vee^{\mathfrak{Q}} B$
and $p_o \neq A \supset^{\mathfrak{Q}} B$
and $p_o \neq A \equiv^{\mathfrak{Q}} B$
and $\sim^{\mathfrak{Q}} A \neq B \wedge^{\mathfrak{Q}} C$
and $\sim^{\mathfrak{Q}} A \neq B \vee^{\mathfrak{Q}} C$
and $\sim^{\mathfrak{Q}} A \neq B \supset^{\mathfrak{Q}} C$
and $\neg\ (B = F_o \wedge A = C) \Longrightarrow \sim^{\mathfrak{Q}} A \neq B \equiv^{\mathfrak{Q}} C$ — $\sim^{\mathfrak{Q}} A$ is the same as $F_o \equiv^{\mathfrak{Q}} A$
and $A \wedge^{\mathfrak{Q}} B \neq C \vee^{\mathfrak{Q}} D$
and $A \wedge^{\mathfrak{Q}} B \neq C \supset^{\mathfrak{Q}} D$
and $A \wedge^{\mathfrak{Q}} B \neq C \equiv^{\mathfrak{Q}} D$
and $A \vee^{\mathfrak{Q}} B \neq C \supset^{\mathfrak{Q}} D$
and $A \vee^{\mathfrak{Q}} B \neq C \equiv^{\mathfrak{Q}} D$
and $A \supset^{\mathfrak{Q}} B \neq C \equiv^{\mathfrak{Q}} D$
by *simp-all*

lemma *pwffs-injectivities* [*induct-simp*]:
shows $\sim^{\mathfrak{Q}} A = \sim^{\mathfrak{Q}} A' \Longrightarrow A = A'$
and $A \wedge^{\mathfrak{Q}} B = A' \wedge^{\mathfrak{Q}} B' \Longrightarrow A = A' \wedge B = B'$
and $A \vee^{\mathfrak{Q}} B = A' \vee^{\mathfrak{Q}} B' \Longrightarrow A = A' \wedge B = B'$
and $A \supset^{\mathfrak{Q}} B = A' \supset^{\mathfrak{Q}} B' \Longrightarrow A = A' \wedge B = B'$
and $A \equiv^{\mathfrak{Q}} B = A' \equiv^{\mathfrak{Q}} B' \Longrightarrow A = A' \wedge B = B'$
by *simp-all*

lemma *pwff-from-neg-pwff* [*elim*!]:
assumes $\sim^{\mathfrak{Q}} A \in$ *pwffs*
shows $A \in$ *pwffs*
using *assms* **by** *cases simp-all*

lemma *pwffs-from-conj-pwff* [*elim*!]:
assumes $A \wedge^{\mathfrak{Q}} B \in$ *pwffs*
shows $\{A, B\} \subseteq$ *pwffs*
using *assms* **by** *cases simp-all*

lemma *pwffs-from-disj-pwff* [*elim*!]:
assumes $A \vee^{\mathfrak{Q}} B \in$ *pwffs*
shows $\{A, B\} \subseteq$ *pwffs*
using *assms* **by** *cases simp-all*

lemma *pwffs-from-imp-pwff* [*elim*!]:
assumes $A \supset^{\mathfrak{Q}} B \in$ *pwffs*
shows $\{A, B\} \subseteq$ *pwffs*
using *assms* **by** *cases simp-all*

lemma *pwffs-from-eqv-pwff* [*elim*!]:
assumes $A \equiv^{\mathfrak{Q}} B \in$ *pwffs*
shows $\{A, B\} \subseteq$ *pwffs*
using *assms* **by** *cases* (*simp-all*, *use F-pwff* **in** *fastforce*)

lemma *pwffs-subset-of-wffso*:
shows *pwffs* $\subseteq$ *wffs*$_o$
proof
fix A
assume $A \in$ *pwffs*
then show $A \in$ *wffs*$_o$
by *induction auto*
qed

lemma *pwff-free-vars-simps* [*simp*]:
shows *T-fv*: *free-vars* $T_o = \{\}$
and *F-fv*: *free-vars* $F_o = \{\}$
and *var-fv*: *free-vars* $(p_o) = \{(p, o)\}$
and *neg-fv*: *free-vars* $(\sim^{\mathfrak{Q}} A) =$ *free-vars* A
and *conj-fv*: *free-vars* $(A \wedge^{\mathfrak{Q}} B) =$ *free-vars* $A \cup$ *free-vars* B
and *disj-fv*: *free-vars* $(A \vee^{\mathfrak{Q}} B) =$ *free-vars* $A \cup$ *free-vars* B

```
  and imp-fv: free-vars (A ⊃^Q B) = free-vars A ∪ free-vars B
  and eqv-fv: free-vars (A ≡^Q B) = free-vars A ∪ free-vars B
  by force+

lemma pwffs-free-vars-are-propositional:
  assumes A ∈ pwffs
  and v ∈ free-vars A
  obtains p where v = (p, o)
using assms by (induction A arbitrary: thesis) auto

lemma is-free-for-in-pwff [intro]:
  assumes A ∈ pwffs
  and v ∈ free-vars A
  shows is-free-for B v A
using assms proof (induction A)
  case (neg-pwff C)
  then show ?case
    using is-free-for-in-neg by simp
next
  case (conj-pwff C D)
  from conj-pwff.prems consider
    (a) v ∈ free-vars C and v ∈ free-vars D
  | (b) v ∈ free-vars C and v ∉ free-vars D
  | (c) v ∉ free-vars C and v ∈ free-vars D
    by auto
  then show ?case
  proof cases
    case a
    then show ?thesis
      using conj-pwff.IH by (intro is-free-for-in-conj)
  next
    case b
    have is-free-for B v C
      by (fact conj-pwff.IH(1)[OF b(1)])
    moreover from b(2) have is-free-for B v D
      using is-free-at-in-free-vars by blast
    ultimately show ?thesis
      by (rule is-free-for-in-conj)
  next
    case c
    from c(1) have is-free-for B v C
      using is-free-at-in-free-vars by blast
    moreover have is-free-for B v D
      by (fact conj-pwff.IH(2)[OF c(2)])
    ultimately show ?thesis
      by (rule is-free-for-in-conj)
  qed
next
  case (disj-pwff C D)
```

```
  from disj-pwff.prems consider
    (a) v ∈ free-vars C and v ∈ free-vars D
  | (b) v ∈ free-vars C and v ∉ free-vars D
  | (c) v ∉ free-vars C and v ∈ free-vars D
    by auto
  then show ?case
  proof cases
    case a
    then show ?thesis
      using disj-pwff.IH by (intro is-free-for-in-disj)
  next
    case b
    have is-free-for B v C
      by (fact disj-pwff.IH(1)[OF b(1)])
    moreover from b(2) have is-free-for B v D
      using is-free-at-in-free-vars by blast
    ultimately show ?thesis
      by (rule is-free-for-in-disj)
  next
    case c
    from c(1) have is-free-for B v C
      using is-free-at-in-free-vars by blast
    moreover have is-free-for B v D
      by (fact disj-pwff.IH(2)[OF c(2)])
    ultimately show ?thesis
      by (rule is-free-for-in-disj)
  qed
next
  case (imp-pwff C D)
  from imp-pwff.prems consider
    (a) v ∈ free-vars C and v ∈ free-vars D
  | (b) v ∈ free-vars C and v ∉ free-vars D
  | (c) v ∉ free-vars C and v ∈ free-vars D
    by auto
  then show ?case
  proof cases
    case a
    then show ?thesis
      using imp-pwff.IH by (intro is-free-for-in-imp)
  next
    case b
    have is-free-for B v C
      by (fact imp-pwff.IH(1)[OF b(1)])
    moreover from b(2) have is-free-for B v D
      using is-free-at-in-free-vars by blast
    ultimately show ?thesis
      by (rule is-free-for-in-imp)
  next
    case c
```

```
    from c(1) have is-free-for B v C
      using is-free-at-in-free-vars by blast
    moreover have is-free-for B v D
      by (fact imp-pwff.IH(2)[OF c(2)])
    ultimately show ?thesis
      by (rule is-free-for-in-imp)
  qed
next
  case (eqv-pwff C D)
  from eqv-pwff.prems consider
    (a) v ∈ free-vars C and v ∈ free-vars D
  | (b) v ∈ free-vars C and v ∉ free-vars D
  | (c) v ∉ free-vars C and v ∈ free-vars D
    by auto
  then show ?case
  proof cases
    case a
    then show ?thesis
      using eqv-pwff.IH by (intro is-free-for-in-equivalence)
  next
    case b
    have is-free-for B v C
      by (fact eqv-pwff.IH(1)[OF b(1)])
    moreover from b(2) have is-free-for B v D
      using is-free-at-in-free-vars by blast
    ultimately show ?thesis
      by (rule is-free-for-in-equivalence)
  next
    case c
    from c(1) have is-free-for B v C
      using is-free-at-in-free-vars by blast
    moreover have is-free-for B v D
      by (fact eqv-pwff.IH(2)[OF c(2)])
    ultimately show ?thesis
      by (rule is-free-for-in-equivalence)
  qed
qed auto
```

4.2 Semantics

Assignment of truth values to propositional variables:

```
definition is-tv-assignment :: (nat ⇒ V) ⇒ bool where
  [iff]: is-tv-assignment φ ⟷ (∀ p. φ p ∈ elts 𝔹)
```

Denotation of a pwff:

```
definition is-pwff-denotation-function where
  [iff]: is-pwff-denotation-function 𝒱 ⟷
    (
      ∀ φ. is-tv-assignment φ ⟶
```

$$
\begin{array}{l}
\quad (\\
\quad\quad \mathcal{V}\ \varphi\ T_o = \mathbf{T} \wedge \\
\quad\quad \mathcal{V}\ \varphi\ F_o = \mathbf{F} \wedge \\
\quad\quad (\forall\, p.\ \mathcal{V}\ \varphi\ (p_o) = \varphi\ p) \wedge \\
\quad\quad (\forall\, A.\ A \in \mathit{pwffs} \longrightarrow \mathcal{V}\ \varphi\ (\sim^{\mathfrak{Q}}\ A) = \sim\ \mathcal{V}\ \varphi\ A) \wedge \\
\quad\quad (\forall\, A\ B.\ A \in \mathit{pwffs} \wedge B \in \mathit{pwffs} \longrightarrow \mathcal{V}\ \varphi\ (A \wedge^{\mathfrak{Q}} B) = \mathcal{V}\ \varphi\ A \wedge \mathcal{V}\ \varphi\ B) \wedge \\
\quad\quad (\forall\, A\ B.\ A \in \mathit{pwffs} \wedge B \in \mathit{pwffs} \longrightarrow \mathcal{V}\ \varphi\ (A \vee^{\mathfrak{Q}} B) = \mathcal{V}\ \varphi\ A \vee \mathcal{V}\ \varphi\ B) \wedge \\
\quad\quad (\forall\, A\ B.\ A \in \mathit{pwffs} \wedge B \in \mathit{pwffs} \longrightarrow \mathcal{V}\ \varphi\ (A \supset^{\mathfrak{Q}} B) = \mathcal{V}\ \varphi\ A \supset \mathcal{V}\ \varphi\ B) \wedge \\
\quad\quad (\forall\, A\ B.\ A \in \mathit{pwffs} \wedge B \in \mathit{pwffs} \longrightarrow \mathcal{V}\ \varphi\ (A \equiv^{\mathfrak{Q}} B) = \mathcal{V}\ \varphi\ A \equiv \mathcal{V}\ \varphi\ B) \\
\quad) \\
)
$$

lemma *pwff-denotation-is-truth-value*:
assumes $A \in \mathit{pwffs}$
and *is-tv-assignment* φ
and *is-pwff-denotation-function* $\mathcal{V}$
shows $\mathcal{V}\ \varphi\ A \in \mathit{elts}\ \mathbb{B}$
using *assms*(*1*) **proof** *induction*
case (*neg-pwff A*)
then have $\mathcal{V}\ \varphi\ (\sim^{\mathfrak{Q}}\ A) = \sim\ \mathcal{V}\ \varphi\ A$
using *assms*(*2*,*3*) **by** *auto*
then show *?case*
using *neg-pwff.IH* **by** *auto*
next
case (*conj-pwff A B*)
then have $\mathcal{V}\ \varphi\ (A \wedge^{\mathfrak{Q}} B) = \mathcal{V}\ \varphi\ A \wedge \mathcal{V}\ \varphi\ B$
using *assms*(*2*,*3*) **by** *auto*
then show *?case*
using *conj-pwff.IH* **by** *auto*
next
case (*disj-pwff A B*)
then have $\mathcal{V}\ \varphi\ (A \vee^{\mathfrak{Q}} B) = \mathcal{V}\ \varphi\ A \vee \mathcal{V}\ \varphi\ B$
using *assms*(*2*,*3*) **by** *auto*
then show *?case*
using *disj-pwff.IH* **by** *auto*
next
case (*imp-pwff A B*)
then have $\mathcal{V}\ \varphi\ (A \supset^{\mathfrak{Q}} B) = \mathcal{V}\ \varphi\ A \supset \mathcal{V}\ \varphi\ B$
using *assms*(*2*,*3*) **by** *blast*
then show *?case*
using *imp-pwff.IH* **by** *auto*
next
case (*eqv-pwff A B*)
then have $\mathcal{V}\ \varphi\ (A \equiv^{\mathfrak{Q}} B) = \mathcal{V}\ \varphi\ A \equiv \mathcal{V}\ \varphi\ B$
using *assms*(*2*,*3*) **by** *blast*
then show *?case*
using *eqv-pwff.IH* **by** *auto*
qed (*use assms*(*2*,*3*) **in** *auto*)

lemma *closed-pwff-is-meaningful-regardless-of-assignment*:
assumes $A \in$ *pwffs*
and *free-vars* $A = \{\}$
and *is-tv-assignment* φ
and *is-tv-assignment* ψ
and *is-pwff-denotation-function* $\mathcal{V}$
shows $\mathcal{V}\ \varphi\ A = \mathcal{V}\ \psi\ A$
using *assms*(*1*,*2*) **proof** *induction*
case *T-pwff*
have $\mathcal{V}\ \varphi\ T_o = \mathbf{T}$
using *assms*(*3*,*5*) **by** *blast*
also have $\ldots = \mathcal{V}\ \psi\ T_o$
using *assms*(*4*,*5*) **by** *force*
finally show *?case* **.**
next
case *F-pwff*
have $\mathcal{V}\ \varphi\ F_o = \mathbf{F}$
using *assms*(*3*,*5*) **by** *blast*
also have $\ldots = \mathcal{V}\ \psi\ F_o$
using *assms*(*4*,*5*) **by** *force*
finally show *?case* **.**
next
case (*var-pwff p*) — impossible case
then show *?case*
by *simp*
next
case (*neg-pwff A*)
from ‹*free-vars* $(\sim^{\mathcal{Q}} A) = \{\}$› **have** *free-vars* $A = \{\}$
by *simp*
have $\mathcal{V}\ \varphi\ (\sim^{\mathcal{Q}} A) = \sim \mathcal{V}\ \varphi\ A$
using *assms*(*3*,*5*) **and** *neg-pwff.hyps* **by** *auto*
also from ‹*free-vars* $A = \{\}$› **have** $\ldots = \sim \mathcal{V}\ \psi\ A$
using *assms*(*3*−*5*) **and** *neg-pwff.IH* **by** *presburger*
also have $\ldots = \mathcal{V}\ \psi\ (\sim^{\mathcal{Q}} A)$
using *assms*(*4*,*5*) **and** *neg-pwff.hyps* **by** *simp*
finally show *?case* **.**
next
case (*conj-pwff A B*)
from ‹*free-vars* $(A \wedge^{\mathcal{Q}} B) = \{\}$› **have** *free-vars* $A = \{\}$ **and** *free-vars* $B = \{\}$
by *simp-all*
have $\mathcal{V}\ \varphi\ (A \wedge^{\mathcal{Q}} B) = \mathcal{V}\ \varphi\ A \wedge \mathcal{V}\ \varphi\ B$
using *assms*(*3*,*5*) **and** *conj-pwff.hyps*(*1*,*2*) **by** *auto*
also from ‹*free-vars* $A = \{\}$› **and** ‹*free-vars* $B = \{\}$› **have** $\ldots = \mathcal{V}\ \psi\ A \wedge \mathcal{V}\ \psi\ B$
using *conj-pwff.IH*(*1*,*2*) **by** *presburger*
also have $\ldots = \mathcal{V}\ \psi\ (A \wedge^{\mathcal{Q}} B)$
using *assms*(*4*,*5*) **and** *conj-pwff.hyps*(*1*,*2*) **by** *fastforce*
finally show *?case* **.**
next
case (*disj-pwff A B*)

from ‹*free-vars* $(A \vee^{\mathcal{Q}} B) = \{\}$› **have** *free-vars* $A = \{\}$ **and** *free-vars* $B = \{\}$
by *simp-all*
have $\mathcal{V}\ \varphi\ (A \vee^{\mathcal{Q}} B) = \mathcal{V}\ \varphi\ A \vee \mathcal{V}\ \varphi\ B$
using *assms*(*3*,*5*) **and** *disj-pwff.hyps*(*1*,*2*) **by** *auto*
also from ‹*free-vars* $A = \{\}$› **and** ‹*free-vars* $B = \{\}$› **have** $\ldots = \mathcal{V}\ \psi\ A \vee \mathcal{V}\ \psi\ B$
using *disj-pwff.IH*(*1*,*2*) **by** *presburger*
also have $\ldots = \mathcal{V}\ \psi\ (A \vee^{\mathcal{Q}} B)$
using *assms*(*4*,*5*) **and** *disj-pwff.hyps*(*1*,*2*) **by** *fastforce*
finally show *?case* **.**
next
case (*imp-pwff A B*)
from ‹*free-vars* $(A \supset^{\mathcal{Q}} B) = \{\}$› **have** *free-vars* $A = \{\}$ **and** *free-vars* $B = \{\}$
by *simp-all*
have $\mathcal{V}\ \varphi\ (A \supset^{\mathcal{Q}} B) = \mathcal{V}\ \varphi\ A \supset \mathcal{V}\ \varphi\ B$
using *assms*(*3*,*5*) **and** *imp-pwff.hyps*(*1*,*2*) **by** *auto*
also from ‹*free-vars* $A = \{\}$› **and** ‹*free-vars* $B = \{\}$› **have** $\ldots = \mathcal{V}\ \psi\ A \supset \mathcal{V}\ \psi\ B$
using *imp-pwff.IH*(*1*,*2*) **by** *presburger*
also have $\ldots = \mathcal{V}\ \psi\ (A \supset^{\mathcal{Q}} B)$
using *assms*(*4*,*5*) **and** *imp-pwff.hyps*(*1*,*2*) **by** *fastforce*
finally show *?case* **.**
next
case (*eqv-pwff A B*)
from ‹*free-vars* $(A \equiv^{\mathcal{Q}} B) = \{\}$› **have** *free-vars* $A = \{\}$ **and** *free-vars* $B = \{\}$
by *simp-all*
have $\mathcal{V}\ \varphi\ (A \equiv^{\mathcal{Q}} B) = \mathcal{V}\ \varphi\ A \equiv \mathcal{V}\ \varphi\ B$
using *assms*(*3*,*5*) **and** *eqv-pwff.hyps*(*1*,*2*) **by** *auto*
also from ‹*free-vars* $A = \{\}$› **and** ‹*free-vars* $B = \{\}$› **have** $\ldots = \mathcal{V}\ \psi\ A \equiv \mathcal{V}\ \psi\ B$
using *eqv-pwff.IH*(*1*,*2*) **by** *presburger*
also have $\ldots = \mathcal{V}\ \psi\ (A \equiv^{\mathcal{Q}} B)$
using *assms*(*4*,*5*) **and** *eqv-pwff.hyps*(*1*,*2*) **by** *fastforce*
finally show *?case* **.**
qed

inductive $\mathcal{V}_B$*-graph* **for** φ **where**
$\mathcal{V}_B$*-graph-T*: $\mathcal{V}_B$*-graph* $\varphi\ T_o$ **T**
| $\mathcal{V}_B$*-graph-F*: $\mathcal{V}_B$*-graph* $\varphi\ F_o$ **F**
| $\mathcal{V}_B$*-graph-var*: $\mathcal{V}_B$*-graph* $\varphi\ (p_o)\ (\varphi\ p)$
| $\mathcal{V}_B$*-graph-neg*: $\mathcal{V}_B$*-graph* $\varphi\ (\sim^{\mathcal{Q}} A)\ (\sim b_A)$ **if** $\mathcal{V}_B$*-graph* $\varphi\ A\ b_A$
| $\mathcal{V}_B$*-graph-conj*: $\mathcal{V}_B$*-graph* $\varphi\ (A \wedge^{\mathcal{Q}} B)\ (b_A \wedge b_B)$ **if** $\mathcal{V}_B$*-graph* $\varphi\ A\ b_A$ **and** $\mathcal{V}_B$*-graph* $\varphi\ B\ b_B$
| $\mathcal{V}_B$*-graph-disj*: $\mathcal{V}_B$*-graph* $\varphi\ (A \vee^{\mathcal{Q}} B)\ (b_A \vee b_B)$ **if** $\mathcal{V}_B$*-graph* $\varphi\ A\ b_A$ **and** $\mathcal{V}_B$*-graph* $\varphi\ B\ b_B$
| $\mathcal{V}_B$*-graph-imp*: $\mathcal{V}_B$*-graph* $\varphi\ (A \supset^{\mathcal{Q}} B)\ (b_A \supset b_B)$ **if** $\mathcal{V}_B$*-graph* $\varphi\ A\ b_A$ **and** $\mathcal{V}_B$*-graph* $\varphi\ B\ b_B$
| $\mathcal{V}_B$*-graph-eqv*: $\mathcal{V}_B$*-graph* $\varphi\ (A \equiv^{\mathcal{Q}} B)\ (b_A \equiv b_B)$ **if** $\mathcal{V}_B$*-graph* $\varphi\ A\ b_A$ **and** $\mathcal{V}_B$*-graph* $\varphi\ B\ b_B$ **and** $A \neq F_o$

lemmas [*intro*!] = $\mathcal{V}_B$*-graph.intros*

lemma $\mathcal{V}_B$*-graph-denotation-is-truth-value* [*elim*!]:
assumes $\mathcal{V}_B$*-graph* $\varphi\ A\ b$
and *is-tv-assignment* φ

shows $b \in \mathit{elts}\ \mathbb{B}$
using *assms* **proof** *induction*
case ($\mathcal{V}_B$*-graph-neg* $A\ b_A$)
show *?case*
using $\mathcal{V}_B$*-graph-neg.IH*[*OF assms*(*2*)] **by** *force*
next
case ($\mathcal{V}_B$*-graph-conj* $A\ b_A\ B\ b_B$)
then show *?case*
using $\mathcal{V}_B$*-graph-conj.IH* **and** *assms*(*2*) **by** *force*
next
case ($\mathcal{V}_B$*-graph-disj* $A\ b_A\ B\ b_B$)
then show *?case*
using $\mathcal{V}_B$*-graph-disj.IH* **and** *assms*(*2*) **by** *force*
next
case ($\mathcal{V}_B$*-graph-imp* $A\ b_A\ B\ b_B$)
then show *?case*
using $\mathcal{V}_B$*-graph-imp.IH* **and** *assms*(*2*) **by** *force*
next
case ($\mathcal{V}_B$*-graph-eqv* $A\ b_A\ B\ b_B$)
then show *?case*
using $\mathcal{V}_B$*-graph-eqv.IH* **and** *assms*(*2*) **by** *force*
qed *simp-all*

lemma $\mathcal{V}_B$*-graph-denotation-uniqueness*:
assumes $A \in \mathit{pwffs}$
and *is-tv-assignment* φ
and $\mathcal{V}_B$*-graph* $\varphi\ A\ b$ **and** $\mathcal{V}_B$*-graph* $\varphi\ A\ b'$
shows $b = b'$
using *assms*(*3*,*1*,*4*) **proof** (*induction arbitrary*: b')
case $\mathcal{V}_B$*-graph-T*
from ‹$\mathcal{V}_B$*-graph* $\varphi\ T_o\ b'$› **show** *?case*
by (*cases rule*: $\mathcal{V}_B$*-graph.cases*) *simp-all*
next
case $\mathcal{V}_B$*-graph-F*
from ‹$\mathcal{V}_B$*-graph* $\varphi\ F_o\ b'$› **show** *?case*
by (*cases rule*: $\mathcal{V}_B$*-graph.cases*) *simp-all*
next
case ($\mathcal{V}_B$*-graph-var* p)
from ‹$\mathcal{V}_B$*-graph* $\varphi\ (p_o)\ b'$› **show** *?case*
by (*cases rule*: $\mathcal{V}_B$*-graph.cases*) *simp-all*
next
case ($\mathcal{V}_B$*-graph-neg* $A\ b_A$)
with ‹$\mathcal{V}_B$*-graph* $\varphi\ (\sim^{\mathfrak{Q}} A)\ b'$› **have** $\mathcal{V}_B$*-graph* $\varphi\ A\ (\sim b')$
proof (*cases rule*: $\mathcal{V}_B$*-graph.cases*)
case ($\mathcal{V}_B$*-graph-neg* $A'\ b_A$)
from ‹$\sim^{\mathfrak{Q}} A = \sim^{\mathfrak{Q}} A'$› **have** $A = A'$
by *simp*
with ‹$\mathcal{V}_B$*-graph* $\varphi\ A'\ b_A$› **have** $\mathcal{V}_B$*-graph* $\varphi\ A\ b_A$
by *simp*

```
    moreover have b_A = ∼ b'
    proof −
      have b_A ∈ elts 𝔹
        by (fact 𝒱_B-graph-denotation-is-truth-value[OF 𝒱_B-graph-neg(3) assms(2)])
      moreover from ‹b_A ∈ elts 𝔹› and 𝒱_B-graph-neg(2) have ∼ b' ∈ elts 𝔹
        by fastforce
      ultimately show ?thesis
        using 𝒱_B-graph-neg(2) by fastforce
    qed
    ultimately show ?thesis
      by blast
  qed simp-all
  moreover from 𝒱_B-graph-neg.prems(1) have A ∈ pwffs
    by (force elim: pwffs.cases)
  moreover have b_A ∈ elts 𝔹 and b' ∈ elts 𝔹 and b_A = ∼ b'
  proof −
    show b_A ∈ elts 𝔹
      by (fact 𝒱_B-graph-denotation-is-truth-value[OF ‹𝒱_B-graph φ A b_A› assms(2)])
    show b' ∈ elts 𝔹
      by (fact 𝒱_B-graph-denotation-is-truth-value[OF ‹𝒱_B-graph φ (∼^𝒬 A) b'› assms(2)])
    show b_A = ∼ b'
      by (fact 𝒱_B-graph-neg(2)[OF ‹A ∈ pwffs› ‹𝒱_B-graph φ A (∼ b')›])
  qed
  ultimately show ?case
    by force
next
  case (𝒱_B-graph-conj A b_A B b_B)
  with ‹𝒱_B-graph φ (A ∧^𝒬 B) b'› obtain b_A' and b_B'
    where b' = b_A' ∧ b_B' and 𝒱_B-graph φ A b_A' and 𝒱_B-graph φ B b_B'
    by (cases rule: 𝒱_B-graph.cases) simp-all
  moreover have A ∈ pwffs and B ∈ pwffs
    using pwffs-from-conj-pwff[OF 𝒱_B-graph-conj.prems(1)] by blast+
  ultimately show ?case
    using 𝒱_B-graph-conj.IH and 𝒱_B-graph-conj.prems(2) by blast
next
  case (𝒱_B-graph-disj A b_A B b_B)
  from ‹𝒱_B-graph φ (A ∨^𝒬 B) b'› obtain b_A' and b_B'
    where b' = b_A' ∨ b_B' and 𝒱_B-graph φ A b_A' and 𝒱_B-graph φ B b_B'
    by (cases rule: 𝒱_B-graph.cases) simp-all
  moreover have A ∈ pwffs and B ∈ pwffs
    using pwffs-from-disj-pwff[OF 𝒱_B-graph-disj.prems(1)] by blast+
  ultimately show ?case
    using 𝒱_B-graph-disj.IH and 𝒱_B-graph-disj.prems(2) by blast
next
  case (𝒱_B-graph-imp A b_A B b_B)
  from ‹𝒱_B-graph φ (A ⊃^𝒬 B) b'› obtain b_A' and b_B'
    where b' = b_A' ⊃ b_B' and 𝒱_B-graph φ A b_A' and 𝒱_B-graph φ B b_B'
    by (cases rule: 𝒱_B-graph.cases) simp-all
  moreover have A ∈ pwffs and B ∈ pwffs
```

using *pwffs-from-imp-pwff*[*OF* $\mathcal{V}_B$*-graph-imp.prems*(*1*)] **by** *blast+*
ultimately show *?case*
using $\mathcal{V}_B$*-graph-imp.IH* **and** $\mathcal{V}_B$*-graph-imp.prems*(*2*) **by** *blast*
next
case ($\mathcal{V}_B$*-graph-eqv* A b_A B b_B)
with ‹$\mathcal{V}_B$*-graph* φ ($A \equiv^{\mathcal{Q}} B$) b'› **obtain** $b_A{}'$ **and** $b_B{}'$
where $b' = b_A{}' \equiv b_B{}'$ **and** $\mathcal{V}_B$*-graph* φ A $b_A{}'$ **and** $\mathcal{V}_B$*-graph* φ B $b_B{}'$
by (*cases rule*: $\mathcal{V}_B$*-graph.cases*) *simp-all*
moreover have $A \in$ *pwffs* **and** $B \in$ *pwffs*
using *pwffs-from-eqv-pwff*[*OF* $\mathcal{V}_B$*-graph-eqv.prems*(*1*)] **by** *blast+*
ultimately show *?case*
using $\mathcal{V}_B$*-graph-eqv.IH* **and** $\mathcal{V}_B$*-graph-eqv.prems*(*2*) **by** *blast*
qed

lemma $\mathcal{V}_B$*-graph-denotation-existence*:
assumes $A \in$ *pwffs*
and *is-tv-assignment* φ
shows $\exists b.$ $\mathcal{V}_B$*-graph* φ A b
using *assms* **proof** *induction*
case (*eqv-pwff* A B)
then obtain b_A **and** b_B **where** $\mathcal{V}_B$*-graph* φ A b_A **and** $\mathcal{V}_B$*-graph* φ B b_B
by *blast*
then show *?case*
proof (*cases* $A \neq F_o$)
case *True*
then show *?thesis*
using *eqv-pwff.IH* **and** *eqv-pwff.prems* **by** *blast*
next
case *False*
then have $A = F_o$
by *blast*
then show *?thesis*
using $\mathcal{V}_B$*-graph-neg*[*OF* ‹$\mathcal{V}_B$*-graph* φ B b_B›] **by** *auto*
qed
qed *blast+*

lemma $\mathcal{V}_B$*-graph-is-functional*:
assumes $A \in$ *pwffs*
and *is-tv-assignment* φ
shows $\exists! b.$ $\mathcal{V}_B$*-graph* φ A b
using *assms* **and** $\mathcal{V}_B$*-graph-denotation-existence* **and** $\mathcal{V}_B$*-graph-denotation-uniqueness* **by** *blast*

definition $\mathcal{V}_B :: (nat \Rightarrow V) \Rightarrow form \Rightarrow V$ **where**
[*simp*]: $\mathcal{V}_B$ φ A = (*THE* $b.$ $\mathcal{V}_B$*-graph* φ A b)

lemma $\mathcal{V}_B$*-equality*:
assumes $A \in$ *pwffs*
and *is-tv-assignment* φ
and $\mathcal{V}_B$*-graph* φ A b

shows $\mathcal{V}_B\ \varphi\ A = b$
unfolding $\mathcal{V}_B$-*def* **using** *assms* **using** $\mathcal{V}_B$-*graph-denotation-uniqueness* **by** *blast*

lemma $\mathcal{V}_B$-*graph-*$\mathcal{V}_B$:
assumes $A \in$ *pwffs*
and *is-tv-assignment* φ
shows $\mathcal{V}_B$-*graph* $\varphi\ A\ (\mathcal{V}_B\ \varphi\ A)$
using $\mathcal{V}_B$-*equality*[*OF assms*] **and** $\mathcal{V}_B$-*graph-is-functional*[*OF assms*] **by** *blast*

named-theorems $\mathcal{V}_B$-*simps*

lemma $\mathcal{V}_B$-T [$\mathcal{V}_B$-*simps*]:
assumes *is-tv-assignment* φ
shows $\mathcal{V}_B\ \varphi\ T_o = \mathbf{T}$
by (*rule* $\mathcal{V}_B$-*equality*[*OF T-pwff assms*], *intro* $\mathcal{V}_B$-*graph-T*)

lemma $\mathcal{V}_B$-F [$\mathcal{V}_B$-*simps*]:
assumes *is-tv-assignment* φ
shows $\mathcal{V}_B\ \varphi\ F_o = \mathbf{F}$
by (*rule* $\mathcal{V}_B$-*equality*[*OF F-pwff assms*], *intro* $\mathcal{V}_B$-*graph-F*)

lemma $\mathcal{V}_B$-*var* [$\mathcal{V}_B$-*simps*]:
assumes *is-tv-assignment* φ
shows $\mathcal{V}_B\ \varphi\ (p_o) = \varphi\ p$
by (*rule* $\mathcal{V}_B$-*equality*[*OF var-pwff assms*], *intro* $\mathcal{V}_B$-*graph-var*)

lemma $\mathcal{V}_B$-*neg* [$\mathcal{V}_B$-*simps*]:
assumes $A \in$ *pwffs*
and *is-tv-assignment* φ
shows $\mathcal{V}_B\ \varphi\ (\sim^{\mathcal{Q}}\ A) = \sim\ \mathcal{V}_B\ \varphi\ A$
by (*rule* $\mathcal{V}_B$-*equality*[*OF neg-pwff*[*OF assms*(*1*)] *assms*(*2*)], *intro* $\mathcal{V}_B$-*graph-neg* $\mathcal{V}_B$-*graph-*$\mathcal{V}_B$[*OF assms*])

lemma $\mathcal{V}_B$-*disj* [$\mathcal{V}_B$-*simps*]:
assumes $A \in$ *pwffs* **and** $B \in$ *pwffs*
and *is-tv-assignment* φ
shows $\mathcal{V}_B\ \varphi\ (A \vee^{\mathcal{Q}} B) = \mathcal{V}_B\ \varphi\ A \vee \mathcal{V}_B\ \varphi\ B$
proof −
from *assms*(*1*,*3*) **have** $\mathcal{V}_B$-*graph* $\varphi\ A\ (\mathcal{V}_B\ \varphi\ A)$
by (*intro* $\mathcal{V}_B$-*graph-*$\mathcal{V}_B$)
moreover from *assms*(*2*,*3*) **have** $\mathcal{V}_B$-*graph* $\varphi\ B\ (\mathcal{V}_B\ \varphi\ B)$
by (*intro* $\mathcal{V}_B$-*graph-*$\mathcal{V}_B$)
ultimately have $\mathcal{V}_B$-*graph* $\varphi\ (A \vee^{\mathcal{Q}} B)\ (\mathcal{V}_B\ \varphi\ A \vee \mathcal{V}_B\ \varphi\ B)$
by (*intro* $\mathcal{V}_B$-*graph-disj*)
with *assms* **show** *?thesis*
using *disj-pwff* **by** (*intro* $\mathcal{V}_B$-*equality*)
qed

lemma $\mathcal{V}_B$-*conj* [$\mathcal{V}_B$-*simps*]:

assumes $A \in$ *pwffs* **and** $B \in$ *pwffs*
and *is-tv-assignment* φ
shows $\mathcal{V}_B\ \varphi\ (A \wedge^{\mathfrak{Q}} B) = \mathcal{V}_B\ \varphi\ A \wedge \mathcal{V}_B\ \varphi\ B$
proof −
from *assms*(*1*,*3*) **have** $\mathcal{V}_B$-*graph* $\varphi\ A\ (\mathcal{V}_B\ \varphi\ A)$
by (*intro* $\mathcal{V}_B$-*graph*-$\mathcal{V}_B$)
moreover from *assms*(*2*,*3*) **have** $\mathcal{V}_B$-*graph* $\varphi\ B\ (\mathcal{V}_B\ \varphi\ B)$
by (*intro* $\mathcal{V}_B$-*graph*-$\mathcal{V}_B$)
ultimately have $\mathcal{V}_B$-*graph* $\varphi\ (A \wedge^{\mathfrak{Q}} B)\ (\mathcal{V}_B\ \varphi\ A \wedge \mathcal{V}_B\ \varphi\ B)$
by (*intro* $\mathcal{V}_B$-*graph-conj*)
with *assms* **show** *?thesis*
using *conj-pwff* **by** (*intro* $\mathcal{V}_B$-*equality*)
qed

lemma $\mathcal{V}_B$-*imp* [$\mathcal{V}_B$-*simps*]:
assumes $A \in$ *pwffs* **and** $B \in$ *pwffs*
and *is-tv-assignment* φ
shows $\mathcal{V}_B\ \varphi\ (A \supset^{\mathfrak{Q}} B) = \mathcal{V}_B\ \varphi\ A \supset \mathcal{V}_B\ \varphi\ B$
proof −
from *assms*(*1*,*3*) **have** $\mathcal{V}_B$-*graph* $\varphi\ A\ (\mathcal{V}_B\ \varphi\ A)$
by (*intro* $\mathcal{V}_B$-*graph*-$\mathcal{V}_B$)
moreover from *assms*(*2*,*3*) **have** $\mathcal{V}_B$-*graph* $\varphi\ B\ (\mathcal{V}_B\ \varphi\ B)$
by (*intro* $\mathcal{V}_B$-*graph*-$\mathcal{V}_B$)
ultimately have $\mathcal{V}_B$-*graph* $\varphi\ (A \supset^{\mathfrak{Q}} B)\ (\mathcal{V}_B\ \varphi\ A \supset \mathcal{V}_B\ \varphi\ B)$
by (*intro* $\mathcal{V}_B$-*graph-imp*)
with *assms* **show** *?thesis*
using *imp-pwff* **by** (*intro* $\mathcal{V}_B$-*equality*)
qed

lemma $\mathcal{V}_B$-*eqv* [$\mathcal{V}_B$-*simps*]:
assumes $A \in$ *pwffs* **and** $B \in$ *pwffs*
and *is-tv-assignment* φ
shows $\mathcal{V}_B\ \varphi\ (A \equiv^{\mathfrak{Q}} B) = \mathcal{V}_B\ \varphi\ A \equiv \mathcal{V}_B\ \varphi\ B$
proof (*cases* $A = F_o$)
case *True*
then show *?thesis*
using $\mathcal{V}_B$-F[*OF assms*(*3*)] **and** $\mathcal{V}_B$-*neg*[*OF assms*(*2*,*3*)] **by** *force*
next
case *False*
from *assms*(*1*,*3*) **have** $\mathcal{V}_B$-*graph* $\varphi\ A\ (\mathcal{V}_B\ \varphi\ A)$
by (*intro* $\mathcal{V}_B$-*graph*-$\mathcal{V}_B$)
moreover from *assms*(*2*,*3*) **have** $\mathcal{V}_B$-*graph* $\varphi\ B\ (\mathcal{V}_B\ \varphi\ B)$
by (*intro* $\mathcal{V}_B$-*graph*-$\mathcal{V}_B$)
ultimately have $\mathcal{V}_B$-*graph* $\varphi\ (A \equiv^{\mathfrak{Q}} B)\ (\mathcal{V}_B\ \varphi\ A \equiv \mathcal{V}_B\ \varphi\ B)$
using *False* **by** (*intro* $\mathcal{V}_B$-*graph-eqv*)
with *assms* **show** *?thesis*
using *eqv-pwff* **by** (*intro* $\mathcal{V}_B$-*equality*)
qed

declare *pwffs.intros* [$\mathcal{V}_B$*-simps*]

lemma *pwff-denotation-function-existence*:
shows *is-pwff-denotation-function* $\mathcal{V}_B$
using $\mathcal{V}_B$*-simps* **by** *simp*

Tautologies:

definition *is-tautology* :: *form* $\Rightarrow$ *bool* **where**
[*iff*]: *is-tautology* $A \longleftrightarrow A \in$ *pwffs* $\wedge$ ($\forall \varphi$. *is-tv-assignment* $\varphi \longrightarrow \mathcal{V}_B\ \varphi\ A = \mathbf{T}$)

lemma *tautology-is-wffo*:
assumes *is-tautology* A
shows $A \in$ *wffs*$_o$
using *assms* **and** *pwffs-subset-of-wffso* **by** *blast*

lemma *propositional-implication-reflexivity-is-tautology*:
shows *is-tautology* ($p_o \supset^{\mathcal{Q}} p_o$)
using $\mathcal{V}_B$*-simps* **by** *simp*

lemma *propositional-principle-of-simplification-is-tautology*:
shows *is-tautology* ($p_o \supset^{\mathcal{Q}} (r_o \supset^{\mathcal{Q}} p_o)$)
using $\mathcal{V}_B$*-simps* **by** *simp*

lemma *closed-pwff-denotation-uniqueness*:
assumes $A \in$ *pwffs* **and** *free-vars* $A = \{\}$
obtains b **where** $\forall \varphi$. *is-tv-assignment* $\varphi \longrightarrow \mathcal{V}_B\ \varphi\ A = b$
using *assms*
by (*meson closed-pwff-is-meaningful-regardless-of-assignment pwff-denotation-function-existence*)

lemma *pwff-substitution-simps*:
shows S $\{(p,\ o) \rightarrowtail A\}\ T_o = T_o$
and S $\{(p,\ o) \rightarrowtail A\}\ F_o = F_o$
and S $\{(p,\ o) \rightarrowtail A\}\ (p'_o) = (\mathit{if}\ p = p'\ \mathit{then}\ A\ \mathit{else}\ (p'_o))$
and S $\{(p,\ o) \rightarrowtail A\}\ (\sim^{\mathcal{Q}} B) = \sim^{\mathcal{Q}}$ (**S** $\{(p,\ o) \rightarrowtail A\}\ B$)
and S $\{(p,\ o) \rightarrowtail A\}\ (B \wedge^{\mathcal{Q}} C) =$ (**S** $\{(p,\ o) \rightarrowtail A\}\ B$) $\wedge^{\mathcal{Q}}$ (**S** $\{(p,\ o) \rightarrowtail A\}\ C$)
and S $\{(p,\ o) \rightarrowtail A\}\ (B \vee^{\mathcal{Q}} C) =$ (**S** $\{(p,\ o) \rightarrowtail A\}\ B$) $\vee^{\mathcal{Q}}$ (**S** $\{(p,\ o) \rightarrowtail A\}\ C$)
and S $\{(p,\ o) \rightarrowtail A\}\ (B \supset^{\mathcal{Q}} C) =$ (**S** $\{(p,\ o) \rightarrowtail A\}\ B$) $\supset^{\mathcal{Q}}$ (**S** $\{(p,\ o) \rightarrowtail A\}\ C$)
and S $\{(p,\ o) \rightarrowtail A\}\ (B \equiv^{\mathcal{Q}} C) =$ (**S** $\{(p,\ o) \rightarrowtail A\}\ B$) $\equiv^{\mathcal{Q}}$ (**S** $\{(p,\ o) \rightarrowtail A\}\ C$)
by *simp-all*

lemma *pwff-substitution-in-pwffs*:
assumes $A \in$ *pwffs* **and** $B \in$ *pwffs*
shows S $\{(p,\ o) \rightarrowtail A\}\ B \in$ *pwffs*
using *assms*(*2*) **proof** *induction*
case *T-pwff*
then show *?case*
using *pwffs.T-pwff* **by** *simp*
next
case *F-pwff*

then show *?case*
using *pwffs.F-pwff* **by** *simp*
next
case (*var-pwff p*)
from *assms*(*1*) **show** *?case*
using *pwffs.var-pwff* **by** *simp*
next
case (*neg-pwff A*)
then show *?case*
using *pwff-substitution-simps*(*4*) **and** *pwffs.neg-pwff* **by** *simp*
next
case (*conj-pwff A B*)
then show *?case*
using *pwff-substitution-simps*(*5*) **and** *pwffs.conj-pwff* **by** *simp*
next
case (*disj-pwff A B*)
then show *?case*
using *pwff-substitution-simps*(*6*) **and** *pwffs.disj-pwff* **by** *simp*
next
case (*imp-pwff A B*)
then show *?case*
using *pwff-substitution-simps*(*7*) **and** *pwffs.imp-pwff* **by** *simp*
next
case (*eqv-pwff A B*)
then show *?case*
using *pwff-substitution-simps*(*8*) **and** *pwffs.eqv-pwff* **by** *simp*
qed

lemma *pwff-substitution-denotation*:
assumes $A \in$ *pwffs* **and** $B \in$ *pwffs*
and *is-tv-assignment* φ
shows $\mathcal{V}_B\ \varphi\ (\mathbf{S}\ \{(p,\ o) \rightarrowtail A\}\ B) = \mathcal{V}_B\ (\varphi(p := \mathcal{V}_B\ \varphi\ A))\ B$
proof −
from *assms*(*1*,*3*) **have** *is-tv-assignment* $(\varphi(p := \mathcal{V}_B\ \varphi\ A))$
using $\mathcal{V}_B$*-graph-denotation-is-truth-value*[*OF* $\mathcal{V}_B$*-graph-*$\mathcal{V}_B$] **by** *simp*
with *assms*(*2*,*1*,*3*) **show** *?thesis*
using $\mathcal{V}_B$*-simps* **and** *pwff-substitution-in-pwffs* **by** *induction auto*
qed

lemma *pwff-substitution-tautology-preservation*:
assumes *is-tautology B* **and** $A \in$ *pwffs*
and $(p,\ o) \in$ *free-vars B*
shows *is-tautology* $(\mathbf{S}\ \{(p,\ o) \rightarrowtail A\}\ B)$
proof (*safe*, *fold is-tv-assignment-def*)
from *assms*(*1*,*2*) **show** $\mathbf{S}\ \{(p,\ o) \rightarrowtail A\}\ B \in$ *pwffs*
using *pwff-substitution-in-pwffs* **by** *blast*
next
fix φ
assume *is-tv-assignment* φ

with *assms*(*1*,*2*) **have** $\mathcal{V}_B\ \varphi\ (\mathbf{S}\ \{(p,\ o) \rightarrowtail A\}\ B) = \mathcal{V}_B\ (\varphi(p := \mathcal{V}_B\ \varphi\ A))\ B$
using *pwff-substitution-denotation* **by** *blast*
moreover from ‹*is-tv-assignment* φ› **and** *assms*(*2*) **have** *is-tv-assignment* $(\varphi(p := \mathcal{V}_B\ \varphi\ A))$
using $\mathcal{V}_B$*-graph-denotation-is-truth-value*[*OF* $\mathcal{V}_B$*-graph-*$\mathcal{V}_B$] **by** *simp*
with *assms*(*1*) **have** $\mathcal{V}_B\ (\varphi(p := \mathcal{V}_B\ \varphi\ A))\ B = \mathbf{T}$
by *fastforce*
ultimately show $\mathcal{V}_B\ \varphi\ \mathbf{S}\ \{(p,\ o) \rightarrowtail A\}\ B = \mathbf{T}$
by (*simp only*:)
qed

lemma *closed-pwff-substitution-free-vars*:
assumes $A \in$ *pwffs* **and** $B \in$ *pwffs*
and *free-vars* $A = \{\}$
and $(p,\ o) \in$ *free-vars* B
shows *free-vars* $(\mathbf{S}\ \{(p,\ o) \rightarrowtail A\}\ B) =$ *free-vars* $B - \{(p,\ o)\}$ (**is** ‹*free-vars* $(\mathbf{S}\ ?\vartheta\ B) =$ -›)
using *assms*(*2*,*4*) **proof** *induction*
case (*conj-pwff* $C\ D$)
have *free-vars* $(\mathbf{S}\ ?\vartheta\ (C \wedge^{\mathfrak{Q}} D)) =$ *free-vars* $((\mathbf{S}\ ?\vartheta\ C) \wedge^{\mathfrak{Q}} (\mathbf{S}\ ?\vartheta\ D))$
by *simp*
also have ... = *free-vars* $(\mathbf{S}\ ?\vartheta\ C) \cup$ *free-vars* $(\mathbf{S}\ ?\vartheta\ D)$
by (*fact conj-fv*)
finally have $*$: *free-vars* $(\mathbf{S}\ ?\vartheta\ (C \wedge^{\mathfrak{Q}} D)) =$ *free-vars* $(\mathbf{S}\ ?\vartheta\ C) \cup$ *free-vars* $(\mathbf{S}\ ?\vartheta\ D)$ **.**
from *conj-pwff.prems* **consider**
$(a)\ (p,\ o) \in$ *free-vars* C **and** $(p,\ o) \in$ *free-vars* D
| $(b)\ (p,\ o) \in$ *free-vars* C **and** $(p,\ o) \notin$ *free-vars* D
| $(c)\ (p,\ o) \notin$ *free-vars* C **and** $(p,\ o) \in$ *free-vars* D
by *auto*
from *this* **and** $*$ **and** *conj-pwff.IH* **show** *?case*
using *free-var-singleton-substitution-neutrality* **by** *cases auto*
next
case (*disj-pwff* $C\ D$)
have *free-vars* $(\mathbf{S}\ ?\vartheta\ (C \vee^{\mathfrak{Q}} D)) =$ *free-vars* $((\mathbf{S}\ ?\vartheta\ C) \vee^{\mathfrak{Q}} (\mathbf{S}\ ?\vartheta\ D))$
by *simp*
also have ... = *free-vars* $(\mathbf{S}\ ?\vartheta\ C) \cup$ *free-vars* $(\mathbf{S}\ ?\vartheta\ D)$
by (*fact disj-fv*)
finally have $*$: *free-vars* $(\mathbf{S}\ ?\vartheta\ (C \vee^{\mathfrak{Q}} D)) =$ *free-vars* $(\mathbf{S}\ ?\vartheta\ C) \cup$ *free-vars* $(\mathbf{S}\ ?\vartheta\ D)$ **.**
from *disj-pwff.prems* **consider**
$(a)\ (p,\ o) \in$ *free-vars* C **and** $(p,\ o) \in$ *free-vars* D
| $(b)\ (p,\ o) \in$ *free-vars* C **and** $(p,\ o) \notin$ *free-vars* D
| $(c)\ (p,\ o) \notin$ *free-vars* C **and** $(p,\ o) \in$ *free-vars* D
by *auto*
from *this* **and** $*$ **and** *disj-pwff.IH* **show** *?case*
using *free-var-singleton-substitution-neutrality* **by** *cases auto*
next
case (*imp-pwff* $C\ D$)
have *free-vars* $(\mathbf{S}\ ?\vartheta\ (C \supset^{\mathfrak{Q}} D)) =$ *free-vars* $((\mathbf{S}\ ?\vartheta\ C) \supset^{\mathfrak{Q}} (\mathbf{S}\ ?\vartheta\ D))$
by *simp*
also have ... = *free-vars* $(\mathbf{S}\ ?\vartheta\ C) \cup$ *free-vars* $(\mathbf{S}\ ?\vartheta\ D)$
by (*fact imp-fv*)

```
  finally have ∗: free-vars (S ?ϑ (C ⊃Ω D)) = free-vars (S ?ϑ C) ∪ free-vars (S ?ϑ D) .
  from imp-pwff.prems consider
    (a) (p, o) ∈ free-vars C and (p, o) ∈ free-vars D
  | (b) (p, o) ∈ free-vars C and (p, o) ∉ free-vars D
  | (c) (p, o) ∉ free-vars C and (p, o) ∈ free-vars D
    by auto
  from this and ∗ and imp-pwff.IH show ?case
    using free-var-singleton-substitution-neutrality by cases auto
next
  case (eqv-pwff C D)
  have free-vars (S ?ϑ (C ≡Ω D)) = free-vars ((S ?ϑ C) ≡Ω (S ?ϑ D))
    by simp
  also have ... = free-vars (S ?ϑ C) ∪ free-vars (S ?ϑ D)
    by (fact eqv-fv)
  finally have ∗: free-vars (S ?ϑ (C ≡Ω D)) = free-vars (S ?ϑ C) ∪ free-vars (S ?ϑ D) .
  from eqv-pwff.prems consider
    (a) (p, o) ∈ free-vars C and (p, o) ∈ free-vars D
  | (b) (p, o) ∈ free-vars C and (p, o) ∉ free-vars D
  | (c) (p, o) ∉ free-vars C and (p, o) ∈ free-vars D
    by auto
  from this and ∗ and eqv-pwff.IH show ?case
    using free-var-singleton-substitution-neutrality by cases auto
qed (use assms(3) in ‹force+›)
```

Substitution in a pwff:

```
definition is-pwff-substitution where
  [iff]: is-pwff-substitution ϑ ⟷ is-substitution ϑ ∧ (∀(x, α) ∈ fmdom′ ϑ. α = o)
```

Tautologous pwff:

```
definition is-tautologous :: form ⇒ bool where
  [iff]: is-tautologous B ⟷ (∃ϑ A. is-tautology A ∧ is-pwff-substitution ϑ ∧ B = S ϑ A)

lemma tautologous-is-wffo:
  assumes is-tautologous A
  shows A ∈ wffs_o
  using assms and substitution-preserves-typing and tautology-is-wffo by blast

lemma implication-reflexivity-is-tautologous:
  assumes A ∈ wffs_o
  shows is-tautologous (A ⊃Ω A)
proof −
  let ?ϑ = {(𝔯, o) ⇀ A}
  have is-tautology (𝔯_o ⊃Ω 𝔯_o)
    by (fact propositional-implication-reflexivity-is-tautology)
  moreover have is-pwff-substitution ?ϑ
    using assms by auto
  moreover have A ⊃Ω A = S ?ϑ (𝔯_o ⊃Ω 𝔯_o)
    by simp
  ultimately show ?thesis
```

by *blast*
qed

lemma *principle-of-simplification-is-tautologous*:
assumes $A \in wffs_o$ **and** $B \in wffs_o$
shows *is-tautologous* $(A \supset^{\mathfrak{Q}} (B \supset^{\mathfrak{Q}} A))$
proof −
let *?ϑ* $= \{(\mathfrak{x},\ o) \rightarrowtail A,\ (\mathfrak{y},\ o) \rightarrowtail B\}$
have *is-tautology* $(\mathfrak{x}_o \supset^{\mathfrak{Q}} (\mathfrak{y}_o \supset^{\mathfrak{Q}} \mathfrak{x}_o))$
by (*fact propositional-principle-of-simplification-is-tautology*)
moreover have *is-pwff-substitution ?ϑ*
using *assms* **by** *auto*
moreover have $A \supset^{\mathfrak{Q}} (B \supset^{\mathfrak{Q}} A) = \mathbf{S}\ ?\vartheta\ (\mathfrak{x}_o \supset^{\mathfrak{Q}} (\mathfrak{y}_o \supset^{\mathfrak{Q}} \mathfrak{x}_o))$
by *simp*
ultimately show *?thesis*
by *blast*
qed

lemma *pseudo-modus-tollens-is-tautologous*:
assumes $A \in wffs_o$ **and** $B \in wffs_o$
shows *is-tautologous* $((A \supset^{\mathfrak{Q}} \sim^{\mathfrak{Q}} B) \supset^{\mathfrak{Q}} (B \supset^{\mathfrak{Q}} \sim^{\mathfrak{Q}} A))$
proof −
let *?ϑ* $= \{(\mathfrak{x},\ o) \rightarrowtail A,\ (\mathfrak{y},\ o) \rightarrowtail B\}$
have *is-tautology* $((\mathfrak{x}_o \supset^{\mathfrak{Q}} \sim^{\mathfrak{Q}} \mathfrak{y}_o) \supset^{\mathfrak{Q}} (\mathfrak{y}_o \supset^{\mathfrak{Q}} \sim^{\mathfrak{Q}} \mathfrak{x}_o))$
using $\mathcal{V}_B$*-simps* **by** (*safe*, *fold is-tv-assignment-def*, *simp only*:) *simp*
moreover have *is-pwff-substitution ?ϑ*
using *assms* **by** *auto*
moreover have $(A \supset^{\mathfrak{Q}} \sim^{\mathfrak{Q}} B) \supset^{\mathfrak{Q}} (B \supset^{\mathfrak{Q}} \sim^{\mathfrak{Q}} A) = \mathbf{S}\ ?\vartheta\ ((\mathfrak{x}_o \supset^{\mathfrak{Q}} \sim^{\mathfrak{Q}} \mathfrak{y}_o) \supset^{\mathfrak{Q}} (\mathfrak{y}_o \supset^{\mathfrak{Q}} \sim^{\mathfrak{Q}} \mathfrak{x}_o))$
by *simp*
ultimately show *?thesis*
by *blast*
qed

end

5 Proof System

theory *Proof-System*
imports
Syntax
begin

5.1 Axioms

inductive-set
axioms :: *form set*
where
axiom-1:
$\mathfrak{g}_{o\to o} \cdot T_o \wedge^{\mathfrak{Q}} \mathfrak{g}_{o\to o} \cdot F_o \equiv^{\mathfrak{Q}} \forall \mathfrak{x}_o.\ \mathfrak{g}_{o\to o} \cdot \mathfrak{x}_o \in axioms$

| *axiom-2*:
  $(\mathfrak{x}_\alpha =_\alpha \mathfrak{y}_\alpha) \supset^{\mathfrak{Q}} (\mathfrak{h}_{\alpha\to o} \cdot \mathfrak{x}_\alpha \equiv^{\mathfrak{Q}} \mathfrak{h}_{\alpha\to o} \cdot \mathfrak{y}_\alpha) \in \mathit{axioms}$
| *axiom-3*:
  $(\mathfrak{f}_{\alpha\to\beta} =_{\alpha\to\beta} \mathfrak{g}_{\alpha\to\beta}) \equiv^{\mathfrak{Q}} \forall \mathfrak{x}_\alpha.\ (\mathfrak{f}_{\alpha\to\beta} \cdot \mathfrak{x}_\alpha =_\beta \mathfrak{g}_{\alpha\to\beta} \cdot \mathfrak{x}_\alpha) \in \mathit{axioms}$
| *axiom-4-1-con*:
  $(\lambda x_\alpha.\ \{\!|c|\!\}_\beta) \cdot A =_\beta \{\!|c|\!\}_\beta \in \mathit{axioms}$ **if** $A \in \mathit{wffs}_\alpha$
| *axiom-4-1-var*:
  $(\lambda x_\alpha.\ y_\beta) \cdot A =_\beta y_\beta \in \mathit{axioms}$ **if** $A \in \mathit{wffs}_\alpha$ **and** $y_\beta \neq x_\alpha$
| *axiom-4-2*:
  $(\lambda x_\alpha.\ x_\alpha) \cdot A =_\alpha A \in \mathit{axioms}$ **if** $A \in \mathit{wffs}_\alpha$
| *axiom-4-3*:
  $(\lambda x_\alpha.\ B \cdot C) \cdot A =_\beta ((\lambda x_\alpha.\ B) \cdot A) \cdot ((\lambda x_\alpha.\ C) \cdot A) \in \mathit{axioms}$
  if $A \in \mathit{wffs}_\alpha$ **and** $B \in \mathit{wffs}_{\gamma\to\beta}$ **and** $C \in \mathit{wffs}_\gamma$
| *axiom-4-4*:
  $(\lambda x_\alpha.\ \lambda y_\gamma.\ B) \cdot A =_{\gamma\to\delta} (\lambda y_\gamma.\ (\lambda x_\alpha.\ B) \cdot A) \in \mathit{axioms}$
  if $A \in \mathit{wffs}_\alpha$ **and** $B \in \mathit{wffs}_\delta$ **and** $(y, \gamma) \notin \{(x, \alpha)\} \cup \mathit{vars}\ A$
| *axiom-4-5*:
  $(\lambda x_\alpha.\ \lambda x_\alpha.\ B) \cdot A =_{\alpha\to\delta} (\lambda x_\alpha.\ B) \in \mathit{axioms}$ **if** $A \in \mathit{wffs}_\alpha$ **and** $B \in \mathit{wffs}_\delta$
| *axiom-5*:
  $\iota \cdot (Q_i \cdot \mathfrak{y}_i) =_i \mathfrak{y}_i \in \mathit{axioms}$

lemma *axioms-are-wffs-of-type-o*:
  shows $\mathit{axioms} \subseteq \mathit{wffs}_o$
  by (*intro subsetI*, *cases rule*: *axioms.cases*) *auto*

5.2 Inference rule R

definition *is-rule-R-app* :: *position* ⇒ *form* ⇒ *form* ⇒ *form* ⇒ *bool* **where**
  [*iff*]: *is-rule-R-app p D C E* ⟷
    (
      ∃ α *A B*.
        $E = A =_\alpha B \wedge A \in \mathit{wffs}_\alpha \wedge B \in \mathit{wffs}_\alpha \wedge$ — E is a well-formed equality
        $A \preceq_p C \wedge$
        $D \in \mathit{wffs}_o \wedge$
        $C\langle\!|p \leftarrow B|\!\rangle \rhd D$
)

lemma *rule-R-original-form-is-wffo*:
  assumes *is-rule-R-app p D C E*
  shows $C \in \mathit{wffs}_o$
  using *assms* **and** *replacement-preserves-typing* **by** *fastforce*

5.3 Proof and derivability

inductive *is-derivable* :: *form* ⇒ *bool* **where**
  dv-axiom: *is-derivable A* **if** $A \in \mathit{axioms}$
| *dv-rule-R*: *is-derivable D* **if** *is-derivable C* **and** *is-derivable E* **and** *is-rule-R-app p D C E*

lemma *derivable-form-is-wffso*:

assumes *is-derivable A*
shows $A \in wffs_o$
using *assms* **and** *axioms-are-wffs-of-type-o* **by** (*fastforce elim*: *is-derivable.cases*)

definition *is-proof-step* :: *form list* ⇒ *nat* ⇒ *bool* **where**
[*iff*]: *is-proof-step* $\mathcal{S}$ $i' \longleftrightarrow$
$\mathcal{S}$! $i' \in$ *axioms* ∨
(∃ *p j k*. {*j*, *k*} ⊆ {*0*..<i'} ∧ *is-rule-R-app p* ($\mathcal{S}$! i') ($\mathcal{S}$! *j*) ($\mathcal{S}$! *k*))

definition *is-proof* :: *form list* ⇒ *bool* **where**
[*iff*]: *is-proof* $\mathcal{S} \longleftrightarrow$ (∀ $i' <$ *length* $\mathcal{S}$. *is-proof-step* $\mathcal{S}$ i')

lemma *common-prefix-is-subproof*:
assumes *is-proof* ($\mathcal{S}$ @ $\mathcal{S}_1$)
and $i' <$ *length* $\mathcal{S}$
shows *is-proof-step* ($\mathcal{S}$ @ $\mathcal{S}_2$) i'
proof −
from *assms*(*2*) **have** ∗: ($\mathcal{S}$ @ $\mathcal{S}_1$) ! i' = ($\mathcal{S}$ @ $\mathcal{S}_2$) ! i'
by (*simp add*: *nth-append*)
moreover from *assms*(*2*) **have** $i' <$ *length* ($\mathcal{S}$ @ $\mathcal{S}_1$)
by *simp*
ultimately obtain *p* **and** *j* **and** *k* **where** ∗∗:
($\mathcal{S}$ @ $\mathcal{S}_1$) ! $i' \in$ *axioms* ∨
{*j*, *k*} ⊆ {*0*..<i'} ∧ *is-rule-R-app p* (($\mathcal{S}$ @ $\mathcal{S}_1$) ! i') (($\mathcal{S}$ @ $\mathcal{S}_1$) ! *j*) (($\mathcal{S}$ @ $\mathcal{S}_1$) ! *k*)
using *assms*(*1*) **by** *fastforce*
then consider
(*axiom*) ($\mathcal{S}$ @ $\mathcal{S}_1$) ! $i' \in$ *axioms*
| (*rule-R*) {*j*, *k*} ⊆ {*0*..<i'} ∧ *is-rule-R-app p* (($\mathcal{S}$ @ $\mathcal{S}_1$) ! i') (($\mathcal{S}$ @ $\mathcal{S}_1$) ! *j*) (($\mathcal{S}$ @ $\mathcal{S}_1$) ! *k*)
by *blast*
then have
($\mathcal{S}$ @ $\mathcal{S}_2$) ! $i' \in$ *axioms* ∨
({*j*, *k*} ⊆ {*0*..<i'} ∧ *is-rule-R-app p* (($\mathcal{S}$ @ $\mathcal{S}_2$) ! i') (($\mathcal{S}$ @ $\mathcal{S}_2$) ! *j*) (($\mathcal{S}$ @ $\mathcal{S}_2$) ! *k*))
proof *cases*
case *axiom*
with ∗ **have** ($\mathcal{S}$ @ $\mathcal{S}_2$) ! $i' \in$ *axioms*
by (*simp only*:)
then show *?thesis* **..**
next
case *rule-R*
with *assms*(*2*) **have** ($\mathcal{S}$ @ $\mathcal{S}_1$) ! *j* = ($\mathcal{S}$ @ $\mathcal{S}_2$) ! *j* **and** ($\mathcal{S}$ @ $\mathcal{S}_1$) ! *k* = ($\mathcal{S}$ @ $\mathcal{S}_2$) ! *k*
by (*simp-all add*: *nth-append*)
then have {*j*, *k*} ⊆ {*0*..<i'} ∧ *is-rule-R-app p* (($\mathcal{S}$ @ $\mathcal{S}_2$) ! i') (($\mathcal{S}$ @ $\mathcal{S}_2$) ! *j*) (($\mathcal{S}$ @ $\mathcal{S}_2$) ! *k*)
using ∗ **and** *rule-R* **by** *simp*
then show *?thesis* **..**
qed
with ∗∗ **show** *?thesis*
by *fastforce*
qed

```
lemma added-suffix-proof-preservation:
  assumes is-proof 𝒮
  and i' < length (𝒮 @ 𝒮') − length 𝒮'
  shows is-proof-step (𝒮 @ 𝒮') i'
  using assms and common-prefix-is-subproof[where 𝒮₁ = []] by simp

lemma append-proof-step-is-proof:
  assumes is-proof 𝒮
  and is-proof-step (𝒮 @ [A]) (length (𝒮 @ [A]) − 1)
  shows is-proof (𝒮 @ [A])
  using assms and added-suffix-proof-preservation by (simp add: All-less-Suc)

lemma added-prefix-proof-preservation:
  assumes is-proof 𝒮'
  and i' ∈ {length 𝒮..<length (𝒮 @ 𝒮')}
  shows is-proof-step (𝒮 @ 𝒮') i'
proof −
  let ?𝒮 = 𝒮 @ 𝒮'
  let ?i = i' − length 𝒮
  from assms(2) have ?𝒮 ! i' = 𝒮' ! ?i and ?i < length 𝒮'
    by (simp-all add: nth-append less-diff-conv2)
  then have is-proof-step ?𝒮 i' = is-proof-step 𝒮' ?i
  proof −
    from assms(1) and ‹?i < length 𝒮'› obtain j and k and p where ∗:
      𝒮' ! ?i ∈ axioms ∨ ({j, k} ⊆ {0..<?i} ∧ is-rule-R-app p (𝒮' ! ?i) (𝒮' ! j) (𝒮' ! k))
      by fastforce
    then consider
      (axiom) 𝒮' ! ?i ∈ axioms
    | (rule-R) {j, k} ⊆ {0..<?i} ∧ is-rule-R-app p (𝒮' ! ?i) (𝒮' ! j) (𝒮' ! k)
      by blast
    then have
      ?𝒮 ! i' ∈ axioms ∨
      (
        {j + length 𝒮, k + length 𝒮} ⊆ {0..<i'} ∧
        is-rule-R-app p (?𝒮 ! i') (?𝒮 ! (j + length 𝒮)) (?𝒮 ! (k + length 𝒮))
      )
    proof cases
      case axiom
      with ‹?𝒮 ! i' = 𝒮' ! ?i› have ?𝒮 ! i' ∈ axioms
        by (simp only:)
      then show ?thesis ..
    next
      case rule-R
      with assms(2) have ?𝒮 ! (j + length 𝒮) = 𝒮' ! j and ?𝒮 ! (k + length 𝒮) = 𝒮' ! k
        by (simp-all add: nth-append)
      with ‹?𝒮 ! i' = 𝒮' ! ?i› and rule-R have
        {j + length 𝒮, k + length 𝒮} ⊆ {0..<i'} ∧
        is-rule-R-app p (?𝒮 ! i') (?𝒮 ! (j + length 𝒮)) (?𝒮 ! (k + length 𝒮))
        by auto
```

```
    then show ?thesis ..
  qed
  with ∗ show ?thesis
    by fastforce
 qed
 with assms(1) and ‹?i < length S'› show ?thesis
  by simp
qed

lemma proof-but-last-is-proof:
 assumes is-proof (S @ [A])
 shows is-proof S
 using assms and common-prefix-is-subproof[where S₁ = [A] and S₂ = []] by simp

lemma proof-prefix-is-proof:
 assumes is-proof (S₁ @ S₂)
 shows is-proof S₁
 using assms and proof-but-last-is-proof
 by (induction S₂ arbitrary: S₁ rule: rev-induct) (simp, metis append.assoc)

lemma single-axiom-is-proof:
 assumes A ∈ axioms
 shows is-proof [A]
 using assms by fastforce

lemma proofs-concatenation-is-proof:
 assumes is-proof S₁ and is-proof S₂
 shows is-proof (S₁ @ S₂)
proof −
 from assms(1) have ∀ i' < length S₁. is-proof-step (S₁ @ S₂) i'
  using added-suffix-proof-preservation by auto
 moreover from assms(2) have ∀ i' ∈ {length S₁..<length (S₁ @ S₂)}. is-proof-step (S₁ @ S₂) i'
  using added-prefix-proof-preservation by auto
 ultimately show ?thesis
  unfolding is-proof-def by (meson atLeastLessThan-iff linorder-not-le)
qed

lemma elem-of-proof-is-wffo:
 assumes is-proof S and A ∈ lset S
 shows A ∈ wffs_o
 using assms and axioms-are-wffs-of-type-o
 unfolding is-rule-R-app-def and is-proof-step-def and is-proof-def
 by (induction S) (simp, metis (full-types) in-mono in-set-conv-nth)

lemma axiom-prepended-to-proof-is-proof:
 assumes is-proof S
 and A ∈ axioms
 shows is-proof ([A] @ S)
 using proofs-concatenation-is-proof[OF single-axiom-is-proof[OF assms(2)] assms(1)] .
```

lemma *axiom-appended-to-proof-is-proof*:
assumes *is-proof* $\mathcal{S}$
and $A \in$ *axioms*
shows *is-proof* ($\mathcal{S}$ @ [A])
using *proofs-concatenation-is-proof*[*OF assms*(*1*) *single-axiom-is-proof*[*OF assms*(*2*)]] .

lemma *rule-R-app-appended-to-proof-is-proof*:
assumes *is-proof* $\mathcal{S}$
and $i_C <$ *length* $\mathcal{S}$ **and** $\mathcal{S}$! $i_C = C$
and $i_E <$ *length* $\mathcal{S}$ **and** $\mathcal{S}$! $i_E = E$
and *is-rule-R-app p D C E*
shows *is-proof* ($\mathcal{S}$ @ [D])
proof −
let *?*$\mathcal{S}$ = $\mathcal{S}$ @ [D]
let *?*i_D = *length* $\mathcal{S}$
from *assms*(*2*,*4*) **have** $i_C <$ *?*i_D **and** $i_E <$ *?*i_D
by *fastforce*+
with *assms*(*3*,*5*,*6*) **have** *is-rule-R-app p* (*?*$\mathcal{S}$! *?*i_D) (*?*$\mathcal{S}$! i_C) (*?*$\mathcal{S}$! i_E)
by (*simp add*: *nth-append*)
with *assms*(*2*,*4*) **have** $\exists\, p\ j\ k.\ \{j, k\} \subseteq \{0..<$*?*$i_D\} \wedge$ *is-rule-R-app p* (*?*$\mathcal{S}$! *?*i_D) (*?*$\mathcal{S}$! j) (*?*$\mathcal{S}$! k)
by *fastforce*
then have *is-proof-step ?*$\mathcal{S}$ (*length ?*$\mathcal{S}$ − *1*)
by *simp*
moreover from *assms*(*1*) **have** $\forall\, i' <$ *length ?*$\mathcal{S}$ − *1*. *is-proof-step ?*$\mathcal{S}$ i'
using *added-suffix-proof-preservation* **by** *auto*
ultimately show *?thesis*
using *less-Suc-eq* **by** *auto*
qed

definition *is-proof-of* :: *form list* ⇒ *form* ⇒ *bool* **where**
[*iff*]: *is-proof-of* $\mathcal{S}$ $A \longleftrightarrow \mathcal{S} \neq [] \wedge$ *is-proof* $\mathcal{S} \wedge$ *last* $\mathcal{S} = A$

lemma *proof-prefix-is-proof-of-last*:
assumes *is-proof* ($\mathcal{S}$ @ $\mathcal{S}'$) **and** $\mathcal{S} \neq []$
shows *is-proof-of* $\mathcal{S}$ (*last* $\mathcal{S}$)
proof −
from *assms*(*1*) **have** *is-proof* $\mathcal{S}$
by (*fact proof-prefix-is-proof*)
with *assms*(*2*) **show** *?thesis*
by *fastforce*
qed

definition *is-theorem* :: *form* ⇒ *bool* **where**
[*iff*]: *is-theorem* $A \longleftrightarrow (\exists \mathcal{S}.$ *is-proof-of* $\mathcal{S}$ $A)$

lemma *proof-form-is-wffo*:
assumes *is-proof-of* $\mathcal{S}$ A
and $B \in$ *lset* $\mathcal{S}$

shows $B \in \mathit{wffs}_o$
using *assms* **and** *elem-of-proof-is-wffo* **by** *blast*

lemma *proof-form-is-theorem*:
assumes *is-proof* $\mathcal{S}$ **and** $\mathcal{S} \neq []$
and $i' < \mathit{length}\ \mathcal{S}$
shows *is-theorem* $(\mathcal{S}\ !\ i')$
proof −
let $?\mathcal{S}_1 = \mathit{take}\ (\mathit{Suc}\ i')\ \mathcal{S}$
from *assms*(*1*) **obtain** $\mathcal{S}_2$ **where** *is-proof* $(?\mathcal{S}_1\ @\ \mathcal{S}_2)$
by (*metis append-take-drop-id*)
then have *is-proof* $?\mathcal{S}_1$
by (*fact proof-prefix-is-proof*)
moreover from *assms*(*3*) **have** *last* $?\mathcal{S}_1 = \mathcal{S}\ !\ i'$
by (*simp add*: *take-Suc-conv-app-nth*)
ultimately show *?thesis*
using *assms*(*2*) **unfolding** *is-proof-of-def* **and** *is-theorem-def* **by** (*metis Zero-neq-Suc take-eq-Nil2*)
qed

theorem *derivable-form-is-theorem*:
assumes *is-derivable A*
shows *is-theorem A*
using *assms* **proof** (*induction rule*: *is-derivable.induct*)
case (*dv-axiom A*)
then have *is-proof* $[A]$
by (*fact single-axiom-is-proof*)
moreover have *last* $[A] = A$
by *simp*
ultimately show *?case*
by *blast*
next
case (*dv-rule-R C E p D*)
obtain $\mathcal{S}_C$ **and** $\mathcal{S}_E$ **where**
is-proof $\mathcal{S}_C$ **and** $\mathcal{S}_C \neq []$ **and** *last* $\mathcal{S}_C = C$ **and**
is-proof $\mathcal{S}_E$ **and** $\mathcal{S}_E \neq []$ **and** *last* $\mathcal{S}_E = E$
using *dv-rule-R.IH* **by** *fastforce*
let $?i_C = \mathit{length}\ \mathcal{S}_C - 1$ **and** $?i_E = \mathit{length}\ \mathcal{S}_C + \mathit{length}\ \mathcal{S}_E - 1$ **and** $?i_D = \mathit{length}\ \mathcal{S}_C + \mathit{length}\ \mathcal{S}_E$
let $?\mathcal{S} = \mathcal{S}_C\ @\ \mathcal{S}_E\ @\ [D]$
from $\langle \mathcal{S}_C \neq [] \rangle$ **have** $?i_C < \mathit{length}\ (\mathcal{S}_C\ @\ \mathcal{S}_E)$ **and** $?i_E < \mathit{length}\ (\mathcal{S}_C\ @\ \mathcal{S}_E)$
using *linorder-not-le* **by** *fastforce*+
moreover have $(\mathcal{S}_C\ @\ \mathcal{S}_E)\ !\ ?i_C = C$ **and** $(\mathcal{S}_C\ @\ \mathcal{S}_E)\ !\ ?i_E = E$
using $\langle \mathcal{S}_C \neq [] \rangle$ **and** $\langle \mathit{last}\ \mathcal{S}_C = C \rangle$
by
(
simp add: *last-conv-nth nth-append*,
metis $\langle \mathit{last}\ \mathcal{S}_E = E \rangle$ $\langle \mathcal{S}_E \neq [] \rangle$ *append-is-Nil-conv last-appendR last-conv-nth length-append*
)
with $\langle$*is-rule-R-app p D C E*$\rangle$ **have** *is-rule-R-app p D* $((\mathcal{S}_C\ @\ \mathcal{S}_E)\ !\ ?i_C)$ $((\mathcal{S}_C\ @\ \mathcal{S}_E)\ !\ ?i_E)$

```
    using ‹(𝒮_C @ 𝒮_E) ! ?i_C = C› by fastforce
  moreover from ‹is-proof 𝒮_C› and ‹is-proof 𝒮_E› have is-proof (𝒮_C @ 𝒮_E)
    by (fact proofs-concatenation-is-proof)
  ultimately have is-proof ((𝒮_C @ 𝒮_E) @ [D])
    using rule-R-app-appended-to-proof-is-proof by presburger
  with ‹𝒮_C ≠ []› show ?case
    unfolding is-proof-of-def and is-theorem-def by (metis snoc-eq-iff-butlast)
qed

theorem theorem-is-derivable-form:
  assumes is-theorem A
  shows is-derivable A
proof −
  from assms obtain 𝒮 where is-proof 𝒮 and 𝒮 ≠ [] and last 𝒮 = A
    by fastforce
  then show ?thesis
  proof (induction length 𝒮 arbitrary: 𝒮 A rule: less-induct)
    case less
    let ?i' = length 𝒮 − 1
    from ‹𝒮 ≠ []› and ‹last 𝒮 = A› have 𝒮 ! ?i' = A
      by (simp add: last-conv-nth)
    from ‹is-proof 𝒮› and ‹𝒮 ≠ []› and ‹last 𝒮 = A› have is-proof-step 𝒮 ?i'
      using added-suffix-proof-preservation[where 𝒮' = []] by simp
    then consider
      (axiom) 𝒮 ! ?i' ∈ axioms
    | (rule-R) ∃ p j k. {j, k} ⊆ {0..<?i'} ∧ is-rule-R-app p (𝒮 ! ?i') (𝒮 ! j) (𝒮 ! k)
      by fastforce
    then show ?case
    proof cases
      case axiom
      with ‹𝒮 ! ?i' = A› show ?thesis
        by (fastforce intro: dv-axiom)
    next
      case rule-R
      then obtain p and j and k
        where {j, k} ⊆ {0..<?i'} and is-rule-R-app p (𝒮 ! ?i') (𝒮 ! j) (𝒮 ! k)
        by force
      let ?𝒮_j = take (Suc j) 𝒮
      let ?𝒮_k = take (Suc k) 𝒮
      obtain 𝒮_j' and 𝒮_k' where 𝒮 = ?𝒮_j @ 𝒮_j' and 𝒮 = ?𝒮_k @ 𝒮_k'
        by (metis append-take-drop-id)
      with ‹is-proof 𝒮› have is-proof (?𝒮_j @ 𝒮_j') and is-proof (?𝒮_k @ 𝒮_k')
        by (simp-all only:)
      moreover
      from ‹𝒮 = ?𝒮_j @ 𝒮_j'› and ‹𝒮 = ?𝒮_k @ 𝒮_k'› and ‹last 𝒮 = A› and ‹{j, k} ⊆ {0..<length 𝒮 −
1}›
      have last 𝒮_j' = A and last 𝒮_k' = A
        using length-Cons and less-le-not-le and take-Suc and take-tl and append.right-neutral
        by (metis atLeastLessThan-iff diff-Suc-1 insert-subset last-appendR take-all-iff)+
```

```
    moreover from ‹𝒮 ≠ []› have ?𝒮_j ≠ [] and ?𝒮_k ≠ []
      by simp-all
    ultimately have is-proof-of ?𝒮_j (last ?𝒮_j) and is-proof-of ?𝒮_k (last ?𝒮_k)
      using proof-prefix-is-proof-of-last [where 𝒮 = ?𝒮_j and 𝒮' = 𝒮_j']
      and proof-prefix-is-proof-of-last [where 𝒮 = ?𝒮_k and 𝒮' = 𝒮_k']
      by fastforce+
    moreover from ‹last 𝒮_j' = A› and ‹last 𝒮_k' = A›
    have length ?𝒮_j < length 𝒮 and length ?𝒮_k < length 𝒮
      using ‹{j, k} ⊆ {0..<length 𝒮 − 1}› by force+
    moreover from calculation(3,4) have last ?𝒮_j = 𝒮 ! j and last ?𝒮_k = 𝒮 ! k
      by (metis Suc-lessD last-snoc linorder-not-le nat-neq-iff take-Suc-conv-app-nth take-all-iff)+
    ultimately have is-derivable (𝒮 ! j) and is-derivable (𝒮 ! k)
      using ‹?𝒮_j ≠ []› and ‹?𝒮_k ≠ []› and less(1) by blast+
    with ‹is-rule-R-app p (𝒮 ! ?i') (𝒮 ! j) (𝒮 ! k)› and ‹𝒮 ! ?i' = A› show ?thesis
      by (blast intro: dv-rule-R)
  qed
 qed
qed

theorem theoremhood-derivability-equivalence:
 shows is-theorem A ⟷ is-derivable A
 using derivable-form-is-theorem and theorem-is-derivable-form by blast

lemma theorem-is-wffo:
 assumes is-theorem A
 shows A ∈ wffs_o
proof −
 from assms obtain 𝒮 where is-proof-of 𝒮 A
   by blast
 then have A ∈ lset 𝒮
   by auto
 with ‹is-proof-of 𝒮 A› show ?thesis
   using proof-form-is-wffo by blast
qed

lemma equality-reflexivity:
 assumes A ∈ wffs_α
 shows is-theorem (A =_α A) (is is-theorem ?A_2)
proof −
 let ?A_1 = (λ𝔯_α. 𝔯_α) ▪ A =_α A
 let ?𝒮 = [?A_1, ?A_2]
 — (.1) Axiom 4.2
 have is-proof-step ?𝒮 0
 proof −
   from assms have ?A_1 ∈ axioms
     by (intro axiom-4-2)
   then show ?thesis
     by simp
 qed
```

```
  — (.2) Rule R: .1,.1
  moreover have is-proof-step ?S 1
  proof −
    let ?p = [«, »]
    have ∃ p j k. {j::nat, k} ⊆ {0..<1} ∧ is-rule-R-app ?p ?A₂ (?S ! j) (?S ! k)
    proof −
      let ?D = ?A₂ and ?j = 0::nat and ?k = 0
      have {?j, ?k} ⊆ {0..<1}
        by simp
      moreover have is-rule-R-app ?p ?A₂ (?S ! ?j) (?S ! ?k)
      proof −
        have (λ𝔯α. 𝔯α) ▪ A ⪯?p (?S ! ?j)
          by force
        moreover have (?S ! ?j)⟪?p ← A⟫ ▷ ?D
          by force
        moreover from ‹A ∈ wffsα› have ?D ∈ wffso
          by (intro equality-wff)
        moreover from ‹A ∈ wffsα› have (λ𝔯α. 𝔯α) ▪ A ∈ wffsα
          by (meson wffs-of-type-simps)
        ultimately show ?thesis
          using ‹A ∈ wffsα› by simp
      qed
      ultimately show ?thesis
        by meson
    qed
    then show ?thesis
      by auto
  qed
  moreover have last ?S = ?A₂
    by simp
  moreover have {0..<length ?S} = {0, 1}
    by (simp add: atLeast0-lessThan-Suc insert-commute)
  ultimately show ?thesis
    unfolding is-theorem-def and is-proof-def and is-proof-of-def
    by (metis One-nat-def Suc-1 length-Cons less-2-cases list.distinct(1) list.size(3))
qed

lemma equality-reflexivity′:
  assumes A ∈ wffsα
  shows is-theorem (A =α A) (is is-theorem ?A₂)
proof (intro derivable-form-is-theorem)
  let ?A₁ = (λ𝔯α. 𝔯α) ▪ A =α A
  — (.1) Axiom 4.2
  from assms have ?A₁ ∈ axioms
    by (intro axiom-4-2)
  then have step-1: is-derivable ?A₁
    by (intro dv-axiom)
  — (.2) Rule R: .1,.1
  then show is-derivable ?A₂
```

```
  proof −
    let ?p = [«, »] and ?C = ?A₁ and ?E = ?A₁ and ?D = ?A₂
    have is-rule-R-app ?p ?D ?C ?E
    proof −
      have (λ𝔵α. 𝔵α) ▪ A ≼?p ?C
        by force
      moreover have ?C⟨?p ← A⟩ ▷ ?D
        by force
      moreover from ‹A ∈ wffsα› have ?D ∈ wffso
        by (intro equality-wff)
      moreover from ‹A ∈ wffsα› have (λ𝔵α. 𝔵α) ▪ A ∈ wffsα
        by (meson wffs-of-type-simps)
      ultimately show ?thesis
        using ‹A ∈ wffsα› by simp
    qed
    with step-1 show ?thesis
      by (blast intro: dv-rule-R)
  qed
qed
```

5.4 Hypothetical proof and derivability

The set of free variables in $\mathcal{X}$ that are exposed to capture at position p in A:

```
definition capture-exposed-vars-at :: position ⇒ form ⇒ 'a ⇒ var set where
  [simp]: capture-exposed-vars-at p A 𝒳 =
    {(x, β) | x β p' E. strict-prefix p' p ∧ λxβ. E ≼p' A ∧ (x, β) ∈ free-vars 𝒳}
```

```
lemma capture-exposed-vars-at-alt-def:
  assumes p ∈ positions A
  shows capture-exposed-vars-at p A 𝒳 = binders-at A p ∩ free-vars 𝒳
  unfolding binders-at-alt-def[OF assms] and in-scope-of-abs-alt-def
  using is-subform-implies-in-positions by auto
```

Inference rule R′:

```
definition rule-R'-side-condition :: form set ⇒ position ⇒ form ⇒ form ⇒ form ⇒ bool where
  [iff]: rule-R'-side-condition ℋ p D C E ⟷
    capture-exposed-vars-at p C E ∩ capture-exposed-vars-at p C ℋ = {}
```

```
lemma rule-R'-side-condition-alt-def:
  fixes ℋ :: form set
  assumes C ∈ wffsα
  shows
    rule-R'-side-condition ℋ p D C (A =α B)
    ⟷
    (
      ∄x β E p'.
        strict-prefix p' p ∧
        λxβ. E ≼p' C ∧
```

$(x,\ \beta) \in$ *free-vars* $(A =_\alpha B) \wedge$
$(\exists\, H \in \mathcal{H}.\ (x,\ \beta) \in$ *free-vars* $H)$
)
proof −
have
capture-exposed-vars-at $p\ C\ (A =_\alpha B)$
=
$\{(x,\ \beta) \mid x\ \beta\ p'\ E.$ *strict-prefix* $p'\ p \wedge \lambda x_\beta.\ E \preceq_{p'} C \wedge (x,\ \beta) \in$ *free-vars* $(A =_\alpha B)\}$
using *assms* **and** *capture-exposed-vars-at-alt-def* **unfolding** *capture-exposed-vars-at-def* **by** *fast*
moreover have
capture-exposed-vars-at $p\ C\ \mathcal{H}$
=
$\{(x,\ \beta) \mid x\ \beta\ p'\ E.$ *strict-prefix* $p'\ p \wedge \lambda x_\beta.\ E \preceq_{p'} C \wedge (x,\ \beta) \in$ *free-vars* $\mathcal{H}\}$
using *assms* **and** *capture-exposed-vars-at-alt-def* **unfolding** *capture-exposed-vars-at-def* **by** *fast*
ultimately have
capture-exposed-vars-at $p\ C\ (A =_\alpha B) \cap$ *capture-exposed-vars-at* $p\ C\ \mathcal{H}$
=
$\{(x,\ \beta) \mid x\ \beta\ p'\ E.$ *strict-prefix* $p'\ p \wedge \lambda x_\beta.\ E \preceq_{p'} C \wedge (x,\ \beta) \in$ *free-vars* $(A =_\alpha B) \wedge$
$(x,\ \beta) \in$ *free-vars* $\mathcal{H}\}$
by *auto*
also have
...
=
$\{(x,\ \beta) \mid x\ \beta\ p'\ E.$ *strict-prefix* $p'\ p \wedge \lambda x_\beta.\ E \preceq_{p'} C \wedge (x,\ \beta) \in$ *free-vars* $(A =_\alpha B) \wedge$
$(\exists\, H \in \mathcal{H}.\ (x,\ \beta) \in$ *free-vars* $H)\}$
by *auto*
finally show *?thesis*
by *fast*
qed

definition *is-rule-R'-app* :: *form set* ⇒ *position* ⇒ *form* ⇒ *form* ⇒ *form* ⇒ *bool* **where**
[*iff*]: *is-rule-R'-app* $\mathcal{H}\ p\ D\ C\ E \longleftrightarrow$ *is-rule-R-app* $p\ D\ C\ E \wedge$ *rule-R'-side-condition* $\mathcal{H}\ p\ D\ C\ E$

lemma *is-rule-R'-app-alt-def*:
shows
is-rule-R'-app $\mathcal{H}\ p\ D\ C\ E$
$\longleftrightarrow$
(
$\exists\, \alpha\ A\ B.$
$E = A =_\alpha B \wedge A \in$ *wffs*$_\alpha \wedge B \in$ *wffs*$_\alpha \wedge$ — E is a well-formed equality
$A \preceq_p C \wedge D \in$ *wffs*$_o \wedge$
$C\langle\!|p \leftarrow B|\!\rangle \rhd D \wedge$
(
$\nexists\, x\ \beta\ E\ p'.$
strict-prefix $p'\ p \wedge$
$\lambda x_\beta.\ E \preceq_{p'} C \wedge$
$(x,\ \beta) \in$ *free-vars* $(A =_\alpha B) \wedge$
$(\exists\, H \in \mathcal{H}.\ (x,\ \beta) \in$ *free-vars* $H)$
)

)
using *rule-R′-side-condition-alt-def* **by** *fastforce*

lemma *rule-R′-preserves-typing*:
assumes *is-rule-R′-app* $\mathcal{H}$ *p D C E*
shows $C \in wffs_o \longleftrightarrow D \in wffs_o$
using *assms* **and** *replacement-preserves-typing* **unfolding** *is-rule-R-app-def* **and** *is-rule-R′-app-def*
by *meson*

abbreviation *is-hyps* :: *form set* ⇒ *bool* **where**
is-hyps $\mathcal{H} \equiv \mathcal{H} \subseteq wffs_o \wedge$ *finite* $\mathcal{H}$

inductive *is-derivable-from-hyps* :: *form set* ⇒ *form* ⇒ *bool* (- ⊢ - [*50*, *50*] *50*) **for** $\mathcal{H}$ **where**
dv-hyp: $\mathcal{H} \vdash A$ **if** $A \in \mathcal{H}$ **and** *is-hyps* $\mathcal{H}$
| *dv-thm*: $\mathcal{H} \vdash A$ **if** *is-theorem A* **and** *is-hyps* $\mathcal{H}$
| *dv-rule-R′*: $\mathcal{H} \vdash D$ **if** $\mathcal{H} \vdash C$ **and** $\mathcal{H} \vdash E$ **and** *is-rule-R′-app* $\mathcal{H}$ *p D C E* **and** *is-hyps* $\mathcal{H}$

lemma *hyp-derivable-form-is-wffso*:
assumes *is-derivable-from-hyps* $\mathcal{H}$ *A*
shows $A \in wffs_o$
using *assms* **and** *theorem-is-wffo* **by** (*cases rule*: *is-derivable-from-hyps.cases*) *auto*

definition *is-hyp-proof-step* :: *form set* ⇒ *form list* ⇒ *form list* ⇒ *nat* ⇒ *bool* **where**
[*iff*]: *is-hyp-proof-step* $\mathcal{H}$ $\mathcal{S}_1$ $\mathcal{S}_2$ $i' \longleftrightarrow$
$\mathcal{S}_2$! $i' \in \mathcal{H} \vee$
$\mathcal{S}_2$! $i' \in$ *lset* $\mathcal{S}_1 \vee$
$(\exists\, p\ j\ k.\ \{j, k\} \subseteq \{0..<i'\} \wedge$ *is-rule-R′-app* $\mathcal{H}$ *p* ($\mathcal{S}_2$! i') ($\mathcal{S}_2$! j) ($\mathcal{S}_2$! k))

type-synonym *hyp-proof* = *form list* × *form list*

definition *is-hyp-proof* :: *form set* ⇒ *form list* ⇒ *form list* ⇒ *bool* **where**
[*iff*]: *is-hyp-proof* $\mathcal{H}$ $\mathcal{S}_1$ $\mathcal{S}_2 \longleftrightarrow (\forall\, i' <$ *length* $\mathcal{S}_2$. *is-hyp-proof-step* $\mathcal{H}$ $\mathcal{S}_1$ $\mathcal{S}_2$ $i')$

lemma *common-prefix-is-hyp-subproof-from*:
assumes *is-hyp-proof* $\mathcal{H}$ $\mathcal{S}_1$ ($\mathcal{S}_2$ @ $\mathcal{S}_2{}'$)
and $i' <$ *length* $\mathcal{S}_2$
shows *is-hyp-proof-step* $\mathcal{H}$ $\mathcal{S}_1$ ($\mathcal{S}_2$ @ $\mathcal{S}_2{}''$) i'
proof −
let *?S* = $\mathcal{S}_2$ @ $\mathcal{S}_2{}'$
from *assms*(*2*) **have** *?S* ! i' = ($\mathcal{S}_2$ @ $\mathcal{S}_2{}''$) ! i'
by (*simp add*: *nth-append*)
moreover from *assms*(*2*) **have** $i' <$ *length ?S*
by *simp*
ultimately obtain *p* **and** *j* **and** *k* **where**
?S ! $i' \in \mathcal{H} \vee$
?S ! $i' \in$ *lset* $\mathcal{S}_1 \vee$
$\{j, k\} \subseteq \{0..<i'\} \wedge$ *is-rule-R′-app* $\mathcal{H}$ *p* (*?S* ! i') (*?S* ! j) (*?S* ! k)
using *assms*(*1*) **unfolding** *is-hyp-proof-def* **and** *is-hyp-proof-step-def* **by** *meson*
then consider

```
    (hyp) ?S ! i' ∈ H
  | (seq) ?S ! i' ∈ lset S₁
  | (rule-R') {j, k} ⊆ {0..<i'} ∧ is-rule-R'-app H p (?S ! i') (?S ! j) (?S ! k)
    by blast
  then have
    (S₂ @ S₂'') ! i' ∈ H ∨
    (S₂ @ S₂'') ! i' ∈ lset S₁ ∨
    ({j, k} ⊆ {0..<i'} ∧ is-rule-R'-app H p ((S₂ @ S₂'') ! i') ((S₂ @ S₂'') ! j) ((S₂ @ S₂'') ! k))
  proof cases
    case hyp
    with assms(2) have (S₂ @ S₂'') ! i' ∈ H
      by (simp add: nth-append)
    then show ?thesis ..
  next
    case seq
    with assms(2) have (S₂ @ S₂'') ! i' ∈ lset S₁
      by (simp add: nth-append)
    then show ?thesis
      by (intro disjI1 disjI2)
  next
    case rule-R'
    with assms(2) have ?S ! j = (S₂ @ S₂'') ! j and ?S ! k = (S₂ @ S₂'') ! k
      by (simp-all add: nth-append)
    with assms(2) and rule-R' have
      {j, k} ⊆ {0..<i'} ∧ is-rule-R'-app H p ((S₂ @ S₂'') ! i') ((S₂ @ S₂'') ! j) ((S₂ @ S₂'') ! k)
      by (metis nth-append)
    then show ?thesis
      by (intro disjI2)
  qed
  then show ?thesis
    unfolding is-hyp-proof-step-def by meson
qed

lemma added-suffix-thms-hyp-proof-preservation:
  assumes is-hyp-proof H S₁ S₂
  shows is-hyp-proof H (S₁ @ S₁') S₂
  using assms by auto

lemma added-suffix-hyp-proof-preservation:
  assumes is-hyp-proof H S₁ S₂
  and i' < length (S₂ @ S₂') − length S₂'
  shows is-hyp-proof-step H S₁ (S₂ @ S₂') i'
  using assms and common-prefix-is-hyp-subproof-from[where S₂' = []] by auto

lemma appended-hyp-proof-step-is-hyp-proof:
  assumes is-hyp-proof H S₁ S₂
  and is-hyp-proof-step H S₁ (S₂ @ [A]) (length (S₂ @ [A]) − 1)
  shows is-hyp-proof H S₁ (S₂ @ [A])
proof (standard, intro allI impI)
```

fix i'
assume $i' < \mathit{length}\ (\mathcal{S}_2\ @\ [A])$
then consider $(a)\ i' < \mathit{length}\ \mathcal{S}_2 \mid (b)\ i' = \mathit{length}\ \mathcal{S}_2$
by *fastforce*
then show *is-hyp-proof-step* $\mathcal{H}\ \mathcal{S}_1\ (\mathcal{S}_2\ @\ [A])\ i'$
proof *cases*
case *a*
with *assms*(*1*) **show** *?thesis*
using *added-suffix-hyp-proof-preservation* **by** *simp*
next
case *b*
with *assms*(*2*) **show** *?thesis*
by *simp*
qed
qed

lemma *added-prefix-hyp-proof-preservation*:
assumes *is-hyp-proof* $\mathcal{H}\ \mathcal{S}_1\ \mathcal{S}_2{}'$
and $i' \in \{\mathit{length}\ \mathcal{S}_2..<\mathit{length}\ (\mathcal{S}_2\ @\ \mathcal{S}_2{}')\}$
shows *is-hyp-proof-step* $\mathcal{H}\ \mathcal{S}_1\ (\mathcal{S}_2\ @\ \mathcal{S}_2{}')\ i'$
proof −
let $?\mathcal{S} = \mathcal{S}_2\ @\ \mathcal{S}_2{}'$
let $?i = i' - \mathit{length}\ \mathcal{S}_2$
from *assms*(*2*) **have** $?\mathcal{S}\ !\ i' = \mathcal{S}_2{}'\ !\ ?i$ **and** $?i < \mathit{length}\ \mathcal{S}_2{}'$
by (*simp-all add*: *nth-append less-diff-conv2*)
then have *is-hyp-proof-step* $\mathcal{H}\ \mathcal{S}_1\ ?\mathcal{S}\ i' =$ *is-hyp-proof-step* $\mathcal{H}\ \mathcal{S}_1\ \mathcal{S}_2{}'\ ?i$
proof −
from *assms*(*1*) **and** ‹$?i < \mathit{length}\ \mathcal{S}_2{}'$› **obtain** j **and** k **and** p **where**
$\mathcal{S}_2{}'\ !\ ?i \in \mathcal{H} \vee$
$\mathcal{S}_2{}'\ !\ ?i \in \mathit{lset}\ \mathcal{S}_1 \vee$
$(\{j, k\} \subseteq \{0..<?i\} \wedge$ *is-rule-R′-app* $\mathcal{H}\ p\ (\mathcal{S}_2{}'\ !\ ?i)\ (\mathcal{S}_2{}'\ !\ j)\ (\mathcal{S}_2{}'\ !\ k))$
unfolding *is-hyp-proof-def* **and** *is-hyp-proof-step-def* **by** *meson*
then consider
(*hyp*) $\mathcal{S}_2{}'\ !\ ?i \in \mathcal{H}$
| (*seq*) $\mathcal{S}_2{}'\ !\ ?i \in \mathit{lset}\ \mathcal{S}_1$
| (*rule-R′*) $\{j, k\} \subseteq \{0..<?i\} \wedge$ *is-rule-R′-app* $\mathcal{H}\ p\ (\mathcal{S}_2{}'\ !\ ?i)\ (\mathcal{S}_2{}'\ !\ j)\ (\mathcal{S}_2{}'\ !\ k)$
by *blast*
then have
$?\mathcal{S}\ !\ i' \in \mathcal{H} \vee$
$?\mathcal{S}\ !\ i' \in \mathit{lset}\ \mathcal{S}_1 \vee$
$(\{j + \mathit{length}\ \mathcal{S}_2, k + \mathit{length}\ \mathcal{S}_2\} \subseteq \{0..<i'\} \wedge$
is-rule-R′-app $\mathcal{H}\ p\ (?\mathcal{S}\ !\ i')\ (?\mathcal{S}\ !\ (j + \mathit{length}\ \mathcal{S}_2))\ (?\mathcal{S}\ !\ (k + \mathit{length}\ \mathcal{S}_2)))$
proof *cases*
case *hyp*
with ‹$?\mathcal{S}\ !\ i' = \mathcal{S}_2{}'\ !\ ?i$› **have** $?\mathcal{S}\ !\ i' \in \mathcal{H}$
by (*simp only*:)
then show *?thesis* **..**
next
case *seq*

```
    with ‹?S ! i' = S2' ! ?i› have ?S ! i' ∈ lset S1
      by (simp only:)
    then show ?thesis
      by (intro disjI1 disjI2)
  next
    case rule-R'
    with assms(2) have ?S ! (j + length S2) = S2' ! j and ?S ! (k + length S2) = S2' ! k
      by (simp-all add: nth-append)
    with ‹?S ! i' = S2' ! ?i› and rule-R' have
      {j + length S2, k + length S2} ⊆ {0..<i'} ∧
      is-rule-R'-app H p (?S ! i') (?S ! (j + length S2)) (?S ! (k + length S2))
      by auto
    then show ?thesis
      by (intro disjI2)
  qed
  with assms(1) and ‹?i < length S2'› show ?thesis
    unfolding is-hyp-proof-def and is-hyp-proof-step-def by meson
qed
with assms(1) and ‹?i < length S2'› show ?thesis
  by simp
qed

lemma hyp-proof-but-last-is-hyp-proof:
  assumes is-hyp-proof H S1 (S2 @ [A])
  shows is-hyp-proof H S1 S2
  using assms and common-prefix-is-hyp-subproof-from[where S2' = [A] and S2'' = []]
  by simp

lemma hyp-proof-prefix-is-hyp-proof:
  assumes is-hyp-proof H S1 (S2 @ S2')
  shows is-hyp-proof H S1 S2
  using assms and hyp-proof-but-last-is-hyp-proof
  by (induction S2' arbitrary: S2 rule: rev-induct) (simp, metis append.assoc)

lemma single-hyp-is-hyp-proof:
  assumes A ∈ H
  shows is-hyp-proof H S1 [A]
  using assms by fastforce

lemma single-thm-is-hyp-proof:
  assumes A ∈ lset S1
  shows is-hyp-proof H S1 [A]
  using assms by fastforce

lemma hyp-proofs-from-concatenation-is-hyp-proof:
  assumes is-hyp-proof H S1 S1' and is-hyp-proof H S2 S2'
  shows is-hyp-proof H (S1 @ S2) (S1' @ S2')
proof (standard, intro allI impI)
  let ?S = S1 @ S2 and ?S' = S1' @ S2'
```

```
  fix i'
  assume i' < length ?S'
  then consider (a) i' < length S1' | (b) i' ∈ {length S1'..<length ?S'}
    by fastforce
  then show is-hyp-proof-step H ?S ?S' i'
  proof cases
    case a
    from ‹is-hyp-proof H S1 S1'› have is-hyp-proof H (S1 @ S2) S1'
      by auto
    with assms(1) and a show ?thesis
      using added-suffix-hyp-proof-preservation[where S1 = S1 @ S2] by auto
  next
    case b
    from assms(2) have is-hyp-proof H (S1 @ S2) S2'
      by auto
    with b show ?thesis
      using added-prefix-hyp-proof-preservation[where S1 = S1 @ S2] by auto
  qed
qed

lemma elem-of-hyp-proof-is-wffo:
  assumes is-hyps H
  and lset S1 ⊆ wffs_o
  and is-hyp-proof H S1 S2
  and A ∈ lset S2
  shows A ∈ wffs_o
using assms proof (induction S2 rule: rev-induct)
  case Nil
  then show ?case
    by simp
next
  case (snoc A' S2)
  from ‹is-hyp-proof H S1 (S2 @ [A'])› have is-hyp-proof H S1 S2
    using hyp-proof-prefix-is-hyp-proof[where S2' = [A']] by presburger
  then show ?case
  proof (cases A ∈ lset S2)
    case True
    with snoc.prems(1,2) and ‹is-hyp-proof H S1 S2› show ?thesis
      by (fact snoc.IH)
  next
    case False
    with snoc.prems(4) have A' = A
      by simp
    with snoc.prems(3) have
      (S2 @ [A]) ! i' ∈ H ∨
      (S2 @ [A]) ! i' ∈ lset S1 ∨
      (S2 @ [A]) ! i' ∈ wffs_o if i' ∈ {0..<length (S2 @ [A])} for i'
      using that by auto
    then have A ∈ wffs_o ∨ A ∈ H ∨ A ∈ lset S1 ∨ length S2 ∉ {0..<Suc (length S2)}
```

```
    by (metis (no-types) length-append-singleton nth-append-length)
   with assms(1) and ‹lset S₁ ⊆ wffs₀› show ?thesis
    using atLeast0-lessThan-Suc by blast
 qed
qed

lemma hyp-prepended-to-hyp-proof-is-hyp-proof:
 assumes is-hyp-proof H S₁ S₂
 and A ∈ H
 shows is-hyp-proof H S₁ ([A] @ S₂)
 using
   hyp-proofs-from-concatenation-is-hyp-proof
   [
     OF single-hyp-is-hyp-proof[OF assms(2)] assms(1),
     where S₁ = []
   ]
 by simp

lemma hyp-appended-to-hyp-proof-is-hyp-proof:
 assumes is-hyp-proof H S₁ S₂
 and A ∈ H
 shows is-hyp-proof H S₁ (S₂ @ [A])
 using
   hyp-proofs-from-concatenation-is-hyp-proof
   [
     OF assms(1) single-hyp-is-hyp-proof[OF assms(2)],
     where S₂ = []
   ]
 by simp

lemma dropped-duplicated-thm-in-hyp-proof-is-hyp-proof:
 assumes is-hyp-proof H (A # S₁) S₂
 and A ∈ lset S₁
 shows is-hyp-proof H S₁ S₂
 using assms by auto

lemma thm-prepended-to-hyp-proof-is-hyp-proof:
 assumes is-hyp-proof H S₁ S₂
 and A ∈ lset S₁
 shows is-hyp-proof H S₁ ([A] @ S₂)
 using hyp-proofs-from-concatenation-is-hyp-proof[OF single-thm-is-hyp-proof[OF assms(2)] assms(1)]
 and dropped-duplicated-thm-in-hyp-proof-is-hyp-proof by simp

lemma thm-appended-to-hyp-proof-is-hyp-proof:
 assumes is-hyp-proof H S₁ S₂
 and A ∈ lset S₁
 shows is-hyp-proof H S₁ (S₂ @ [A])
 using hyp-proofs-from-concatenation-is-hyp-proof[OF assms(1) single-thm-is-hyp-proof[OF assms(2)]]
 and dropped-duplicated-thm-in-hyp-proof-is-hyp-proof by simp
```

lemma *rule-R′-app-appended-to-hyp-proof-is-hyp-proof*:
assumes *is-hyp-proof* $\mathcal{H}$ $\mathcal{S}'$ $\mathcal{S}$
and $i_C < \mathit{length}\ \mathcal{S}$ **and** $\mathcal{S}\ !\ i_C = C$
and $i_E < \mathit{length}\ \mathcal{S}$ **and** $\mathcal{S}\ !\ i_E = E$
and *is-rule-R′-app* $\mathcal{H}$ p D C E
shows *is-hyp-proof* $\mathcal{H}$ $\mathcal{S}'$ $(\mathcal{S} \mathbin{@} [D])$
proof (*standard*, *intro allI impI*)
let $?\mathcal{S} = \mathcal{S} \mathbin{@} [D]$
fix i'
assume $i' < \mathit{length}\ ?\mathcal{S}$
then consider (*a*) $i' < \mathit{length}\ \mathcal{S}$ | (*b*) $i' = \mathit{length}\ \mathcal{S}$
by *fastforce*
then show *is-hyp-proof-step* $\mathcal{H}$ $\mathcal{S}'$ $(\mathcal{S} \mathbin{@} [D])$ i'
proof *cases*
case *a*
with *assms*(*1*) **show** *?thesis*
using *added-suffix-hyp-proof-preservation* **by** *auto*
next
case *b*
let $?i_D = \mathit{length}\ \mathcal{S}$
from *assms*(*2*,*4*) **have** $i_C < ?i_D$ **and** $i_E < ?i_D$
by *fastforce+*
with *assms*(*3*,*5*,*6*) **have** *is-rule-R′-app* $\mathcal{H}$ p $(?\mathcal{S}\ !\ ?i_D)$ $(?\mathcal{S}\ !\ i_C)$ $(?\mathcal{S}\ !\ i_E)$
by (*simp add*: *nth-append*)
with *assms*(*2*,*4*) **have**
$\exists\, p\ j\ k.\ \{j, k\} \subseteq \{0..<?i_D\} \wedge$ *is-rule-R′-app* $\mathcal{H}$ p $(?\mathcal{S}\ !\ ?i_D)$ $(?\mathcal{S}\ !\ j)$ $(?\mathcal{S}\ !\ k)$
by (*intro exI*)*+ auto*
then have *is-hyp-proof-step* $\mathcal{H}$ $\mathcal{S}'$ $?\mathcal{S}$ $(\mathit{length}\ ?\mathcal{S} - 1)$
by *simp*
moreover from *b* **have** $i' = \mathit{length}\ ?\mathcal{S} - 1$
by *simp*
ultimately show *?thesis*
by *fast*
qed
qed

definition *is-hyp-proof-of* :: *form set* ⇒ *form list* ⇒ *form list* ⇒ *form* ⇒ *bool* **where**
[*iff*]: *is-hyp-proof-of* $\mathcal{H}$ $\mathcal{S}_1$ $\mathcal{S}_2$ A ⟷
is-hyps $\mathcal{H}$ ∧
is-proof $\mathcal{S}_1$ ∧
$\mathcal{S}_2 \neq []$ ∧
is-hyp-proof $\mathcal{H}$ $\mathcal{S}_1$ $\mathcal{S}_2$ ∧
last $\mathcal{S}_2 = A$

lemma *hyp-proof-prefix-is-hyp-proof-of-last*:
assumes *is-hyps* $\mathcal{H}$
and *is-proof* $\mathcal{S}''$
and *is-hyp-proof* $\mathcal{H}$ $\mathcal{S}''$ $(\mathcal{S} \mathbin{@} \mathcal{S}')$ **and** $\mathcal{S} \neq []$

shows *is-hyp-proof-of* $\mathcal{H}$ $\mathcal{S}''$ $\mathcal{S}$ (*last* $\mathcal{S}$)
using *assms* **and** *hyp-proof-prefix-is-hyp-proof* **by** *simp*

theorem *hyp-derivability-implies-hyp-proof-existence*:
assumes $\mathcal{H} \vdash A$
shows $\exists \mathcal{S}_1\ \mathcal{S}_2$. *is-hyp-proof-of* $\mathcal{H}$ $\mathcal{S}_1$ $\mathcal{S}_2$ A
using *assms* **proof** (*induction rule*: *is-derivable-from-hyps.induct*)
case (*dv-hyp* A)
from ‹$A \in \mathcal{H}$› **have** *is-hyp-proof* $\mathcal{H}$ [] [A]
by (*fact single-hyp-is-hyp-proof*)
moreover have *last* [A] = A
by *simp*
moreover have *is-proof* []
by *simp*
ultimately show *?case*
using ‹*is-hyps* $\mathcal{H}$› **unfolding** *is-hyp-proof-of-def* **by** (*meson list.discI*)
next
case (*dv-thm* A)
then obtain $\mathcal{S}$ **where** *is-proof* $\mathcal{S}$ **and** $\mathcal{S} \neq []$ **and** *last* $\mathcal{S} = A$
by *fastforce*
then have *is-hyp-proof* $\mathcal{H}$ $\mathcal{S}$ [A]
using *single-thm-is-hyp-proof* **by** *auto*
with ‹*is-hyps* $\mathcal{H}$› **and** ‹*is-proof* $\mathcal{S}$› **have** *is-hyp-proof-of* $\mathcal{H}$ $\mathcal{S}$ [A] A
by *fastforce*
then show *?case*
by (*intro exI*)
next
case (*dv-rule-R'* C E p D)
from *dv-rule-R'.IH* **obtain** $\mathcal{S}_C$ **and** $\mathcal{S}_C{}'$ **and** $\mathcal{S}_E$ **and** $\mathcal{S}_E{}'$ **where**
is-hyp-proof $\mathcal{H}$ $\mathcal{S}_C{}'$ $\mathcal{S}_C$ **and** *is-proof* $\mathcal{S}_C{}'$ **and** $\mathcal{S}_C \neq []$ **and** *last* $\mathcal{S}_C = C$ **and**
is-hyp-proof $\mathcal{H}$ $\mathcal{S}_E{}'$ $\mathcal{S}_E$ **and** *is-proof* $\mathcal{S}_E{}'$ **and** $\mathcal{S}_E \neq []$ **and** *last* $\mathcal{S}_E = E$
by *auto*
let *?i*$_C$ = *length* $\mathcal{S}_C$ − *1* **and** *?i*$_E$ = *length* $\mathcal{S}_C$ + *length* $\mathcal{S}_E$ − *1* **and** *?i*$_D$ = *length* $\mathcal{S}_C$ + *length* $\mathcal{S}_E$
let *?*$\mathcal{S}$ = $\mathcal{S}_C$ @ $\mathcal{S}_E$ @ [D]
from ‹$\mathcal{S}_C \neq []$› **have** *?i*$_C$ < *length* ($\mathcal{S}_C$ @ $\mathcal{S}_E$) **and** *?i*$_E$ < *length* ($\mathcal{S}_C$ @ $\mathcal{S}_E$)
using *linorder-not-le* **by** *fastforce+*
moreover have ($\mathcal{S}_C$ @ $\mathcal{S}_E$) ! *?i*$_C$ = C **and** ($\mathcal{S}_C$ @ $\mathcal{S}_E$) ! *?i*$_E$ = E
using ‹$\mathcal{S}_C \neq []$› **and** ‹*last* $\mathcal{S}_C$ = C› **and** ‹$\mathcal{S}_E \neq []$› **and** ‹*last* $\mathcal{S}_E$ = E›
by
(
simp add: *last-conv-nth nth-append*,
metis append-is-Nil-conv last-appendR last-conv-nth length-append
)
with ‹*is-rule-R'-app* $\mathcal{H}$ p D C E› **have** *is-rule-R'-app* $\mathcal{H}$ p D (($\mathcal{S}_C$ @ $\mathcal{S}_E$) ! *?i*$_C$) (($\mathcal{S}_C$ @ $\mathcal{S}_E$) ! *?i*$_E$)
by *fastforce*
moreover from ‹*is-hyp-proof* $\mathcal{H}$ $\mathcal{S}_C{}'$ $\mathcal{S}_C$› **and** ‹*is-hyp-proof* $\mathcal{H}$ $\mathcal{S}_E{}'$ $\mathcal{S}_E$›
have *is-hyp-proof* $\mathcal{H}$ ($\mathcal{S}_C{}'$ @ $\mathcal{S}_E{}'$) ($\mathcal{S}_C$ @ $\mathcal{S}_E$)
by (*fact hyp-proofs-from-concatenation-is-hyp-proof*)

```
  ultimately have is-hyp-proof ℋ (𝒮_C' @ 𝒮_E') ((𝒮_C @ 𝒮_E) @ [D])
    using rule-R'-app-appended-to-hyp-proof-is-hyp-proof
    by presburger
  moreover from ‹is-proof 𝒮_C'› and ‹is-proof 𝒮_E'› have is-proof (𝒮_C' @ 𝒮_E')
    by (fact proofs-concatenation-is-proof)
  ultimately have is-hyp-proof-of ℋ (𝒮_C' @ 𝒮_E') ((𝒮_C @ 𝒮_E) @ [D]) D
    using ‹is-hyps ℋ› by fastforce
  then show ?case
    by (intro exI)
qed

theorem hyp-proof-existence-implies-hyp-derivability:
  assumes ∃𝒮_1 𝒮_2. is-hyp-proof-of ℋ 𝒮_1 𝒮_2 A
  shows ℋ ⊢ A
proof −
  from assms obtain 𝒮_1 and 𝒮_2
    where is-hyps ℋ and is-proof 𝒮_1 and 𝒮_2 ≠ [] and is-hyp-proof ℋ 𝒮_1 𝒮_2 and last 𝒮_2 = A
    by fastforce
  then show ?thesis
  proof (induction length 𝒮_2 arbitrary: 𝒮_2 A rule: less-induct)
    case less
    let ?i' = length 𝒮_2 − 1
    from ‹𝒮_2 ≠ []› and ‹last 𝒮_2 = A› have 𝒮_2 ! ?i' = A
      by (simp add: last-conv-nth)
    from ‹is-hyp-proof ℋ 𝒮_1 𝒮_2› and ‹𝒮_2 ≠ []› have is-hyp-proof-step ℋ 𝒮_1 𝒮_2 ?i'
      by simp
    then consider
      (hyp) 𝒮_2 ! ?i' ∈ ℋ
    | (seq) 𝒮_2 ! ?i' ∈ lset 𝒮_1
    | (rule-R') ∃p j k. {j, k} ⊆ {0..<?i'} ∧ is-rule-R'-app ℋ p (𝒮_2 ! ?i') (𝒮_2 ! j) (𝒮_2 ! k)
      by force
    then show ?case
    proof cases
      case hyp
      with ‹𝒮_2 ! ?i' = A› and ‹is-hyps ℋ› show ?thesis
        by (fastforce intro: dv-hyp)
    next
      case seq
      from ‹𝒮_2 ! ?i' ∈ lset 𝒮_1› and ‹𝒮_2 ! ?i' = A›
      obtain j where 𝒮_1 ! j = A and 𝒮_1 ≠ [] and j < length 𝒮_1
        by (metis empty-iff in-set-conv-nth list.set(1))
      with ‹is-proof 𝒮_1› have is-proof (take (Suc j) 𝒮_1) and take (Suc j) 𝒮_1 ≠ []
        using proof-prefix-is-proof[where 𝒮_1 = take (Suc j) 𝒮_1 and 𝒮_2 = drop (Suc j) 𝒮_1]
        by simp-all
      moreover from ‹𝒮_1 ! j = A› and ‹j < length 𝒮_1› have last (take (Suc j) 𝒮_1) = A
        by (simp add: take-Suc-conv-app-nth)
      ultimately have is-proof-of (take (Suc j) 𝒮_1) A
        by fastforce
      then have is-theorem A
```

```
      using is-theorem-def by blast
    with ‹is-hyps ℋ› show ?thesis
      by (intro dv-thm)
  next
    case rule-R′
    then obtain p and j and k
      where {j, k} ⊆ {0..<?i′} and is-rule-R′-app ℋ p (𝒮₂ ! ?i′) (𝒮₂ ! j) (𝒮₂ ! k)
      by force
    let ?𝒮ⱼ = take (Suc j) 𝒮₂ and ?𝒮ₖ = take (Suc k) 𝒮₂
    obtain 𝒮ⱼ′ and 𝒮ₖ′ where 𝒮₂ = ?𝒮ⱼ @ 𝒮ⱼ′ and 𝒮₂ = ?𝒮ₖ @ 𝒮ₖ′
      by (metis append-take-drop-id)
    then have is-hyp-proof ℋ 𝒮₁ (?𝒮ⱼ @ 𝒮ⱼ′) and is-hyp-proof ℋ 𝒮₁ (?𝒮ₖ @ 𝒮ₖ′)
      by (simp-all only: ‹is-hyp-proof ℋ 𝒮₁ 𝒮₂›)
    moreover from ‹𝒮₂ ≠ []› and ‹𝒮₂ = ?𝒮ⱼ @ 𝒮ⱼ′› and ‹𝒮₂ = ?𝒮ₖ @ 𝒮ₖ′› and ‹last 𝒮₂ = A›
    have last 𝒮ⱼ′ = A and last 𝒮ₖ′ = A
      using ‹{j, k} ⊆ {0..<length 𝒮₂ − 1}› and take-tl and less-le-not-le and append.right-neutral
      by (metis atLeastLessThan-iff insert-subset last-appendR length-tl take-all-iff)+
    moreover from ‹𝒮₂ ≠ []› have ?𝒮ⱼ ≠ [] and ?𝒮ₖ ≠ []
      by simp-all
    ultimately have is-hyp-proof-of ℋ 𝒮₁ ?𝒮ⱼ (last ?𝒮ⱼ) and is-hyp-proof-of ℋ 𝒮₁ ?𝒮ₖ (last ?𝒮ₖ)
      using hyp-proof-prefix-is-hyp-proof-of-last
        [OF ‹is-hyps ℋ› ‹is-proof 𝒮₁› ‹is-hyp-proof ℋ 𝒮₁ (?𝒮ⱼ @ 𝒮ⱼ′)› ‹?𝒮ⱼ ≠ []›]
      and hyp-proof-prefix-is-hyp-proof-of-last
        [OF ‹is-hyps ℋ› ‹is-proof 𝒮₁› ‹is-hyp-proof ℋ 𝒮₁ (?𝒮ₖ @ 𝒮ₖ′)› ‹?𝒮ₖ ≠ []›]
      by fastforce+
    moreover from ‹last 𝒮ⱼ′ = A› and ‹last 𝒮ₖ′ = A›
    have length ?𝒮ⱼ < length 𝒮₂ and length ?𝒮ₖ < length 𝒮₂
      using ‹{j, k} ⊆ {0..<length 𝒮₂ − 1}› by force+
    moreover from calculation(3,4) have last ?𝒮ⱼ = 𝒮₂ ! j and last ?𝒮ₖ = 𝒮₂ ! k
      by (metis Suc-lessD last-snoc linorder-not-le nat-neq-iff take-Suc-conv-app-nth take-all-iff)+
    ultimately have ℋ ⊢ 𝒮₂ ! j and ℋ ⊢ 𝒮₂ ! k
      using ‹is-hyps ℋ›
      and less(1)[OF ‹length ?𝒮ⱼ < length 𝒮₂›] and less(1)[OF ‹length ?𝒮ₖ < length 𝒮₂›]
      by fast+
    with ‹is-hyps ℋ› and ‹𝒮₂ ! ?i′ = A› show ?thesis
      using ‹is-rule-R′-app ℋ p (𝒮₂ ! ?i′) (𝒮₂ ! j) (𝒮₂ ! k)› by (blast intro: dv-rule-R′)
  qed
 qed
qed

theorem hypothetical-derivability-proof-existence-equivalence:
 shows ℋ ⊢ A ⟷ (∃𝒮₁ 𝒮₂. is-hyp-proof-of ℋ 𝒮₁ 𝒮₂ A)
 using hyp-derivability-implies-hyp-proof-existence and hyp-proof-existence-implies-hyp-derivability ..

proposition derivability-from-no-hyps-theoremhood-equivalence:
 shows {} ⊢ A ⟷ is-theorem A
proof
 assume {} ⊢ A
 then show is-theorem A
```

```
  proof (induction rule: is-derivable-from-hyps.induct)
    case (dv-rule-R' C E p D)
    from ‹is-rule-R'-app {} p D C E› have is-rule-R-app p D C E
      by simp
    moreover from ‹is-theorem C› and ‹is-theorem E› have is-derivable C and is-derivable E
      using theoremhood-derivability-equivalence by (simp-all only:)
    ultimately have is-derivable D
      by (fastforce intro: dv-rule-R)
    then show ?case
      using theoremhood-derivability-equivalence by (simp only:)
  qed simp
next
  assume is-theorem A
  then show {} ⊢ A
    by (blast intro: dv-thm)
qed

abbreviation is-derivable-from-no-hyps (⊢ - [50] 50) where
  ⊢ A ≡ {} ⊢ A

corollary derivability-implies-hyp-derivability:
  assumes ⊢ A and is-hyps ℋ
  shows ℋ ⊢ A
  using assms and derivability-from-no-hyps-theoremhood-equivalence and dv-thm by simp

lemma axiom-is-derivable-from-no-hyps:
  assumes A ∈ axioms
  shows ⊢ A
  using derivability-from-no-hyps-theoremhood-equivalence
  and derivable-form-is-theorem[OF dv-axiom[OF assms]] by (simp only:)

lemma axiom-is-derivable-from-hyps:
  assumes A ∈ axioms and is-hyps ℋ
  shows ℋ ⊢ A
  using assms and axiom-is-derivable-from-no-hyps and derivability-implies-hyp-derivability by blast

lemma rule-R [consumes 2, case-names occ-subform replacement]:
  assumes ⊢ C and ⊢ A =α B
  and A ≼p C and C⟨p ← B⟩ ▷ D
  shows ⊢ D
proof −
  from assms(1,2) have is-derivable C and is-derivable (A =α B)
    using derivability-from-no-hyps-theoremhood-equivalence
    and theoremhood-derivability-equivalence by blast+
  moreover have is-rule-R-app p D C (A =α B)
  proof −
    from assms(1−4) have D ∈ wffs_o and A ∈ wffs_α and B ∈ wffs_α
      by (meson hyp-derivable-form-is-wffso replacement-preserves-typing wffs-from-equality)+
    with assms(3,4) show ?thesis
```

by *fastforce*
qed
ultimately have *is-derivable D*
by (*rule dv-rule-R*)
then show *?thesis*
using *derivability-from-no-hyps-theoremhood-equivalence* **and** *derivable-form-is-theorem* **by** *simp*
qed

lemma *rule-R′* [*consumes 2*, *case-names occ-subform replacement no-capture*]:
assumes $\mathcal{H} \vdash C$ **and** $\mathcal{H} \vdash A =_\alpha B$
and $A \preceq_p C$ **and** $C\langle\!|p \leftarrow B|\!\rangle \rhd D$
and *rule-R′-side-condition* $\mathcal{H}$ *p D C* $(A =_\alpha B)$
shows $\mathcal{H} \vdash D$
using *assms*(*1*,*2*) **proof** (*rule dv-rule-R′*)
from *assms*(*1*) **show** *is-hyps* $\mathcal{H}$
by (*blast elim*: *is-derivable-from-hyps.cases*)
moreover from *assms*(*1−4*) **have** $D \in wffs_o$
by (*meson hyp-derivable-form-is-wffso replacement-preserves-typing wffs-from-equality*)
ultimately show *is-rule-R′-app* $\mathcal{H}$ *p D C* $(A =_\alpha B)$
using *assms*(*2−5*) **and** *hyp-derivable-form-is-wffso* **and** *wffs-from-equality*
unfolding *is-rule-R-app-def* **and** *is-rule-R′-app-def* **by** *metis*
qed

end

6 Elementary Logic

theory *Elementary-Logic*
imports
Proof-System
Propositional-Wff
begin

no-notation *funcset* (**infixr** $\rightarrow$ *60*)
notation *funcset* (**infixr** $\nrightarrow$ *60*)

6.1 Proposition 5200

proposition *prop-5200*:
assumes $A \in wffs_\alpha$
shows $\vdash A =_\alpha A$
using *assms* **and** *equality-reflexivity* **and** *dv-thm* **by** *simp*

corollary *hyp-prop-5200*:
assumes *is-hyps* $\mathcal{H}$ **and** $A \in wffs_\alpha$
shows $\mathcal{H} \vdash A =_\alpha A$
using *derivability-implies-hyp-derivability*[*OF prop-5200*[*OF assms*(*2*)] *assms*(*1*)] **.**

6.2 Proposition 5201 (Equality Rules)

proposition *prop-5201-1*:
assumes $\mathcal{H} \vdash A$ **and** $\mathcal{H} \vdash A \equiv^{\mathfrak{Q}} B$
shows $\mathcal{H} \vdash B$
proof −
from *assms*(*2*) **have** $\mathcal{H} \vdash A =_o B$
unfolding *equivalence-def* **.**
with *assms*(*1*) **show** *?thesis*
by (*rule rule-R'*[**where** $p = []$]) *auto*
qed

proposition *prop-5201-2*:
assumes $\mathcal{H} \vdash A =_\alpha B$
shows $\mathcal{H} \vdash B =_\alpha A$
proof −
have $\mathcal{H} \vdash A =_\alpha A$
proof (*rule hyp-prop-5200*)
from *assms* **show** *is-hyps* $\mathcal{H}$
by (*blast elim*: *is-derivable-from-hyps.cases*)
show $A \in wffs_\alpha$
by (*fact hyp-derivable-form-is-wffso*[*OF assms*, *THEN wffs-from-equality*(*1*)])
qed
from *this* **and** *assms* **show** *?thesis*
by (*rule rule-R'*[**where** $p = [«,»]$]) (*force*+, *fastforce dest*: *subforms-from-app*)
qed

proposition *prop-5201-3*:
assumes $\mathcal{H} \vdash A =_\alpha B$ **and** $\mathcal{H} \vdash B =_\alpha C$
shows $\mathcal{H} \vdash A =_\alpha C$
using *assms* **by** (*rule rule-R'*[**where** $p = [»]$]) (*force*+, *fastforce dest*: *subforms-from-app*)

proposition *prop-5201-4*:
assumes $\mathcal{H} \vdash A =_{\alpha\to\beta} B$ **and** $\mathcal{H} \vdash C =_\alpha D$
shows $\mathcal{H} \vdash A \cdot C =_\beta B \cdot D$
proof −
have $\mathcal{H} \vdash A \cdot C =_\beta A \cdot C$
proof (*rule hyp-prop-5200*)
from *assms* **show** *is-hyps* $\mathcal{H}$
by (*blast elim*: *is-derivable-from-hyps.cases*)
from *assms* **have** $A \in wffs_{\alpha\to\beta}$ **and** $C \in wffs_\alpha$
using *hyp-derivable-form-is-wffso* **and** *wffs-from-equality* **by** *blast*+
then show $A \cdot C \in wffs_\beta$
by *auto*
qed
from *this* **and** *assms*(*1*) **have** $\mathcal{H} \vdash A \cdot C =_\beta B \cdot C$
by (*rule rule-R'*[**where** $p = [»,«]$]) (*force*+, *fastforce dest*: *subforms-from-app*)
from *this* **and** *assms*(*2*) **show** *?thesis*
by (*rule rule-R'*[**where** $p = [»,»]$]) (*force*+, *fastforce dest*: *subforms-from-app*)
qed

proposition *prop-5201-5*:
assumes $\mathcal{H} \vdash A =_{\alpha\to\beta} B$ **and** $C \in \mathit{wffs}_\alpha$
shows $\mathcal{H} \vdash A \bullet C =_\beta B \bullet C$
proof −
have $\mathcal{H} \vdash A \bullet C =_\beta A \bullet C$
proof (*rule hyp-prop-5200*)
from *assms*(*1*) **show** *is-hyps* $\mathcal{H}$
by (*blast elim*: *is-derivable-from-hyps.cases*)
have $A \in \mathit{wffs}_{\alpha\to\beta}$
by (*fact hyp-derivable-form-is-wffso*[*OF assms*(*1*), *THEN wffs-from-equality*(*1*)])
with *assms*(*2*) **show** $A \bullet C \in \mathit{wffs}_\beta$
by *auto*
qed
from *this* **and** *assms*(*1*) **show** *?thesis*
by (*rule rule-R'*[**where** p = [»,«]]) (*force*+, *fastforce dest*: *subforms-from-app*)
qed

proposition *prop-5201-6*:
assumes $\mathcal{H} \vdash C =_\alpha D$ **and** $A \in \mathit{wffs}_{\alpha\to\beta}$
shows $\mathcal{H} \vdash A \bullet C =_\beta A \bullet D$
proof −
have $\mathcal{H} \vdash A \bullet C =_\beta A \bullet C$
proof (*rule hyp-prop-5200*)
from *assms*(*1*) **show** *is-hyps* $\mathcal{H}$
by (*blast elim*: *is-derivable-from-hyps.cases*)
have $C \in \mathit{wffs}_\alpha$
by (*fact hyp-derivable-form-is-wffso*[*OF assms*(*1*), *THEN wffs-from-equality*(*1*)])
with *assms*(*2*) **show** $A \bullet C \in \mathit{wffs}_\beta$
by *auto*
qed
from *this* **and** *assms*(*1*) **show** *?thesis*
by (*rule rule-R'*[**where** p = [»,»]]) (*force*+, *fastforce dest*: *subforms-from-app*)
qed

lemmas *Equality-Rules* = *prop-5201-1 prop-5201-2 prop-5201-3 prop-5201-4 prop-5201-5 prop-5201-6*

6.3 Proposition 5202 (Rule RR)

proposition *prop-5202*:
assumes $\vdash A =_\alpha B \vee \vdash B =_\alpha A$
and $p \in \mathit{positions}\ C$ **and** $A \preceq_p C$ **and** $C\langle\!|p \leftarrow B|\!\rangle \rhd D$
and $\mathcal{H} \vdash C$
shows $\mathcal{H} \vdash D$
proof −
from *assms*(*5*) **have** $\vdash C =_o C$
using *prop-5200* **and** *hyp-derivable-form-is-wffso* **by** *blast*
moreover from *assms*(*1*) **consider** (*a*) $\vdash A =_\alpha B$ | (*b*) $\vdash B =_\alpha A$
by *blast*

then have $\vdash A =_\alpha B$
by *cases* (*assumption*, *fact Equality-Rules*(*2*))
ultimately have $\vdash C =_o D$
by (*rule rule-R*[**where** $p = \langle\!\langle \# \; p]$) (*use assms*(*2*−*4*) **in** *auto*)
then have $\mathcal{H} \vdash C =_o D$
proof −
from *assms*(*5*) **have** *is-hyps* $\mathcal{H}$
by (*blast elim*: *is-derivable-from-hyps.cases*)
with $\langle \vdash C =_o D \rangle$ **show** *?thesis*
by (*fact derivability-implies-hyp-derivability*)
qed
with *assms*(*5*) **show** *?thesis*
by (*rule Equality-Rules*(*1*)[*unfolded equivalence-def*])
qed

lemmas *rule-RR* = *prop-5202*

6.4 Proposition 5203

proposition *prop-5203*:
assumes $A \in wffs_\alpha$ **and** $B \in wffs_\beta$
and $\forall\, v \in vars\ A.\ \neg\ is\text{-}bound\ v\ B$
shows $\vdash (\lambda x_\alpha.\ B) \cdot A =_\beta \mathbf{S}\ \{(x,\ \alpha) \rightarrowtail A\}\ B$
using *assms*(*2*,*1*,*3*) **proof** *induction*
case (*var-is-wff* β *y*)
then show *?case*
proof (*cases* $y_\beta = x_\alpha$)
case *True*
then have $\alpha = \beta$
by *simp*
moreover from *assms*(*1*) **have** $\vdash (\lambda x_\alpha.\ x_\alpha) \cdot A =_\alpha A$
using *axiom-4-2* **by** (*intro axiom-is-derivable-from-no-hyps*)
moreover have S $\{(x,\ \alpha) \rightarrowtail A\}\ (x_\alpha) = A$
by *force*
ultimately show *?thesis*
using *True* **by** (*simp only*:)
next
case *False*
with *assms*(*1*) **have** $\vdash (\lambda x_\alpha.\ y_\beta) \cdot A =_\beta y_\beta$
using *axiom-4-1-var* **by** (*intro axiom-is-derivable-from-no-hyps*)
moreover from *False* **have S** $\{(x,\ \alpha) \rightarrowtail A\}\ (y_\beta) = y_\beta$
by *auto*
ultimately show *?thesis*
by (*simp only*:)
qed
next
case (*con-is-wff* β *c*)
from *assms*(*1*) **have** $\vdash (\lambda x_\alpha.\ \{\!|c|\!\}_\beta) \cdot A =_\beta \{\!|c|\!\}_\beta$
using *axiom-4-1-con* **by** (*intro axiom-is-derivable-from-no-hyps*)

```
  moreover have S {(x, α) ↣ A} (⦇c⦈β) = ⦇c⦈β
    by auto
  ultimately show ?case
    by (simp only:)
next
  case (app-is-wff γ β D C)
  from app-is-wff.prems(2) have not-bound-subforms: ∀ v ∈ vars A. ¬ is-bound v D ∧ ¬ is-bound v C
    using is-bound-in-app-homomorphism by fast
  from ‹D ∈ wffs_{γ→β}› have ⊢ (λx_α. D) • A =_{γ→β} S {(x, α) ↣ A} D
    using app-is-wff.IH(1)[OF assms(1)] and not-bound-subforms by simp
  moreover from ‹C ∈ wffs_γ› have ⊢ (λx_α. C) • A =_γ S {(x, α) ↣ A} C
    using app-is-wff.IH(2)[OF assms(1)] and not-bound-subforms by simp
  moreover have ⊢ (λx_α. D • C) • A =_β ((λx_α. D) • A) • ((λx_α. C) • A)
   using axiom-is-derivable-from-no-hyps[OF axiom-4-3[OF assms(1) ‹D ∈ wffs_{γ→β}› ‹C ∈ wffs_γ›]] .
  ultimately show ?case
    using Equality-Rules(3,4) and substitute.simps(3) by presburger
next
  case (abs-is-wff β D γ y)
  then show ?case
  proof (cases y_γ = x_α)
    case True
    then have ⊢ (λx_α. λy_γ. D) • A =_{γ→β} λy_γ. D
      using axiom-is-derivable-from-no-hyps[OF axiom-4-5[OF assms(1) abs-is-wff.hyps(1)]] by fast
    moreover from True have S {(x, α) ↣ A} (λy_γ. D) = λy_γ. D
      using empty-substitution-neutrality
      by (simp add: singleton-substitution-simps(4) fmdrop-fmupd-same)
    ultimately show ?thesis
      by (simp only:)
  next
    case False
    have binders-at (λy_γ. D) [«] = {(y, γ)}
      by simp
    then have is-bound (y, γ) (λy_γ. D)
      by fastforce
    with abs-is-wff.prems(2) have (y, γ) ∉ vars A
      by blast
    with ‹y_γ ≠ x_α› have ⊢ (λx_α. λy_γ. D) • A =_{γ→β} λy_γ. (λx_α. D) • A
      using axiom-4-4[OF assms(1) abs-is-wff.hyps(1)] and axiom-is-derivable-from-no-hyps by blast
    moreover have ⊢ (λx_α. D) • A =_β S {(x, α) ↣ A} D
    proof −
      have ∀ p. y_γ ≼« # p λy_γ. D ⟶ y_γ ≼p D
        using subforms-from-abs by fastforce
      from abs-is-wff.prems(2) have ∀ v ∈ vars A. ¬ is-bound v D
        using is-bound-in-abs-body by fast
      then show ?thesis
        by (fact abs-is-wff.IH[OF assms(1)])
    qed
    ultimately have ⊢ (λx_α. λy_γ. D) • A =_{γ→β} λy_γ. S {(x, α) ↣ A} D
      by (rule rule-R[where p = [»,«]]) force+
```

with *False* **show** *?thesis*
by *simp*
qed
qed

6.5 Proposition 5204

proposition *prop-5204*:
assumes $A \in \mathit{wffs}_\alpha$ **and** $B \in \mathit{wffs}_\beta$ **and** $C \in \mathit{wffs}_\beta$
and $\vdash B =_\beta C$
and $\forall\, v \in \mathit{vars}\ A.\ \neg\ \mathit{is\text{-}bound}\ v\ B \wedge \neg\ \mathit{is\text{-}bound}\ v\ C$
shows $\vdash$ **S** $\{(x,\, \alpha) \rightarrowtail A\}$ $(B =_\beta C)$
proof −
have $\vdash (\lambda x_\alpha.\ B) \centerdot A =_\beta (\lambda x_\alpha.\ B) \centerdot A$
proof −
have $(\lambda x_\alpha.\ B) \centerdot A \in \mathit{wffs}_\beta$
using *assms*(*1*,*2*) **by** *auto*
then show *?thesis*
by (*fact prop-5200*)
qed
from *this* **and** *assms*(*4*) **have** $\vdash (\lambda x_\alpha.\ B) \centerdot A =_\beta (\lambda x_\alpha.\ C) \centerdot A$
by (*rule rule-R*[**where** $p = [\mathrm{»},\mathrm{«},\mathrm{«}]$]) *force*+
moreover from *assms*(*1*,*2*,*5*) **have** $\vdash (\lambda x_\alpha.\ B) \centerdot A =_\beta$ **S** $\{(x,\, \alpha) \rightarrowtail A\}$ B
using *prop-5203* **by** *auto*
moreover from *assms*(*1*,*3*,*5*) **have** $\vdash (\lambda x_\alpha.\ C) \centerdot A =_\beta$ **S** $\{(x,\, \alpha) \rightarrowtail A\}$ C
using *prop-5203* **by** *auto*
ultimately have $\vdash$ (**S** $\{(x,\, \alpha) \rightarrowtail A\}$ $B) =_\beta$ (**S** $\{(x,\, \alpha) \rightarrowtail A\}$ $C)$
using *Equality-Rules*(*2*,*3*) **by** *blast*
then show *?thesis*
by *simp*
qed

6.6 Proposition 5205 (η-conversion)

proposition *prop-5205*:
shows $\vdash \mathfrak{f}_{\alpha\to\beta} =_{\alpha\to\beta} (\lambda y_\alpha.\ \mathfrak{f}_{\alpha\to\beta} \centerdot y_\alpha)$
proof −
{
fix y
assume $y_\alpha \neq \mathfrak{x}_\alpha$
let $?A = \lambda y_\alpha.\ \mathfrak{f}_{\alpha\to\beta} \centerdot y_\alpha$
have $\vdash (\mathfrak{f}_{\alpha\to\beta} =_{\alpha\to\beta} ?A) =_o \forall \mathfrak{x}_\alpha.\ (\mathfrak{f}_{\alpha\to\beta} \centerdot \mathfrak{x}_\alpha =_\beta ?A \centerdot \mathfrak{x}_\alpha)$
proof −
have $\vdash (\mathfrak{f}_{\alpha\to\beta} =_{\alpha\to\beta} \mathfrak{g}_{\alpha\to\beta}) =_o \forall \mathfrak{x}_\alpha.\ (\mathfrak{f}_{\alpha\to\beta} \centerdot \mathfrak{x}_\alpha =_\beta \mathfrak{g}_{\alpha\to\beta} \centerdot \mathfrak{x}_\alpha)$ (**is** $\vdash ?B =_o ?C$)
using *axiom-3*[*unfolded equivalence-def*] **by** (*rule axiom-is-derivable-from-no-hyps*)
have $\vdash$ **S** $\{(\mathfrak{g},\, \alpha\to\beta) \rightarrowtail ?A\}$ $(?B =_o ?C)$
proof −
have $?A \in \mathit{wffs}_{\alpha\to\beta}$ **and** $?B \in \mathit{wffs}_o$ **and** $?C \in \mathit{wffs}_o$
by *auto*

moreover have $\forall\, v \in \mathit{vars}\ ?A.\ \neg\ \mathit{is\text{-}bound}\ v\ ?B \wedge \neg\ \mathit{is\text{-}bound}\ v\ ?C$
proof
fix v
assume $v \in \mathit{vars}\ ?A$
have $\mathit{vars}\ ?B = \{(\mathfrak{f},\ \alpha{\to}\beta),\ (\mathfrak{g},\ \alpha{\to}\beta)\}$ **and** $\mathit{vars}\ ?C = \{(\mathfrak{f},\ \alpha{\to}\beta),\ (\mathfrak{x},\ \alpha),\ (\mathfrak{g},\ \alpha{\to}\beta)\}$
by *force+*
with ‹$y_\alpha \neq \mathfrak{x}_\alpha$› **have** $(y,\ \alpha) \notin \mathit{vars}\ ?B$ **and** $(y,\ \alpha) \notin \mathit{vars}\ ?C$
by *force+*
then have $\neg\ \mathit{is\text{-}bound}\ (y,\ \alpha)\ ?B$ **and** $\neg\ \mathit{is\text{-}bound}\ (y,\ \alpha)\ ?C$
using *absent-var-is-not-bound* **by** *blast+*
moreover have $\neg\ \mathit{is\text{-}bound}\ (\mathfrak{f},\ \alpha{\to}\beta)\ ?B$ **and** $\neg\ \mathit{is\text{-}bound}\ (\mathfrak{f},\ \alpha{\to}\beta)\ ?C$
by *code-simp+*
moreover from ‹$v \in \mathit{vars}\ ?A$› **have** $v \in \{(y,\ \alpha),\ (\mathfrak{f},\ \alpha{\to}\beta)\}$
by *auto*
ultimately show $\neg\ \mathit{is\text{-}bound}\ v\ ?B \wedge \neg\ \mathit{is\text{-}bound}\ v\ ?C$
by *fast*
qed
ultimately show *?thesis*
using ‹$\vdash\ ?B =_o\ ?C$› **and** *prop-5204* **by** *presburger*
qed
then show *?thesis*
by *simp*
qed
moreover have $\vdash\ ?A \cdot \mathfrak{x}_\alpha =_\beta \mathfrak{f}_{\alpha\to\beta} \cdot \mathfrak{x}_\alpha$
proof −
have $\mathfrak{x}_\alpha \in \mathit{wffs}_\alpha$ **and** $\mathfrak{f}_{\alpha\to\beta} \cdot y_\alpha \in \mathit{wffs}_\beta$
by *auto*
moreover have $\forall\, v \in \mathit{vars}\ (\mathfrak{x}_\alpha).\ \neg\ \mathit{is\text{-}bound}\ v\ (\mathfrak{f}_{\alpha\to\beta} \cdot y_\alpha)$
using ‹$y_\alpha \neq \mathfrak{x}_\alpha$› **by** *auto*
moreover have S $\{(y,\ \alpha) \rightarrowtail \mathfrak{x}_\alpha\}\ (\mathfrak{f}_{\alpha\to\beta} \cdot y_\alpha) = \mathfrak{f}_{\alpha\to\beta} \cdot \mathfrak{x}_\alpha$
by *simp*
ultimately show *?thesis*
using *prop-5203* **by** *metis*
qed
ultimately have $\vdash\ (\mathfrak{f}_{\alpha\to\beta} =_{\alpha\to\beta} ?A) =_o \forall\, \mathfrak{x}_\alpha.\ (\mathfrak{f}_{\alpha\to\beta} \cdot \mathfrak{x}_\alpha =_\beta \mathfrak{f}_{\alpha\to\beta} \cdot \mathfrak{x}_\alpha)$
by (*rule rule-R*[**where** $p = [\text{»},\text{»},\text{«},\text{»}]$]) *force+*
moreover have $\vdash\ (\mathfrak{f}_{\alpha\to\beta} =_{\alpha\to\beta} \mathfrak{f}_{\alpha\to\beta}) =_o \forall\, \mathfrak{x}_\alpha.\ (\mathfrak{f}_{\alpha\to\beta} \cdot \mathfrak{x}_\alpha =_\beta \mathfrak{f}_{\alpha\to\beta} \cdot \mathfrak{x}_\alpha)$
proof −
let $?A = \mathfrak{f}_{\alpha\to\beta}$
have $\vdash\ (\mathfrak{f}_{\alpha\to\beta} =_{\alpha\to\beta} \mathfrak{g}_{\alpha\to\beta}) =_o \forall\, \mathfrak{x}_\alpha.\ (\mathfrak{f}_{\alpha\to\beta} \cdot \mathfrak{x}_\alpha =_\beta \mathfrak{g}_{\alpha\to\beta} \cdot \mathfrak{x}_\alpha)$ (**is** $\vdash\ ?B =_o\ ?C$)
using *axiom-3*[*unfolded equivalence-def*] **by** (*rule axiom-is-derivable-from-no-hyps*)
have $\vdash$ **S** $\{(\mathfrak{g},\ \alpha{\to}\beta) \rightarrowtail ?A\}\ (?B =_o\ ?C)$
proof −
have $?A \in \mathit{wffs}_{\alpha\to\beta}$ **and** $?B \in \mathit{wffs}_o$ **and** $?C \in \mathit{wffs}_o$
by *auto*
moreover have $\forall\, v \in \mathit{vars}\ ?A.\ \neg\ \mathit{is\text{-}bound}\ v\ ?B \wedge \neg\ \mathit{is\text{-}bound}\ v\ ?C$
proof
fix v
assume $v \in \mathit{vars}\ ?A$

have *vars ?B* = {(𝔣, *α*→*β*), (𝔤, *α*→*β*)} **and** *vars ?C* = {(𝔣, *α*→*β*), (𝔵, *α*), (𝔤, *α*→*β*)}
by *force+*
with ‹$y_\alpha \neq \mathfrak{x}_\alpha$› **have** (*y*, *α*) ∉ *vars ?B* **and** (*y*, *α*) ∉ *vars ?C*
by *force+*
then have ¬ *is-bound* (*y*, *α*) *?B* **and** ¬ *is-bound* (*y*, *α*) *?C*
using *absent-var-is-not-bound* **by** *blast+*
moreover have ¬ *is-bound* (𝔣, *α*→*β*) *?B* **and** ¬ *is-bound* (𝔣, *α*→*β*) *?C*
by *code-simp+*
moreover from ‹*v* ∈ *vars ?A* ›**have** *v* ∈ {(*y*, *α*), (𝔣, *α*→*β*)}
by *auto*
ultimately show ¬ *is-bound v ?B* ∧ ¬ *is-bound v ?C*
by *fast*
qed
ultimately show *?thesis*
using ‹⊢ *?B* =$_o$ *?C*› **and** *prop-5204* **by** *presburger*
qed
then show *?thesis*
by *simp*
qed
ultimately have ⊢ $\mathfrak{f}_{\alpha\to\beta} =_{\alpha\to\beta} (\lambda y_\alpha.\ \mathfrak{f}_{\alpha\to\beta} \cdot y_\alpha)$
using *Equality-Rules*(*1*)[*unfolded equivalence-def*] **and** *Equality-Rules*(*2*) **and** *prop-5200*
by (*metis wffs-of-type-intros*(*1*))
}
note *x-neq-y* = *this*
then have §*6*: ⊢ $\mathfrak{f}_{\alpha\to\beta} =_{\alpha\to\beta} \lambda\mathfrak{y}_\alpha.\ \mathfrak{f}_{\alpha\to\beta} \cdot \mathfrak{y}_\alpha$ (**is** ⊢ *?B* =$_-$ *?C*)
by *simp*
then have §*7*: ⊢ $(\lambda\mathfrak{x}_\alpha.\ \mathfrak{f}_{\alpha\to\beta} \cdot \mathfrak{x}_\alpha) =_{\alpha\to\beta} (\lambda\mathfrak{y}_\alpha.\ (\lambda\mathfrak{x}_\alpha.\ \mathfrak{f}_{\alpha\to\beta} \cdot \mathfrak{x}_\alpha) \cdot \mathfrak{y}_\alpha)$
proof −
let *?A* = $\lambda\mathfrak{x}_\alpha.\ \mathfrak{f}_{\alpha\to\beta} \cdot \mathfrak{x}_\alpha$
have *?A* ∈ *wffs*$_{\alpha\to\beta}$ **and** *?B* ∈ *wffs*$_{\alpha\to\beta}$ **and** *?C* ∈ *wffs*$_{\alpha\to\beta}$
by *auto*
moreover have ∀ *v* ∈ *vars ?A*. ¬ *is-bound v ?B* ∧ ¬ *is-bound v ?C*
proof
fix *v*
assume *v* ∈ *vars ?A*
have ¬ *is-bound* (𝔵, *α*) *?B* **and** ¬ *is-bound* (𝔵, *α*) *?C*
by *code-simp+*
moreover have ¬ *is-bound* (𝔣, *α*→*β*) *?B* **and** ¬ *is-bound* (𝔣, *α*→*β*) *?C*
by *code-simp+*
moreover from ‹*v* ∈ *vars ?A* ›**have** *v* ∈ {(𝔵, *α*), (𝔣, *α*→*β*)}
by *auto*
ultimately show ¬ *is-bound v ?B* ∧ ¬ *is-bound v ?C*
by *fast*
qed
ultimately have ⊢ **S** {(𝔣, *α*→*β*) ⇀ *?A*} (*?B* =$_{\alpha\to\beta}$ *?C*)
using §*6* **and** *prop-5204* **by** *presburger*
then show *?thesis*
by *simp*
qed

have $\vdash (\lambda \mathfrak{x}_\alpha.\ \mathfrak{f}_{\alpha\to\beta} \cdot \mathfrak{x}_\alpha) =_{\alpha\to\beta} (\lambda \mathfrak{y}_\alpha.\ \mathfrak{f}_{\alpha\to\beta} \cdot \mathfrak{y}_\alpha)$
proof −
have $\vdash (\lambda \mathfrak{x}_\alpha.\ \mathfrak{f}_{\alpha\to\beta} \cdot \mathfrak{x}_\alpha) \cdot \mathfrak{y}_\alpha =_\beta \mathfrak{f}_{\alpha\to\beta} \cdot \mathfrak{y}_\alpha$
proof −
have $\mathfrak{y}_\alpha \in \mathit{wffs}_\alpha$ **and** $\mathfrak{f}_{\alpha\to\beta} \cdot \mathfrak{x}_\alpha \in \mathit{wffs}_\beta$
by *auto*
moreover have $\forall\, v \in \mathit{vars}\ (\mathfrak{y}_\alpha).\ \neg\ \mathit{is\text{-}bound}\ v\ (\mathfrak{f}_{\alpha\to\beta} \cdot \mathfrak{x}_\alpha)$
by *simp*
moreover have S $\{(\mathfrak{x},\ \alpha) \rightarrowtail \mathfrak{y}_\alpha\}\ (\mathfrak{f}_{\alpha\to\beta} \cdot \mathfrak{x}_\alpha) = \mathfrak{f}_{\alpha\to\beta} \cdot \mathfrak{y}_\alpha$
by *simp*
ultimately show *?thesis*
using *prop-5203* **by** *metis*
qed
from §*7* **and** *this* **show** *?thesis*
by (*rule rule-R* [**where** $p = [\text{»},\text{«}]$]) *force+*
qed
with §*6* **and** *x-neq-y*[*of y*] **show** *?thesis*
using *Equality-Rules*(*2*,*3*) **by** *blast*
qed

6.7 Proposition 5206 (α-conversion)

proposition *prop-5206*:
assumes $A \in \mathit{wffs}_\alpha$
and $(z,\ \beta) \notin \mathit{free\text{-}vars}\ A$
and *is-free-for* $(z_\beta)\ (x,\ \beta)\ A$
shows $\vdash (\lambda x_\beta.\ A) =_{\beta\to\alpha} (\lambda z_\beta.\ \mathbf{S}\ \{(x,\ \beta) \rightarrowtail z_\beta\}\ A)$
proof −
have *is-substitution* $\{(x,\ \beta) \rightarrowtail z_\beta\}$
by *auto*
from *this* **and** *assms*(*1*) **have S** $\{(x,\ \beta) \rightarrowtail z_\beta\}\ A \in \mathit{wffs}_\alpha$
by (*fact substitution-preserves-typing*)
obtain y **where** $(y,\ \beta) \notin \{(x,\ \beta),\ (z,\ \beta)\} \cup \mathit{vars}\ A$
proof −
have *finite* $(\{(x,\ \beta),\ (z,\ \beta)\} \cup \mathit{vars}\ A)$
using *vars-form-finiteness* **by** *blast*
with *that* **show** *?thesis*
using *fresh-var-existence* **by** *metis*
qed
then have $(y,\ \beta) \neq (x,\ \beta)$ **and** $(y,\ \beta) \neq (z,\ \beta)$ **and** $(y,\ \beta) \notin \mathit{vars}\ A$ **and** $(y,\ \beta) \notin \mathit{free\text{-}vars}\ A$
using *free-vars-in-all-vars* **by** *auto*
have §*1*: $\vdash (\lambda x_\beta.\ A) =_{\beta\to\alpha} (\lambda y_\beta.\ (\lambda x_\beta.\ A) \cdot y_\beta)$
proof −
let $?A = \lambda x_\beta.\ A$
have $*$: $\vdash \mathfrak{f}_{\beta\to\alpha} =_{\beta\to\alpha} (\lambda y_\beta.\ \mathfrak{f}_{\beta\to\alpha} \cdot y_\beta)$ (**is** $\vdash\ ?B =_{\text{-}}\ ?C$)
by (*fact prop-5205*)
moreover have $\vdash$ **S** $\{(\mathfrak{f},\ \beta\to\alpha) \rightarrowtail\ ?A\}\ (?B =_{\beta\to\alpha}\ ?C)$
proof −
from *assms*(*1*) **have** $?A \in \mathit{wffs}_{\beta\to\alpha}$ **and** $?B \in \mathit{wffs}_{\beta\to\alpha}$ **and** $?C \in \mathit{wffs}_{\beta\to\alpha}$

```
      by auto
    moreover have ∀ v ∈ vars ?A. ¬ is-bound v ?B ∧ ¬ is-bound v ?C
    proof
      fix v
      assume v ∈ vars ?A
      then consider (a) v = (x, β) | (b) v ∈ vars A
        by fastforce
      then show ¬ is-bound v ?B ∧ ¬ is-bound v ?C
      proof cases
        case a
        then show ?thesis
          using ‹(y, β) ≠ (x, β)› by force
      next
        case b
        then have ¬ is-bound v ?B
          by simp
        moreover have ¬ is-bound v ?C
          using b and ‹(y, β) ∉ vars A› by code-simp force
        ultimately show ?thesis
          by blast
      qed
    qed
    ultimately show ?thesis
      using prop-5204 and ∗ by presburger
  qed
  ultimately show ?thesis
    by simp
qed
then have §2: ⊢ (λx_β. A) =_{β→α} (λy_β. S {(x, β) ⤇ y_β} A)
proof −
  have ⊢ (λx_β. A) · y_β =_α S {(x, β) ⤇ y_β} A (is ⊢ (λx_β. ?B) · ?A =_ -)
  proof −
    have ?A ∈ wffs_β and ?B ∈ wffs_α
      by blast fact
    moreover have ∀ v ∈ vars ?A. ¬ is-bound v ?B
      using ‹(y, β) ∉ vars A› and absent-var-is-not-bound by auto
    ultimately show ?thesis
      by (fact prop-5203)
  qed
  with §1 show ?thesis
    by (rule rule-R [where p = [»,«]]) force+
qed
moreover
have §3: ⊢ (λz_β. S {(x, β) ⤇ z_β} A) =_{β→α} (λy_β. (λz_β. S {(x, β) ⤇ z_β} A) · y_β)
proof −
  let ?A = λz_β. S {(x, β) ⤇ z_β} A
  have ∗: ⊢ f_{β→α} =_{β→α} (λy_β. f_{β→α} · y_β) (is ⊢ ?B =_ ?C)
    by (fact prop-5205)
  moreover have ⊢ S {(f, β→α) ⤇ ?A} (?B =_{β→α} ?C)
```

proof −
have *?A* ∈ $wffs_{\beta\to\alpha}$ **and** *?B* ∈ $wffs_{\beta\to\alpha}$ **and** *?C* ∈ $wffs_{\beta\to\alpha}$
using ‹**S** {(*x*, *β*) ⇀ z_β} *A* ∈ $wffs_\alpha$› **by** *auto*
moreover have ∀ *v* ∈ *vars ?A*. ¬ *is-bound v ?B* ∧ ¬ *is-bound v ?C*
proof
fix *v*
assume *v* ∈ *vars ?A*
then consider (*a*) *v* = (*z*, *β*) | (*b*) *v* ∈ *vars* (**S** {(*x*, *β*) ⇀ z_β} *A*)
by *fastforce*
then show ¬ *is-bound v ?B* ∧ ¬ *is-bound v ?C*
proof *cases*
case *a*
then show *?thesis*
using ‹(*y*, *β*) ≠ (*z*, *β*)› **by** *auto*
next
case *b*
then have ¬ *is-bound v ?B*
by *simp*
moreover from *b* **and** ‹(*y*, *β*) ∉ *vars A*› **and** ‹(*y*, *β*) ≠ (*z*, *β*)› **have** *v* ≠ (*y*, *β*)
using *renaming-substitution-minimal-change* **by** *blast*
then have ¬ *is-bound v ?C*
by *code-simp simp*
ultimately show *?thesis*
by *blast*
qed
qed
ultimately show *?thesis*
by *simp*
qed
then have §*4*: ⊢ (λz_β. **S** {(*x*, *β*) ⇀ z_β} *A*) =$_{\beta\to\alpha}$ (λy_β. **S** {(*x*, *β*) ⇀ y_β} *A*)
proof −
have ⊢ (λz_β. **S** {(*x*, *β*) ⇀ z_β} *A*) ▪ y_β =$_\alpha$ **S** {(*x*, *β*) ⇀ y_β} *A* (**is** ⊢ (λz_β. *?B*) ▪ *?A* =$_-$ -)
proof −
have *?A* ∈ $wffs_\beta$ **and** *?B* ∈ $wffs_\alpha$
by *blast fact*
moreover from ‹(*y*, *β*) ∉ *vars A*› **and** ‹(*y*, *β*) ≠ (*z*, *β*)› **have** ∀ *v* ∈ *vars ?A*. ¬ *is-bound v ?B*
using *absent-var-is-not-bound* **and** *renaming-substitution-minimal-change* **by** *auto*
ultimately have ⊢ (λz_β. **S** {(*x*, *β*) ⇀ z_β} *A*) ▪ y_β =$_\alpha$ **S** {(*z*, *β*) ⇀ y_β} **S** {(*x*, *β*) ⇀ z_β} *A*
using *prop-5203* **by** *fast*
moreover have S {(*z*, *β*) ⇀ y_β} **S** {(*x*, *β*) ⇀ z_β} *A* = **S** {(*x*, *β*) ⇀ y_β} *A*
by (*fact renaming-substitution-composability*[*OF assms*(*2*,*3*)])
ultimately show *?thesis*
by (*simp only*:)
qed
with §*3* **show** *?thesis*
by (*rule rule-R* [**where** *p* = [»,«]]) *auto*

qed
ultimately show *?thesis*
using *Equality-Rules(2,3)* **by** *blast*
qed

lemmas α = *prop-5206*

6.8 Proposition 5207 (β-conversion)

context
begin

private lemma *bound-var-renaming-equality*:
assumes $A \in wffs_\alpha$
and $z_\gamma \neq y_\gamma$
and $(z, \gamma) \notin vars\ A$
shows $\vdash A =_\alpha rename\text{-}bound\text{-}var\ (y, \gamma)\ z\ A$
using *assms* **proof** *induction*
case (*var-is-wff* α x)
then show *?case*
using *prop-5200* **by** *force*
next
case (*con-is-wff* α c)
then show *?case*
using *prop-5200* **by** *force*
next
case (*app-is-wff* α β A B)
then show *?case*
using *Equality-Rules(4)* **by** *auto*
next
case (*abs-is-wff* β A α x)
then show *?case*
proof (*cases* $(y, \gamma) = (x, \alpha)$)
case *True*
have $\vdash \lambda y_\gamma.\ A =_{\gamma\to\beta} \lambda y_\gamma.\ A$
by (*fact abs-is-wff.hyps*[*THEN prop-5200*[*OF wffs-of-type-intros(4)*]])
moreover have $\vdash A =_\beta rename\text{-}bound\text{-}var\ (y, \gamma)\ z\ A$
using *abs-is-wff.IH*[*OF assms(2)*] **and** *abs-is-wff.prems(2)* **by** *fastforce*
ultimately have $\vdash \lambda y_\gamma.\ A =_{\gamma\to\beta} \lambda y_\gamma.\ rename\text{-}bound\text{-}var\ (y, \gamma)\ z\ A$
by (*rule rule-R*[**where** p = [»,«]]) *force+*
moreover
have
$\vdash \lambda y_\gamma.\ rename\text{-}bound\text{-}var\ (y, \gamma)\ z\ A$
$=_{\gamma\to\beta}$
$\lambda z_\gamma.$ **S** $\{(y, \gamma) \rightarrowtail z_\gamma\}$ (*rename-bound-var* $(y, \gamma)\ z\ A$)
proof −
have *rename-bound-var* $(y, \gamma)\ z\ A \in wffs_\beta$
using *hyp-derivable-form-is-wffso*[*OF* ‹$\vdash A =_\beta rename\text{-}bound\text{-}var\ (y, \gamma)\ z\ A$›]
by (*blast dest*: *wffs-from-equality*)

```
    moreover from abs-is-wff.prems(2) have (z, γ) ∉ free-vars (rename-bound-var (y, γ) z A)
      using rename-bound-var-free-vars[OF abs-is-wff.hyps assms(2)] by simp
    moreover from abs-is-wff.prems(2) have is-free-for (z_γ) (y, γ) (rename-bound-var (y, γ) z A)
      using is-free-for-in-rename-bound-var[OF abs-is-wff.hyps assms(2)] by simp
    ultimately show ?thesis
      using α by fast
  qed
  ultimately have ⊢ λy_γ. A =_{γ→β} λz_γ. S {(y, γ) ⇀ z_γ} (rename-bound-var (y, γ) z A)
    by (rule Equality-Rules(3))
  then show ?thesis
    using True by auto
next
  case False
  have ⊢ λx_α. A =_{α→β} λx_α. A
    by (fact abs-is-wff.hyps[THEN prop-5200[OF wffs-of-type-intros(4)]])
  moreover have ⊢ A =_β rename-bound-var (y, γ) z A
    using abs-is-wff.IH[OF assms(2)] and abs-is-wff.prems(2) by fastforce
  ultimately have ⊢ λx_α. A =_{α→β} λx_α. rename-bound-var (y, γ) z A
    by (rule rule-R[where p = [»,«]]) force+
  then show ?thesis
    using False by auto
qed
qed

proposition prop-5207:
  assumes A ∈ wffs_α and B ∈ wffs_β
  and is-free-for A (x, α) B
  shows ⊢ (λx_α. B) • A =_β S {(x, α) ⇀ A} B
using assms proof (induction form-size B arbitrary: B β rule: less-induct)
  case less
  from less(3,1,2,4) show ?case
  proof (cases B rule: wffs-of-type-cases)
    case (var-is-wff y)
    then show ?thesis
    proof (cases y_β = x_α)
      case True
      then have α = β
        by simp
      moreover from assms(1) have ⊢ (λx_α. x_α) • A =_α A
        using axiom-4-2 by (intro axiom-is-derivable-from-no-hyps)
      moreover have S {(x, α) ⇀ A} (x_α) = A
        by force
      ultimately show ?thesis
        unfolding True and var-is-wff by simp
    next
      case False
      with assms(1) have ⊢ (λx_α. y_β) • A =_β y_β
        using axiom-4-1-var by (intro axiom-is-derivable-from-no-hyps)
      moreover from False have S {(x, α) ⇀ A} (y_β) = y_β
```

```
    by auto
   ultimately show ?thesis
    unfolding False and var-is-wff by simp
 qed
next
 case (con-is-wff c)
  from assms(1) have ⊢ (λx_α. ⦇c⦈_β) ∙ A =_β ⦇c⦈_β
   using axiom-4-1-con by (intro axiom-is-derivable-from-no-hyps)
 moreover have S {(x, α) ↣ A} (⦇c⦈_β) = ⦇c⦈_β
   by auto
 ultimately show ?thesis
   by (simp only: con-is-wff)
next
 case (app-is-wff γ D C)
 have form-size D < form-size B and form-size C < form-size B
   unfolding app-is-wff(1) by simp-all
 from less(4)[unfolded app-is-wff(1)] have is-free-for A (x, α) D and is-free-for A (x, α) C
   using is-free-for-from-app by iprover+
 from ‹is-free-for A (x, α) D› have ⊢ (λx_α. D) ∙ A =_{γ→β} S {(x, α) ↣ A} D
   by (fact less(1)[OF ‹form-size D < form-size B› assms(1) app-is-wff(2)])
 moreover from ‹is-free-for A (x, α) C› have ⊢ (λx_α. C) ∙ A =_γ S {(x, α) ↣ A} C
   by (fact less(1)[OF ‹form-size C < form-size B› assms(1) app-is-wff(3)])
 moreover have ⊢ (λx_α. D ∙ C) ∙ A =_β ((λx_α. D) ∙ A) ∙ ((λx_α. C) ∙ A)
   by (fact axiom-4-3[OF assms(1) app-is-wff(2,3), THEN axiom-is-derivable-from-no-hyps])
 ultimately show ?thesis
   unfolding app-is-wff(1) using Equality-Rules(3,4) and substitute.simps(3) by presburger
next
 case (abs-is-wff δ D γ y)
 then show ?thesis
 proof (cases y_γ = x_α)
   case True
   with abs-is-wff(1) have ⊢ (λx_α. λy_γ. D) ∙ A =_β λy_γ. D
     using axiom-4-5[OF assms(1) abs-is-wff(3)] by (simp add: axiom-is-derivable-from-no-hyps)
   moreover have S {(x, α) ↣ A} (λy_γ. D) = λy_γ. D
     using True by (simp add: empty-substitution-neutrality fmdrop-fmupd-same)
   ultimately show ?thesis
     unfolding abs-is-wff(2) by (simp only:)
 next
   case False
   have form-size D < form-size B
     unfolding abs-is-wff(2) by simp
   have is-free-for A (x, α) D
     using is-free-for-from-abs[OF less(4)[unfolded abs-is-wff(2)]] and ‹y_γ ≠ x_α› by blast
   have ⊢ (λx_α. (λy_γ. D)) ∙ A =_β λy_γ. S {(x, α) ↣ A} D
   proof (cases (y, γ) ∉ vars A)
     case True
     with ‹y_γ ≠ x_α› have ⊢ (λx_α. λy_γ. D) ∙ A =_{γ→δ} λy_γ. (λx_α. D) ∙ A
       using axiom-4-4[OF assms(1) abs-is-wff(3)] and axiom-is-derivable-from-no-hyps by auto
     moreover have ⊢ (λx_α. D) ∙ A =_δ S {(x, α) ↣ A} D
```

```
    by
      (
        fact less(1)
          [OF ‹form-size D < form-size B› assms(1) ‹D ∈ wffs_δ› ‹is-free-for A (x, α) D›]
      )
  ultimately show ?thesis
    unfolding abs-is-wff(1) by (rule rule-R[where p = [»,«]]) force+
next
  case False
  have finite (vars {A, D})
    using vars-form-finiteness and vars-form-set-finiteness by simp
  then obtain z where (z, γ) ∉ ({(x, α), (y, γ)} ∪ vars {A, D})
    using fresh-var-existence by (metis Un-insert-left finite.simps insert-is-Un)
  then have z_γ ≠ x_α and z_γ ≠ y_γ and (z, γ) ∉ vars {A, D}
    by simp-all
  then show ?thesis
  proof (cases (x, α) ∉ free-vars D)
    case True
    define D′ where D′ = S {(y, γ) ⇀ z_γ} D
    have is-substitution {(y, γ) ⇀ z_γ}
      by auto
    with ‹D ∈ wffs_δ› and D′-def have D′ ∈ wffs_δ
      using substitution-preserves-typing by blast
    then have ⊢ (λx_α. λz_γ. D′) ▪ A =_{γ→δ} λz_γ. (λx_α. D′) ▪ A
      using ‹z_γ ≠ x_α› and ‹(z, γ) ∉ vars {A, D}› and axiom-4-4[OF assms(1)]
      and axiom-is-derivable-from-no-hyps
      by auto
    moreover have §2: ⊢ (λx_α. D′) ▪ A =_δ D′
    proof −
      have form-size D′ = form-size D
        unfolding D′-def by (fact renaming-substitution-preserves-form-size)
      then have form-size D′ < form-size B
        using ‹form-size D < form-size B› by simp
      moreover from ‹z_γ ≠ x_α› have is-free-for A (x, α) D′
        unfolding D′-def and is-free-for-def
        using substitution-preserves-freeness[OF True] and is-free-at-in-free-vars
        by fast
      ultimately have ⊢ (λx_α. D′) ▪ A =_δ S {(x, α) ⇀ A} D′
        using less(1) and assms(1) and ‹D′ ∈ wffs_δ› by simp
      moreover from ‹z_γ ≠ x_α› have (x, α) ∉ free-vars D′
        unfolding D′-def using substitution-preserves-freeness[OF True] by fast
      then have S {(x, α) ⇀ A} D′ = D′
        by (fact free-var-singleton-substitution-neutrality)
      ultimately show ?thesis
        by (simp only:)
    qed
    ultimately have §3: ⊢ (λx_α. λz_γ. D′) ▪ A =_{γ→δ} λz_γ. D′ (is ‹⊢ ?A3›)
      by (rule rule-R[where p = [»,«]]) force+
    moreover have §4: ⊢ (λy_γ. D) =_{γ→δ} λz_γ. D′
```

proof −
have $(z, \gamma) \notin$ *free-vars* D
using ‹$(z, \gamma) \notin$ *vars* $\{A, D\}$› **and** *free-vars-in-all-vars-set* **by** *auto*
moreover have *is-free-for* (z_γ) (y, γ) D
using ‹$(z, \gamma) \notin$ *vars* $\{A, D\}$› **and** *absent-var-is-free-for* **by** *force*
ultimately have $\vdash \lambda y_\gamma.\ D =_{\gamma\to\delta} \lambda z_\gamma.$ **S** $\{(y, \gamma) \rightarrowtail z_\gamma\}$ D
using α[*OF* ‹$D \in$ *wffs*$_\delta$›] **by** *fast*
then show *?thesis*
using *D′-def* **by** *blast*
qed
ultimately have §5: $\vdash (\lambda x_\alpha.\ \lambda y_\gamma.\ D) \cdot A =_{\gamma\to\delta} \lambda y_\gamma.\ D$
proof −
note *rule-RR′* = *rule-RR*[*OF disjI2*]
have §5$_1$: $\vdash (\lambda x_\alpha.\ \lambda y_\gamma.\ D) \cdot A =_{\gamma\to\delta} \lambda z_\gamma.\ D'$ (**is** ‹$\vdash$ *?A5*$_1$›)
by (*rule rule-RR′*[*OF* §4, **where** $p = [\langle, \rangle, \langle, \langle]$ **and** C = *?A3*]) (*use* §3 **in** ‹*force*+›)
show *?thesis*
by (*rule rule-RR′*[*OF* §4, **where** $p = [\rangle]$ **and** C = *?A5*$_1$]) (*use* §5$_1$ **in** ‹*force*+›)
qed
then show *?thesis*
using *free-var-singleton-substitution-neutrality*[*OF* ‹$(x, \alpha) \notin$ *free-vars* D›]
by (*simp only*: ‹$\beta = \gamma\to\delta$›)
next
case *False*
have $(y, \gamma) \notin$ *free-vars* A
proof (*rule ccontr*)
assume $\neg$ $(y, \gamma) \notin$ *free-vars* A
moreover from ‹$\neg$ $(x, \alpha) \notin$ *free-vars* D› **obtain** p
where $p \in$ *positions* D **and** *is-free-at* (x, α) p D
using *free-vars-in-is-free-at* **by** *blast*
then have ‹ # $p \in$ *positions* $(\lambda y_\gamma.\ D)$ **and** *is-free-at* (x, α) (‹ # p) $(\lambda y_\gamma.\ D)$
using *is-free-at-to-abs*[*OF* ‹*is-free-at* (x, α) p D›] **and** ‹$y_\gamma \neq x_\alpha$› **by** (*simp*, *fast*)
moreover have *in-scope-of-abs* (y, γ) (‹ # p) $(\lambda y_\gamma.\ D)$
by *force*
ultimately have $\neg$ *is-free-for* A (x, α) $(\lambda y_\gamma.\ D)$
by *blast*
with ‹*is-free-for* A (x, α) B›[*unfolded abs-is-wff*(2)] **show** *False*
by *contradiction*
qed
define A' **where** A' = *rename-bound-var* (y, γ) z A
have $A' \in$ *wffs*$_\alpha$
unfolding *A′-def* **by** (*fact rename-bound-var-preserves-typing*[*OF assms*(1)])
from ‹$(z, \gamma) \notin$ *vars* $\{A, D\}$› **have** $(y, \gamma) \notin$ *vars* A'
using
old-var-not-free-not-occurring-after-rename
[
OF assms(1) ‹$z_\gamma \neq y_\gamma$› ‹$(y, \gamma) \notin$ *free-vars* A›
]
unfolding *A′-def* **by** *simp*
from *A′-def* **have** §6: $\vdash A =_\alpha A'$

```
      using bound-var-renaming-equality[OF assms(1) ‹z_γ ≠ y_γ›] and ‹(z, γ) ∉ vars {A, D}›
      by simp
    moreover have §7: ⊢ (λx_α. λy_γ. D) ▪ A' =_{γ→δ} λy_γ. (λx_α. D) ▪ A' (is ‹⊢ ?A7›)
      using axiom-4-4[OF ‹A' ∈ wffs_α› ‹D ∈ wffs_δ›]
      and ‹(y, γ) ∉ vars A'› and ‹y_γ ≠ x_α› and axiom-is-derivable-from-no-hyps
      by auto
    ultimately have §8: ⊢ (λx_α. λy_γ. D) ▪ A =_{γ→δ} λy_γ. (λx_α. D) ▪ A
    proof −
      note rule-RR' = rule-RR[OF disjI2]
      have §8_1: ⊢ (λx_α. λy_γ. D) ▪ A =_{γ→δ} λy_γ. (λx_α. D) ▪ A' (is ‹⊢ ?A8_1›)
        by (rule rule-RR'[OF §6, where p = [«,»,»] and C = ?A7]) (use §7 in ‹force+›)
      show ?thesis
        by (rule rule-RR'[OF §6, where p = [»,«,»] and C = ?A8_1]) (use §8_1 in ‹force+›)
    qed
    moreover have form-size D < form-size B
      unfolding abs-is-wff(2) by (simp only: form-size.simps(4) lessI)
    with assms(1) have §9: ⊢ (λx_α. D) ▪ A =_δ S {(x, α) ⇀ A} D
      using less(1) and ‹D ∈ wffs_δ› and ‹is-free-for A (x, α) D› by (simp only:)
    ultimately show ?thesis
      unfolding ‹β = γ→δ› by (rule rule-R[where p = [»,«]]) force+
  qed
 qed
 then show ?thesis
  unfolding abs-is-wff(2) using False and singleton-substitution-simps(4) by simp
 qed
qed
qed

end
```

6.9 Proposition 5208

```
proposition prop-5208:
 assumes vs ≠ [] and B ∈ wffs_β
 shows ⊢ ▪^Ω_⋆ (λ^Ω_⋆ vs B) (map FVar vs) =_β B
using assms(1) proof (induction vs rule: list-nonempty-induct)
 case (single v)
 obtain x and α where v = (x, α)
  by fastforce
 then have ▪^Ω_⋆ (λ^Ω_⋆ [v] B) (map FVar [v]) = (λx_α. B) ▪ x_α
  by simp
 moreover have ⊢ (λx_α. B) ▪ x_α =_β B
 proof −
  have is-free-for (x_α) (x, α) B
   by fastforce
  then have ⊢ (λx_α. B) ▪ x_α =_β S {(x, α) ⇀ x_α} B
   by (rule prop-5207 [OF wffs-of-type-intros(1) assms(2)])
  then show ?thesis
   using identity-singleton-substitution-neutrality by (simp only:)
```

```
  qed
  ultimately show ?case
    by (simp only:)
next
  case (cons v vs)
  obtain x and α where v = (x, α)
    by fastforce
  have ⊢ ·^Q_⋆ (λ^Q_⋆ (v # vs) B) (map FVar (v # vs)) =_β ·^Q_⋆ (λ^Q_⋆ vs B) (map FVar vs)
  proof −
    have ·^Q_⋆ (λ^Q_⋆ (v # vs) B) (map FVar (v # vs)) ∈ wffs_β
    proof −
      have λ^Q_⋆ (v # vs) B ∈ wffs_{foldr (→) (map snd (v # vs)) β}
        using generalized-abs-wff [OF assms(2)] by blast
      moreover
      have ∀ k < length (map FVar (v # vs)). map FVar (v # vs) ! k ∈ wffs_{map snd (v # vs) ! k}
      proof safe
        fix k
        assume ∗: k < length (map FVar (v # vs))
        moreover obtain x and α where (v # vs) ! k = (x, α)
          by fastforce
        with ∗ have map FVar (v # vs) ! k = x_α and map snd (v # vs) ! k = α
          by (metis length-map nth-map snd-conv)+
        ultimately show map FVar (v # vs) ! k ∈ wffs_{map snd (v # vs) ! k}
          by fastforce
      qed
      ultimately show ?thesis
        using generalized-app-wff[where As = map FVar (v # vs) and ts = map snd (v # vs)] by
simp
    qed
    then have
      ⊢ ·^Q_⋆ (λ^Q_⋆ (v # vs) B) (map FVar (v # vs)) =_β ·^Q_⋆ (λ^Q_⋆ (v # vs) B) (map FVar (v # vs))
      by (fact prop-5200)
    then have
      ⊢ ·^Q_⋆ (λ^Q_⋆ (v # vs) B) (map FVar (v # vs)) =_β ·^Q_⋆ ((λ^Q_⋆ (v # vs) B) · FVar v) (map FVar
vs)
      by simp
    moreover have ⊢ (λ^Q_⋆ (v # vs) B) · FVar v =_{foldr (→) (map snd vs) β} (λ^Q_⋆ vs B)
    proof −
      have ⊢ (λ^Q_⋆ (v # vs) B) · FVar v =_{foldr (→) (map snd vs) β} S {v ⤚ FVar v} (λ^Q_⋆ vs B)
      proof −
        from ‹v = (x, α)› have λ^Q_⋆ (v # vs) B = λx_α. λ^Q_⋆ vs B
          by simp
        have λ^Q_⋆ vs B ∈ wffs_{foldr (→) (map snd vs) β}
          using generalized-abs-wff[OF assms(2)] by blast
        moreover have is-free-for (x_α) (x, α) (λ^Q_⋆ vs B)
          by fastforce
        ultimately
        have ⊢ (λx_α. λ^Q_⋆ vs B) · x_α =_{foldr (→) (map snd vs) β} S {(x, α) ⤚ x_α} λ^Q_⋆ vs B
```

```
      by (rule prop-5207 [OF wffs-of-type-intros(1)])
    with ‹v = (x, α)› show ?thesis
      by simp
  qed
 then show ?thesis
  using identity-singleton-substitution-neutrality by (simp only:)
 qed
 ultimately show ?thesis
 proof (induction rule: rule-R [where p = [»] @ replicate (length vs) «])
  case occ-subform
  then show ?case
   unfolding equality-of-type-def using leftmost-subform-in-generalized-app
   by (metis append-Cons append-Nil is-subform-at.simps(3) length-map)
 next
  case replacement
  then show ?case
   unfolding equality-of-type-def using leftmost-subform-in-generalized-app-replacement
   and is-subform-implies-in-positions and leftmost-subform-in-generalized-app
   by (metis append-Cons append-Nil length-map replace-right-app)
 qed
qed
moreover have ⊢ ·^Ω⋆ (λ^Ω⋆ vs B) (map FVar vs) =β B
 by (fact cons.IH)
ultimately show ?case
 by (rule rule-R [where p = [»]]) auto
qed
```

6.10 Proposition 5209

```
proposition prop-5209:
 assumes A ∈ wffs_α and B ∈ wffs_β and C ∈ wffs_β
 and ⊢ B =β C
 and is-free-for A (x, α) (B =β C)
 shows ⊢ S {(x, α) ⇀ A} (B =β C)
proof −
 have ⊢ (λx_α. B) · A =β (λx_α. B) · A
 proof −
  have (λx_α. B) · A ∈ wffs_β
   using assms(1,2) by blast
  then show ?thesis
   by (fact prop-5200)
 qed
 from this and assms(4) have ⊢ (λx_α. B) · A =β (λx_α. C) · A
  by (rule rule-R [where p = [»,«,«]]) force+
 moreover have ⊢ (λx_α. B) · A =β S {(x, α) ⇀ A} B
 proof −
  from assms(5)[unfolded equality-of-type-def] have is-free-for A (x, α) (Q_β · B)
   by (rule is-free-for-from-app)
  then have is-free-for A (x, α) B
```

```
    by (rule is-free-for-from-app)
  with assms(1,2) show ?thesis
    by (rule prop-5207)
 qed
 moreover have ⊢ (λx_α. C) ▪ A =_β S {(x, α) ⇀ A} C
 proof −
  from assms(5)[unfolded equality-of-type-def] have is-free-for A (x, α) C
    by (rule is-free-for-from-app)
  with assms(1,3) show ?thesis
    by (rule prop-5207)
 qed
 ultimately have ⊢ (S {(x, α) ⇀ A} B) =_β (S {(x, α) ⇀ A} C)
  using Equality-Rules(2,3) by blast
 then show ?thesis
  by simp
qed
```

6.11 Proposition 5210

```
proposition prop-5210:
 assumes B ∈ wffs_β
 shows ⊢ T_o =_o (B =_β B)
proof −
 have §1:
  ⊢
    ((λ𝔫_β. 𝔫_β) =_{β→β} (λ𝔫_β. 𝔫_β))
    =_o
    ∀𝔵_β. ((λ𝔫_β. 𝔫_β) ▪ 𝔵_β =_β (λ𝔫_β. 𝔫_β) ▪ 𝔵_β)
 proof −
  have ⊢ (𝔣_{β→β} =_{β→β} 𝔤_{β→β}) =_o ∀𝔵_β. (𝔣_{β→β} ▪ 𝔵_β =_β 𝔤_{β→β} ▪ 𝔵_β) (is ⊢ ?B =_o ?C)
    using axiom-3[unfolded equivalence-def] by (rule axiom-is-derivable-from-no-hyps)
  moreover have (λ𝔫_β. 𝔫_β) ∈ wffs_{β→β} and ?B ∈ wffs_o and ?C ∈ wffs_o
    by auto
  moreover have is-free-for (λ𝔫_β. 𝔫_β) (𝔣, β→β) (?B =_o ?C)
    by simp
  ultimately have ⊢ S {(𝔣, β→β) ⇀ (λ𝔫_β. 𝔫_β)} (?B =_o ?C) (is ⊢ ?S)
    using prop-5209 by presburger
  moreover have ?S =
    (
      (λ𝔫_β. 𝔫_β) =_{β→β} 𝔤_{β→β}) =_o ∀𝔵_β. ((λ𝔫_β. 𝔫_β) ▪ 𝔵_β =_β 𝔤_{β→β} ▪ 𝔵_β
    ) (is - = ?B' =_o ?C')
    by simp
  ultimately have ⊢ ?B' =_o ?C'
    by (simp only:)
  moreover from ‹(λ𝔫_β. 𝔫_β) ∈ wffs_{β→β}› have ?B' ∈ wffs_o and ?C' ∈ wffs_o
    by auto
  moreover have is-free-for (λ𝔫_β. 𝔫_β) (𝔤, β→β) (?B' =_o ?C')
    by simp
  ultimately have ⊢ S {(𝔤, β→β) ⇀ (λ𝔫_β. 𝔫_β)} (?B' =_o ?C') (is ⊢ ?S')
```

```
    using prop-5209[OF ‹(λ𝔫β. 𝔫β) ∈ wffsβ→β›] by blast
  then show ?thesis
    by simp
 qed
 then have ⊢ (λ𝔵β. To) =β→o (λ𝔵β. (𝔵β =β 𝔵β))
 proof −
  have λ𝔫β. 𝔫β ∈ wffsβ→β
    by blast
  then have ⊢ λ𝔫β. 𝔫β =β→β λ𝔫β. 𝔫β
    by (fact prop-5200)
  with §1 have ⊢ ∀𝔵β. ((λ𝔫β. 𝔫β) ▪ 𝔵β =β (λ𝔫β. 𝔫β) ▪ 𝔵β)
    using rule-R and is-subform-at.simps(1) by blast
  moreover have ⊢ (λ𝔫β. 𝔫β) ▪ 𝔵β =β 𝔵β
    using axiom-4-2[OF wffs-of-type-intros(1)] by (rule axiom-is-derivable-from-no-hyps)
  ultimately have ⊢ ∀𝔵β. (𝔵β =β (λ𝔫β. 𝔫β) ▪ 𝔵β)
    by (rule rule-R[where p = [»,«,«,»]]) auto
  from this and ‹⊢ (λ𝔫β. 𝔫β) ▪ 𝔵β =β 𝔵β› have ⊢ ∀𝔵β. (𝔵β =β 𝔵β)
    by (rule rule-R[where p = [»,«,»]]) auto
  then show ?thesis
    unfolding forall-def and PI-def by (fold equality-of-type-def)
 qed
 from this and assms have 3: ⊢ (λ𝔵β. To) ▪ B =o (λ𝔵β. (𝔵β =β 𝔵β)) ▪ B
  by (rule Equality-Rules(5))
 then show ?thesis
 proof −
  have ⊢ (λ𝔵β. To) ▪ B =o To
    using prop-5207[OF assms true-wff] by fastforce
  from 3 and this have ⊢ To =o (λ𝔵β. (𝔵β =β 𝔵β)) ▪ B
    by (rule rule-R[where p = [«,»]]) auto
  moreover have ⊢ (λ𝔵β. (𝔵β =β 𝔵β)) ▪ B =o (B =β B)
  proof −
    have 𝔵β =β 𝔵β ∈ wffso and is-free-for B (𝔵, β) (𝔵β =β 𝔵β)
      by (blast, intro is-free-for-in-equality is-free-for-in-var)
    moreover have S {(𝔵, β) ⇀ B} (𝔵β =β 𝔵β) = (B =β B)
      by simp
    ultimately show ?thesis
      using prop-5207[OF assms] by metis
  qed
  ultimately show ?thesis
    by (rule rule-R [where p = [»]]) auto
 qed
qed
```

6.12 Proposition 5211

```
proposition prop-5211:
 shows ⊢ (To ∧^Q To) =o To
proof −
 have const-T-wff: (λxo. To) ∈ wffso→o for x
```

```
 by blast
 have §1: ⊢ (λ𝔶o. T o) ▪ T o ∧^Q (λ𝔶o. T o) ▪ F o =o ∀𝔵o. (λ𝔶o. T o) ▪ 𝔵o
 proof −
  have ⊢ 𝔤o→o ▪ T o ∧^Q 𝔤o→o ▪ F o =o ∀𝔵o. 𝔤o→o ▪ 𝔵o (is ⊢ ?B =o ?C)
   using axiom-1[unfolded equivalence-def] by (rule axiom-is-derivable-from-no-hyps)
  moreover have ?B ∈ wffs o and ?C ∈ wffs o
   by auto
  moreover have is-free-for (λ𝔶o. T o) (𝔤, o→o) (?B =o ?C)
   by simp
  ultimately have ⊢ S {(𝔤, o→o) ⇀ (λ𝔶o. T o)} (?B =o ?C)
   using const-T-wff and prop-5209 by presburger
  then show ?thesis
   by simp
 qed
 then have ⊢ T o ∧^Q T o =o ∀𝔵o. T o
 proof −
  have T-β-redex: ⊢ (λ𝔶o. T o) ▪ A =o T o if A ∈ wffs o for A
   using that and prop-5207[OF that true-wff] by fastforce
  from §1 and T-β-redex[OF true-wff]
  have ⊢ T o ∧^Q (λ𝔶o. T o) ▪ F o =o ∀𝔵o. (λ𝔶o. T o) ▪ 𝔵o
   by (rule rule-R[where p = [«,»,«,»]]) force+
  from this and T-β-redex[OF false-wff] have ⊢ T o ∧^Q T o =o ∀𝔵o. (λ𝔶o. T o) ▪ 𝔵o
   by (rule rule-R[where p = [«,»,»]]) force+
  from this and T-β-redex[OF wffs-of-type-intros(1)] show ?thesis
   by (rule rule-R[where p = [»,»,«]]) force+
 qed
 moreover have ⊢ T o =o ∀𝔵o. T o
  using prop-5210[OF const-T-wff] by simp
 ultimately show ?thesis
  using Equality-Rules(2,3) by blast
qed

lemma true-is-derivable:
 shows ⊢ T o
 unfolding true-def using Q-wff by (rule prop-5200)
```

6.13 Proposition 5212

```
proposition prop-5212:
 shows ⊢ T o ∧^Q T o
proof −
 have ⊢ T o
  by (fact true-is-derivable)
 moreover have ⊢ (T o ∧^Q T o) =o T o
  by (fact prop-5211)
 then have ⊢ T o ≡^Q (T o ∧^Q T o)
  unfolding equivalence-def by (fact Equality-Rules(2))
 ultimately show ?thesis
  by (rule Equality-Rules(1))
```

qed

6.14 Proposition 5213

proposition *prop-5213*:
assumes $\vdash A =_\alpha B$ **and** $\vdash C =_\beta D$
shows $\vdash (A =_\alpha B) \wedge^{\mathfrak{Q}} (C =_\beta D)$
proof −
from *assms* **have** $A \in wffs_\alpha$ **and** $C \in wffs_\beta$
using *hyp-derivable-form-is-wffso* **and** *wffs-from-equality* **by** *blast+*
have $\vdash T_o =_o (A =_\alpha A)$
by (*fact prop-5210*[*OF* ‹$A \in wffs_\alpha$›])
moreover have $\vdash A =_\alpha B$
by *fact*
ultimately have $\vdash T_o =_o (A =_\alpha B)$
by (*rule rule-R*[**where** $p = [\text{»},\text{»}]$]) *force+*
have $\vdash T_o =_o (C =_\beta C)$
by (*fact prop-5210*[*OF* ‹$C \in wffs_\beta$›])
moreover have $\vdash C =_\beta D$
by *fact*
ultimately have $\vdash T_o =_o (C =_\beta D)$
by (*rule rule-R*[**where** $p = [\text{»},\text{»}]$]) *force+*
then show *?thesis*
proof −
have $\vdash T_o \wedge^{\mathfrak{Q}} T_o$
by (*fact prop-5212*)
from *this* **and** ‹$\vdash T_o =_o (A =_\alpha B)$› **have** $\vdash (A =_\alpha B) \wedge^{\mathfrak{Q}} T_o$
by (*rule rule-R*[**where** $p = [\text{«},\text{»}]$]) *force+*
from *this* **and** ‹$\vdash T_o =_o (C =_\beta D)$› **show** *?thesis*
by (*rule rule-R*[**where** $p = [\text{»}]$]) *force+*
qed
qed

6.15 Proposition 5214

proposition *prop-5214*:
shows $\vdash T_o \wedge^{\mathfrak{Q}} F_o =_o F_o$
proof −
have *id-on-o-is-wff*: $(\lambda \mathfrak{x}_o.\ \mathfrak{x}_o) \in wffs_{o\to o}$
by *blast*
have §*1*: $\vdash (\lambda \mathfrak{x}_o.\ \mathfrak{x}_o) \cdot T_o \wedge^{\mathfrak{Q}} (\lambda \mathfrak{x}_o.\ \mathfrak{x}_o) \cdot F_o =_o \forall \mathfrak{x}_o.\ (\lambda \mathfrak{x}_o.\ \mathfrak{x}_o) \cdot \mathfrak{x}_o$
proof −
have $\vdash \mathfrak{g}_{o\to o} \cdot T_o \wedge^{\mathfrak{Q}} \mathfrak{g}_{o\to o} \cdot F_o =_o \forall \mathfrak{x}_o.\ \mathfrak{g}_{o\to o} \cdot \mathfrak{x}_o$ (**is** $\vdash$ *?B* $=_o$ *?C*)
using *axiom-1*[*unfolded equivalence-def*] **by** (*rule axiom-is-derivable-from-no-hyps*)
moreover have *?B* $\in wffs_o$ **and** *?C* $\in wffs_o$ **and** *is-free-for* $(\lambda \mathfrak{x}_o.\ \mathfrak{x}_o)$ $(\mathfrak{g},\ o\to o)$ (*?B* $=_o$ *?C*)
by *auto*
ultimately have $\vdash$ **S** $\{(\mathfrak{g},\ o\to o) \rightarrowtail (\lambda \mathfrak{x}_o.\ \mathfrak{x}_o)\}$ (*?B* $=_o$ *?C*)
using *id-on-o-is-wff* **and** *prop-5209* **by** *presburger*
then show *?thesis*

```
    by simp
  qed
  then have ⊢ T o ∧^Q F o =o ∀𝔯o. 𝔯o
  proof −
    have id-β-redex: ⊢ (λ𝔯o. 𝔯o) • A =o A if A ∈ wffs_o for A
      by (fact axiom-is-derivable-from-no-hyps[OF axiom-4-2[OF that]])
    from §1 and id-β-redex[OF true-wff]
    have ⊢ T o ∧^Q (λ𝔯o. 𝔯o) • F o =o ∀𝔯o. (λ𝔯o. 𝔯o) • 𝔯o
      by (rule rule-R[where p = [«,»,«,»]]) force+
    from this and id-β-redex[OF false-wff] have ⊢ T o ∧^Q F o =o ∀𝔯o. (λ𝔯o. 𝔯o) • 𝔯o
      by (rule rule-R[where p = [«,»,»]]) force+
    from this and id-β-redex[OF wffs-of-type-intros(1)] show ?thesis
      by (rule rule-R[where p = [»,»,«]]) force+
  qed
  then show ?thesis
    by simp
qed
```

6.16 Proposition 5215 (Universal Instantiation)

```
proposition prop-5215:
  assumes 𝓗 ⊢ ∀ x_α. B and A ∈ wffs_α
  and is-free-for A (x, α) B
  shows 𝓗 ⊢ S {(x, α) ⇀ A} B
proof −
  from assms(1) have is-hyps 𝓗
    by (blast elim: is-derivable-from-hyps.cases)
  from assms(1) have 𝓗 ⊢ (λ𝔯α. T o) =α→o (λx_α. B)
    by simp
  with assms(2) have 𝓗 ⊢ (λ𝔯α. T o) • A =o (λx_α. B) • A
    by (intro Equality-Rules(5))
  then have 𝓗 ⊢ T o =o S {(x, α) ⇀ A} B
  proof −
    have 𝓗 ⊢ (λ𝔯α. T o) • A =o T o
    proof −
      have ⊢ (λ𝔯α. T o) • A =o T o
        using prop-5207[OF assms(2) true-wff is-free-for-in-true] and derived-substitution-simps(1)
        by (simp only:)
      from this and ‹is-hyps 𝓗› show ?thesis
        by (rule derivability-implies-hyp-derivability)
    qed
    moreover have 𝓗 ⊢ (λx_α. B) • A =o S {(x, α) ⇀ A} B
    proof −
      have B ∈ wffs_o
        using hyp-derivable-form-is-wffso[OF assms(1)] by (fastforce elim: wffs-from-forall)
      with assms(2,3) have ⊢ (λx_α. B) • A =o S {(x, α) ⇀ A} B
        using prop-5207 by (simp only:)
      from this and ‹is-hyps 𝓗› show ?thesis
        by (rule derivability-implies-hyp-derivability)
```

qed
ultimately show *?thesis*
using ‹$\mathcal{H} \vdash (\lambda \mathfrak{x}_\alpha.\ T_o) \cdot A =_o (\lambda x_\alpha.\ B) \cdot A$› **and** *Equality-Rules(2,3)* **by** *meson*
qed
then show *?thesis*
proof −
have $\mathcal{H} \vdash T_o$
by (*fact derivability-implies-hyp-derivability*[*OF true-is-derivable* ‹*is-hyps* $\mathcal{H}$›])
from *this* **and** ‹$\mathcal{H} \vdash T_o =_o \mathbf{S}\ \{(x,\ \alpha) \rightarrowtail A\}\ B$› **show** *?thesis*
by (*rule Equality-Rules(1)*[*unfolded equivalence-def*])
qed
qed

lemmas $\forall I$ = *prop-5215*

6.17 Proposition 5216

proposition *prop-5216*:
assumes $A \in \mathit{wffs}_o$
shows $\vdash (T_o \wedge^{\mathcal{Q}} A) =_o A$
proof −
let *?B* = $\lambda \mathfrak{x}_o.\ (T_o \wedge^{\mathcal{Q}} \mathfrak{x}_o =_o \mathfrak{x}_o)$
have *B-is-wff*: *?B* $\in \mathit{wffs}_{o \to o}$
by *auto*
have §*1*: $\vdash$ *?B* $\cdot$ $T_o \wedge^{\mathcal{Q}}$ *?B* $\cdot$ $F_o =_o \forall \mathfrak{x}_o.$ *?B* $\cdot$ $\mathfrak{x}_o$
proof −
have $\vdash \mathfrak{g}_{o \to o} \cdot T_o \wedge^{\mathcal{Q}} \mathfrak{g}_{o \to o} \cdot F_o =_o \forall \mathfrak{x}_o.\ \mathfrak{g}_{o \to o} \cdot \mathfrak{x}_o$ (**is** $\vdash$ *?C* $=_o$ *?D*)
using *axiom-1*[*unfolded equivalence-def*] **by** (*rule axiom-is-derivable-from-no-hyps*)
moreover have *?C* $\in \mathit{wffs}_o$ **and** *?D* $\in \mathit{wffs}_o$ **and** *is-free-for ?B* ($\mathfrak{g}$, $o \to o$) (*?C* $=_o$ *?D*)
by *auto*
ultimately have $\vdash \mathbf{S}\ \{(\mathfrak{g},\ o \to o) \rightarrowtail$ *?B*$\}$ (*?C* $=_o$ *?D*)
using *B-is-wff* **and** *prop-5209* **by** *presburger*
then show *?thesis*
by *simp*
qed
have $*$: *is-free-for A* ($\mathfrak{x}$, *o*) ($T_o \wedge^{\mathcal{Q}} \mathfrak{x}_o =_o \mathfrak{x}_o$) **for** *A*
by (*intro is-free-for-in-conj is-free-for-in-equality is-free-for-in-true is-free-for-in-var*)
have $\vdash (T_o \wedge^{\mathcal{Q}} T_o =_o T_o) \wedge^{\mathcal{Q}} (T_o \wedge^{\mathcal{Q}} F_o =_o F_o)$
by (*fact prop-5213*[*OF prop-5211 prop-5214*])
moreover
have $\vdash (T_o \wedge^{\mathcal{Q}} T_o =_o T_o) \wedge^{\mathcal{Q}} (T_o \wedge^{\mathcal{Q}} F_o =_o F_o) =_o \forall \mathfrak{x}_o.\ (T_o \wedge^{\mathcal{Q}} \mathfrak{x}_o =_o \mathfrak{x}_o)$
proof −
have *B-β-redex*: $\vdash$ *?B* $\cdot$ $A =_o (T_o \wedge^{\mathcal{Q}} A =_o A)$ **if** $A \in \mathit{wffs}_o$ **for** *A*
proof −
have $T_o \wedge^{\mathcal{Q}} \mathfrak{x}_o =_o \mathfrak{x}_o \in \mathit{wffs}_o$
by *blast*
moreover have $\mathbf{S}\ \{(\mathfrak{x},\ o) \rightarrowtail A\}\ (T_o \wedge^{\mathcal{Q}} \mathfrak{x}_o =_o \mathfrak{x}_o) = (T_o \wedge^{\mathcal{Q}} A =_o A)$
by *simp*
ultimately show *?thesis*

using $*$ **and** *prop-5207*[*OF that*] **by** *metis*
qed
from §*1* **and** *B-β-redex*[*OF true-wff*]
have $\vdash (T_o \wedge^{\mathfrak{Q}} T_o =_o T_o) \wedge^{\mathfrak{Q}} ?B \cdot F_o =_o \forall \mathfrak{x}_o.\ ?B \cdot \mathfrak{x}_o$
by (*rule rule-R*[**where** $p = [«,»,«,»]]$) *force+*
from *this* **and** *B-β-redex*[*OF false-wff*]
have $\vdash (T_o \wedge^{\mathfrak{Q}} T_o =_o T_o) \wedge^{\mathfrak{Q}} (T_o \wedge^{\mathfrak{Q}} F_o =_o F_o) =_o \forall \mathfrak{x}_o.\ ?B \cdot \mathfrak{x}_o$
by (*rule rule-R*[**where** $p = [«,»,»]]$) *force+*
from *this* **and** *B-β-redex*[*OF wffs-of-type-intros*(*1*)] **show** *?thesis*
by (*rule rule-R*[**where** $p = [»,»,«]]$) *force+*
qed
ultimately have $\vdash \forall \mathfrak{x}_o.\ (T_o \wedge^{\mathfrak{Q}} \mathfrak{x}_o =_o \mathfrak{x}_o)$
by (*rule rule-R*[**where** $p = []]$) *fastforce+*
show *?thesis*
using $\forall I[OF \langle \vdash \forall \mathfrak{x}_o.\ (T_o \wedge^{\mathfrak{Q}} \mathfrak{x}_o =_o \mathfrak{x}_o)\rangle$ *assms* $*]$ **by** *simp*
qed

6.18 Proposition 5217

proposition *prop-5217*:
shows $\vdash (T_o =_o F_o) =_o F_o$
proof −
let $?B = \lambda \mathfrak{x}_o.\ (T_o =_o \mathfrak{x}_o)$
have *B-is-wff*: $?B \in wffs_{o \to o}$
by *auto*
have $*$: *is-free-for* A ($\mathfrak{x}$, o) $(T_o =_o \mathfrak{x}_o)$ **for** A
by (*intro is-free-for-in-equality is-free-for-in-true is-free-for-in-var*)
have §*1*: $\vdash ?B \cdot T_o \wedge^{\mathfrak{Q}} ?B \cdot F_o =_o \forall \mathfrak{x}_o.\ ?B \cdot \mathfrak{x}_o$
proof −
have $\vdash \mathfrak{g}_{o \to o} \cdot T_o \wedge^{\mathfrak{Q}} \mathfrak{g}_{o \to o} \cdot F_o =_o \forall \mathfrak{x}_o.\ \mathfrak{g}_{o \to o} \cdot \mathfrak{x}_o$ (**is** $\vdash ?C =_o ?D$)
using *axiom-1*[*unfolded equivalence-def*] **by** (*rule axiom-is-derivable-from-no-hyps*)
moreover have $?C \in wffs_o$ **and** $?D \in wffs_o$ **and** *is-free-for* $?B$ ($\mathfrak{g}$, $o \to o$) $(?C =_o ?D)$
by *auto*
ultimately have $\vdash \mathbf{S}\ \{(\mathfrak{g},\ o \to o) \rightarrowtail ?B\}\ (?C =_o ?D)$
using *B-is-wff* **and** *prop-5209* **by** *presburger*
then show *?thesis*
by *simp*
qed
then have $\vdash (T_o =_o T_o) \wedge^{\mathfrak{Q}} (T_o =_o F_o) =_o \forall \mathfrak{x}_o.\ (T_o =_o \mathfrak{x}_o)$ (**is** $\vdash ?A$)
proof −
have *B-β-redex*: $\vdash ?B \cdot A =_o (T_o =_o A)$ **if** $A \in wffs_o$ **for** A
proof −
have $T_o =_o \mathfrak{x}_o \in wffs_o$
by *auto*
moreover have $\mathbf{S}\ \{(\mathfrak{x},\ o) \rightarrowtail A\}\ (T_o =_o \mathfrak{x}_o) = (T_o =_o A)$
by *simp*
ultimately show *?thesis*
using $*$ **and** *prop-5207*[*OF that*] **by** *metis*
qed

from §*1* **and** *B-β-redex*[*OF true-wff*] **have** $\vdash (T_o =_o T_o) \wedge^{\mathfrak{Q}} ?B \cdot F_o =_o \forall \mathfrak{x}_o.\ ?B \cdot \mathfrak{x}_o$
by (*rule rule-R*[**where** $p = [«,»,«,»]]$) *force+*
from *this* **and** *B-β-redex*[*OF false-wff*]
have $\vdash (T_o =_o T_o) \wedge^{\mathfrak{Q}} (T_o =_o F_o) =_o \forall \mathfrak{x}_o.\ ?B \cdot \mathfrak{x}_o$
by (*rule rule-R*[**where** $p = [«,»,»]]$) *force+*
from *this* **and** *B-β-redex*[*OF wffs-of-type-intros*(*1*)] **show** *?thesis*
by (*rule rule-R*[**where** $p = [»,»,«]]$) *force+*
qed
from *prop-5210*[*OF true-wff*] **have** $\vdash T_o \wedge^{\mathfrak{Q}} (T_o =_o F_o) =_o \forall \mathfrak{x}_o.\ (T_o =_o \mathfrak{x}_o)$
by (*rule rule-RR*[*OF disjI2*, **where** $p = [«,»,«,»]$ **and** $C = ?A$]) (*force+*, *fact*)
from *this* **and** *prop-5216*[**where** $A = T_o =_o F_o$]
have $\vdash (T_o =_o F_o) =_o \forall \mathfrak{x}_o.\ (T_o =_o \mathfrak{x}_o)$
by (*rule rule-R* [**where** $p = [«,»]]$) *force+*
moreover have §*5*:
$\vdash ((\lambda \mathfrak{x}_o.\ T_o) =_{o\to o} (\lambda \mathfrak{x}_o.\ \mathfrak{x}_o)) =_o \forall \mathfrak{x}_o.\ ((\lambda \mathfrak{x}_o.\ T_o) \cdot \mathfrak{x}_o =_o (\lambda \mathfrak{x}_o.\ \mathfrak{x}_o) \cdot \mathfrak{x}_o)$
proof −
have $\vdash (\mathfrak{f}_{o\to o} =_{o\to o} \mathfrak{g}_{o\to o}) =_o \forall \mathfrak{x}_o.\ (\mathfrak{f}_{o\to o} \cdot \mathfrak{x}_o =_o \mathfrak{g}_{o\to o} \cdot \mathfrak{x}_o)$ (**is** $\vdash ?C =_o ?D$)
using *axiom-3*[*unfolded equivalence-def*] **by** (*rule axiom-is-derivable-from-no-hyps*)
moreover have *is-free-for* $((\lambda \mathfrak{x}_o.\ T_o))$ $(\mathfrak{f},\ o\to o)$ $(?C =_o ?D)$
by *fastforce*
moreover have $(\lambda \mathfrak{x}_o.\ T_o) \in wffs_{o\to o}$ **and** $?C \in wffs_o$ **and** $?D \in wffs_o$
by *auto*
ultimately have $\vdash$ **S** $\{(\mathfrak{f},\ o\to o) \rightarrowtail (\lambda \mathfrak{x}_o.\ T_o)\}$ $(?C =_o ?D)$
using *prop-5209* **by** *presburger*
then have $\vdash ((\lambda \mathfrak{x}_o.\ T_o) =_{o\to o} \mathfrak{g}_{o\to o}) =_o \forall \mathfrak{x}_o.\ ((\lambda \mathfrak{x}_o.\ T_o) \cdot \mathfrak{x}_o =_o \mathfrak{g}_{o\to o} \cdot \mathfrak{x}_o)$
(**is** $\vdash ?C' =_o ?D'$)
by *simp*
moreover have *is-free-for* $((\lambda \mathfrak{x}_o.\ \mathfrak{x}_o))$ $(\mathfrak{g},\ o\to o)$ $(?C' =_o ?D')$
by *fastforce*
moreover have $(\lambda \mathfrak{x}_o.\ \mathfrak{x}_o) \in wffs_{o\to o}$ **and** $?C' \in wffs_o$ **and** $?D' \in wffs_o$
using $\langle(\lambda \mathfrak{x}_o.\ T_o) \in wffs_{o\to o}\rangle$ **by** *auto*
ultimately have $\vdash$ **S** $\{(\mathfrak{g},\ o\to o) \rightarrowtail (\lambda \mathfrak{x}_o.\ \mathfrak{x}_o)\}$ $(?C' =_o ?D')$
using *prop-5209* **by** *presburger*
then show *?thesis*
by *simp*
qed
then have $\vdash F_o =_o \forall \mathfrak{x}_o.\ (T_o =_o \mathfrak{x}_o)$
proof −
have $\vdash (\lambda \mathfrak{x}_o.\ T_o) \cdot \mathfrak{x}_o =_o T_o$
using *prop-5208*[**where** $vs = [(\mathfrak{x},\ o)]$] **and** *true-wff* **by** *simp*
with §*5* **have** $*$:
$\vdash ((\lambda \mathfrak{x}_o.\ T_o) =_{o\to o} (\lambda \mathfrak{x}_o.\ \mathfrak{x}_o)) =_o \forall \mathfrak{x}_o.\ (T_o =_o (\lambda \mathfrak{x}_o.\ \mathfrak{x}_o) \cdot \mathfrak{x}_o)$
by (*rule rule-R*[**where** $p = [»,»,«,«,»]]$) *force+*
have $\vdash (\lambda \mathfrak{x}_o.\ \mathfrak{x}_o) \cdot \mathfrak{x}_o =_o \mathfrak{x}_o$
using *prop-5208*[**where** $vs = [(\mathfrak{x},\ o)]$] **by** *fastforce*
with $*$ **have** $\vdash ((\lambda \mathfrak{x}_o.\ T_o) =_{o\to o} (\lambda \mathfrak{x}_o.\ \mathfrak{x}_o)) =_o \forall \mathfrak{x}_o.\ (T_o =_o \mathfrak{x}_o)$
by (*rule rule-R*[**where** $p = [»,»,«,»]]$) *force+*
then show *?thesis*
by *simp*

```
  qed
  ultimately show ?thesis
    using Equality-Rules(2,3) by blast
qed
```

6.19 Proposition 5218

```
proposition prop-5218:
  assumes A ∈ wffs_o
  shows ⊢ (T_o =_o A) =_o A
proof −
  let ?B = λ𝔵_o. ((T_o =_o 𝔵_o) =_o 𝔵_o)
  have B-is-wff: ?B ∈ wffs_{o→o}
    by auto
  have §1: ⊢ ?B ▪ T_o ∧^Q ?B ▪ F_o =_o ∀𝔵_o. ?B ▪ 𝔵_o
  proof −
    have ⊢ 𝔤_{o→o} ▪ T_o ∧^Q 𝔤_{o→o} ▪ F_o =_o ∀𝔵_o. 𝔤_{o→o} ▪ 𝔵_o (is ⊢ ?C =_o ?D)
      using axiom-1[unfolded equivalence-def] by (rule axiom-is-derivable-from-no-hyps)
    moreover have ?C ∈ wffs_o and ?D ∈ wffs_o and is-free-for ?B (𝔤, o→o) (?C =_o ?D)
      by auto
    ultimately have ⊢ S {(𝔤, o→o) ⇀ ?B} (?C =_o ?D)
      using prop-5209[OF B-is-wff] by presburger
    then show ?thesis
      by simp
  qed
  have ∗: is-free-for A (𝔵, o) ((T_o =_o 𝔵_o) =_o 𝔵_o) for A
    by (intro is-free-for-in-equality is-free-for-in-true is-free-for-in-var)
  have §2:
    ⊢
      ((T_o =_o T_o) =_o T_o) ∧^Q ((T_o =_o F_o) =_o F_o)
      =_o
      ∀𝔵_o. ((T_o =_o 𝔵_o) =_o 𝔵_o)
  proof −
    have B-β-redex: ⊢ ?B ▪ A =_o ((T_o =_o A) =_o A) if A ∈ wffs_o for A
    proof −
      have (T_o =_o 𝔵_o) =_o 𝔵_o ∈ wffs_o
        by auto
      moreover have S {(𝔵, o) ⇀ A} ((T_o =_o 𝔵_o) =_o 𝔵_o) = ((T_o =_o A) =_o A)
        by simp
      ultimately show ?thesis
        using ∗ and prop-5207[OF that] by metis
    qed
    from §1 and B-β-redex[OF true-wff]
    have ⊢ ((T_o =_o T_o) =_o T_o) ∧^Q ?B ▪ F_o =_o ∀𝔵_o. ?B ▪ 𝔵_o
      by (rule rule-R[where p = [«,»,«,»]]) force+
    from this and B-β-redex[OF false-wff]
    have ⊢ ((T_o =_o T_o) =_o T_o) ∧^Q ((T_o =_o F_o) =_o F_o) =_o ∀𝔵_o. ?B ▪ 𝔵_o
      by (rule rule-R[where p = [«,»,»]]) force+
    from this and B-β-redex[OF wffs-of-type-intros(1)] show ?thesis
```

by (*rule rule-R*[**where** p = [»,»,«]]) *force*+
qed
have §*3*: $\vdash$ $(T_o =_o T_o) =_o T_o$
by (*fact Equality-Rules*(*2*)[*OF prop-5210* [*OF true-wff*]])
have $\vdash$ $((T_o =_o T_o) =_o T_o) \wedge^{\mathcal{Q}} ((T_o =_o F_o) =_o F_o)$
by (*fact prop-5213*[*OF* §*3 prop-5217*])
from *this* **and** §*2* **have** §*4*: $\vdash$ $\forall \mathfrak{x}_o.\ ((T_o =_o \mathfrak{x}_o) =_o \mathfrak{x}_o)$
by (*rule rule-R*[**where** p = []]) *fastforce*+
then show *?thesis*
using $\forall I$[*OF* §*4 assms* $*$] **by** *simp*
qed

6.20 Proposition 5219 (Rule T)

proposition *prop-5219-1*:
assumes $A \in wffs_o$
shows $\mathcal{H} \vdash A \longleftrightarrow \mathcal{H} \vdash T_o =_o A$
proof *safe*
assume $\mathcal{H} \vdash A$
then have *is-hyps* $\mathcal{H}$
by (*blast dest*: *is-derivable-from-hyps.cases*)
then have $\mathcal{H} \vdash (T_o =_o A) =_o A$
by (*fact derivability-implies-hyp-derivability*[*OF prop-5218*[*OF assms*]])
with ‹$\mathcal{H} \vdash A$› **show** $\mathcal{H} \vdash T_o =_o A$
using *Equality-Rules*(*1*)[*unfolded equivalence-def*] **and** *Equality-Rules*(*2*) **by** *blast*
next
assume $\mathcal{H} \vdash T_o =_o A$
then have *is-hyps* $\mathcal{H}$
by (*blast dest*: *is-derivable-from-hyps.cases*)
then have $\mathcal{H} \vdash (T_o =_o A) =_o A$
by (*fact derivability-implies-hyp-derivability*[*OF prop-5218*[*OF assms*]])
with ‹$\mathcal{H} \vdash T_o =_o A$› **show** $\mathcal{H} \vdash A$
by (*rule Equality-Rules*(*1*)[*unfolded equivalence-def*])
qed

proposition *prop-5219-2*:
assumes $A \in wffs_o$
shows $\mathcal{H} \vdash A \longleftrightarrow \mathcal{H} \vdash A =_o T_o$
using *prop-5219-1*[*OF assms*] **and** *Equality-Rules*(*2*) **by** *blast*

lemmas *rule-T* = *prop-5219-1 prop-5219-2*

6.21 Proposition 5220 (Universal Generalization)

context
begin

private lemma *const-true-α-conversion*:
shows $\vdash$ $(\lambda x_\alpha.\ T_o) =_{\alpha\to o} (\lambda z_\alpha.\ T_o)$
proof −

have $(z,\ \alpha) \notin$ *free-vars* T_o **and** *is-free-for* $(z_\alpha)\ (x,\ \alpha)\ T_o$
by *auto*
then have $\vdash (\lambda x_\alpha.\ T_o) =_{\alpha\to o} \lambda z_\alpha.$ **S** $\{(x,\ \alpha) \rightarrowtail z_\alpha\}\ T_o$
by (*rule prop-5206*[*OF true-wff*])
then show *?thesis*
by *simp*
qed

proposition *prop-5220*:
assumes $\mathcal{H} \vdash A$
and $(x,\ \alpha) \notin$ *free-vars* $\mathcal{H}$
shows $\mathcal{H} \vdash \forall\, x_\alpha.\ A$
proof −
from ‹$\mathcal{H} \vdash A$› **have** *is-hyps* $\mathcal{H}$
by (*blast dest*: *is-derivable-from-hyps.cases*)
have $\mathcal{H} \vdash A$
by *fact*
then have §*2*: $\mathcal{H} \vdash T_o =_o A$
using *rule-T*(*1*)[*OF hyp-derivable-form-is-wffso*[*OF* ‹$\mathcal{H} \vdash A$›]] **by** *simp*
have §*3*: $\mathcal{H} \vdash (\lambda \mathfrak{x}_\alpha.\ T_o) =_{\alpha\to o} (\lambda x_\alpha.\ T_o)$
by (*fact derivability-implies-hyp-derivability*[*OF const-true-α-conversion* ‹*is-hyps* $\mathcal{H}$›])
from §*3* **and** §*2* **have** $\mathcal{H} \vdash \lambda \mathfrak{x}_\alpha.\ T_o =_{\alpha\to o} \lambda x_\alpha.\ A$
proof (*induction rule*: *rule-R'*[**where** $p = [\text{»},\ \text{«}]$])
case *no-capture*
have $*$: [»,«] $\in$ *positions* $(\lambda \mathfrak{x}_\alpha.\ T_o =_{\alpha\to o} \lambda x_\alpha.\ T_o)$
by *simp*
show *?case*
unfolding *rule-R'-side-condition-def* **and** *capture-exposed-vars-at-alt-def*[*OF* $*$] **using** *assms*(*2*)
by *simp*
qed *force*+
then show *?thesis*
unfolding *forall-def*[*unfolded PI-def*, *folded equality-of-type-def*] **.**
qed

end

lemmas *Gen* = *prop-5220*

proposition *generalized-Gen*:
assumes $\mathcal{H} \vdash A$
and *lset vs* $\cap$ *free-vars* $\mathcal{H} = \{\}$
shows $\mathcal{H} \vdash \forall^{\mathfrak{Q}}_{\star}$ *vs A*
using *assms*(*2*) **proof** (*induction vs*)
case *Nil*
then show *?case*
using *assms*(*1*) **by** *simp*
next
case (*Cons v vs*)
obtain x **and** α **where** $v = (x,\ \alpha)$

by *fastforce*
with *Cons.prems* **have** *lset vs* ∩ *free-vars* $\mathcal{H} = \{\}$ **and** $(x, \alpha) \notin$ *free-vars* $\mathcal{H}$
by *simp-all*
from ‹*lset vs* ∩ *free-vars* $\mathcal{H} = \{\}$› **have** $\mathcal{H} \vdash \forall^{\mathfrak{Q}}{}_{\star}$ *vs A*
by (*fact Cons.IH*)
with ‹$(x, \alpha) \notin$ *free-vars* $\mathcal{H}$› **and** ‹$v = (x, \alpha)$› **show** *?case*
using *Gen* **by** *simp*
qed

6.22 Proposition 5221 (Substitution)

context
begin

private lemma *prop-5221-aux*:
assumes $\mathcal{H} \vdash B$
and $(x, \alpha) \notin$ *free-vars* $\mathcal{H}$
and *is-free-for A* (x, α) *B*
and $A \in \mathit{wffs}_{\alpha}$
shows $\mathcal{H} \vdash \mathbf{S}\ \{(x, \alpha) \rightarrowtail A\}\ B$
proof −
have $\mathcal{H} \vdash B$
by *fact*
from *this* **and** *assms*(*2*) **have** $\mathcal{H} \vdash \forall\, x_{\alpha}.\ B$
by (*rule Gen*)
from *this* **and** *assms*(*4*,*3*) **show** *?thesis*
by (*rule* $\forall I$)
qed

proposition *prop-5221*:
assumes $\mathcal{H} \vdash B$
and *is-substitution* ϑ
and $\forall\, v \in \mathit{fmdom}'\ \vartheta.$ *var-name v* $\notin$ *free-var-names* $\mathcal{H} \wedge$ *is-free-for* (ϑ \$\$! *v*) *v B*
and $\vartheta \neq \{\$\$\}$
shows $\mathcal{H} \vdash \mathbf{S}\ \vartheta\ B$
proof −
obtain *xs* **and** *As*
where *lset xs* = *fmdom′* ϑ — i.e., $x^1_{\alpha_1}, \ldots, x^n_{\alpha_n}$
and *As* = *map* (($$!) ϑ) *xs* — i.e., $A^1_{\alpha_1}, \ldots, A^n_{\alpha_n}$
and *length xs* = *card* (*fmdom′* ϑ)
by (*metis distinct-card finite-distinct-list finite-fmdom′*)
then have *distinct xs*
by (*simp add*: *card-distinct*)
from ‹*lset xs* = *fmdom′* ϑ› **and** ‹*As* = *map* (($$!) ϑ) *xs*› **have** *lset As* = *fmran′* ϑ
by (*intro subset-antisym subsetI*) (*force simp add*: *fmlookup-dom′-iff fmlookup-ran′-iff*)+
from *assms*(*1*) **have** *finite* (*var-name* ‘ (*vars B* ∪ *vars* (*lset As*) ∪ *vars* $\mathcal{H}$))
by (*cases rule*: *is-derivable-from-hyps.cases*) (*simp-all add*: *finite-Domain vars-form-finiteness*)
then obtain *ys* — i.e., $y^1_{\alpha_1}, \ldots, y^n_{\alpha_n}$
where *length ys* = *length xs*

and *distinct ys*
and *ys-fresh*:
(*var-name ‘ lset ys*) ∩ (*var-name ‘* (*vars B* ∪ *vars* (*lset As*) ∪ *vars* $\mathcal{H}$ ∪ *lset xs*)) = {}
and *map var-type ys* = *map var-type xs*
using *fresh-var-list-existence* **by** (*metis image-Un*)
have *length xs* = *length As*
by (*simp add*: ‹*As* = *map* (($$!) ϑ) *xs*›)
— $\mathcal{H} \vdash \mathsf{S}\begin{smallmatrix} x^1_{\alpha_1} \dots x^k_{\alpha_k} x^{k+1}_{\alpha_{k+1}} \dots x^n_{\alpha_n} \\ A^1_{\alpha_1} \dots A^k_{\alpha_k} y^{k+1}_{\alpha_{k+1}} \dots y^n_{\alpha_n} \end{smallmatrix} B$
have $\mathcal{H} \vdash \mathbf{S}$ (*fmap-of-list* (*zip xs* (*take k As* @ *drop k* (*map FVar ys*)))) *B* **if** $k \leq$ *length xs* **for** *k*
using *that* **proof** (*induction k*)
case *0*
have $\mathcal{H} \vdash \mathbf{S}$ (*fmap-of-list* (*zip xs* (*map FVar ys*))) *B*
using ‹*length ys* = *length xs*›
and ‹*length xs* = *length As*›
and ‹(*var-name ‘ lset ys*) ∩ (*var-name ‘* (*vars B* ∪ *vars* (*lset As*) ∪ *vars* $\mathcal{H}$ ∪ *lset xs*)) = {}›
and ‹*lset xs* = *fmdom′* ϑ›
and ‹*distinct ys*›
and *assms*(*3*)
and ‹*map var-type ys* = *map var-type xs*›
and ‹*distinct xs*›
and ‹*length xs* = *card* (*fmdom′* ϑ)›
proof (*induction ys xs As arbitrary*: ϑ *rule*: *list-induct3*)
case *Nil*
with *assms*(*1*) **show** *?case*
using *empty-substitution-neutrality* **by** *auto*
next
— In the following:

- $\vartheta = \{x^1_{\alpha_1} \rightarrowtail y^1_{\alpha_1}, \dots, x^n_{\alpha_n} \rightarrowtail y^n_{\alpha_n}\}$
- $?\vartheta = \{x^2_{\alpha_2} \rightarrowtail y^2_{\alpha_2}, \dots, x^n_{\alpha_n} \rightarrowtail y^n_{\alpha_n}\}$
- $v_x = x^1_{\alpha_1}$, and $v_y = y^1_{\alpha_1}$

case (*Cons* v_y *ys* v_x *xs A′ As′*)
let *?*ϑ = *fmap-of-list* (*zip xs* (*map FVar ys*))
from *Cons.hyps*(*1*) **have** *lset xs* = *fmdom′ ?*ϑ
by *simp*
from *Cons.hyps*(*1*) **and** *Cons.prems*(*6*) **have** *fmran′ ?*ϑ = *FVar ‘ lset ys*
by *force*
have *is-substitution ?*ϑ
unfolding *is-substitution-def* **proof**
fix *v*
assume *v* ∈ *fmdom′ ?*ϑ
with ‹*lset xs* = *fmdom′ ?*ϑ› **obtain** *k* **where** *v* = *xs* ! *k* **and** *k* < *length xs*
by (*metis in-set-conv-nth*)
moreover obtain α **where** *var-type v* = α
by *blast*
moreover from ‹*k* < *length xs*› **and** ‹*v* = *xs* ! *k*› **have** *?*ϑ $$! *v* = (*map FVar ys*) ! *k*
using *Cons.hyps*(*1*) **and** *Cons.prems*(*6*) **by** *auto*

moreover from *this* **and** ‹$k <$ *length xs*› **obtain** y **and** β **where** *?ϑ* $$! $v = y_\beta$
using *Cons.hyps*(*1*) **by** *force*
ultimately have $\alpha = \beta$
using *Cons.hyps*(*1*) **and** *Cons.prems*(*5*)
by (*metis form.inject*(*1*) *list.inject list.simps*(*9*) *nth-map snd-conv*)
then show *case v of* $(x, \alpha) \Rightarrow$ *?ϑ* $$! $(x, \alpha) \in$ *wffs*$_\alpha$
using ‹*?ϑ* $$! $v = y_\beta$› **and** ‹*var-type* $v = \alpha$› **by** *fastforce*
qed
have $v_x \notin$ *fmdom′ ?ϑ*
using *Cons.prems*(*6*) **and** ‹*lset xs* = *fmdom′ ?ϑ*› **by** *auto*
obtain x **and** α **where** $v_x = (x, \alpha)$
by *fastforce*
have *FVar* $v_y \in$ *wffs*$_\alpha$
using *Cons.prems*(*5*) **and** *surj-pair*[*of* v_y] **unfolding** ‹$v_x = (x, \alpha)$› **by** *fastforce*
have *distinct xs*
using *Cons.prems*(*6*) **by** *fastforce*
moreover have *ys-fresh′*:
(*var-name* ‘ *lset ys*) ∩ (*var-name* ‘ (*vars B* ∪ *vars* (*lset As′*) ∪ *vars* $\mathcal{H}$ ∪ *lset xs*)) = {}
proof −
have *vars* (*lset* (*A′* # *As′*)) = *vars* {*A′*} ∪ *vars* (*lset As′*)
by *simp*
moreover have *var-name* ‘ (*lset* (v_x # *xs*)) = {*var-name* v_x} ∪ *var-name* ‘ (*lset xs*)
by *simp*
moreover from *Cons.prems*(*1*) **have**
var-name ‘ *lset ys*
∩
(
var-name ‘ (*vars B*) ∪ *var-name* ‘ (*vars* (*lset* (*A′* # *As′*))) ∪ *var-name* ‘ (*vars* $\mathcal{H}$)
∪ *var-name* ‘ (*lset* (v_x # *xs*))
)
= {}
by (*simp add*: *image-Un*)
ultimately have
var-name ‘ *lset ys*
∩
(
var-name ‘ (*vars B*) ∪ *var-name* ‘ (*vars* (*lset As′*)) ∪ *var-name* ‘ (*vars* $\mathcal{H}$)
∪ *var-name* ‘ (*lset* (v_x # *xs*))
)
= {}
by *fast*
then show *?thesis*
by (*simp add*: *image-Un*)
qed
moreover have *distinct ys*
using *Cons.prems*(*3*) **by** *auto*
moreover have ∀ *v* ∈ *fmdom′ ?ϑ*. *var-name v* ∉ *free-var-names* $\mathcal{H}$ ∧ *is-free-for* (*?ϑ* $$! *v*) *v B*
proof
fix *v*

assume $v \in \mathit{fmdom'}\ ?\vartheta$
with *Cons.hyps*(*1*) **obtain** y **where** $?\vartheta\ \$\$!\ v = \mathit{FVar}\ y$ **and** $y \in \mathit{lset}\ ys$
by (*metis* (*mono-tags*, *lifting*) *fmap-of-zipped-list-range image-iff length-map list.set-map*)
moreover from *Cons.prems*(*2*,*4*) **have** *var-name* $v \notin \mathit{free\text{-}var\text{-}names}\ \mathcal{H}$
using ‹$\mathit{lset}\ xs = \mathit{fmdom'}\ ?\vartheta$› **and** ‹$v \in \mathit{fmdom'}\ ?\vartheta$› **by** *auto*
moreover from ‹$y \in \mathit{lset}\ ys$› **have** $y \notin \mathit{vars}\ B$
using *ys-fresh'* **by** *blast*
then have *is-free-for* (*FVar* y) $v\ B$
by (*intro absent-var-is-free-for*)
ultimately show *var-name* $v \notin \mathit{free\text{-}var\text{-}names}\ \mathcal{H} \wedge \mathit{is\text{-}free\text{-}for}\ (?\vartheta\ \$\$!\ v)\ v\ B$
by *simp*
qed
moreover have $\mathit{map}\ \mathit{var\text{-}type}\ ys = \mathit{map}\ \mathit{var\text{-}type}\ xs$
using *Cons.prems*(*5*) **by** *simp*
moreover have $\mathit{length}\ xs = \mathit{card}\ (\mathit{fmdom'}\ ?\vartheta)$
by (*fact distinct-card*[*OF* ‹*distinct xs*›, *unfolded* ‹$\mathit{lset}\ xs = \mathit{fmdom'}\ ?\vartheta$›, *symmetric*])
— $\mathcal{H} \vdash \mathsf{S}\ \substack{x^2_{\alpha_2} \cdots x^n_{\alpha_n} \\ y^2_{\alpha_2} \cdots y^n_{\alpha_n}}\ B$
ultimately have $\mathcal{H} \vdash \mathbf{S}\ ?\vartheta\ B$
using *Cons.IH* **and** ‹$\mathit{lset}\ xs = \mathit{fmdom'}\ ?\vartheta$› **by** *blast*
moreover from *Cons.prems*(*2*,*4*) **have** $(x, \alpha) \notin \mathit{free\text{-}vars}\ \mathcal{H}$
using ‹$v_x = (x, \alpha)$› **by** *auto*
moreover have *is-free-for* (*FVar* v_y) (x, α) (**S** $?\vartheta\ B$)
proof −
have $v_y \notin \mathit{fmdom'}\ ?\vartheta$
using *Cons.prems*(*1*) **and** ‹$\mathit{lset}\ xs = \mathit{fmdom'}\ ?\vartheta$› **by** *force*
moreover have $\mathit{fmran'}\ ?\vartheta = \mathit{lset}\ (\mathit{map}\ \mathit{FVar}\ ys)$
using *Cons.hyps*(*1*) **and** ‹*distinct xs*› **by** *simp*
then have $v_y \notin \mathit{vars}\ (\mathit{fmran'}\ ?\vartheta)$
using *Cons.prems*(*3*) **by** *force*
moreover have $v_y \notin \mathit{vars}\ B$
using *Cons.prems*(*1*) **by** *fastforce*
ultimately have $v_y \notin \mathit{vars}\ (\mathbf{S}\ ?\vartheta\ B)$
by (*rule excluded-var-from-substitution*[*OF* ‹*is-substitution* $?\vartheta$›])
then show *?thesis*
by (*fact absent-var-is-free-for*)
qed
— $\mathcal{H} \vdash \mathsf{S}\ \substack{x^1_{\alpha_1} \\ y^1_{\alpha_1}}\ \mathsf{S}\ \substack{x^2_{\alpha_2} \cdots x^n_{\alpha_n} \\ y^2_{\alpha_2} \cdots y^n_{\alpha_n}}\ B$
ultimately have $\mathcal{H} \vdash \mathbf{S}\ \{(x, \alpha) \rightarrowtail \mathit{FVar}\ v_y\}\ (\mathbf{S}\ ?\vartheta\ B)$
using ‹$\mathit{FVar}\ v_y \in \mathit{wffs}_\alpha$› **by** (*rule prop-5221-aux*)
— $\mathsf{S}\ \substack{x^1_{\alpha_1} \\ y^1_{\alpha_1}}\ \mathsf{S}\ \substack{x^2_{\alpha_2} \cdots x^n_{\alpha_n} \\ y^2_{\alpha_2} \cdots y^n_{\alpha_n}}\ B = \mathsf{S}\ \substack{x^1_{\alpha_1} \cdots x^n_{\alpha_n} \\ y^1_{\alpha_1} \cdots y^n_{\alpha_n}}\ B$
moreover have $\mathbf{S}\ \{v_x \rightarrowtail \mathit{FVar}\ v_y\}\ \mathbf{S}\ ?\vartheta\ B = \mathbf{S}\ (\{v_x \rightarrowtail \mathit{FVar}\ v_y\} +\!\!+_f\ ?\vartheta)\ B$
proof −
have $v_x \notin \mathit{lset}\ ys$
using *Cons.prems*(*1*) **by** *fastforce*
then have $\mathbf{S}\ \{v_x \rightarrowtail \mathit{FVar}\ v_y\}\ (\mathit{FVar}\ y) = \mathit{FVar}\ y$ **if** $y \in \mathit{lset}\ ys$ **for** y
using *that* **and** *free-var-singleton-substitution-neutrality* **and** *surj-pair*[*of* y] **by** *fastforce*
with ‹$\mathit{fmran'}\ ?\vartheta = \mathit{FVar}\ `\ \mathit{lset}\ ys$› **have** $\mathit{fmmap}\ (\lambda A'.\ \mathbf{S}\ \{v_x \rightarrowtail \mathit{FVar}\ v_y\}\ A')\ ?\vartheta = ?\vartheta$

```
      by (fastforce intro: fmap.map-ident-strong)
    with ‹v_x ∉ fmdom' ?ϑ› show ?thesis
      using ‹∀ v ∈ fmdom' ?ϑ. var-name v ∉ free-var-names ℋ ∧ is-free-for (?ϑ $$! v) v B›
      and substitution-consolidation by auto
  qed
  — ℋ ⊢ Ṣ x¹_α₁ ... xⁿ_αₙ / y¹_α₁ ... yⁿ_αₙ B
  ultimately show ?case
    using ‹v_x = (x, α)› and ‹v_x ∉ fmdom' ?ϑ› and fmap-singleton-comm by fastforce
 qed
 with 0 and that show ?case
  by auto
next
 case (Suc k)
 let ?ps = λk. zip xs (take k As @ drop k (map FVar ys))
 let ?y = ys ! k and ?A = As ! k
 let ?ϑ = λk. fmap-of-list (?ps k)
 let ?ϑ' = λk. fmap-of-list (map (λ(v', A'). (v', S {?y ↣ ?A} A')) (?ps k))
 have fmdom' (?ϑ k') = lset xs for k'
  by (simp add: ‹length xs = length As› ‹length ys = length xs›)
 have fmdom' (?ϑ' k') = lset xs for k'
  using ‹length xs = length As› and ‹length ys = length xs› and fmdom'-fmap-of-list by simp
 have ?y ∈ lset ys
  using Suc.prems ‹length ys = length xs› by simp
 have ∀ j < length ys. ys ! j ∉ vars (ℋ::form set) ∧ ys ! j ∉ vars B
  using ‹(var-name ‘ lset ys) ∩ (var-name ‘ (vars B ∪ vars (lset As) ∪ vars ℋ ∪ lset xs)) = {}›
  by force
 obtain n_y and α_y where (n_y, α_y) = ?y
  using surj-pair[of ?y] by fastforce
 moreover have ?A ∈ wffs_α_y
 proof −
  from Suc.prems and ‹(n_y, α_y) = ?y› have var-type (xs ! k) = α_y
   using ‹length ys = length xs› and ‹map var-type ys = map var-type xs› and Suc-le-lessD
   by (metis nth-map snd-conv)
  with Suc.prems and assms(2) and ‹lset xs = fmdom' ϑ› and ‹As = map (($$!) ϑ) xs› show
?thesis
   using less-eq-Suc-le and nth-mem by fastforce
 qed
 ultimately have is-substitution {?y ↣ ?A}
  by auto
 have wfs: is-substitution (?ϑ k) for k
 unfolding is-substitution-def proof
  fix v
  assume v ∈ fmdom' (?ϑ k)
  with ‹fmdom' (?ϑ k) = lset xs› obtain j where v = xs ! j and j < length xs
   by (fastforce simp add: in-set-conv-nth)
  obtain α where var-type v = α
   by blast
  show case v of (x, α) ⇒ (?ϑ k) $$! (x, α) ∈ wffs_α
  proof (cases j < k)
```

```
    case True
    with ‹j < length xs› and ‹v = xs ! j› have (?ϑ k) $$! v = As ! j
      using ‹distinct xs› and ‹length xs = length As› and ‹length ys = length xs› by force
    with assms(2) ‹v = xs ! j› and ‹v ∈ fmdom' (?ϑ k)› and ‹var-type v = α› and ‹j < length xs›
    have (?ϑ k) $$! v ∈ wffs_α
      using ‹As = map (($$!) ϑ) xs› and ‹fmdom' (?ϑ k) = lset xs› and ‹lset xs = fmdom' ϑ›
      by auto
    then show ?thesis
      using ‹var-type v = α› by force
  next
    case False
    with ‹j < length xs› and ‹v = xs ! j› have (?ϑ k) $$! v = FVar (ys ! j)
      using ‹distinct xs› and ‹length xs = length As› and ‹length ys = length xs› by force
    with ‹j < length xs› and ‹v = xs ! j› and ‹var-type v = α› and ‹length ys = length xs›
    have (?ϑ k) $$! v ∈ wffs_α
      using ‹map var-type ys = map var-type xs› and surj-pair[of ys ! j]
      by (metis nth-map snd-conv wffs-of-type-intros(1))
    then show ?thesis
      using ‹var-type v = α› by force
  qed
qed
have ϑ'-alt-def: ?ϑ' k = fmap-of-list
  (zip xs
    (take k (map (λA'. S {?y ↣ ?A} A') As)
    @
    (drop k (map (λA'. S {?y ↣ ?A} A') (map FVar ys)))))
proof −
  have
    fmap-of-list (zip xs (map (λA'. S {?y ↣ ?A} A') (take k As @ drop k (map FVar ys))))
    =
    fmap-of-list
      (zip xs
        (map (λA'. S {?y ↣ ?A} A') (take k As)
        @
        (drop k (map (λA'. S {?y ↣ ?A} A') (map FVar ys)))))
    by (simp add: drop-map)
  then show ?thesis
    by (metis take-map zip-map2)
qed
— 𝓗 ⊢ Ş [x^1_α1 ... x^k_αk x^{k+1}_α{k+1} ... x^n_αn / A^1_α1 ... A^k_αk y^{k+1}_α{k+1} ... y^n_αn] B
have 𝓗 ⊢ S (?ϑ k) B
  by (fact Suc.IH[OF Suc-leD[OF Suc.prems]])
— 𝓗 ⊢ Ş [y^{k+1}_α{k+1} / A^{k+1}_α{k+1}] Ş [x^1_α1 ... x^k_αk x^{k+1}_α{k+1} ... x^n_αn / A^1_α1 ... A^k_αk y^{k+1}_α{k+1} ... y^n_αn] B
then have 𝓗 ⊢ S {?y ↣ ?A} S (?ϑ k) B
proof −
  from ‹(n_y, α_y) = ?y› and ‹length ys = length xs› have (n_y, α_y) ∉ free-vars 𝓗
    using ‹∀j < length ys. ys ! j ∉ vars (𝓗::form set) ∧ ys ! j ∉ vars B›
```

```
  and free-vars-in-all-vars-set and Suc-le-lessD[OF Suc.prems] by fastforce
moreover have is-free-for ?A (n_y, α_y) (S (?ϑ k) B)
proof −
  have is-substitution (fmdrop (xs ! k) (?ϑ k))
    using wfs and ‹fmdom′ (?ϑ k) = lset xs› by force
  moreover from Suc-le-lessD[OF Suc.prems] have var-type (xs ! k) = var-type (ys ! k)
    using ‹length ys = length xs› and ‹map var-type ys = map var-type xs› by (metis nth-map)
  then have is-substitution {xs ! k ⟼ FVar ?y}
    unfolding is-substitution-def using ‹(n_y, α_y) = ?y›
    by (intro ballI) (clarsimp, metis snd-eqD wffs-of-type-intros(1))
  moreover have (xs ! k) ∉ fmdom′ (fmdrop (xs ! k) (?ϑ k))
    by simp
  moreover have
    ∀ v ∈ fmdom′ (fmdrop (xs ! k) (?ϑ k)). ?y ∉ vars (fmdrop (xs ! k) (?ϑ k) $$! v)
  proof
    fix v
    assume v ∈ fmdom′ (fmdrop (xs ! k) (?ϑ k))
    then have v ∈ fmdom′ (?ϑ k)
      by simp
    with ‹fmdom′ (?ϑ k) = lset xs› obtain j where v = xs ! j and j < length xs and j ≠ k
      using ‹v ∈ fmdom′ (fmdrop (xs ! k) (?ϑ k))›
      and ‹(xs ! k) ∉ fmdom′ (fmdrop (xs ! k) (?ϑ k))› by (metis in-set-conv-nth)
    then show ?y ∉ vars ((fmdrop (xs ! k) (?ϑ k)) $$! v)
    proof (cases j < k)
      case True
      with ‹j < length xs› and ‹v = xs ! j› have (?ϑ k) $$! v = As ! j
        using ‹distinct xs› and ‹length xs = length As› and ‹length ys = length xs› by force
      moreover from ‹j < length xs› and ‹length xs = length As› have ?y ∉ vars (As ! j)
        using ‹?y ∈ lset ys› and ys-fresh by fastforce
      ultimately show ?thesis
        using ‹v ∈ fmdom′ (fmdrop (xs ! k) (?ϑ k))› by auto
    next
      case False
      with ‹j < length xs› and ‹v = xs ! j› have (?ϑ k) $$! v = FVar (ys ! j)
        using ‹distinct xs› and ‹length xs = length As› and ‹length ys = length xs› by force
      moreover from Suc-le-lessD[OF Suc.prems] and ‹j ≠ k› have ?y ≠ ys ! j
        by (simp add: ‹distinct ys› ‹j < length xs› ‹length ys = length xs› nth-eq-iff-index-eq)
      ultimately show ?thesis
        using ‹v ∈ fmdom′ (fmdrop (xs ! k) (?ϑ k))›
        and ‹xs ! k ∉ fmdom′ (fmdrop (xs ! k) (?ϑ k))› and surj-pair[of ys ! j] by fastforce
    qed
  qed
  moreover from ‹k < length xs› and ‹length ys = length xs› have ?y ∉ vars B
    by (simp add: ‹∀ j < length ys. ys ! j ∉ vars H ∧ ys ! j ∉ vars B›)
  moreover have is-free-for ?A (xs ! k) B
  proof −
    from Suc-le-lessD[OF Suc.prems] and ‹lset xs = fmdom′ ϑ› have xs ! k ∈ fmdom′ ϑ
      using nth-mem by blast
    moreover from Suc.prems and ‹As = map (($$!) ϑ) xs› have ϑ $$! (xs ! k) = ?A
```

```
    by fastforce
  ultimately show ?thesis
    using assms(3) by simp
qed
moreover
have ∀ v ∈ fmdom' (fmdrop (xs ! k) (?ϑ k)). is-free-for (fmdrop (xs ! k) (?ϑ k) $$! v) v B
proof
  fix v
  assume v ∈ fmdom' (fmdrop (xs ! k) (?ϑ k))
  then have v ∈ fmdom' (?ϑ k)
    by simp
  with ‹fmdom' (?ϑ k) = lset xs› obtain j where v = xs ! j and j < length xs and j ≠ k
    using ‹v ∈ fmdom' (fmdrop (xs ! k) (?ϑ k))›
    and ‹(xs ! k) ∉ fmdom' (fmdrop (xs ! k) (?ϑ k))› by (metis in-set-conv-nth)
  then show is-free-for (fmdrop (xs ! k) (?ϑ k) $$! v) v B
  proof (cases j < k)
    case True
    with ‹j < length xs› and ‹v = xs ! j› have (?ϑ k) $$! v = As ! j
      using ‹distinct xs› and ‹length xs = length As› and ‹length ys = length xs› by force
    moreover have is-free-for (As ! j) v B
    proof −
      from ‹j < length xs› and ‹lset xs = fmdom' ϑ› and ‹v = xs ! j› have v ∈ fmdom' ϑ
        using nth-mem by blast
      moreover have ϑ $$! v = As ! j
        by (simp add: ‹As = map (($$!) ϑ) xs› ‹j < length xs› ‹v = xs ! j›)
      ultimately show ?thesis
        using assms(3) by simp
      qed
    ultimately show ?thesis
      using ‹v ∈ fmdom' (fmdrop (xs ! k) (?ϑ k))› by auto
  next
    case False
    with ‹j < length xs› and ‹v = xs ! j› have (?ϑ k) $$! v = FVar (ys ! j)
      using ‹distinct xs› and ‹length xs = length As› and ‹length ys = length xs› by force
    moreover from ‹j < length xs› and ‹length ys = length xs› have ys ! j ∉ vars B
      using ‹∀ j < length ys. ys ! j ∉ vars H ∧ ys ! j ∉ vars B› by simp
    then have is-free-for (FVar (ys ! j)) v B
      by (fact absent-var-is-free-for)
    ultimately show ?thesis
      using ‹v ∈ fmdom' (fmdrop (xs ! k) (?ϑ k))› by auto
  qed
qed
ultimately have is-free-for ?A (ys ! k) S ({xs ! k ↣ FVar ?y} ++f fmdrop (xs ! k) (?ϑ k)) B
  using is-free-for-with-renaming-substitution by presburger
moreover have S ({xs ! k ↣ FVar ?y} ++f fmdrop (xs ! k) (?ϑ k)) B = S (?ϑ k) B
  using ‹length xs = length As› and ‹length ys = length xs› and Suc-le-eq and Suc.prems
  and ‹distinct xs› by simp
ultimately show ?thesis
  unfolding ‹(n_y, α_y) = ?y› by simp
```

```
  qed
  ultimately show ?thesis
    using prop-5221-aux[OF ‹ℋ ⊢ S (?ϑ k) B›] and ‹?A ∈ wffs_{α_y}› and ‹(n_y, α_y) = ?y› by metis
qed
— Ş ^{y^{k+1}_{α_{k+1}}}_{A^{k+1}_{α_{k+1}}} Ş ^{x^1_{α_1} ... x^k_{α_k} x^{k+1}_{α_{k+1}} ... x^n_{α_n}}_{A^1_{α_1} ... A^k_{α_k} y^{k+1}_{α_{k+1}} ... y^n_{α_n}} B = Ş ^{x^1_{α_1} ... x^k_{α_k} x^{k+1}_{α_{k+1}} x^{k+2}_{α_{k+2}} ... x^n_{α_n}}_{A^1_{α_1} ... A^k_{α_k} A^{k+1}_{α_{k+1}} y^{k+2}_{α_{k+2}} ... y^n_{α_n}} B
moreover have S {?y ⇀ ?A} S (?ϑ k) B = S (?ϑ (Suc k)) B
proof −
  have S {?y ⇀ ?A} S (?ϑ k) B = S {?y ⇀ ?A} ++_f (?ϑ' k) B
  proof −
    have ?y ∉ fmdom' (?ϑ k)
      using ‹?y ∈ lset ys› and ‹fmdom' (?ϑ k) = lset xs› and ys-fresh by blast
    moreover have (?ϑ' k) = fmmap (λA'. S {?y ⇀ ?A} A') (?ϑ k)
      using ‹length xs = length As› and ‹length ys = length xs› by simp
    moreover have ∀ v' ∈ fmdom' (?ϑ k). is-free-for (?ϑ k $$! v') v' B
    proof
      fix v'
      assume v' ∈ fmdom' (?ϑ k)
      with ‹fmdom' (?ϑ k) = lset xs› obtain j where v' = xs ! j and j < length xs
        by (metis in-set-conv-nth)
      obtain α where var-type v' = α
        by blast
      show is-free-for (?ϑ k $$! v') v' B
      proof (cases j < k)
        case True
        with ‹j < length xs› and ‹v' = xs ! j› have (?ϑ k) $$! v' = As ! j
          using ‹distinct xs› and ‹length xs = length As› and ‹length ys = length xs› by force
        moreover from ‹lset xs = fmdom' ϑ› and assms(3) have is-free-for (As ! j) (xs ! j) B
          by (metis ‹As = map (($$!) ϑ) xs› ‹j < length xs› nth-map nth-mem)
        ultimately show ?thesis
          using ‹v' = xs ! j› by (simp only:)
      next
        case False
        with ‹j < length xs› and ‹v' = xs ! j› have (?ϑ k) $$! v' = FVar (ys ! j)
          using ‹distinct xs› and ‹length xs = length As› and ‹length ys = length xs› by force
        moreover from ‹j < length xs› have is-free-for (FVar (ys ! j)) (xs ! j) B
          using ‹∀ j < length ys. ys ! j ∉ vars ℋ ∧ ys ! j ∉ vars B› and ‹length ys = length xs›
          and absent-var-is-free-for by presburger
        ultimately show ?thesis
          using ‹v' = xs ! j› by (simp only:)
      qed
    qed
    ultimately show ?thesis
      using substitution-consolidation by simp
  qed
  also have ... = S {?y ⇀ ?A} ++_f (?ϑ (Suc k)) B
  proof −
    have ?ϑ' k = ?ϑ (Suc k)
    proof (intro fsubset-antisym[unfolded fmsubset-alt-def] fmpredI)
```

```
{
 fix v' and A'
 assume ?ϑ' k $$ v' = Some A'
 then have v' ∈ fmdom' (?ϑ' k)
   by (intro fmdom'I)
 then obtain j where j < length xs and xs ! j = v'
   using ‹fmdom' (?ϑ' k) = lset xs› by (metis in-set-conv-nth)
 then consider (a) j < k | (b) j = k | (c) j ∈ {k<..< length xs}
   by fastforce
 then show ?ϑ (Suc k) $$ v' = Some A'
 proof cases
   case a
   with ϑ'-alt-def and ‹distinct xs› and ‹j < length xs›
   have ?ϑ' k $$ (xs ! j) = Some (take k (map (λA'. S {?y ↣ ?A} A') As) ! j)
     using ‹length xs = length As› and ‹length ys = length xs› by auto
   also from a and Suc.prems have ... = Some (S {?y ↣ ?A} (As ! j))
     using ‹length xs = length As› by auto
   also have ... = Some (As ! j)
   proof −
     from Suc.prems and ‹length ys = length xs› have Suc k ≤ length ys
       by (simp only:)
     moreover have j < length As
       using ‹j < length xs› and ‹length xs = length As› by (simp only:)
     ultimately have ?y ∉ vars (As ! j)
       using ys-fresh by force
     then show ?thesis
       using free-var-singleton-substitution-neutrality and free-vars-in-all-vars by blast
   qed
   also from a and ‹xs ! j = v'› and ‹distinct xs› have ... = ?ϑ (Suc k) $$ v'
     using ‹j < length xs› and ‹length xs = length As› and ‹length ys = length xs›
     by fastforce
   finally show ?thesis
     using ‹?ϑ' k $$ v' = Some A'› and ‹xs ! j = v'› by simp
 next
   case b
   then have
     ?ϑ' k $$ (xs ! k) = Some (drop k (map (λA'. S {?y ↣ ?A} A') (map FVar ys)) ! 0)
     using ‹distinct xs› and ‹j < length xs› and ‹length xs = length As›
     and ‹length ys = length xs› and fmap-of-list-nth-split by simp
   also from Suc.prems have ... = Some (S {?y ↣ ?A} (FVar ?y))
     using ‹length ys = length xs› by simp
   also from ‹(n_y, α_y) = ys ! k› have ... = Some ?A
     by (metis singleton-substitution-simps(1))
   also from b and ‹xs ! j = v'› and ‹distinct xs› have ... = ?ϑ (Suc k) $$ v'
     using ‹j < length xs› and ‹length xs = length As› and ‹length ys = length xs›
     by fastforce
   finally show ?thesis
     using b and ‹?ϑ' k $$ v' = Some A'› and ‹xs ! j = v'› by force
 next
```

case c
then have $j > k$
by *simp*
with ϑ'*-alt-def* **and** ‹*distinct xs*› **and** ‹$j <$ *length xs*› **have**
$?\vartheta'\ k\ \$\$\ (xs\ !\ j) = Some\ (drop\ k\ (map\ (\lambda A'.\ \mathbf{S}\ \{?y \rightarrowtail ?A\}\ A')\ (map\ FVar\ ys))\ !\ (j - k))$
using *fmap-of-list-nth-split* **and** ‹*length xs = length As*› **and** ‹*length ys = length xs*›
by *simp*
also from *Suc.prems* **and** c **have** $\ldots = Some\ (\mathbf{S}\ \{?y \rightarrowtail ?A\}\ (FVar\ (ys\ !\ j)))$
using ‹*length ys = length xs*› **by** *simp*
also from *Suc-le-lessD*[*OF Suc.prems*] **and** ‹*distinct ys*› **have** $\ldots = Some\ (FVar\ (ys\ !\ j))$
using ‹$j <$ *length xs*› **and** ‹$k < j$› **and** ‹*length ys = length xs*›
by (*metis nless-le nth-eq-iff-index-eq prod.exhaust-sel singleton-substitution-simps*(*1*))
also from c **and** ‹*distinct xs*› **have** $\ldots = ?\vartheta\ (Suc\ k)\ \$\$\ v'$
using ‹$xs\ !\ j = v'$› **and** ‹*length xs = length As*› **and** ‹*length ys = length xs*› **by** *force*
finally show *?thesis*
using ‹$?\vartheta'\ k\ \$\$\ v' = Some\ A'$› **and** ‹$xs\ !\ j = v'$› **by** *force*
qed
}
note ϑ*-k-in-Sub-k = this*
{
fix v' **and** A'
assume $?\vartheta\ (Suc\ k)\ \$\$\ v' = Some\ A'$
then have $v' \in fmdom'\ (?\vartheta\ (Suc\ k))$
by (*intro fmdom'I*)
then obtain j **where** $j <$ *length xs* **and** $xs\ !\ j = v'$
using ‹$fmdom'\ (?\vartheta\ (Suc\ k)) =$ *lset xs*› **by** (*metis in-set-conv-nth*)
then consider $(a)\ j < k \mid (b)\ j = k \mid (c)\ j \in \{k<..<$ *length xs*$\}$
by *fastforce*
with ‹$j <$ *length xs*› **and** ‹$xs\ !\ j = v'$› **and** ϑ*-k-in-Sub-k* **show** $?\vartheta'\ k\ \$\$\ v' = Some\ A'$
using ‹$\bigwedge k'.\ fmdom'\ (?\vartheta'\ k') =$ *lset xs*› **and** ‹$?\vartheta\ (Suc\ k)\ \$\$\ v' = Some\ A'$›
by (*metis* (*mono-tags*, *lifting*) *fmlookup-dom'-iff nth-mem*)+
}
qed
then show *?thesis*
by *presburger*
qed
also have $\ldots = \mathbf{S}\ (?\vartheta\ (Suc\ k))\ B$
proof −
have $?\vartheta\ (Suc\ k)\ \$\$\ ?y = None$
using ‹$?y \in$ *lset ys*› ‹$\bigwedge k'.\ fmdom'\ (?\vartheta\ k') =$ *lset xs*› **and** *ys-fresh* **by** *blast*
moreover from *Suc-le-lessD*[*OF Suc.prems*] **have** $?y \notin vars\ B$
using ‹$\forall j <$ *length ys*. $ys\ !\ j \notin vars\ \mathcal{H} \wedge ys\ !\ j \notin vars\ B$› **and** ‹*length ys = length xs*›
by *auto*
ultimately show *?thesis*
by (*rule substitution-absorption*)
qed
finally show *?thesis* **.**
qed
— $\mathcal{H} \vdash \mathsf{S} \begin{smallmatrix} x^1_{\alpha_1} \ldots x^k_{\alpha_k} x^{k+1}_{\alpha_{k+1}} x^{k+2}_{\alpha_{k+2}} \ldots x^n_{\alpha_n} \\ A^1_{\alpha_1} \ldots A^k_{\alpha_k} A^{k+1}_{\alpha_{k+1}} y^{k+2}_{\alpha_{k+2}} \ldots y^n_{\alpha_n} \end{smallmatrix} B$

ultimately show *?case*
by (*simp only*:)
qed
— $\mathcal{H} \vdash \mathsf{S}\,{}^{x^1_{\alpha_1} \ldots\, x^n_{\alpha_n}}_{A^1_{\alpha_1} \ldots\, A^n_{\alpha_n}}\, B$
then have $\mathcal{H} \vdash \mathbf{S}$ (*fmap-of-list* (*zip xs As*)) *B*
using ‹*length xs* = *length As*› **and** ‹*length ys* = *length xs*› **by** *force*
moreover have *fmap-of-list* (*zip xs As*) = *ϑ*
proof (*intro fsubset-antisym*[*unfolded fmsubset-alt-def*] *fmpredI*)
fix *v* **and** *A*
assume *fmap-of-list* (*zip xs As*) $$ *v* = *Some A*
with ‹*lset xs* = *fmdom′ ϑ*› **have** *v* ∈ *fmdom′ ϑ*
by (*fast dest*: *fmap-of-list-SomeD set-zip-leftD*)
with ‹*fmap-of-list* (*zip xs As*) $$ *v* = *Some A*› **and** ‹*As* = *map* (($$!) *ϑ*) *xs*› **show** *ϑ* $$ *v* = *Some A*
by
(*simp add*: *map-of-zip-map fmap-of-list.rep-eq split*: *if-splits*)
(*meson fmdom′-notI option.exhaust-sel*)
next
fix *v* **and** *A*
assume *ϑ* $$ *v* = *Some A*
with ‹*As* = *map* (($$!) *ϑ*) *xs*› **show** *fmap-of-list* (*zip xs As*) $$ *v* = *Some A*
using ‹*lset xs* = *fmdom′ ϑ*› **by** (*simp add*: *fmap-of-list.rep-eq fmdom′I map-of-zip-map*)
qed
ultimately show *?thesis*
by (*simp only*:)
qed

end

lemmas *Sub* = *prop-5221*

6.23 Proposition 5222 (Rule of Cases)

lemma *forall-α-conversion*:
assumes *A* ∈ *wffs*$_o$
and (*z*, *β*) ∉ *free-vars A*
and *is-free-for* (z_β) (*x*, *β*) *A*
shows ⊢ ∀ x_β. *A* =$_o$ ∀ z_β. **S** {(*x*, *β*) ⇀ z_β} *A*
proof −
from *assms*(*1*) **have** ∀ x_β. *A* ∈ *wffs*$_o$
by (*intro forall-wff*)
then have ⊢ ∀ x_β. *A* =$_o$ ∀ x_β. *A*
by (*fact prop-5200*)
moreover from *assms* **have** ⊢ (λx_β. *A*) =$_{\beta\to o}$ (λz_β. **S** {(*x*, *β*) ⇀ z_β} *A*)
by (*rule prop-5206*)
ultimately show *?thesis*
unfolding *forall-def* **and** *PI-def* **by** (*rule rule-R* [**where** *p* = [»,»]]) *force*+
qed

proposition *prop-5222*:
assumes $\mathcal{H} \vdash \mathbf{S}\ \{(x,\ o) \rightarrowtail T_o\}\ A$ **and** $\mathcal{H} \vdash \mathbf{S}\ \{(x,\ o) \rightarrowtail F_o\}\ A$
and $A \in \mathit{wffs}_o$
shows $\mathcal{H} \vdash A$
proof −
from *assms(1)* **have** *is-hyps* $\mathcal{H}$
by (*blast elim*: *is-derivable-from-hyps.cases*)
have §*1*: $\mathcal{H} \vdash T_o =_o (\lambda x_o.\ A) \cdot T_o$
proof −
have $\vdash (\lambda x_o.\ A) \cdot T_o =_o \mathbf{S}\ \{(x,\ o) \rightarrowtail T_o\}\ A$
using *prop-5207*[*OF true-wff assms(3) closed-is-free-for*] **by** *simp*
from *this* **and** *assms(1)* **have** $\mathcal{H} \vdash (\lambda x_o.\ A) \cdot T_o$
using *rule-RR*[*OF disjI2*, **where** $p = []$] **by** *fastforce*
moreover have $(\lambda x_o.\ A) \cdot T_o \in \mathit{wffs}_o$
by (*fact hyp-derivable-form-is-wffso*[*OF* ‹$\mathcal{H} \vdash (\lambda x_o.\ A) \cdot T_o$›])
ultimately show *?thesis*
using *rule-T(1)* **by** *blast*
qed
moreover have §*2*: $\mathcal{H} \vdash T_o =_o (\lambda x_o.\ A) \cdot F_o$
proof −
have $\vdash (\lambda x_o.\ A) \cdot F_o =_o \mathbf{S}\ \{(x,\ o) \rightarrowtail F_o\}\ A$
using *prop-5207*[*OF false-wff assms(3) closed-is-free-for*] **by** *simp*
from *this* **and** *assms(2)* **have** $\mathcal{H} \vdash (\lambda x_o.\ A) \cdot F_o$
using *rule-RR*[*OF disjI2*, **where** $p = []$] **by** *fastforce*
moreover have $(\lambda x_o.\ A) \cdot F_o \in \mathit{wffs}_o$
by (*fact hyp-derivable-form-is-wffso*[*OF* ‹$\mathcal{H} \vdash (\lambda x_o.\ A) \cdot F_o$›])
ultimately show *?thesis*
using *rule-T(1)* **by** *blast*
qed
moreover from *prop-5212* **and** ‹*is-hyps* $\mathcal{H}$› **have** §*3*: $\mathcal{H} \vdash T_o \wedge^{\mathcal{Q}} T_o$
by (*rule derivability-implies-hyp-derivability*)
ultimately have $\mathcal{H} \vdash (\lambda x_o.\ A) \cdot T_o \wedge^{\mathcal{Q}} (\lambda x_o.\ A) \cdot F_o$
proof −
from §*3* **and** §*1* **have** $\mathcal{H} \vdash (\lambda x_o.\ A) \cdot T_o \wedge^{\mathcal{Q}} T_o$
by (*rule rule-R'*[**where** $p = [«,»]$]) (*force*+, *fastforce dest*: *subforms-from-app*)
from *this* **and** §*2* **show** *?thesis*
by (*rule rule-R'*[**where** $p = [»]$]) (*force*+, *fastforce dest*: *subforms-from-app*)
qed
moreover have $\vdash (\lambda x_o.\ A) \cdot T_o \wedge^{\mathcal{Q}} (\lambda x_o.\ A) \cdot F_o =_o \forall x_o.\ A$
proof −
have $\mathfrak{g}_{o\to o} \cdot \mathfrak{x}_o \in \mathit{wffs}_o$
by *blast*
have $\vdash \mathfrak{g}_{o\to o} \cdot T_o \wedge^{\mathcal{Q}} \mathfrak{g}_{o\to o} \cdot F_o =_o \forall \mathfrak{x}_o.\ \mathfrak{g}_{o\to o} \cdot \mathfrak{x}_o$
using *axiom-1*[*unfolded equivalence-def*] **by** (*rule axiom-is-derivable-from-no-hyps*)
— By α-conversion
then have $\vdash \mathfrak{g}_{o\to o} \cdot T_o \wedge^{\mathcal{Q}} \mathfrak{g}_{o\to o} \cdot F_o =_o \forall x_o.\ \mathfrak{g}_{o\to o} \cdot x_o$ (**is** $\vdash$ *?B* $=_o$ *?C*)
proof −
have $\vdash \forall \mathfrak{x}_o.\ \mathfrak{g}_{o\to o} \cdot \mathfrak{x}_o =_o \forall x_o.\ \mathfrak{g}_{o\to o} \cdot x_o$

```
    proof (cases x = 𝔵)
      case True
      from ‹𝔤_{o→o} ▪ 𝔵_o ∈ wffs_o› have ⊢ ∀𝔵_o. 𝔤_{o→o} ▪ 𝔵_o =_o ∀𝔵_o. 𝔤_{o→o} ▪ 𝔵_o
        by (fact prop-5200[OF forall-wff])
      with True show ?thesis
        using identity-singleton-substitution-neutrality by simp
    next
      case False
      from ‹𝔤_{o→o} ▪ 𝔵_o ∈ wffs_o›
      have ⊢ ∀𝔵_o. 𝔤_{o→o} ▪ 𝔵_o =_o ∀x_o. S {(𝔵, o) ⇀ x_o} (𝔤_{o→o} ▪ 𝔵_o)
        by
          (rule forall-α-conversion)
          (simp add: False, intro is-free-for-to-app is-free-for-in-var)
      then show ?thesis
        by force
    qed
    with ‹⊢ 𝔤_{o→o} ▪ T_o ∧^Ω 𝔤_{o→o} ▪ F_o =_o ∀𝔵_o. 𝔤_{o→o} ▪ 𝔵_o› show ?thesis
      using Equality-Rules(3) by blast
  qed
  — By Sub
  then have ∗: ⊢ (λx_o. A) ▪ T_o ∧^Ω (λx_o. A) ▪ F_o =_o ∀x_o. (λx_o. A) ▪ x_o
  proof −
    let ?ϑ = {(𝔤, o→o) ⇀ λx_o. A}
    from assms(3) have is-substitution ?ϑ
      by auto
    moreover have
      ∀v ∈ fmdom′ ?ϑ.
        var-name v ∉ free-var-names ({}::form set) ∧ is-free-for (?ϑ $$! v) v (?B =_o ?C)
      by (code-simp, (unfold atomize-conj[symmetric])?, simp)+ blast
    moreover have ?ϑ ≠ {$$}
      by simp
    ultimately have ⊢ S ?ϑ (?B =_o ?C)
      by (rule Sub [OF ‹⊢ ?B =_o ?C›])
    then show ?thesis
      by simp
  qed
  — By λ-conversion
  then show ?thesis
  proof −
    have ⊢ (λx_o. A) ▪ x_o =_o A
      using prop-5208[where vs = [(x, o)]] and assms(3) by simp
    from ∗ and this show ?thesis
      by (rule rule-R[where p = [»,»,«]]) force+
  qed
qed
ultimately have ℋ ⊢ ∀x_o. A
  using rule-RR and is-subform-at.simps(1) by (blast intro: empty-is-position)
then show ?thesis
proof −
```

have *is-free-for* (x_o) $(x,\ o)$ A
by *fastforce*
from ‹$\mathcal{H} \vdash \forall\, x_o.\ A$› **and** *wffs-of-type-intros(1)* **and** *this* **show** *?thesis*
by (*rule* $\forall I$[*of* $\mathcal{H}$ x o A x_o, *unfolded identity-singleton-substitution-neutrality*])
qed
qed

lemmas *Cases* = *prop-5222*

6.24 Proposition 5223

proposition *prop-5223*:
shows $\vdash (T_o \supset^{\mathcal{Q}} \mathfrak{y}_o) =_o \mathfrak{y}_o$
proof −
have $\vdash (T_o \supset^{\mathcal{Q}} \mathfrak{y}_o) =_o (T_o =_o (T_o \wedge^{\mathcal{Q}} \mathfrak{y}_o))$
proof −
let *?A* = $(\lambda \mathfrak{x}_o.\ \lambda \mathfrak{y}_o.\ (\mathfrak{x}_o \equiv^{\mathcal{Q}} \mathfrak{x}_o \wedge^{\mathcal{Q}} \mathfrak{y}_o)) \cdot T_o \cdot \mathfrak{y}_o$
have *?A* $\in$ *wffs*$_o$
by *force*
then have $\vdash$ *?A* $=_o$ *?A*
by (*fact prop-5200*)
then have $\vdash (T_o \supset^{\mathcal{Q}} \mathfrak{y}_o) =_o$ *?A*
unfolding *imp-fun-def* **and** *imp-op-def* **.**
moreover
have $\vdash (\lambda \mathfrak{x}_o.\ \lambda \mathfrak{y}_o.\ (\mathfrak{x}_o \equiv^{\mathcal{Q}} \mathfrak{x}_o \wedge^{\mathcal{Q}} \mathfrak{y}_o)) \cdot T_o =_{o \to o} \lambda \mathfrak{y}_o.\ (T_o \equiv^{\mathcal{Q}} T_o \wedge^{\mathcal{Q}} \mathfrak{y}_o)$
proof −
have $\lambda \mathfrak{y}_o.\ (\mathfrak{x}_o \equiv^{\mathcal{Q}} \mathfrak{x}_o \wedge^{\mathcal{Q}} \mathfrak{y}_o) \in$ *wffs*$_{o \to o}$
by *auto*
moreover
have *is-free-for* T_o $(\mathfrak{x},\ o)$ $(\lambda \mathfrak{y}_o.\ (\mathfrak{x}_o \equiv^{\mathcal{Q}} \mathfrak{x}_o \wedge^{\mathcal{Q}} \mathfrak{y}_o))$
by *fastforce*
moreover
have S $\{(\mathfrak{x},\ o) \rightarrowtail T_o\}$ $(\lambda \mathfrak{y}_o.\ (\mathfrak{x}_o \equiv^{\mathcal{Q}} \mathfrak{x}_o \wedge^{\mathcal{Q}} \mathfrak{y}_o)) = (\lambda \mathfrak{y}_o.\ (T_o \equiv^{\mathcal{Q}} T_o \wedge^{\mathcal{Q}} \mathfrak{y}_o))$
by *simp*
ultimately show *?thesis*
using *prop-5207*[*OF true-wff*] **by** *metis*
qed
ultimately have $*$: $\vdash (T_o \supset^{\mathcal{Q}} \mathfrak{y}_o) =_o (\lambda \mathfrak{y}_o.\ (T_o \equiv^{\mathcal{Q}} T_o \wedge^{\mathcal{Q}} \mathfrak{y}_o)) \cdot \mathfrak{y}_o$
by (*rule rule-R* [**where** p = [»,«]]) *force+*
have $T_o \equiv^{\mathcal{Q}} T_o \wedge^{\mathcal{Q}} \mathfrak{y}_o \in$ *wffs*$_o$
by *auto*
then have $\vdash (\lambda \mathfrak{y}_o.\ (T_o \equiv^{\mathcal{Q}} T_o \wedge^{\mathcal{Q}} \mathfrak{y}_o)) \cdot \mathfrak{y}_o =_o (T_o \equiv^{\mathcal{Q}} T_o \wedge^{\mathcal{Q}} \mathfrak{y}_o)$
using *prop-5208*[**where** *vs* = [$(\mathfrak{y},\ o)$]] **by** *simp*
from $*$ **and** *this* **show** *?thesis*
by (*rule rule-R*[**where** p = [»]]) *force+*
qed
with *prop-5218* **have** $\vdash (T_o \supset^{\mathcal{Q}} \mathfrak{y}_o) =_o (T_o \wedge^{\mathcal{Q}} \mathfrak{y}_o)$
using *rule-R* **and** *Equality-Rules(3)* **by** (*meson conj-op-wff true-wff wffs-of-type-intros(1)*)
with *prop-5216* **show** *?thesis*

using *rule-R* **and** *Equality-Rules*(*3*) **by** (*meson conj-op-wff true-wff wffs-of-type-intros*(*1*))
qed

corollary *generalized-prop-5223*:
assumes $A \in \mathit{wffs}_o$
shows $\vdash (T_o \supset^{\mathcal{Q}} A) =_o A$
proof −
have $T_o \supset^{\mathcal{Q}} \mathfrak{y}_o \in \mathit{wffs}_o$ **and** *is-free-for A* ($\mathfrak{y}$, *o*) $((T_o \supset^{\mathcal{Q}} \mathfrak{y}_o) =_o \mathfrak{y}_o)$
by (*blast*, *intro is-free-for-in-equality is-free-for-in-imp is-free-for-in-true is-free-for-in-var*)
from *this*(*2*) **have** $\vdash \mathbf{S}\ \{(\mathfrak{y},\ o) \rightarrowtail A\}\ ((T_o \supset^{\mathcal{Q}} \mathfrak{y}_o) =_o \mathfrak{y}_o)$
by (*rule prop-5209*[*OF assms* ‹$T_o \supset^{\mathcal{Q}} \mathfrak{y}_o \in \mathit{wffs}_o$› *wffs-of-type-intros*(*1*) *prop-5223*])
then show *?thesis*
by *simp*
qed

6.25 Proposition 5224 (Modus Ponens)

proposition *prop-5224*:
assumes $\mathcal{H} \vdash A$ **and** $\mathcal{H} \vdash A \supset^{\mathcal{Q}} B$
shows $\mathcal{H} \vdash B$
proof −
have $\mathcal{H} \vdash A \supset^{\mathcal{Q}} B$
by *fact*
moreover from *assms*(*1*) **have** $A \in \mathit{wffs}_o$
by (*fact hyp-derivable-form-is-wffso*)
from *this* **and** *assms*(*1*) **have** $\mathcal{H} \vdash A =_o T_o$
using *rule-T*(*2*) **by** *blast*
ultimately have $\mathcal{H} \vdash T_o \supset^{\mathcal{Q}} B$
by (*rule rule-R'*[**where** *p* = [«,»]]) (*force*+, *fastforce dest*: *subforms-from-app*)
have $\vdash (T_o \supset^{\mathcal{Q}} B) =_o B$
proof −
let *?ϑ* = $\{(\mathfrak{y},\ o) \rightarrowtail B\}$
have $B \in \mathit{wffs}_o$
by (*fact hyp-derivable-form-is-wffso*[*OF assms*(*2*), *THEN wffs-from-imp-op*(*2*)])
then have *is-substitution ?ϑ*
by *simp*
moreover have
$\forall\, v \in \mathit{fmdom}'\ ?\vartheta.$
var-name v ∉ *free-var-names* ({}::*form set*) ∧
is-free-for (*?ϑ* $$! *v*) *v* $((T_o \supset^{\mathcal{Q}} \mathfrak{y}_o) =_o \mathfrak{y}_o)$
by (*code-simp*, (*unfold atomize-conj*[*symmetric*])*?*, *simp*)+ *blast*
moreover have *?ϑ* ≠ {$$}
by *simp*
ultimately have $\vdash \mathbf{S}\ ?\vartheta\ ((T_o \supset^{\mathcal{Q}} \mathfrak{y}_o) =_o \mathfrak{y}_o)$
by (*rule Sub*[*OF prop-5223*])
then show *?thesis*
by *simp*
qed
then show *?thesis*

by (*rule rule-RR*[*OF disjI1*, **where** $p = []$]) (*use* ‹$\mathcal{H} \vdash T_o \supset^{\mathcal{Q}} B$› **in** ‹*force*+›)
qed

lemmas *MP* = *prop-5224*

corollary *generalized-modus-ponens*:
assumes $\mathcal{H} \vdash hs \supset^{\mathcal{Q}}_{\star} B$ **and** $\forall H \in lset\ hs.\ \mathcal{H} \vdash H$
shows $\mathcal{H} \vdash B$
using *assms* **proof** (*induction hs arbitrary*: *B rule*: *rev-induct*)
case *Nil*
then show *?case*
by *simp*
next
case (*snoc H' hs*)
from ‹$\forall H \in lset\ (hs\ @\ [H']).\ \mathcal{H} \vdash H$› **have** $\mathcal{H} \vdash H'$
by *simp*
moreover have $\mathcal{H} \vdash H' \supset^{\mathcal{Q}} B$
proof −
from ‹$\mathcal{H} \vdash (hs\ @\ [H']) \supset^{\mathcal{Q}}_{\star} B$› **have** $\mathcal{H} \vdash hs \supset^{\mathcal{Q}}_{\star} (H' \supset^{\mathcal{Q}} B)$
by *simp*
moreover from ‹$\forall H \in lset\ (hs\ @\ [H']).\ \mathcal{H} \vdash H$› **have** $\forall H \in lset\ hs.\ \mathcal{H} \vdash H$
by *simp*
ultimately show *?thesis*
by (*elim snoc.IH*)
qed
ultimately show *?case*
by (*rule MP*)
qed

6.26 Proposition 5225

proposition *prop-5225*:
shows $\vdash \prod \alpha \cdot \mathfrak{f}_{\alpha\to o} \supset^{\mathcal{Q}} \mathfrak{f}_{\alpha\to o} \cdot \mathfrak{x}_\alpha$
proof −
have $\mathfrak{f}_{\alpha\to o} \cdot \mathfrak{x}_\alpha \in wffs_o$
by *blast*
have §*1*:
⊢
$\prod \alpha \cdot \mathfrak{f}_{\alpha\to o} \supset^{\mathcal{Q}} (((\lambda \mathfrak{f}_{\alpha\to o}.\ \mathfrak{f}_{\alpha\to o} \cdot \mathfrak{x}_\alpha) \cdot (\lambda \mathfrak{x}_\alpha.\ T_o))$
$=_o$
$((\lambda \mathfrak{f}_{\alpha\to o}.\ \mathfrak{f}_{\alpha\to o} \cdot \mathfrak{x}_\alpha) \cdot \mathfrak{f}_{\alpha\to o}))$
proof −
let *?θ* = $\{(\mathfrak{h}, (\alpha\to o)\to o) \rightarrowtail \lambda \mathfrak{f}_{\alpha\to o}.\ \mathfrak{f}_{\alpha\to o} \cdot \mathfrak{x}_\alpha, (\mathfrak{x}, \alpha\to o) \rightarrowtail \lambda \mathfrak{x}_\alpha.\ T_o, (\mathfrak{y}, \alpha\to o) \rightarrowtail \mathfrak{f}_{\alpha\to o}\}$
and *?A* = $(\mathfrak{x}_{\alpha\to o} =_{\alpha\to o} \mathfrak{y}_{\alpha\to o}) \supset^{\mathcal{Q}} (\mathfrak{h}_{(\alpha\to o)\to o} \cdot \mathfrak{x}_{\alpha\to o} \equiv^{\mathcal{Q}} \mathfrak{h}_{(\alpha\to o)\to o} \cdot \mathfrak{y}_{\alpha\to o})$
have ⊢ *?A*
by (*fact axiom-is-derivable-from-no-hyps*[*OF axiom-2*])
moreover have $\lambda \mathfrak{f}_{\alpha\to o}.\ \mathfrak{f}_{\alpha\to o} \cdot \mathfrak{x}_\alpha \in wffs_{(\alpha\to o)\to o}$ **and** $\lambda \mathfrak{x}_\alpha.\ T_o \in wffs_{\alpha\to o}$
and $\mathfrak{f}_{\alpha\to o} \in wffs_{\alpha\to o}$
by *blast*+

then have *is-substitution ?ϑ*
by *simp*
moreover have
$\forall\, v \in$ *fmdom′ ?ϑ*. *var-name v* $\notin$ *free-var-names* ({}::*form set*) $\wedge$ *is-free-for* (*?ϑ* $$! *v*) *v ?A*
by (*code-simp*, (*unfold atomize-conj*[*symmetric*])*?*, *simp*)+ *blast*
moreover have *?ϑ* $\neq$ {$$}
by *simp*
ultimately have $\vdash$ **S** *?ϑ ?A*
by (*rule Sub*)
then show *?thesis*
by *simp*
qed
have $\vdash \prod \alpha \cdot \mathfrak{f}_{\alpha\to o} \supset^{\mathfrak{Q}} (T_o =_o \mathfrak{f}_{\alpha\to o} \cdot \mathfrak{x}_\alpha)$
proof −
have
$\vdash (\lambda \mathfrak{f}_{\alpha\to o}.\ \mathfrak{f}_{\alpha\to o} \cdot \mathfrak{x}_\alpha) \cdot (\lambda \mathfrak{x}_\alpha.\ T_o) =_o (\lambda \mathfrak{x}_\alpha.\ T_o) \cdot \mathfrak{x}_\alpha$
(**is** $\vdash (\lambda ?x_{?\beta}.\ ?B) \cdot ?A =_o ?C$)
proof −
have $\vdash (\lambda ?x_{?\beta}.\ ?B) \cdot ?A =_o$ **S** $\{(?x,\ ?\beta) \rightarrowtail ?A\}$ *?B*
using *prop-5207*[*OF wffs-of-type-intros*(*4*)[*OF true-wff*] ‹*?B* $\in$ *wffs*$_o$›] **by** *fastforce*
then show *?thesis*
by *simp*
qed
moreover have $\vdash (\lambda \mathfrak{x}_\alpha.\ T_o) \cdot \mathfrak{x}_\alpha =_o T_o$
using *prop-5208*[**where** *vs* = [($\mathfrak{x}$, α)]] **and** *true-wff* **by** *simp*
ultimately have $\vdash (\lambda \mathfrak{f}_{\alpha\to o}.\ \mathfrak{f}_{\alpha\to o} \cdot \mathfrak{x}_\alpha) \cdot (\lambda \mathfrak{x}_\alpha.\ T_o) =_o T_o$
by (*rule Equality-Rules*(*3*))
from §*1* **and** *this* **have** $\vdash \prod \alpha \cdot \mathfrak{f}_{\alpha\to o} \supset^{\mathfrak{Q}} (T_o =_o ((\lambda \mathfrak{f}_{\alpha\to o}.\ \mathfrak{f}_{\alpha\to o} \cdot \mathfrak{x}_\alpha) \cdot \mathfrak{f}_{\alpha\to o}))$
by (*rule rule-R*[**where** *p* = [»,«,»]]) *force*+
moreover have $\vdash (\lambda \mathfrak{f}_{\alpha\to o}.\ \mathfrak{f}_{\alpha\to o} \cdot \mathfrak{x}_\alpha) \cdot \mathfrak{f}_{\alpha\to o} =_o \mathfrak{f}_{\alpha\to o} \cdot \mathfrak{x}_\alpha$
using *prop-5208*[**where** *vs* = [($\mathfrak{f}$, $\alpha\to o$)]] **by** *force*
ultimately show *?thesis*
by (*rule rule-R*[**where** *p* = [»,»]]) *force*+
qed
from *this* **and** *prop-5218*[*OF* ‹$\mathfrak{f}_{\alpha\to o} \cdot \mathfrak{x}_\alpha \in$ *wffs*$_o$›] **show** *?thesis*
by (*rule rule-R*[**where** *p* = [»]]) *auto*
qed

6.27 Proposition 5226

proposition *prop-5226*:
assumes $A \in$ *wffs*$_\alpha$ **and** $B \in$ *wffs*$_o$
and *is-free-for A* (x, α) *B*
shows $\vdash \forall\, x_\alpha.\ B \supset^{\mathfrak{Q}}$ **S** $\{(x, \alpha) \rightarrowtail A\}$ *B*
proof −
have $\vdash \prod \alpha \cdot (\lambda x_\alpha.\ B) \supset^{\mathfrak{Q}} (\lambda x_\alpha.\ B) \cdot A$
proof −
let *?ϑ* = {($\mathfrak{f}$, $\alpha\to o$) $\rightarrowtail \lambda x_\alpha.\ B$, ($\mathfrak{x}$, α) $\rightarrowtail A$}
have $\vdash \prod \alpha \cdot \mathfrak{f}_{\alpha\to o} \supset^{\mathfrak{Q}} \mathfrak{f}_{\alpha\to o} \cdot \mathfrak{x}_\alpha$ (**is** $\vdash$ *?C*)

by (*fact prop-5225*)
moreover from *assms* **have** *is-substitution ?ϑ*
by *auto*
moreover have
$\forall\, v \in \mathit{fmdom}'\ ?\vartheta.\ \mathit{var\text{-}name}\ v \notin \mathit{free\text{-}var\text{-}names}\ (\{\}::\mathit{form\ set}) \wedge \mathit{is\text{-}free\text{-}for}\ (?\vartheta\ \$\$!\ v)\ v\ ?C$
by (*code-simp*, (*unfold atomize-conj*[*symmetric*])*?*, *fastforce*)+ *blast*
moreover have $?\vartheta \neq \{\$\$\}$
by *simp*
ultimately have $\vdash$ **S** *?ϑ ?C*
by (*rule Sub*)
moreover have S $?\vartheta\ ?C = \prod \alpha \cdot (\lambda x_\alpha.\ B) \supset^{\mathfrak{Q}} (\lambda x_\alpha.\ B) \cdot A$
by *simp*
ultimately show *?thesis*
by (*simp only*:)
qed
moreover from *assms* **have** $\vdash (\lambda x_\alpha.\ B) \cdot A =_o$ **S** $\{(x,\ \alpha) \rightarrowtail A\}\ B$
by (*rule prop-5207*)
ultimately show *?thesis*
by (*rule rule-R* [**where** $p = [\text{»}]$]) *force*+
qed

6.28 Proposition 5227

corollary *prop-5227*:
shows $\vdash F_o \supset^{\mathfrak{Q}} \mathfrak{x}_o$
proof −
have $\vdash \forall \mathfrak{x}_o.\ \mathfrak{x}_o \supset^{\mathfrak{Q}}$ **S** $\{(\mathfrak{x},\ o) \rightarrowtail \mathfrak{x}_o\}\ (\mathfrak{x}_o)$
by (*rule prop-5226*) *auto*
then show *?thesis*
using *identity-singleton-substitution-neutrality* **by** *simp*
qed

corollary *generalized-prop-5227*:
assumes $A \in \mathit{wffs}_o$
shows $\vdash F_o \supset^{\mathfrak{Q}} A$
proof −
let $?\vartheta = \{(\mathfrak{x},\ o) \rightarrowtail A\}$ **and** $?B = F_o \supset^{\mathfrak{Q}} \mathfrak{x}_o$
from *assms* **have** *is-substitution ?ϑ*
by *simp*
moreover have *is-free-for A* $(\mathfrak{x},\ o)$ *?B*
by (*intro is-free-for-in-false is-free-for-in-imp is-free-for-in-var*)
then have
$\forall\, v \in \mathit{fmdom}'\ ?\vartheta.\ \mathit{var\text{-}name}\ v \notin \mathit{free\text{-}var\text{-}names}\ (\{\}::\mathit{form\ set}) \wedge \mathit{is\text{-}free\text{-}for}\ (?\vartheta\ \$\$!\ v)\ v\ ?B$
by *force*
ultimately have $\vdash$ **S** $\{(\mathfrak{x},\ o) \rightarrowtail A\}\ (F_o \supset^{\mathfrak{Q}} \mathfrak{x}_o)$
using *Sub*[*OF prop-5227*, **where** $\vartheta = ?\vartheta$] **by** *fastforce*
then show *?thesis*
by *simp*
qed

6.29 Proposition 5228

proposition *prop-5228*:
shows $\vdash (T_o \supset^{\mathfrak{Q}} T_o) =_o T_o$
and $\vdash (T_o \supset^{\mathfrak{Q}} F_o) =_o F_o$
and $\vdash (F_o \supset^{\mathfrak{Q}} T_o) =_o T_o$
and $\vdash (F_o \supset^{\mathfrak{Q}} F_o) =_o T_o$
proof −
show $\vdash (T_o \supset^{\mathfrak{Q}} T_o) =_o T_o$ **and** $\vdash (T_o \supset^{\mathfrak{Q}} F_o) =_o F_o$
using *generalized-prop-5223* **by** *blast+*
next
have $\vdash F_o \supset^{\mathfrak{Q}} F_o$ **and** $\vdash F_o \supset^{\mathfrak{Q}} T_o$
using *generalized-prop-5227* **by** *blast+*
then show $\vdash (F_o \supset^{\mathfrak{Q}} T_o) =_o T_o$ **and** $\vdash (F_o \supset^{\mathfrak{Q}} F_o) =_o T_o$
using *rule-T(2)* **by** *blast+*
qed

6.30 Proposition 5229

lemma *false-in-conj-provability*:
assumes $A \in \mathit{wffs}_o$
shows $\vdash F_o \wedge^{\mathfrak{Q}} A \equiv^{\mathfrak{Q}} F_o$
proof −
have $\vdash (\lambda \mathfrak{x}_o.\ \lambda \mathfrak{y}_o.\ (\mathfrak{x}_o \equiv^{\mathfrak{Q}} \mathfrak{x}_o \wedge^{\mathfrak{Q}} \mathfrak{y}_o)) \cdot F_o \cdot A$
by (*intro generalized-prop-5227*[*OF assms*, *unfolded imp-op-def imp-fun-def*])
moreover have
$\vdash$
$(\lambda \mathfrak{x}_o.\ \lambda \mathfrak{y}_o.\ (\mathfrak{x}_o \equiv^{\mathfrak{Q}} \mathfrak{x}_o \wedge^{\mathfrak{Q}} \mathfrak{y}_o)) \cdot F_o$
$=_{o \to o}$
$\lambda \mathfrak{y}_o.\ (F_o \equiv^{\mathfrak{Q}} F_o \wedge^{\mathfrak{Q}} \mathfrak{y}_o)$
(**is** $\vdash (\lambda ?x_{?\beta}.\ ?A) \cdot ?B =_{?\gamma} ?C$)
proof −
have $?B \in \mathit{wffs}_{?\beta}$ **and** $?A \in \mathit{wffs}_{?\gamma}$ **and** *is-free-for* $?B$ $(?x,\ ?\beta)$ $?A$
by *auto*
then have $\vdash (\lambda ?x_{?\beta}.\ ?A) \cdot ?B =_{?\gamma}$ **S** $\{(?x,\ ?\beta) \rightarrowtail ?B\}$ $?A$
by (*rule prop-5207*)
moreover have S $\{(?x,\ ?\beta) \rightarrowtail ?B\}$ $?A = ?C$
by *simp*
ultimately show *?thesis*
by (*simp only*:)
qed
ultimately have $\vdash (\lambda \mathfrak{y}_o.\ (F_o \equiv^{\mathfrak{Q}} F_o \wedge^{\mathfrak{Q}} \mathfrak{y}_o)) \cdot A$
by (*rule rule-R*[**where** $p = [\![]\!]$]) *auto*
moreover have
$\vdash$
$(\lambda \mathfrak{y}_o.\ (F_o \equiv^{\mathfrak{Q}} F_o \wedge^{\mathfrak{Q}} \mathfrak{y}_o)) \cdot A$
$=_o$
$(F_o \equiv^{\mathfrak{Q}} F_o \wedge^{\mathfrak{Q}} A)$
(**is** $\vdash (\lambda ?x_{?\beta}.\ ?A) \cdot ?B =_o ?C$)
proof −

```
    have ?B ∈ wffs_?β and ?A ∈ wffs_o
      using assms by auto
    moreover have is-free-for ?B (?x, ?β) ?A
      by (intro is-free-for-in-equivalence is-free-for-in-conj is-free-for-in-false) fastforce
    ultimately have ⊢ (λ?x_?β. ?A) ▪ ?B =_o S {(?x, ?β) ⇀ ?B} ?A
      by (rule prop-5207)
    moreover
    have S {(?x, ?β) ⇀ ?B} ?A = ?C
      by simp
    ultimately show ?thesis
      by (simp only:)
  qed
  ultimately have ⊢ F_o ≡^Q F_o ∧^Q A
    by (rule rule-R[where p = []]) auto
  then show ?thesis
    unfolding equivalence-def by (rule Equality-Rules(2))
qed

proposition prop-5229:
  shows ⊢ (T_o ∧^Q T_o) =_o T_o
  and ⊢ (T_o ∧^Q F_o) =_o F_o
  and ⊢ (F_o ∧^Q T_o) =_o F_o
  and ⊢ (F_o ∧^Q F_o) =_o F_o
proof −
  show ⊢ (T_o ∧^Q T_o) =_o T_o and ⊢ (T_o ∧^Q F_o) =_o F_o
    using prop-5216 by blast+
next
  show ⊢ (F_o ∧^Q T_o) =_o F_o and ⊢ (F_o ∧^Q F_o) =_o F_o
    using false-in-conj-provability and true-wff and false-wff by simp-all
qed
```

6.31 Proposition 5230

```
proposition prop-5230:
  shows ⊢ (T_o ≡^Q T_o) =_o T_o
  and ⊢ (T_o ≡^Q F_o) =_o F_o
  and ⊢ (F_o ≡^Q T_o) =_o F_o
  and ⊢ (F_o ≡^Q F_o) =_o T_o
proof −
  show ⊢ (T_o ≡^Q T_o) =_o T_o and ⊢ (T_o ≡^Q F_o) =_o F_o
    unfolding equivalence-def using prop-5218 by blast+
next
  show ⊢ (F_o ≡^Q F_o) =_o T_o
    unfolding equivalence-def by (rule Equality-Rules(2)[OF prop-5210[OF false-wff]])
next
  have §1: ⊢ (F_o ≡^Q T_o) ⊃^Q ((λ𝔵_o. (𝔵_o ≡^Q F_o)) ▪ F_o ≡^Q (λ𝔵_o. (𝔵_o ≡^Q F_o)) ▪ T_o)
  proof −
    let ?ϑ = {(𝔥, o→o) ⇀ λ𝔵_o. (𝔵_o ≡^Q F_o), (𝔵, o) ⇀ F_o, (𝔶, o) ⇀ T_o}
      and ?A = (𝔵_o =_o 𝔶_o) ⊃^Q (𝔥_{o→o} ▪ 𝔵_o ≡^Q 𝔥_{o→o} ▪ 𝔶_o)
```

have $\vdash$ *?A*
by (*fact axiom-is-derivable-from-no-hyps*[*OF axiom-2*])
moreover have *is-substitution ?ϑ*
by *auto*
moreover have
$\forall\, v \in$ *fmdom′ ?ϑ*. *var-name v* $\notin$ *free-var-names* ({}::*form set*) $\wedge$ *is-free-for* (*?ϑ* $$! *v*) *v ?A*
by (*code-simp*, *unfold atomize-conj*[*symmetric*], *simp*, *force*)+ *blast*
moreover have *?ϑ* $\neq$ {$$}
by *simp*
ultimately have $\vdash$ **S** *?ϑ ?A*
by (*rule Sub*)
then show *?thesis*
by *simp*
qed
then have §*2*: $\vdash$ $(F_o \equiv^{\mathfrak{Q}} T_o) \supset^{\mathfrak{Q}} ((F_o \equiv^{\mathfrak{Q}} F_o) \equiv^{\mathfrak{Q}} (T_o \equiv^{\mathfrak{Q}} F_o))$ (**is** $\vdash$ *?A2*)
proof −
have *is-free-for A* ($\mathfrak{x}$, *o*) ($\mathfrak{x}_o \equiv^{\mathfrak{Q}} F_o$) **for** *A*
by *code-simp blast*
have *β-reduction*: $\vdash$ $(\lambda\mathfrak{x}_o.\ (\mathfrak{x}_o \equiv^{\mathfrak{Q}} F_o)) \cdot A =_o (A \equiv^{\mathfrak{Q}} F_o)$ **if** $A \in$ *wffs*$_o$ **for** *A*
using
prop-5207
[
OF that equivalence-wff[*OF wffs-of-type-intros*(*1*) *false-wff*]
‹*is-free-for A* ($\mathfrak{x}$, *o*) ($\mathfrak{x}_o \equiv^{\mathfrak{Q}} F_o$)›
]
by *simp*
from §*1* **and** *β-reduction*[*OF false-wff*] **have**
$\vdash$ $(F_o =_o T_o) \supset^{\mathfrak{Q}} ((F_o \equiv^{\mathfrak{Q}} F_o) \equiv^{\mathfrak{Q}} (\lambda\mathfrak{x}_o.\ (\mathfrak{x}_o \equiv^{\mathfrak{Q}} F_o)) \cdot T_o)$
by (*rule rule-R*[**where** *p* = [»,«,»]]) *force*+
from *this* **and** *β-reduction*[*OF true-wff*] **show** *?thesis*
by (*rule rule-R*[**where** *p* = [»,»]]) *force*+
qed
then have §*3*: $\vdash$ $(F_o \equiv^{\mathfrak{Q}} T_o) \supset^{\mathfrak{Q}} F_o$
proof −
note *r1* = *rule-RR*[*OF disjI1*] **and** *r2* = *rule-RR*[*OF disjI2*]
have §*3*$_1$: $\vdash$ $(F_o \equiv^{\mathfrak{Q}} T_o) \supset^{\mathfrak{Q}} ((F_o \equiv^{\mathfrak{Q}} F_o) \equiv^{\mathfrak{Q}} F_o)$ (**is** ‹$\vdash$ *?A3*$_1$›)
by (*rule r1*[*OF prop-5218*[*OF false-wff*], **where** *p* = [»,»] **and** *C* = *?A2*]) (*use* §*2* **in** ‹*force*+›)
have §*3*$_2$: $\vdash$ $(F_o \equiv^{\mathfrak{Q}} T_o) \supset^{\mathfrak{Q}} (T_o \equiv^{\mathfrak{Q}} F_o)$ (**is** ‹$\vdash$ *?A3*$_2$›)
by (*rule r2*[*OF prop-5210*[*OF false-wff*], **where** *p* = [»,«,»] **and** *C* = *?A3*$_1$]) (*use* §*3*$_1$ **in** ‹*force*+›)
show *?thesis*
by (*rule r1*[*OF prop-5218*[*OF false-wff*], **where** *p* = [»] **and** *C* = *?A3*$_2$]) (*use* §*3*$_2$ **in** ‹*force*+›)
qed
then have $\vdash$ $(F_o \equiv^{\mathfrak{Q}} T_o) \equiv^{\mathfrak{Q}} ((F_o \equiv^{\mathfrak{Q}} T_o) \wedge^{\mathfrak{Q}} F_o)$
proof −
have
$\vdash$
$(\lambda\mathfrak{x}_o.\ \lambda\mathfrak{y}_o.\ (\mathfrak{x}_o \equiv^{\mathfrak{Q}} \mathfrak{x}_o \wedge^{\mathfrak{Q}} \mathfrak{y}_o)) \cdot (F_o \equiv^{\mathfrak{Q}} T_o)$
$=_{o\to o}$
S $\{(\mathfrak{x},\ o) \rightarrowtail F_o \equiv^{\mathfrak{Q}} T_o\}$ $(\lambda\mathfrak{y}_o.\ (\mathfrak{x}_o \equiv^{\mathfrak{Q}} \mathfrak{x}_o \wedge^{\mathfrak{Q}} \mathfrak{y}_o))$

by (*rule prop-5207*) *auto*
from §*3*[*unfolded imp-op-def imp-fun-def*] **and** *this*
have $\vdash (\lambda \mathfrak{y}_o.\ ((F_o \equiv^{\mathcal{Q}} T_o) \equiv^{\mathcal{Q}} (F_o \equiv^{\mathcal{Q}} T_o) \wedge^{\mathcal{Q}} \mathfrak{y}_o)) \cdot F_o$
by (*rule rule-R*[**where** $p = [\langle]$]) *force+*
moreover have
$\vdash$
$(\lambda \mathfrak{y}_o.\ ((F_o \equiv^{\mathcal{Q}} T_o) \equiv^{\mathcal{Q}} (F_o \equiv^{\mathcal{Q}} T_o) \wedge^{\mathcal{Q}} \mathfrak{y}_o)) \cdot F_o$
$=_o$
S $\{(\mathfrak{y},\ o) \rightarrowtail F_o\}$ $((F_o \equiv^{\mathcal{Q}} T_o) \equiv^{\mathcal{Q}} (F_o \equiv^{\mathcal{Q}} T_o) \wedge^{\mathcal{Q}} \mathfrak{y}_o)$
by (*rule prop-5207*) *auto*
ultimately show *?thesis*
by (*rule rule-R*[**where** $p = []$]) *force+*
qed
moreover have §*5*: $\vdash \mathfrak{x}_o \wedge^{\mathcal{Q}} F_o \equiv^{\mathcal{Q}} F_o$
proof −
from *prop-5229*(*2*,*4*) **have**
$\vdash$ **S** $\{(\mathfrak{x},\ o) \rightarrowtail T_o\}$ $(\mathfrak{x}_o \wedge^{\mathcal{Q}} F_o \equiv^{\mathcal{Q}} F_o)$ **and** $\vdash$ **S** $\{(\mathfrak{x},\ o) \rightarrowtail F_o\}$ $(\mathfrak{x}_o \wedge^{\mathcal{Q}} F_o \equiv^{\mathcal{Q}} F_o)$
by *simp-all*
moreover have $\mathfrak{x}_o \wedge^{\mathcal{Q}} F_o \equiv^{\mathcal{Q}} F_o \in \mathit{wffs}_o$
by *auto*
ultimately show *?thesis*
by (*rule Cases*)
qed
then have $\vdash (F_o \equiv^{\mathcal{Q}} T_o) \wedge^{\mathcal{Q}} F_o \equiv^{\mathcal{Q}} F_o$
proof −
let *?ϑ* = $\{(\mathfrak{x},\ o) \rightarrowtail F_o \equiv^{\mathcal{Q}} T_o\}$
have *is-substitution ?ϑ*
by *auto*
moreover have $\forall\, v \in \mathit{fmdom'}\ ?\vartheta.$
var-name v $\notin$ *free-var-names* ({}::*form set*) $\wedge$ *is-free-for* (*?ϑ* $$! *v*) *v* $(\mathfrak{x}_o \wedge^{\mathcal{Q}} F_o \equiv^{\mathcal{Q}} F_o)$
by *simp*
moreover have *?ϑ* $\neq$ {$$}
by *simp*
ultimately have $\vdash$ **S** *?ϑ* $(\mathfrak{x}_o \wedge^{\mathcal{Q}} F_o \equiv^{\mathcal{Q}} F_o)$
by (*rule Sub*[*OF* ‹$\vdash \mathfrak{x}_o \wedge^{\mathcal{Q}} F_o \equiv^{\mathcal{Q}} F_o$›])
then show *?thesis*
by *simp*
qed
ultimately show $\vdash (F_o \equiv^{\mathcal{Q}} T_o) =_o F_o$
unfolding *equivalence-def* **by** (*rule Equality-Rules*(*3*))
qed

6.32 Proposition 5231

proposition *prop-5231*:
shows $\vdash \sim^{\mathcal{Q}} T_o =_o F_o$
and $\vdash \sim^{\mathcal{Q}} F_o =_o T_o$
using *prop-5230*(*3*,*4*) **unfolding** *neg-def* **and** *equivalence-def* **and** *equality-of-type-def* **.**

6.33 Proposition 5232

lemma *disj-op-alt-def-provability*:
assumes $A \in \mathit{wffs}_o$ **and** $B \in \mathit{wffs}_o$
shows $\vdash A \vee^{\mathfrak{Q}} B =_o \sim^{\mathfrak{Q}} (\sim^{\mathfrak{Q}} A \wedge^{\mathfrak{Q}} \sim^{\mathfrak{Q}} B)$
proof −
let $?C = (\lambda \mathfrak{x}_o.\ \lambda \mathfrak{y}_o.\ \sim^{\mathfrak{Q}} (\sim^{\mathfrak{Q}} \mathfrak{x}_o \wedge^{\mathfrak{Q}} \sim^{\mathfrak{Q}} \mathfrak{y}_o)) \cdot A \cdot B$
from *assms* **have** $?C \in \mathit{wffs}_o$
by *blast*
have $(\sim^{\mathfrak{Q}} (\sim^{\mathfrak{Q}} \mathfrak{x}_o \wedge^{\mathfrak{Q}} \sim^{\mathfrak{Q}} \mathfrak{y}_o)) \in \mathit{wffs}_o$
by *auto*
moreover obtain z **where** $(z,\ o) \notin \{(\mathfrak{x},\ o), (\mathfrak{y},\ o)\}$ **and** $(z,\ o) \notin \mathit{free\text{-}vars}\ A$
using *free-vars-form-finiteness* **and** *fresh-var-existence*
by (*metis Un-iff Un-insert-right free-vars-form.simps*(*1*,*3*) *inf-sup-aci*(*5*) *sup-bot-left*)
then have $(z,\ o) \notin \mathit{free\text{-}vars}\ (\sim^{\mathfrak{Q}} (\sim^{\mathfrak{Q}} \mathfrak{x}_o \wedge^{\mathfrak{Q}} \sim^{\mathfrak{Q}} \mathfrak{y}_o))$
by *simp*
moreover have *is-free-for* (z_o) $(\mathfrak{y},\ o)$ $(\sim^{\mathfrak{Q}} (\sim^{\mathfrak{Q}} \mathfrak{x}_o \wedge^{\mathfrak{Q}} \sim^{\mathfrak{Q}} \mathfrak{y}_o))$
by (*intro is-free-for-in-conj is-free-for-in-neg is-free-for-in-var*)
ultimately have
$\vdash (\lambda \mathfrak{y}_o.\ \sim^{\mathfrak{Q}} (\sim^{\mathfrak{Q}} \mathfrak{x}_o \wedge^{\mathfrak{Q}} \sim^{\mathfrak{Q}} \mathfrak{y}_o)) =_{o \to o} (\lambda z_o.\ \mathbf{S}\ \{(\mathfrak{y},\ o) \rightarrowtail z_o\}\ \sim^{\mathfrak{Q}} (\sim^{\mathfrak{Q}} \mathfrak{x}_o \wedge^{\mathfrak{Q}} \sim^{\mathfrak{Q}} \mathfrak{y}_o))$
by (*rule* α)
then have $\vdash (\lambda \mathfrak{y}_o.\ \sim^{\mathfrak{Q}} (\sim^{\mathfrak{Q}} \mathfrak{x}_o \wedge^{\mathfrak{Q}} \sim^{\mathfrak{Q}} \mathfrak{y}_o)) =_{o \to o} (\lambda z_o.\ \sim^{\mathfrak{Q}} (\sim^{\mathfrak{Q}} \mathfrak{x}_o \wedge^{\mathfrak{Q}} \sim^{\mathfrak{Q}} z_o))$
by *simp*
from *prop-5200*[*OF* ‹$?C \in \mathit{wffs}_o$›] **and** *this* **have**
$\vdash$
$(\lambda \mathfrak{x}_o.\ \lambda z_o.\ \sim^{\mathfrak{Q}} (\sim^{\mathfrak{Q}} \mathfrak{x}_o \wedge^{\mathfrak{Q}} \sim^{\mathfrak{Q}} z_o)) \cdot A \cdot B$
$=_o$
$(\lambda \mathfrak{x}_o.\ \lambda \mathfrak{y}_o.\ \sim^{\mathfrak{Q}} (\sim^{\mathfrak{Q}} \mathfrak{x}_o \wedge^{\mathfrak{Q}} \sim^{\mathfrak{Q}} \mathfrak{y}_o)) \cdot A \cdot B$
by (*rule rule-R*[**where** $p = [«,»,«,«,«]]$) *force*+
moreover have $\lambda z_o.\ \sim^{\mathfrak{Q}} (\sim^{\mathfrak{Q}} \mathfrak{x}_o \wedge^{\mathfrak{Q}} \sim^{\mathfrak{Q}} z_o) \in \mathit{wffs}_{o \to o}$
by *blast*
have
$\vdash$
$(\lambda \mathfrak{x}_o.\ \lambda z_o.\ \sim^{\mathfrak{Q}} (\sim^{\mathfrak{Q}} \mathfrak{x}_o \wedge^{\mathfrak{Q}} \sim^{\mathfrak{Q}} z_o)) \cdot A$
$=_{o \to o}$
$\mathbf{S}\ \{(\mathfrak{x},\ o) \rightarrowtail A\}\ (\lambda z_o.\ \sim^{\mathfrak{Q}} (\sim^{\mathfrak{Q}} \mathfrak{x}_o \wedge^{\mathfrak{Q}} \sim^{\mathfrak{Q}} z_o))$
by
(*rule prop-5207*)
(
fact, *blast*, *intro is-free-for-in-neg is-free-for-in-conj is-free-for-to-abs*,
(*fastforce simp add*: ‹$(z,\ o) \notin \mathit{free\text{-}vars}\ A$›)+
)
then have $\vdash (\lambda \mathfrak{x}_o.\ \lambda z_o.\ \sim^{\mathfrak{Q}} (\sim^{\mathfrak{Q}} \mathfrak{x}_o \wedge^{\mathfrak{Q}} \sim^{\mathfrak{Q}} z_o)) \cdot A =_{o \to o} (\lambda z_o.\ \sim^{\mathfrak{Q}} (\sim^{\mathfrak{Q}} A \wedge^{\mathfrak{Q}} \sim^{\mathfrak{Q}} z_o))$
using ‹$(z,\ o) \notin \mathit{free\text{-}vars}\ (\sim^{\mathfrak{Q}} (\sim^{\mathfrak{Q}} \mathfrak{x}_o \wedge^{\mathfrak{Q}} \sim^{\mathfrak{Q}} \mathfrak{y}_o))$› **by** *simp*
ultimately have
$\vdash (\lambda z_o.\ \sim^{\mathfrak{Q}} (\sim^{\mathfrak{Q}} A \wedge^{\mathfrak{Q}} \sim^{\mathfrak{Q}} z_o)) \cdot B =_o (\lambda \mathfrak{x}_o.\ \lambda \mathfrak{y}_o.\ \sim^{\mathfrak{Q}} (\sim^{\mathfrak{Q}} \mathfrak{x}_o \wedge^{\mathfrak{Q}} \sim^{\mathfrak{Q}} \mathfrak{y}_o)) \cdot A \cdot B$
by (*rule rule-R*[**where** $p = [«,»,«]]$) *force*+
moreover have $\vdash (\lambda z_o.\ \sim^{\mathfrak{Q}} (\sim^{\mathfrak{Q}} A \wedge^{\mathfrak{Q}} \sim^{\mathfrak{Q}} z_o)) \cdot B =_o \mathbf{S}\ \{(z,\ o) \rightarrowtail B\}\ (\sim^{\mathfrak{Q}} (\sim^{\mathfrak{Q}} A \wedge^{\mathfrak{Q}} \sim^{\mathfrak{Q}} z_o))$
by
(*rule prop-5207*)

(
fact, *blast intro*: *assms(1)*, *intro is-free-for-in-neg is-free-for-in-conj*,
use ‹$(z,\ o) \notin$ *free-vars* A› *is-free-at-in-free-vars* **in** ‹*fastforce+*›
)
moreover have S $\{(z,\ o) \rightarrowtail B\}$ $(\sim^{\mathfrak{Q}}\ (\sim^{\mathfrak{Q}}\ A \wedge^{\mathfrak{Q}} \sim^{\mathfrak{Q}}\ z_o)) = \sim^{\mathfrak{Q}}\ (\sim^{\mathfrak{Q}}\ A \wedge^{\mathfrak{Q}} \sim^{\mathfrak{Q}}\ B)$
using *free-var-singleton-substitution-neutrality*[*OF* ‹$(z,\ o) \notin$ *free-vars* A›] **by** *simp*
ultimately have $\vdash (\lambda \mathfrak{x}_o.\ \lambda \mathfrak{y}_o.\ \sim^{\mathfrak{Q}}\ (\sim^{\mathfrak{Q}}\ \mathfrak{x}_o \wedge^{\mathfrak{Q}} \sim^{\mathfrak{Q}}\ \mathfrak{y}_o)) \cdot A \cdot B =_o \sim^{\mathfrak{Q}}\ (\sim^{\mathfrak{Q}}\ A \wedge^{\mathfrak{Q}} \sim^{\mathfrak{Q}}\ B)$
using *Equality-Rules(2,3)* **by** *metis*
then show *?thesis*
by *simp*
qed

context begin

private lemma *prop-5232-aux*:
assumes $\vdash \sim^{\mathfrak{Q}}\ (A \wedge^{\mathfrak{Q}} B) =_o C$
and $\vdash \sim^{\mathfrak{Q}}\ A' =_o A$ **and** $\vdash \sim^{\mathfrak{Q}}\ B' =_o B$
shows $\vdash A' \vee^{\mathfrak{Q}} B' =_o C$
proof −
let *?D* $= \sim^{\mathfrak{Q}}\ (A \wedge^{\mathfrak{Q}} B) =_o C$
from *assms(2)* **have** $\vdash \sim^{\mathfrak{Q}}\ (\sim^{\mathfrak{Q}}\ A' \wedge^{\mathfrak{Q}} B) =_o C$ (**is** ‹$\vdash$ *?A1*›)
by (*rule rule-RR*[*OF disjI2*, **where** $p = [«,»,»,«,»]$ **and** $C =$ *?D*]) (*use assms(1)* **in** ‹*force+*›)
from *assms(3)* **have** $\vdash \sim^{\mathfrak{Q}}\ (\sim^{\mathfrak{Q}}\ A' \wedge^{\mathfrak{Q}} \sim^{\mathfrak{Q}}\ B') =_o C$
by (*rule rule-RR*[*OF disjI2*, **where** $p = [«,»,»,»]$ **and** $C =$ *?A1*]) (*use* ‹$\vdash$ *?A1*› **in** ‹*force+*›)
moreover from *assms(2,3)* **have** $A' \in wffs_o$ **and** $B' \in wffs_o$
using *hyp-derivable-form-is-wffso* **by** (*blast dest*: *wffs-from-equality wffs-from-neg*)+
then have $\vdash A' \vee^{\mathfrak{Q}} B' =_o \sim^{\mathfrak{Q}}\ (\sim^{\mathfrak{Q}}\ A' \wedge^{\mathfrak{Q}} \sim^{\mathfrak{Q}}\ B')$
by (*rule disj-op-alt-def-provability*)
ultimately show *?thesis*
using *prop-5201-3* **by** *blast*
qed

proposition *prop-5232*:
shows $\vdash (T_o \vee^{\mathfrak{Q}}\ T_o) =_o T_o$
and $\vdash (T_o \vee^{\mathfrak{Q}}\ F_o) =_o T_o$
and $\vdash (F_o \vee^{\mathfrak{Q}}\ T_o) =_o T_o$
and $\vdash (F_o \vee^{\mathfrak{Q}}\ F_o) =_o F_o$
proof −
from *prop-5231(2)* **have** $\vdash \sim^{\mathfrak{Q}}\ F_o =_o T_o$ (**is** ‹$\vdash$ *?A*›) **.**
from *prop-5229(4)* **have** $\vdash \sim^{\mathfrak{Q}}\ (F_o \wedge^{\mathfrak{Q}} F_o) =_o T_o$
by (*rule rule-RR*[*OF disjI2*, **where** $p = [«,»,»]$ **and** $C =$ *?A*]) (*use* ‹$\vdash$ *?A*› **in** ‹*force+*›)
from *prop-5232-aux*[*OF this prop-5231(1) prop-5231(1)*] **show** $\vdash (T_o \vee^{\mathfrak{Q}}\ T_o) =_o T_o$ **.**
from *prop-5229(3)* **have** $\vdash \sim^{\mathfrak{Q}}\ (F_o \wedge^{\mathfrak{Q}} T_o) =_o T_o$
by (*rule rule-RR*[*OF disjI2*, **where** $p = [«,»,»]$ **and** $C =$ *?A*]) (*use* ‹$\vdash$ *?A*› **in** ‹*force+*›)
from *prop-5232-aux*[*OF this prop-5231(1) prop-5231(2)*] **show** $\vdash (T_o \vee^{\mathfrak{Q}}\ F_o) =_o T_o$ **.**
from *prop-5229(2)* **have** $\vdash \sim^{\mathfrak{Q}}\ (T_o \wedge^{\mathfrak{Q}} F_o) =_o T_o$
by (*rule rule-RR*[*OF disjI2*, **where** $p = [«,»,»]$ **and** $C =$ *?A*]) (*use* ‹$\vdash$ *?A*› **in** ‹*force+*›)
from *prop-5232-aux*[*OF this prop-5231(2) prop-5231(1)*] **show** $\vdash (F_o \vee^{\mathfrak{Q}}\ T_o) =_o T_o$ **.**
next

from *prop-5231*(*1*) **have** $\vdash \sim^{\mathcal{Q}} T_o =_o F_o$ (**is** ‹$\vdash$ *?A*›) **.**
from *prop-5229*(*1*) **have** $\vdash \sim^{\mathcal{Q}} (T_o \wedge^{\mathcal{Q}} T_o) =_o F_o$
by (*rule rule-RR*[*OF disjI2*, **where** $p = [«,»,»]$ **and** C = *?A*]) (*use* ‹$\vdash$ *?A*› **in** ‹*force*+›)
from *prop-5232-aux*[*OF this prop-5231*(*2*) *prop-5231*(*2*)] **show** $\vdash (F_o \vee^{\mathcal{Q}} F_o) =_o F_o$ **.**
qed

end

6.34 Proposition 5233

context begin

private lemma *lem-prop-5233-no-free-vars*:
assumes $A \in$ *pwffs* **and** *free-vars* $A = \{\}$
shows $(\forall \varphi.$ *is-tv-assignment* $\varphi \longrightarrow \mathcal{V}_B\ \varphi\ A = \mathbf{T}) \longrightarrow \vdash A =_o T_o$ (**is** *?*$A_T \longrightarrow$ -)
and $(\forall \varphi.$ *is-tv-assignment* $\varphi \longrightarrow \mathcal{V}_B\ \varphi\ A = \mathbf{F}) \longrightarrow \vdash A =_o F_o$ (**is** *?*$A_F \longrightarrow$ -)
proof −
from *assms* **have** (*?*$A_T \longrightarrow \vdash A =_o T_o) \wedge$ (*?*$A_F \longrightarrow \vdash A =_o F_o$)
proof *induction*
case *T-pwff*
have $\vdash T_o =_o T_o$
by (*rule prop-5200*[*OF true-wff*])
moreover have $\forall \varphi.$ *is-tv-assignment* $\varphi \longrightarrow \mathcal{V}_B\ \varphi\ T_o = \mathbf{T}$
using $\mathcal{V}_B$*-T* **by** *blast*
then have $\neg\ (\forall \varphi.$ *is-tv-assignment* $\varphi \longrightarrow \mathcal{V}_B\ \varphi\ T_o = \mathbf{F})$
by (*auto simp*: *inj-eq*)
ultimately show *?case*
by *blast*
next
case *F-pwff*
have $\vdash F_o =_o F_o$
by (*rule prop-5200*[*OF false-wff*])
moreover have $\forall \varphi.$ *is-tv-assignment* $\varphi \longrightarrow \mathcal{V}_B\ \varphi\ F_o = \mathbf{F}$
using $\mathcal{V}_B$*-F* **by** *blast*
then have $\neg\ (\forall \varphi.$ *is-tv-assignment* $\varphi \longrightarrow \mathcal{V}_B\ \varphi\ F_o = \mathbf{T})$
by (*auto simp*: *inj-eq*)
ultimately show *?case*
by *blast*
next
case (*var-pwff p*) — impossible case
then show *?case*
by *simp*
next
case (*neg-pwff B*)
from *neg-pwff.hyps* **have** $\sim^{\mathcal{Q}} B \in$ *pwffs* **and** *free-vars* $B = \{\}$
using *neg-pwff.prems* **by** (*force*, *auto elim*: *free-vars-form.elims*)
consider
(*a*) $\forall \varphi.$ *is-tv-assignment* $\varphi \longrightarrow \mathcal{V}_B\ \varphi\ B = \mathbf{T}$
| (*b*) $\forall \varphi.$ *is-tv-assignment* $\varphi \longrightarrow \mathcal{V}_B\ \varphi\ B = \mathbf{F}$

```
    using closed-pwff-denotation-uniqueness[OF neg-pwff.hyps ‹free-vars B = {}›]
    and neg-pwff.hyps[THEN 𝒱_B-graph-denotation-is-truth-value[OF 𝒱_B-graph-𝒱_B]]
    by (auto dest: tv-cases) metis
  then show ?case
  proof cases
    case a
    with ‹free-vars B = {}› have ⊢ T_o =_o B
      using neg-pwff.IH and Equality-Rules(2) by blast
    from prop-5231(1)[unfolded neg-def, folded equality-of-type-def] and this
    have ⊢ ∼𝒬 B =_o F_o
      unfolding neg-def[folded equality-of-type-def] by (rule rule-R[where p = [«,»,»]]) force+
    moreover from a have ∀φ. is-tv-assignment φ ⟶ 𝒱_B φ (∼𝒬 B) = F
      using 𝒱_B-neg[OF neg-pwff.hyps] by simp
    ultimately show ?thesis
      by (auto simp: inj-eq)
  next
    case b
    with ‹free-vars B = {}› have ⊢ F_o =_o B
      using neg-pwff.IH and Equality-Rules(2) by blast
    then have ⊢ ∼𝒬 B =_o T_o
      unfolding neg-def[folded equality-of-type-def]
      using rule-T(2)[OF hyp-derivable-form-is-wffso] by blast
    moreover from b have ∀φ. is-tv-assignment φ ⟶ 𝒱_B φ (∼𝒬 B) = T
      using 𝒱_B-neg[OF neg-pwff.hyps] by simp
    ultimately show ?thesis
      by (auto simp: inj-eq)
  qed
next
  case (conj-pwff B C)
  from conj-pwff.prems have free-vars B = {} and free-vars C = {}
    by simp-all
  with conj-pwff.hyps obtain b and b'
    where B-den: ∀φ. is-tv-assignment φ ⟶ 𝒱_B φ B = b
    and C-den: ∀φ. is-tv-assignment φ ⟶ 𝒱_B φ C = b'
    using closed-pwff-denotation-uniqueness by metis
  then have b ∈ elts 𝔹 and b' ∈ elts 𝔹
    using closed-pwff-denotation-uniqueness[OF conj-pwff.hyps(1) ‹free-vars B = {}›]
    and closed-pwff-denotation-uniqueness[OF conj-pwff.hyps(2) ‹free-vars C = {}›]
    and conj-pwff.hyps[THEN 𝒱_B-graph-denotation-is-truth-value[OF 𝒱_B-graph-𝒱_B]]
    by force+
  with conj-pwff.hyps consider
    (a) b = T and b' = T
  | (b) b = T and b' = F
  | (c) b = F and b' = T
  | (d) b = F and b' = F
    by auto
  then show ?case
  proof cases
    case a
```

from *prop-5229(1)* **have** $\vdash T_o \wedge^{\mathcal{Q}} T_o =_o T_o$ (**is** ‹$\vdash$ *?A1*›) **.**
from *B-den*[*unfolded a(1)*] **and** ‹*free-vars* $B = \{\}$› **have** $\vdash B =_o T_o$
using *conj-pwff.IH(1)* **by** *simp*
then have $\vdash B \wedge^{\mathcal{Q}} T_o =_o T_o$ (**is** ‹$\vdash$ *?A2*›)
by (*rule rule-RR*[*OF disjI2*, **where** $p = [«,»,«,»]$ **and** C = *?A1*]) (*use* ‹$\vdash$ *?A1*› **in** ‹*force*+›)
from *C-den*[*unfolded a(2)*] **and** ‹*free-vars* $C = \{\}$› **have** $\vdash C =_o T_o$
using *conj-pwff.IH(2)* **by** *simp*
then have $\vdash B \wedge^{\mathcal{Q}} C =_o T_o$
by (*rule rule-RR*[*OF disjI2*, **where** $p = [«,»,»]$ **and** C = *?A2*]) (*use* ‹$\vdash$ *?A2*› **in** ‹*force*+›)
then have $(\forall \varphi.$ *is-tv-assignment* $\varphi \longrightarrow \mathcal{V}_B\ \varphi\ (B \wedge^{\mathcal{Q}} C) = \mathbf{T}) \longrightarrow \vdash B \wedge^{\mathcal{Q}} C =_o T_o$
by *blast*
moreover have $\forall \varphi.$ *is-tv-assignment* $\varphi \longrightarrow \mathcal{V}_B\ \varphi\ (B \wedge^{\mathcal{Q}} C) \neq \mathbf{F}$
using $\mathcal{V}_B$*-conj*[*OF conj-pwff.hyps*] **and** *B-den*[*unfolded a(1)*] **and** *C-den*[*unfolded a(2)*]
by (*auto simp*: *inj-eq*)
ultimately show *?thesis*
by *force*
next
case *b*
from *prop-5229(2)* **have** $\vdash T_o \wedge^{\mathcal{Q}} F_o =_o F_o$ (**is** ‹$\vdash$ *?A1*›) **.**
from *B-den*[*unfolded b(1)*] **and** ‹*free-vars* $B = \{\}$› **have** $\vdash B =_o T_o$
using *conj-pwff.IH(1)* **by** *simp*
then have $\vdash B \wedge^{\mathcal{Q}} F_o =_o F_o$ (**is** ‹$\vdash$ *?A2*›)
by (*rule rule-RR*[*OF disjI2*, **where** $p = [«,»,«,»]$ **and** C = *?A1*]) (*use* ‹$\vdash$ *?A1*› **in** ‹*force*+›)
from *C-den*[*unfolded b(2)*] **and** ‹*free-vars* $C = \{\}$› **have** $\vdash C =_o F_o$
using *conj-pwff.IH(2)* **by** *simp*
then have $\vdash B \wedge^{\mathcal{Q}} C =_o F_o$
by (*rule rule-RR*[*OF disjI2*, **where** $p = [«,»,»]$ **and** C = *?A2*]) (*use* ‹$\vdash$ *?A2*› **in** ‹*force*+›)
then have $(\forall \varphi.$ *is-tv-assignment* $\varphi \longrightarrow \mathcal{V}_B\ \varphi\ (B \wedge^{\mathcal{Q}} C) = \mathbf{F}) \longrightarrow \vdash B \wedge^{\mathcal{Q}} C =_o F_o$
by *blast*
moreover have $\forall \varphi.$ *is-tv-assignment* $\varphi \longrightarrow \mathcal{V}_B\ \varphi\ (B \wedge^{\mathcal{Q}} C) \neq \mathbf{T}$
using $\mathcal{V}_B$*-conj*[*OF conj-pwff.hyps*] **and** *B-den*[*unfolded b(1)*] **and** *C-den*[*unfolded b(2)*]
by (*auto simp*: *inj-eq*)
ultimately show *?thesis*
by *force*
next
case *c*
from *prop-5229(3)* **have** $\vdash F_o \wedge^{\mathcal{Q}} T_o =_o F_o$ (**is** ‹$\vdash$ *?A1*›) **.**
from *B-den*[*unfolded c(1)*] **and** ‹*free-vars* $B = \{\}$› **have** $\vdash B =_o F_o$
using *conj-pwff.IH(1)* **by** *simp*
then have $\vdash B \wedge^{\mathcal{Q}} T_o =_o F_o$ (**is** ‹$\vdash$ *?A2*›)
by (*rule rule-RR*[*OF disjI2*, **where** $p = [«,»,«,»]$ **and** C = *?A1*]) (*use* ‹$\vdash$ *?A1*› **in** ‹*force*+›)
from *C-den*[*unfolded c(2)*] **and** ‹*free-vars* $C = \{\}$› **have** $\vdash C =_o T_o$
using *conj-pwff.IH(2)* **by** *simp*
then have $\vdash B \wedge^{\mathcal{Q}} C =_o F_o$
by (*rule rule-RR*[*OF disjI2*, **where** $p = [«,»,»]$ **and** C = *?A2*]) (*use* ‹$\vdash$ *?A2*› **in** ‹*force*+›)
then have $(\forall \varphi.$ *is-tv-assignment* $\varphi \longrightarrow \mathcal{V}_B\ \varphi\ (B \wedge^{\mathcal{Q}} C) = \mathbf{F}) \longrightarrow \vdash B \wedge^{\mathcal{Q}} C =_o F_o$
by *blast*
moreover have $\forall \varphi.$ *is-tv-assignment* $\varphi \longrightarrow \mathcal{V}_B\ \varphi\ (B \wedge^{\mathcal{Q}} C) \neq \mathbf{T}$
using $\mathcal{V}_B$*-conj*[*OF conj-pwff.hyps*] **and** *B-den*[*unfolded c(1)*] **and** *C-den*[*unfolded c(2)*]

```
    by (auto simp: inj-eq)
  ultimately show ?thesis
    by force
next
  case d
  from prop-5229(4) have ⊢ F_o ∧^Ω F_o =_o F_o (is ‹⊢ ?A1›) .
  from B-den[unfolded d(1)] and ‹free-vars B = {}› have ⊢ B =_o F_o
    using conj-pwff.IH(1) by simp
  then have ⊢ B ∧^Ω F_o =_o F_o  (is ‹⊢ ?A2›)
    by (rule rule-RR[OF disjI2, where p = [«,»,«,»] and C = ?A1]) (use ‹⊢ ?A1› in ‹force+›)
  from C-den[unfolded d(2)] and ‹free-vars C = {}› have ⊢ C =_o F_o
    using conj-pwff.IH(2) by simp
  then have ⊢ B ∧^Ω C =_o F_o
    by (rule rule-RR[OF disjI2, where p = [«,»,»] and C = ?A2]) (use ‹⊢ ?A2› in ‹force+›)
  then have (∀φ. is-tv-assignment φ ⟶ V_B φ (B ∧^Ω C) = F) ⟶ ⊢ B ∧^Ω C =_o F_o
    by blast
  moreover have ∀φ. is-tv-assignment φ ⟶ V_B φ (B ∧^Ω C) ≠ T
    using V_B-conj[OF conj-pwff.hyps] and B-den[unfolded d(1)] and C-den[unfolded d(2)]
    by (auto simp: inj-eq)
  ultimately show ?thesis
    by force
qed
next
  case (disj-pwff B C)
  from disj-pwff.prems have free-vars B = {} and free-vars C = {}
    by simp-all
  with disj-pwff.hyps obtain b and b'
    where B-den: ∀φ. is-tv-assignment φ ⟶ V_B φ B = b
    and C-den: ∀φ. is-tv-assignment φ ⟶ V_B φ C = b'
    using closed-pwff-denotation-uniqueness by metis
  then have b ∈ elts 𝔹 and b' ∈ elts 𝔹
    using closed-pwff-denotation-uniqueness[OF disj-pwff.hyps(1) ‹free-vars B = {}›]
    and closed-pwff-denotation-uniqueness[OF disj-pwff.hyps(2) ‹free-vars C = {}›]
    and disj-pwff.hyps[THEN V_B-graph-denotation-is-truth-value[OF V_B-graph-V_B]]
    by force+
  with disj-pwff.hyps consider
    (a) b = T and b' = T
  | (b) b = T and b' = F
  | (c) b = F and b' = T
  | (d) b = F and b' = F
    by auto
  then show ?case
  proof cases
    case a
    from prop-5232(1) have ⊢ T_o ∨^Ω T_o =_o T_o (is ‹⊢ ?A1›) .
    from B-den[unfolded a(1)] and ‹free-vars B = {}› have ⊢ B =_o T_o
      using disj-pwff.IH(1) by simp
    then have ⊢ B ∨^Ω T_o =_o T_o (is ‹⊢ ?A2›)
      by (rule rule-RR[OF disjI2, where p = [«,»,«,»] and C = ?A1]) (use ‹⊢ ?A1› in ‹force+›)
```

from *C-den*[*unfolded a*(*2*)] **and** ‹*free-vars* $C = \{\}$› **have** $\vdash C =_o T_o$
using *disj-pwff.IH*(*2*) **by** *simp*
then have $\vdash B \vee^{\mathfrak{Q}} C =_o T_o$
by (*rule rule-RR*[*OF disjI2*, **where** $p = [«,»,»]$ **and** $C = ?A2$]) (*use* ‹$\vdash ?A2$› **in** ‹*force*+›)
then have $(\forall \varphi.\ \textit{is-tv-assignment}\ \varphi \longrightarrow \mathcal{V}_B\ \varphi\ (B \vee^{\mathfrak{Q}} C) = \mathbf{T}) \longrightarrow \vdash B \vee^{\mathfrak{Q}} C =_o T_o$
by *blast*
moreover have $\forall \varphi.\ \textit{is-tv-assignment}\ \varphi \longrightarrow \mathcal{V}_B\ \varphi\ (B \vee^{\mathfrak{Q}} C) \neq \mathbf{F}$
using $\mathcal{V}_B$*-disj*[*OF disj-pwff.hyps*] **and** *B-den*[*unfolded a*(*1*)] **and** *C-den*[*unfolded a*(*2*)]
by (*auto simp*: *inj-eq*)
ultimately show *?thesis*
by *force*
next
case *b*
from *prop-5232*(*2*) **have** $\vdash T_o \vee^{\mathfrak{Q}} F_o =_o T_o$ (**is** ‹$\vdash ?A1$›) **.**
from *B-den*[*unfolded b*(*1*)] **and** ‹*free-vars* $B = \{\}$› **have** $\vdash B =_o T_o$
using *disj-pwff.IH*(*1*) **by** *simp*
then have $\vdash B \vee^{\mathfrak{Q}} F_o =_o T_o$ (**is** ‹$\vdash ?A2$›)
by (*rule rule-RR*[*OF disjI2*, **where** $p = [«,»,«,»]$ **and** $C = ?A1$]) (*use* ‹$\vdash ?A1$› **in** ‹*force*+›)
from *C-den*[*unfolded b*(*2*)] **and** ‹*free-vars* $C = \{\}$› **have** $\vdash C =_o F_o$
using *disj-pwff.IH*(*2*) **by** *simp*
then have $\vdash B \vee^{\mathfrak{Q}} C =_o T_o$
by (*rule rule-RR*[*OF disjI2*, **where** $p = [«,»,»]$ **and** $C = ?A2$]) (*use* ‹$\vdash ?A2$› **in** ‹*force*+›)
then have $(\forall \varphi.\ \textit{is-tv-assignment}\ \varphi \longrightarrow \mathcal{V}_B\ \varphi\ (B \vee^{\mathfrak{Q}} C) = \mathbf{T}) \longrightarrow \vdash B \vee^{\mathfrak{Q}} C =_o T_o$
by *blast*
moreover have $\forall \varphi.\ \textit{is-tv-assignment}\ \varphi \longrightarrow \mathcal{V}_B\ \varphi\ (B \vee^{\mathfrak{Q}} C) \neq \mathbf{F}$
using $\mathcal{V}_B$*-disj*[*OF disj-pwff.hyps*] **and** *B-den*[*unfolded b*(*1*)] **and** *C-den*[*unfolded b*(*2*)]
by (*auto simp*: *inj-eq*)
ultimately show *?thesis*
by *force*
next
case *c*
from *prop-5232*(*3*) **have** $\vdash F_o \vee^{\mathfrak{Q}} T_o =_o T_o$ (**is** ‹$\vdash ?A1$›) **.**
from *B-den*[*unfolded c*(*1*)] **and** ‹*free-vars* $B = \{\}$› **have** $\vdash B =_o F_o$
using *disj-pwff.IH*(*1*) **by** *simp*
then have $\vdash B \vee^{\mathfrak{Q}} T_o =_o T_o$ (**is** ‹$\vdash ?A2$›)
by (*rule rule-RR*[*OF disjI2*, **where** $p = [«,»,«,»]$ **and** $C = ?A1$]) (*use* ‹$\vdash ?A1$› **in** ‹*force*+›)
from *C-den*[*unfolded c*(*2*)] **and** ‹*free-vars* $C = \{\}$› **have** $\vdash C =_o T_o$
using *disj-pwff.IH*(*2*) **by** *simp*
then have $\vdash B \vee^{\mathfrak{Q}} C =_o T_o$
by (*rule rule-RR*[*OF disjI2*, **where** $p = [«,»,»]$ **and** $C = ?A2$]) (*use* ‹$\vdash ?A2$› **in** ‹*force*+›)
then have $(\forall \varphi.\ \textit{is-tv-assignment}\ \varphi \longrightarrow \mathcal{V}_B\ \varphi\ (B \vee^{\mathfrak{Q}} C) = \mathbf{T}) \longrightarrow \vdash B \vee^{\mathfrak{Q}} C =_o T_o$
by *blast*
moreover have $\forall \varphi.\ \textit{is-tv-assignment}\ \varphi \longrightarrow \mathcal{V}_B\ \varphi\ (B \vee^{\mathfrak{Q}} C) \neq \mathbf{F}$
using $\mathcal{V}_B$*-disj*[*OF disj-pwff.hyps*] **and** *B-den*[*unfolded c*(*1*)] **and** *C-den*[*unfolded c*(*2*)]
by (*auto simp*: *inj-eq*)
ultimately show *?thesis*
by *force*
next
case *d*

```
    from prop-5232(4) have ⊢ F_o ∨^Ω F_o =_o F_o (is ‹⊢ ?A1›) .
    from B-den[unfolded d(1)] and ‹free-vars B = {}› have ⊢ B =_o F_o
      using disj-pwff.IH(1) by simp
    then have ⊢ B ∨^Ω F_o =_o F_o (is ‹⊢ ?A2›)
      by (rule rule-RR[OF disjI2, where p = [«,»,«,»] and C = ?A1]) (use ‹⊢ ?A1› in ‹force+›)
    from C-den[unfolded d(2)] and ‹free-vars C = {}› have ⊢ C =_o F_o
      using disj-pwff.IH(2) by simp
    then have ⊢ B ∨^Ω C =_o F_o
      by (rule rule-RR[OF disjI2, where p = [«,»,»] and C = ?A2]) (use ‹⊢ ?A2› in ‹force+›)
    then have (∀φ. is-tv-assignment φ ⟶ V_B φ (B ∨^Ω C) = T) ⟶ ⊢ B ∨^Ω C =_o F_o
      by blast
    moreover have ∀φ. is-tv-assignment φ ⟶ V_B φ (B ∨^Ω C) ≠ T
      using V_B-disj[OF disj-pwff.hyps] and B-den[unfolded d(1)] and C-den[unfolded d(2)]
      by (auto simp: inj-eq)
    ultimately show ?thesis
      using ‹⊢ B ∨^Ω C =_o F_o› by auto
  qed
next
  case (imp-pwff B C)
  from imp-pwff.prems have free-vars B = {} and free-vars C = {}
    by simp-all
  with imp-pwff.hyps obtain b and b'
    where B-den: ∀φ. is-tv-assignment φ ⟶ V_B φ B = b
    and C-den: ∀φ. is-tv-assignment φ ⟶ V_B φ C = b'
    using closed-pwff-denotation-uniqueness by metis
  then have b ∈ elts 𝔹 and b' ∈ elts 𝔹
    using closed-pwff-denotation-uniqueness[OF imp-pwff.hyps(1) ‹free-vars B = {}›]
    and closed-pwff-denotation-uniqueness[OF imp-pwff.hyps(2) ‹free-vars C = {}›]
    and imp-pwff.hyps[THEN V_B-graph-denotation-is-truth-value[OF V_B-graph-V_B]]
    by force+
  with imp-pwff.hyps consider
    (a) b = T and b' = T
  | (b) b = T and b' = F
  | (c) b = F and b' = T
  | (d) b = F and b' = F
    by auto
  then show ?case
  proof cases
    case a
    from prop-5228(1) have ⊢ T_o ⊃^Ω T_o =_o T_o (is ‹⊢ ?A1›) .
    from B-den[unfolded a(1)] and ‹free-vars B = {}› have ⊢ B =_o T_o
      using imp-pwff.IH(1) by simp
    then have ⊢ B ⊃^Ω T_o =_o T_o (is ‹⊢ ?A2›)
      by (rule rule-RR[OF disjI2, where p = [«,»,«,»] and C = ?A1]) (use ‹⊢ ?A1› in ‹force+›)
    from C-den[unfolded a(2)] and ‹free-vars C = {}› have ⊢ C =_o T_o
      using imp-pwff.IH(2) by simp
    then have ⊢ B ⊃^Ω C =_o T_o
      by (rule rule-RR[OF disjI2, where p = [«,»,»] and C = ?A2]) (use ‹⊢ ?A2› in ‹force+›)
    then have (∀φ. is-tv-assignment φ ⟶ V_B φ (B ⊃^Ω C) = T) ⟶ ⊢ B ⊃^Ω C =_o T_o
```

by *blast*
moreover have $\forall \varphi$. *is-tv-assignment* $\varphi \longrightarrow \mathcal{V}_B\ \varphi\ (B \supset^{\mathcal{Q}} C) \neq \mathbf{F}$
using $\mathcal{V}_B$*-imp*[*OF imp-pwff.hyps*] **and** *B-den*[*unfolded a*(*1*)] **and** *C-den*[*unfolded a*(*2*)]
by (*auto simp*: *inj-eq*)
ultimately show *?thesis*
by *force*
next
case *b*
from *prop-5228*(*2*) **have** $\vdash T_o \supset^{\mathcal{Q}} F_o =_o F_o$ (**is** ‹$\vdash$ *?A1*›) **.**
from *B-den*[*unfolded b*(*1*)] **and** ‹*free-vars* $B = \{\}$› **have** $\vdash B =_o T_o$
using *imp-pwff.IH*(*1*) **by** *simp*
then have $\vdash B \supset^{\mathcal{Q}} F_o =_o F_o$ (**is** ‹$\vdash$ *?A2*›)
by (*rule rule-RR*[*OF disjI2*, **where** $p = [«,»,«,»]$ **and** C = *?A1*]) (*use* ‹$\vdash$ *?A1*› **in** ‹*force*+›)
from *C-den*[*unfolded b*(*2*)] **and** ‹*free-vars* $C = \{\}$› **have** $\vdash C =_o F_o$
using *imp-pwff.IH*(*2*) **by** *simp*
then have $\vdash B \supset^{\mathcal{Q}} C =_o F_o$
by (*rule rule-RR*[*OF disjI2*, **where** $p = [«,»,»]$ **and** C = *?A2*]) (*use* ‹$\vdash$ *?A2*› **in** ‹*force*+›)
then have $(\forall \varphi$. *is-tv-assignment* $\varphi \longrightarrow \mathcal{V}_B\ \varphi\ (B \supset^{\mathcal{Q}} C) = \mathbf{F}) \longrightarrow \vdash B \supset^{\mathcal{Q}} C =_o F_o$
by *blast*
moreover have $\forall \varphi$. *is-tv-assignment* $\varphi \longrightarrow \mathcal{V}_B\ \varphi\ (B \supset^{\mathcal{Q}} C) \neq \mathbf{T}$
using $\mathcal{V}_B$*-imp*[*OF imp-pwff.hyps*] **and** *B-den*[*unfolded b*(*1*)] **and** *C-den*[*unfolded b*(*2*)]
by (*auto simp*: *inj-eq*)
ultimately show *?thesis*
by *force*
next
case *c*
from *prop-5228*(*3*) **have** $\vdash F_o \supset^{\mathcal{Q}} T_o =_o T_o$ (**is** ‹$\vdash$ *?A1*›) **.**
from *B-den*[*unfolded c*(*1*)] **and** ‹*free-vars* $B = \{\}$› **have** $\vdash B =_o F_o$
using *imp-pwff.IH*(*1*) **by** *simp*
then have $\vdash B \supset^{\mathcal{Q}} T_o =_o T_o$ (**is** ‹$\vdash$ *?A2*›)
by (*rule rule-RR*[*OF disjI2*, **where** $p = [«,»,«,»]$ **and** C = *?A1*]) (*use* ‹$\vdash$ *?A1*› **in** ‹*force*+›)
from *C-den*[*unfolded c*(*2*)] **and** ‹*free-vars* $C = \{\}$› **have** $\vdash C =_o T_o$
using *imp-pwff.IH*(*2*) **by** *simp*
then have $\vdash B \supset^{\mathcal{Q}} C =_o T_o$
by (*rule rule-RR*[*OF disjI2*, **where** $p = [«,»,»]$ **and** C = *?A2*]) (*use* ‹$\vdash$ *?A2*› **in** ‹*force*+›)
then have $(\forall \varphi$. *is-tv-assignment* $\varphi \longrightarrow \mathcal{V}_B\ \varphi\ (B \supset^{\mathcal{Q}} C) = \mathbf{T}) \longrightarrow \vdash B \supset^{\mathcal{Q}} C =_o T_o$
by *blast*
moreover have $\forall \varphi$. *is-tv-assignment* $\varphi \longrightarrow \mathcal{V}_B\ \varphi\ (B \supset^{\mathcal{Q}} C) \neq \mathbf{F}$
using $\mathcal{V}_B$*-imp*[*OF imp-pwff.hyps*] **and** *B-den*[*unfolded c*(*1*)] **and** *C-den*[*unfolded c*(*2*)]
by (*auto simp*: *inj-eq*)
ultimately show *?thesis*
by *force*
next
case *d*
from *prop-5228*(*4*) **have** $\vdash F_o \supset^{\mathcal{Q}} F_o =_o T_o$ (**is** ‹$\vdash$ *?A1*›) **.**
from *B-den*[*unfolded d*(*1*)] **and** ‹*free-vars* $B = \{\}$› **have** $\vdash B =_o F_o$
using *imp-pwff.IH*(*1*) **by** *simp*
then have $\vdash B \supset^{\mathcal{Q}} F_o =_o T_o$ (**is** ‹$\vdash$ *?A2*›)
by (*rule rule-RR*[*OF disjI2*, **where** $p = [«,»,«,»]$ **and** C = *?A1*]) (*use* ‹$\vdash$ *?A1*› **in** ‹*force*+›)

```
    from C-den[unfolded d(2)] and ‹free-vars C = {}› have ⊢ C =o Fo
      using imp-pwff.IH(2) by simp
    then have ⊢ B ⊃Q C =o To
      by (rule rule-RR[OF disjI2, where p = [«,»,»] and C = ?A2]) (use ‹⊢ ?A2› in ‹force+›)
    then have (∀φ. is-tv-assignment φ ⟶ 𝒱B φ (B ⊃Q C) = T) ⟶ ⊢ B ⊃Q C =o To
      by blast
    moreover have ∀φ. is-tv-assignment φ ⟶ 𝒱B φ (B ⊃Q C) ≠ F
      using 𝒱B-imp[OF imp-pwff.hyps] and B-den[unfolded d(1)] and C-den[unfolded d(2)]
      by (auto simp: inj-eq)
    ultimately show ?thesis
      by force
  qed
next
  case (eqv-pwff B C)
  from eqv-pwff.prems have free-vars B = {} and free-vars C = {}
    by simp-all
  with eqv-pwff.hyps obtain b and b'
    where B-den: ∀φ. is-tv-assignment φ ⟶ 𝒱B φ B = b
    and C-den: ∀φ. is-tv-assignment φ ⟶ 𝒱B φ C = b'
    using closed-pwff-denotation-uniqueness by metis
  then have b ∈ elts 𝔹 and b' ∈ elts 𝔹
    using closed-pwff-denotation-uniqueness[OF eqv-pwff.hyps(1) ‹free-vars B = {}›]
    and closed-pwff-denotation-uniqueness[OF eqv-pwff.hyps(2) ‹free-vars C = {}›]
    and eqv-pwff.hyps[THEN 𝒱B-graph-denotation-is-truth-value[OF 𝒱B-graph-𝒱B]]
    by force+
  with eqv-pwff.hyps consider
    (a) b = T and b' = T
  | (b) b = T and b' = F
  | (c) b = F and b' = T
  | (d) b = F and b' = F
    by auto
  then show ?case
  proof cases
    case a
    from prop-5230(1) have ⊢ (To ≡Q To) =o To (is ‹⊢ ?A1›) .
    from B-den[unfolded a(1)] and ‹free-vars B = {}› have ⊢ B =o To
      using eqv-pwff.IH(1) by simp
    then have ⊢ (B ≡Q To) =o To (is ‹⊢ ?A2›)
      by (rule rule-RR[OF disjI2, where p = [«,»,«,»] and C = ?A1]) (use ‹⊢ ?A1› in ‹force+›)
    from C-den[unfolded a(2)] and ‹free-vars C = {}› have ⊢ C =o To
      using eqv-pwff.IH(2) by simp
    then have ⊢ (B ≡Q C) =o To
      by (rule rule-RR[OF disjI2, where p = [«,»,»] and C = ?A2]) (use ‹⊢ ?A2› in ‹force+›)
    then have (∀φ. is-tv-assignment φ ⟶ 𝒱B φ (B ≡Q C) = T) ⟶ ⊢ (B ≡Q C) =o To
      by blast
    moreover have ∀φ. is-tv-assignment φ ⟶ 𝒱B φ (B ≡Q C) ≠ F
      using 𝒱B-eqv[OF eqv-pwff.hyps] and B-den[unfolded a(1)] and C-den[unfolded a(2)]
      by (auto simp: inj-eq)
    ultimately show ?thesis
```

by *force*
next
case *b*
from *prop-5230(2)* **have** $\vdash (T_o \equiv^{\mathcal{Q}} F_o) =_o F_o$ (**is** ‹$\vdash$ *?A1*›) .
from *B-den*[*unfolded b(1)*] **and** ‹*free-vars* $B = \{\}$› **have** $\vdash B =_o T_o$
using *eqv-pwff.IH(1)* **by** *simp*
then have $\vdash (B \equiv^{\mathcal{Q}} F_o) =_o F_o$ (**is** ‹$\vdash$ *?A2*›)
by (*rule rule-RR*[*OF disjI2*, **where** $p = [«,»,«,»]$ **and** C = *?A1*]) (*use* ‹$\vdash$ *?A1*› **in** ‹*force*+›)
from *C-den*[*unfolded b(2)*] **and** ‹*free-vars* $C = \{\}$› **have** $\vdash C =_o F_o$
using *eqv-pwff.IH(2)* **by** *simp*
then have $\vdash (B \equiv^{\mathcal{Q}} C) =_o F_o$
by (*rule rule-RR*[*OF disjI2*, **where** $p = [«,»,»]$ **and** C = *?A2*]) (*use* ‹$\vdash$ *?A2*› **in** ‹*force*+›)
then have $(\forall \varphi.$ *is-tv-assignment* $\varphi \longrightarrow \mathcal{V}_B\ \varphi\ (B \equiv^{\mathcal{Q}} C) = \mathbf{F}) \longrightarrow \vdash (B \equiv^{\mathcal{Q}} C) =_o F_o$
by *blast*
moreover have $\forall \varphi.$ *is-tv-assignment* $\varphi \longrightarrow \mathcal{V}_B\ \varphi\ (B \equiv^{\mathcal{Q}} C) \neq \mathbf{T}$
using $\mathcal{V}_B$*-eqv*[*OF eqv-pwff.hyps*] **and** *B-den*[*unfolded b(1)*] **and** *C-den*[*unfolded b(2)*]
by (*auto simp*: *inj-eq*)
ultimately show *?thesis*
by *force*
next
case *c*
from *prop-5230(3)* **have** $\vdash (F_o \equiv^{\mathcal{Q}} T_o) =_o F_o$ (**is** ‹$\vdash$ *?A1*›) .
from *B-den*[*unfolded c(1)*] **and** ‹*free-vars* $B = \{\}$› **have** $\vdash B =_o F_o$
using *eqv-pwff.IH(1)* **by** *simp*
then have $\vdash (B \equiv^{\mathcal{Q}} T_o) =_o F_o$ (**is** ‹$\vdash$ *?A2*›)
by (*rule rule-RR*[*OF disjI2*, **where** $p = [«,»,«,»]$ **and** C = *?A1*]) (*use* ‹$\vdash$ *?A1*› **in** ‹*force*+›)
from *C-den*[*unfolded c(2)*] **and** ‹*free-vars* $C = \{\}$› **have** $\vdash C =_o T_o$
using *eqv-pwff.IH(2)* **by** *simp*
then have $\vdash (B \equiv^{\mathcal{Q}} C) =_o F_o$
by (*rule rule-RR*[*OF disjI2*, **where** $p = [«,»,»]$ **and** C = *?A2*]) (*use* ‹$\vdash$ *?A2*› **in** ‹*force*+›)
then have $(\forall \varphi.$ *is-tv-assignment* $\varphi \longrightarrow \mathcal{V}_B\ \varphi\ (B \equiv^{\mathcal{Q}} C) = \mathbf{F}) \longrightarrow \vdash (B \equiv^{\mathcal{Q}} C) =_o F_o$
by *blast*
moreover have $\forall \varphi.$ *is-tv-assignment* $\varphi \longrightarrow \mathcal{V}_B\ \varphi\ (B \equiv^{\mathcal{Q}} C) \neq \mathbf{T}$
using $\mathcal{V}_B$*-eqv*[*OF eqv-pwff.hyps*] **and** *B-den*[*unfolded c(1)*] **and** *C-den*[*unfolded c(2)*]
by (*auto simp*: *inj-eq*)
ultimately show *?thesis*
by *force*
next
case *d*
from *prop-5230(4)* **have** $\vdash (F_o \equiv^{\mathcal{Q}} F_o) =_o T_o$ (**is** ‹$\vdash$ *?A1*›) .
from *B-den*[*unfolded d(1)*] **and** ‹*free-vars* $B = \{\}$› **have** $\vdash B =_o F_o$
using *eqv-pwff.IH(1)* **by** *simp*
then have $\vdash (B \equiv^{\mathcal{Q}} F_o) =_o T_o$ (**is** ‹$\vdash$ *?A2*›)
by (*rule rule-RR*[*OF disjI2*, **where** $p = [«,»,«,»]$ **and** C = *?A1*]) (*use* ‹$\vdash$ *?A1*› **in** ‹*force*+›)
from *C-den*[*unfolded d(2)*] **and** ‹*free-vars* $C = \{\}$› **have** $\vdash C =_o F_o$
using *eqv-pwff.IH(2)* **by** *simp*
then have $\vdash (B \equiv^{\mathcal{Q}} C) =_o T_o$
by (*rule rule-RR*[*OF disjI2*, **where** $p = [«,»,»]$ **and** C = *?A2*]) (*use* ‹$\vdash$ *?A2*› **in** ‹*force*+›)
then have $(\forall \varphi.$ *is-tv-assignment* $\varphi \longrightarrow \mathcal{V}_B\ \varphi\ (B \equiv^{\mathcal{Q}} C) = \mathbf{T}) \longrightarrow \vdash (B \equiv^{\mathcal{Q}} C) =_o T_o$

```
      by blast
    moreover have ∀φ. is-tv-assignment φ ⟶ 𝒱_B φ (B ≡^Ω C) ≠ F
      using 𝒱_B-eqv[OF eqv-pwff.hyps] and B-den[unfolded d(1)] and C-den[unfolded d(2)]
      by (auto simp: inj-eq)
    ultimately show ?thesis
      by force
  qed
 qed
 then show ?A_T ⟶ ⊢ A =_o T_o and ?A_F ⟶ ⊢ A =_o F_o
  by blast+
qed

proposition prop-5233:
 assumes is-tautology A
 shows ⊢ A
proof −
 have finite (free-vars A)
  using free-vars-form-finiteness by presburger
 from this and assms show ?thesis
 proof (induction free-vars A arbitrary: A)
  case empty
  from empty(2) have A ∈ pwffs and ∀φ. is-tv-assignment φ ⟶ 𝒱_B φ A = T
   unfolding is-tautology-def by blast+
  with empty(1) have ⊢ A =_o T_o
   using lem-prop-5233-no-free-vars(1) by (simp only:)
  then show ?case
   using rule-T(2)[OF tautology-is-wffo[OF empty(2)]] by (simp only:)
 next
  case (insert v F)
  from insert.prems have A ∈ pwffs
   by blast
  with insert.hyps(4) obtain p where v = (p, o)
   using pwffs-free-vars-are-propositional by blast
  from ‹v = (p, o)› and insert.hyps(4) have
   is-tautology (S {(p, o) ⇀ T_o} A) and is-tautology (S {(p, o) ⇀ F_o} A)
   using pwff-substitution-tautology-preservation [OF insert.prems] by blast+
  moreover from insert.hyps(2,4) and ‹v = (p, o)› and ‹A ∈ pwffs›
  have free-vars (S {(p, o) ⇀ T_o} A) = F and free-vars (S {(p, o) ⇀ F_o} A) = F
   using closed-pwff-substitution-free-vars and T-pwff and F-pwff and T-fv and F-fv
   by (metis Diff-insert-absorb insertI1)+
  ultimately have ⊢ S {(p, o) ⇀ T_o} A and ⊢ S {(p, o) ⇀ F_o} A
   using insert.hyps(3) by (simp-all only:)
  from this and tautology-is-wffo[OF insert.prems] show ?case
   by (rule Cases)
 qed
qed

end
```

6.35 Proposition 5234 (Rule P)

According to the proof in [2], if $[A^1 \wedge \cdots \wedge A^n] \supset B$ is tautologous, then clearly $A^1 \supset (\ldots (A^n \supset B) \ldots)$ is also tautologous. Since this is not clear to us, we prove instead the version of Rule P found in [1]:

proposition *tautologous-horn-clause-is-hyp-derivable*:
assumes *is-hyps* $\mathcal{H}$ **and** *is-hyps* $\mathcal{G}$
and $\forall A \in \mathcal{G}.\ \mathcal{H} \vdash A$
and *lset hs* $= \mathcal{G}$
and *is-tautologous* (*hs* $\supset^{\mathfrak{Q}}_{\star}$ B)
shows $\mathcal{H} \vdash B$
proof −
from *assms*(5) **obtain** ϑ **and** C
where *is-tautology* C
and *is-substitution* ϑ
and $\forall (x, \alpha) \in$ *fmdom'* $\vartheta.\ \alpha = o$
and *hs* $\supset^{\mathfrak{Q}}_{\star}$ B = **S** ϑ C
by *blast*
then have $\vdash$ *hs* $\supset^{\mathfrak{Q}}_{\star}$ B
proof (*cases* $\vartheta = \{\$\$\}$)
case *True*
with ‹*hs* $\supset^{\mathfrak{Q}}_{\star}$ B = **S** ϑ C› **have** C = *hs* $\supset^{\mathfrak{Q}}_{\star}$ B
using *empty-substitution-neutrality* **by** *simp*
with ‹*hs* $\supset^{\mathfrak{Q}}_{\star}$ B = **S** ϑ C› **and** ‹*is-tautology* C› **show** *?thesis*
using *prop-5233* **by** (*simp only*:)
next
case *False*
from ‹*is-tautology* C› **have** $\vdash$ C **and** $C \in$ *pwffs*
using *prop-5233* **by** *simp-all*
moreover have
$\forall v \in$ *fmdom'* ϑ. *var-name* $v \notin$ *free-var-names* ({}::*form set*) $\wedge$ *is-free-for* (ϑ $\$\$!$ v) v C
proof
fix v
assume $v \in$ *fmdom'* ϑ
then show *var-name* $v \notin$ *free-var-names* ({}::*form set*) $\wedge$ *is-free-for* (ϑ $\$\$!$ v) v C
proof (*cases* $v \in$ *free-vars* C)
case *True*
with ‹$C \in$ *pwffs*› **show** *?thesis*
using *is-free-for-in-pwff* **by** *simp*
next
case *False*
then have *is-free-for* (ϑ $\$\$!$ v) v C
unfolding *is-free-for-def* **using** *is-free-at-in-free-vars* **by** *blast*
then show *?thesis*
by *simp*
qed
qed
ultimately show *?thesis*
using *False* **and** ‹*is-substitution* ϑ› **and** *Sub*

by (*simp add*: ‹*hs* $\supset^{\mathfrak{Q}}_{\star}$ B = **S** ϑ C›[*unfolded generalized-imp-op-def*])
qed
from *this* **and** *assms*(*1*) **have** $\mathcal{H} \vdash$ *hs* $\supset^{\mathfrak{Q}}_{\star}$ B
by (*rule derivability-implies-hyp-derivability*)
with *assms*(*3*,*4*) **show** *?thesis*
using *generalized-modus-ponens* **by** *blast*
qed

corollary *tautologous-is-hyp-derivable*:
assumes *is-hyps* $\mathcal{H}$
and *is-tautologous* B
shows $\mathcal{H} \vdash B$
using *assms* **and** *tautologous-horn-clause-is-hyp-derivable*[**where** $\mathcal{G} = \{\}$] **by** *simp*

lemmas *prop-5234* = *tautologous-horn-clause-is-hyp-derivable tautologous-is-hyp-derivable*

lemmas *rule-P* = *prop-5234*

6.36 Proposition 5235

proposition *prop-5235*:
assumes $A \in$ *pwffs* **and** $B \in$ *pwffs*
and $(x, \alpha) \notin$ *free-vars* A
shows $\vdash \forall x_\alpha.\ (A \vee^{\mathfrak{Q}} B) \supset^{\mathfrak{Q}} (A \vee^{\mathfrak{Q}} \forall x_\alpha.\ B)$
proof −
have §*1*: $\vdash \forall x_\alpha.\ (T_o \vee^{\mathfrak{Q}} B) \supset^{\mathfrak{Q}} (T_o \vee^{\mathfrak{Q}} \forall x_\alpha.\ B)$
proof (*intro rule-P*(*2*))
show *is-tautologous* $(\forall x_\alpha.\ (T_o \vee^{\mathfrak{Q}} B) \supset^{\mathfrak{Q}} T_o \vee^{\mathfrak{Q}} \forall x_\alpha.\ B)$
proof −
let *?ϑ* = $\{(\mathfrak{x},\ o) \rightarrowtail \forall x_\alpha.\ (T_o \vee^{\mathfrak{Q}} B), (\mathfrak{y},\ o) \rightarrowtail \forall x_\alpha.\ B\}$ **and** *?C* = $\mathfrak{x}_o \supset^{\mathfrak{Q}} (T_o \vee^{\mathfrak{Q}} (\mathfrak{y}_o))$
have *is-tautology ?C*
using $\mathcal{V}_B$*-simps* **by** *simp*
moreover from *assms*(*2*) **have** *is-pwff-substitution ?ϑ*
using *pwffs-subset-of-wffso* **by** *fastforce*
moreover have $\forall x_\alpha.\ (T_o \vee^{\mathfrak{Q}} B) \supset^{\mathfrak{Q}} T_o \vee^{\mathfrak{Q}} \forall x_\alpha.\ B$ = **S** *?ϑ ?C*
by *simp*
ultimately show *?thesis*
by *blast*
qed
qed *simp*
have §*2*: $\vdash \forall x_\alpha.\ B \supset^{\mathfrak{Q}} (F_o \vee^{\mathfrak{Q}} \forall x_\alpha.\ B)$
proof (*intro rule-P*(*2*))
show *is-tautologous* $(\forall x_\alpha.\ B \supset^{\mathfrak{Q}} (F_o \vee^{\mathfrak{Q}} \forall x_\alpha.\ B))$
proof −
let *?ϑ* = $\{(\mathfrak{x},\ o) \rightarrowtail \forall x_\alpha.\ B\}$ **and** *?C* = $\mathfrak{x}_o \supset^{\mathfrak{Q}} (F_o \vee^{\mathfrak{Q}} (\mathfrak{x}_o))$
have *is-tautology* $(\mathfrak{x}_o \supset^{\mathfrak{Q}} (F_o \vee^{\mathfrak{Q}} (\mathfrak{x}_o)))$ (**is** ‹*is-tautology ?C*›)
using $\mathcal{V}_B$*-simps* **by** *simp*
moreover from *assms*(*2*) **have** *is-pwff-substitution ?ϑ*
using *pwffs-subset-of-wffso* **by** *auto*

```
    moreover have ∀ x_α. B ⊃^Q (F_o ∨^Q ∀ x_α. B) = S ?ϑ ?C
      by simp
    ultimately show ?thesis
      by blast
  qed
qed simp
have §3: ⊢ B ≡^Q (F_o ∨^Q B)
proof (intro rule-P(2))
  show is-tautologous (B ≡^Q (F_o ∨^Q B))
  proof −
    let ?ϑ = {(𝔯, o) ⇀ B} and ?C = 𝔯_o ≡^Q (F_o ∨^Q (𝔯_o))
    have is-tautology ?C
      using V_B-simps by simp
    moreover from assms(2) have is-pwff-substitution ?ϑ
      using pwffs-subset-of-wffso by auto
    moreover have B ≡^Q (F_o ∨^Q B) = S ?ϑ ?C
      by simp
    ultimately show ?thesis
      by blast
  qed
qed simp
from §2 and §3[unfolded equivalence-def] have §4:
  ⊢ ∀ x_α. (F_o ∨^Q B) ⊃^Q (F_o ∨^Q ∀ x_α. B)
  by (rule rule-R[where p = [«,»,»,«]]) force+
obtain p where (p, o) ∉ vars (∀ x_α. (A ∨^Q B) ⊃^Q (A ∨^Q ∀ x_α. B))
  by (meson fresh-var-existence vars-form-finiteness)
then have (p, o) ≠ (x, α) and (p, o) ∉ vars A and (p, o) ∉ vars B
  by simp-all
from ‹(p, o) ∉ vars B› have sub: S {(p, o) ⇀ C} B = B for C
  using free-var-singleton-substitution-neutrality and free-vars-in-all-vars by blast
have §5: ⊢ ∀ x_α. (p_o ∨^Q B) ⊃^Q (p_o ∨^Q ∀ x_α. B) (is ‹⊢ ?C›)
proof −
  from sub and §1 have ⊢ S {(p, o) ⇀ T_o} ?C
    using ‹(p, o) ≠ (x, α)› by auto
  moreover from sub and §4 have ⊢ S {(p, o) ⇀ F_o} ?C
    using ‹(p, o) ≠ (x, α)› by auto
  moreover from assms(2) have ?C ∈ wffs_o
    using pwffs-subset-of-wffso by auto
  ultimately show ?thesis
    by (rule Cases)
qed
then show ?thesis
proof −
  let ?ϑ = {(p, o) ⇀ A}
  from assms(1) have is-substitution ?ϑ
    using pwffs-subset-of-wffso by auto
  moreover have
    ∀ v ∈ fmdom′ ?ϑ. var-name v ∉ free-var-names ({}::form set) ∧ is-free-for (?ϑ $$! v) v ?C
  proof
```

```
    fix v
    assume v ∈ fmdom' ?ϑ
    then have v = (p, o)
      by simp
    with assms(3) and ‹(p, o) ∉ vars B› have is-free-for (?ϑ $$! v) v ?C
      using occurs-in-vars
      by (intro is-free-for-in-imp is-free-for-in-forall is-free-for-in-disj) auto
    moreover have var-name v ∉ free-var-names ({}::form set)
      by simp
    ultimately show var-name v ∉ free-var-names ({}::form set) ∧ is-free-for (?ϑ $$! v) v ?C
      unfolding ‹v = (p, o)› by blast
  qed
  moreover have ?ϑ ≠ {$$}
    by simp
  ultimately have ⊢ S ?ϑ ?C
    by (rule Sub[OF §5])
  moreover have S ?ϑ ?C = ∀ x_α. (A ∨^Q B) ⊃^Q (A ∨^Q ∀ x_α. B)
    using ‹(p, o) ≠ (x, α)› and sub[of A] by simp fast
  ultimately show ?thesis
    by (simp only:)
 qed
qed
```

6.37 Proposition 5237 ($\supset \forall$ Rule)

The proof in [2] uses the pseudo-rule Q and the axiom 5 of $\mathcal{F}$. Therefore, we prove such axiom, following the proof of Theorem 143 in [1]:

```
context begin

private lemma prop-5237-aux:
 assumes A ∈ wffs_o and B ∈ wffs_o
 and (x, α) ∉ free-vars A
 shows ⊢ ∀ x_α. (A ⊃^Q B) ≡^Q (A ⊃^Q (∀ x_α. B))
proof −
 have is-tautology (𝔯_o ≡^Q (T_o ⊃^Q 𝔯_o)) (is ‹is-tautology ?C_1›)
   using V_B-simps by simp
 have is-tautology (𝔯_o ⊃^Q (𝔯_o ≡^Q (F_o ⊃^Q 𝔫_o))) (is ‹is-tautology ?C_2›)
   using V_B-simps by simp
 have §1: ⊢ ∀ x_α. B ≡^Q (T_o ⊃^Q ∀ x_α. B)
 proof (intro rule-P(2))
   show is-tautologous (∀ x_α. B ≡^Q (T_o ⊃^Q ∀ x_α. B))
   proof −
     let ?ϑ = {(𝔯, o) ⇀ ∀ x_α. B}
     from assms(2) have is-pwff-substitution ?ϑ
       using pwffs-subset-of-wffso by auto
     moreover have ∀ x_α. B ≡^Q (T_o ⊃^Q ∀ x_α. B) = S ?ϑ ?C_1
       by simp
     ultimately show ?thesis
       using ‹is-tautology ?C_1› by blast
```

qed
qed *simp*
have §*2*: $\vdash\ B \equiv^{\mathcal{Q}} (T_o \supset^{\mathcal{Q}} B)$
proof (*intro rule-P(2)*)
show *is-tautologous* $(B \equiv^{\mathcal{Q}} T_o \supset^{\mathcal{Q}} B)$
proof −
let *?ϑ* $= \{(\mathfrak{x},\ o) \rightarrowtail B\}$
from *assms(2)* **have** *is-pwff-substitution ?ϑ*
using *pwffs-subset-of-wffso* **by** *auto*
moreover have $B \equiv^{\mathcal{Q}} T_o \supset^{\mathcal{Q}} B = \mathbf{S}\ ?\vartheta\ ?C_1$
by *simp*
ultimately show *?thesis*
using ‹*is-tautology ?C*$_1$› **by** *blast*
qed
qed *simp*
have $\vdash\ T_o$
by (*fact true-is-derivable*)
then have §*3*: $\vdash\ \forall\, x_\alpha.\ T_o$
using *Gen* **by** *simp*
have §*4*: $\vdash\ \forall\, x_\alpha.\ T_o \equiv^{\mathcal{Q}} (F_o \supset^{\mathcal{Q}} \forall\, x_\alpha.\ B)$
proof (*intro rule-P(1)*[**where** $\mathcal{G} = \{\forall\, x_\alpha.\ T_o\}$])
show *is-tautologous* $([\forall\, x_\alpha.\ T_o] \supset^{\mathcal{Q}}_{\star} (\forall\, x_\alpha.\ T_o \equiv^{\mathcal{Q}} (F_o \supset^{\mathcal{Q}} \forall\, x_\alpha.\ B)))$
proof −
let *?ϑ* $= \{(\mathfrak{x},\ o) \rightarrowtail \forall\, x_\alpha.\ T_o,\ (\mathfrak{y},\ o) \rightarrowtail \forall\, x_\alpha.\ B\}$
from *assms(2)* **have** *is-pwff-substitution ?ϑ*
using *pwffs-subset-of-wffso* **by** *auto*
moreover have $[\forall\, x_\alpha.\ T_o] \supset^{\mathcal{Q}}_{\star} (\forall\, x_\alpha.\ T_o \equiv^{\mathcal{Q}} (F_o \supset^{\mathcal{Q}} \forall\, x_\alpha.\ B)) = \mathbf{S}\ ?\vartheta\ ?C_2$
by *simp*
ultimately show *?thesis*
using ‹*is-tautology ?C*$_2$› **by** *blast*
qed
qed (*use* §*3* **in** *fastforce*)+
have §*5*: $\vdash\ T_o \equiv^{\mathcal{Q}} (F_o \supset^{\mathcal{Q}} B)$
proof (*intro rule-P(2)*)
show *is-tautologous* $(T_o \equiv^{\mathcal{Q}} (F_o \supset^{\mathcal{Q}} B))$
proof −
let *?ϑ* $= \{(\mathfrak{x},\ o) \rightarrowtail B\}$ **and** *?C* $= T_o \equiv^{\mathcal{Q}} (F_o \supset^{\mathcal{Q}} \mathfrak{x}_o)$
have *is-tautology ?C*
using $\mathcal{V}_B$*-simps* **by** *simp*
moreover from *assms(2)* **have** *is-pwff-substitution ?ϑ*
using *pwffs-subset-of-wffso* **by** *auto*
moreover have $T_o \equiv^{\mathcal{Q}} (F_o \supset^{\mathcal{Q}} B) = \mathbf{S}\ ?\vartheta\ ?C$
by *simp*
ultimately show *?thesis*
by *blast*
qed
qed *simp*
from §*4* **and** §*5* **have** §*6*: $\vdash\ \forall\, x_\alpha.\ (F_o \supset^{\mathcal{Q}} B) \equiv^{\mathcal{Q}} (F_o \supset^{\mathcal{Q}} \forall\, x_\alpha.\ B)$
unfolding *equivalence-def* **by** (*rule rule-R*[**where** $p = [«,»,»,«]]$) *force*+

```
  from §1 and §2 have §7: ⊢ ∀ x_α. (T_o ⊃^Ω B) ≡^Ω (T_o ⊃^Ω ∀ x_α. B)
    unfolding equivalence-def by (rule rule-R[where p = [«,»,»,«]]) force+
  obtain p where (p, o) ∉ vars B and p ≠ x
    using fresh-var-existence and vars-form-finiteness by (metis finite-insert insert-iff)
  from ‹(p, o) ∉ vars B› have sub: S {(p, o) ⇀ C} B = B for C
    using free-var-singleton-substitution-neutrality and free-vars-in-all-vars by blast
  have §8: ⊢ ∀ x_α. (p_o ⊃^Ω B) ≡^Ω (p_o ⊃^Ω ∀ x_α. B) (is ‹⊢ ?C_3›)
  proof −
    from sub and §7 have ⊢ S {(p, o) ⇀ T_o} ?C_3
      using ‹p ≠ x› by auto
    moreover from sub and §6 have ⊢ S {(p, o) ⇀ F_o} ?C_3
      using ‹p ≠ x› by auto
    moreover from assms(2) have ?C_3 ∈ wffs_o
      using pwffs-subset-of-wffso by auto
    ultimately show ?thesis
      by (rule Cases)
  qed
  then show ?thesis
  proof −
    let ?ϑ = {(p, o) ⇀ A}
    from assms(1) have is-substitution ?ϑ
      using pwffs-subset-of-wffso by auto
    moreover have
      ∀ v ∈ fmdom′ ?ϑ. var-name v ∉ free-var-names ({}::form set) ∧ is-free-for (?ϑ $$! v) v ?C_3
    proof
      fix v
      assume v ∈ fmdom′ ?ϑ
      then have v = (p, o)
        by simp
      with assms(3) and ‹(p, o) ∉ vars B› have is-free-for (?ϑ $$! v) v ?C_3
        using occurs-in-vars
        by (intro is-free-for-in-imp is-free-for-in-forall is-free-for-in-equivalence) auto
      moreover have var-name v ∉ free-var-names ({}::form set)
        by simp
      ultimately show var-name v ∉ free-var-names ({}::form set) ∧ is-free-for (?ϑ $$! v) v ?C_3
        unfolding ‹v = (p, o)› by blast
    qed
    moreover have ?ϑ ≠ {$$}
      by simp
    ultimately have ⊢ S ?ϑ ?C_3
      by (rule Sub[OF §8])
    moreover have S ?ϑ ?C_3 = ∀ x_α. (A ⊃^Ω B) ≡^Ω (A ⊃^Ω ∀ x_α. B)
      using ‹p ≠ x› and sub[of A] by simp
    ultimately show ?thesis
      by (simp only:)
  qed
qed

proposition prop-5237:
```

assumes *is-hyps* $\mathcal{H}$
and $\mathcal{H} \vdash A \supset^{\mathcal{Q}} B$
and $(x, \alpha) \notin$ *free-vars* $(\{A\} \cup \mathcal{H})$
shows $\mathcal{H} \vdash A \supset^{\mathcal{Q}} (\forall\, x_\alpha.\ B)$
proof −
have $\mathcal{H} \vdash A \supset^{\mathcal{Q}} B$
by *fact*
with *assms*(*3*) **have** $\mathcal{H} \vdash \forall\, x_\alpha.\ (A \supset^{\mathcal{Q}} B)$
using *Gen* **by** *simp*
moreover have $\mathcal{H} \vdash \forall\, x_\alpha.\ (A \supset^{\mathcal{Q}} B) \equiv^{\mathcal{Q}} (A \supset^{\mathcal{Q}} (\forall\, x_\alpha.\ B))$
proof −
from *assms*(*2*) **have** $A \in \mathit{wffs}_o$ **and** $B \in \mathit{wffs}_o$
using *hyp-derivable-form-is-wffso* **by** (*blast dest*: *wffs-from-imp-op*)+
with *assms*(*1*,*3*) **show** *?thesis*
using *prop-5237-aux* **and** *derivability-implies-hyp-derivability* **by** *simp*
qed
ultimately show *?thesis*
by (*rule Equality-Rules*(*1*))
qed

lemmas $\supset\forall$ = *prop-5237*

corollary *generalized-prop-5237*:
assumes *is-hyps* $\mathcal{H}$
and $\mathcal{H} \vdash A \supset^{\mathcal{Q}} B$
and $\forall\, v \in S.\ v \notin$ *free-vars* $(\{A\} \cup \mathcal{H})$
and *lset vs* = S
shows $\mathcal{H} \vdash A \supset^{\mathcal{Q}} (\forall^{\mathcal{Q}}{}_\star\ vs\ B)$
using *assms* **proof** (*induction vs arbitrary*: S)
case *Nil*
then show *?case*
by *simp*
next
case (*Cons v vs*)
obtain x **and** α **where** $v = (x, \alpha)$
by *fastforce*
from *Cons.prems*(*3*) **have** $*$: $\forall\, v' \in S.\ v' \notin$ *free-vars* $(\{A\} \cup \mathcal{H})$
by *blast*
then show *?case*
proof (*cases v* ∈ *lset vs*)
case *True*
with *Cons.prems*(*4*) **have** *lset vs* = S
by *auto*
with *assms*(*1*,*2*) **and** $*$ **have** $\mathcal{H} \vdash A \supset^{\mathcal{Q}} \forall^{\mathcal{Q}}{}_\star\ vs\ B$
by (*fact Cons.IH*)
with *True* **and** ‹*lset vs* = S› **and** ‹$v = (x, \alpha)$› **and** $*$ **have** $\mathcal{H} \vdash A \supset^{\mathcal{Q}} (\forall\, x_\alpha.\ \forall^{\mathcal{Q}}{}_\star\ vs\ B)$
using *prop-5237*[*OF assms*(*1*)] **by** *simp*
with ‹$v = (x, \alpha)$› **show** *?thesis*
by *simp*

```
  next
    case False
    with ‹lset (v # vs) = S› have lset vs = S − {v}
      by auto
    moreover from ∗ have ∀ v′ ∈ S − {v}. v′ ∉ free-vars ({A} ∪ ℋ)
      by blast
    ultimately have ℋ ⊢ A ⊃^𝔔 ∀^𝔔_⋆ vs B
      using assms(1,2) by (intro Cons.IH)
    moreover from Cons.prems(4) and ‹v = (x, α)› and ∗ have (x, α) ∉ free-vars ({A} ∪ ℋ)
      by auto
    ultimately have ℋ ⊢ A ⊃^𝔔 (∀ x_α. ∀^𝔔_⋆ vs B)
      using assms(1) by (intro prop-5237)
    with ‹v = (x, α)› show ?thesis
      by simp
  qed
qed

end
```

6.38 Proposition 5238

```
context begin

private lemma prop-5238-aux:
  assumes A ∈ wffs_α and B ∈ wffs_α
  shows ⊢ ((λx_β. A) =_{β→α} (λx_β. B)) ≡^𝔔 ∀ x_β. (A =_α B)
proof −
  have §1:
    ⊢ (𝔣_{β→α} =_{β→α} 𝔤_{β→α}) ≡^𝔔 ∀𝔵_β. (𝔣_{β→α} ▪ 𝔵_β =_α 𝔤_{β→α} ▪ 𝔵_β) (is ‹⊢ - ≡^𝔔 ∀𝔵_β. ?C_1›)
    by (fact axiom-is-derivable-from-no-hyps[OF axiom-3])
  then have §2:
    ⊢ (𝔣_{β→α} =_{β→α} 𝔤_{β→α}) ≡^𝔔 ∀ x_β. (𝔣_{β→α} ▪ x_β =_α 𝔤_{β→α} ▪ x_β) (is ‹⊢ ?C_2›)
  proof (cases x = 𝔵)
    case True
    with §1 show ?thesis
      by (simp only:)
  next
    case False
    have ?C_1 ∈ wffs_o
      by blast
    moreover from False have (x, β) ∉ free-vars ?C_1
      by simp
    moreover have is-free-for (x_β) (𝔵, β) ?C_1
      by (intro is-free-for-in-equality is-free-for-to-app) simp-all
    ultimately have ⊢ λ𝔵_β. ?C_1 =_{β→o} λx_β. (S {(𝔵, β) ⇀ x_β} ?C_1)
      by (rule α)
    from §1 and this show ?thesis
      by (rule rule-R[where p = [»,»]]) force+
  qed
```

then have §*3*:
  $\vdash ((\lambda x_\beta.\ A) =_{\beta\to\alpha} (\lambda x_\beta.\ B)) \equiv^{\mathfrak{Q}} \forall\, x_\beta.\ ((\lambda x_\beta.\ A) \cdot x_\beta =_\alpha (\lambda x_\beta.\ B) \cdot x_\beta)$
proof −
  let *?ϑ* = $\{(\mathfrak{f},\ \beta\to\alpha) \rightarrowtail \lambda x_\beta.\ A,\ (\mathfrak{g},\ \beta\to\alpha) \rightarrowtail \lambda x_\beta.\ B\}$
  have $\lambda x_\beta.\ A \in \mathit{wffs}_{\beta\to\alpha}$ **and** $\lambda x_\beta.\ B \in \mathit{wffs}_{\beta\to\alpha}$
    by (*blast intro*: *assms*(*1*,*2*))+
  then have *is-substitution ?ϑ*
    by *simp*
  moreover have
    $\forall\, v \in \mathit{fmdom'}\ ?\vartheta.$ *var-name* $v \notin$ *free-var-names* ({}::*form set*) $\wedge$ *is-free-for* (*?ϑ* $$! v) v *?C*$_2$
  proof
    fix v
    assume $v \in$ *fmdom'* *?ϑ*
    then consider (*a*) $v = (\mathfrak{f},\ \beta\to\alpha)$ | (*b*) $v = (\mathfrak{g},\ \beta\to\alpha)$
      by *fastforce*
    then show *var-name* $v \notin$ *free-var-names* ({}::*form set*) $\wedge$ *is-free-for* (*?ϑ* $$! v) v *?C*$_2$
    proof *cases*
      case *a*
      have $(x,\ \beta) \notin$ *free-vars* $(\lambda x_\beta.\ A)$
        by *simp*
      then have *is-free-for* $(\lambda x_\beta.\ A)$ $(\mathfrak{f},\ \beta\to\alpha)$ *?C*$_2$
        unfolding *equivalence-def*
        by (*intro is-free-for-in-equality is-free-for-in-forall is-free-for-to-app*, *simp-all*)
      with *a* **show** *?thesis*
        by *force*
    next
      case *b*
      have $(x,\ \beta) \notin$ *free-vars* $(\lambda x_\beta.\ B)$
        by *simp*
      then have *is-free-for* $(\lambda x_\beta.\ B)$ $(\mathfrak{g},\ \beta\to\alpha)$ *?C*$_2$
        unfolding *equivalence-def*
        by (*intro is-free-for-in-equality is-free-for-in-forall is-free-for-to-app*, *simp-all*)
      with *b* **show** *?thesis*
        by *force*
    qed
  qed
  moreover have *?ϑ* $\neq$ {$$}
    by *simp*
  ultimately have $\vdash$ **S** *?ϑ* *?C*$_2$
    by (*rule Sub*[*OF* §*2*])
  then show *?thesis*
    by *simp*
qed
then have §*4*: $\vdash ((\lambda x_\beta.\ A) =_{\beta\to\alpha} (\lambda x_\beta.\ B)) \equiv^{\mathfrak{Q}} \forall\, x_\beta.\ (A =_\alpha (\lambda x_\beta.\ B) \cdot x_\beta)$
proof −
  have $\vdash (\lambda x_\beta.\ A) \cdot x_\beta =_\alpha A$
    using *prop-5208*[**where** *vs* = [$(x,\ \beta)$]] **and** *assms*(*1*) **by** *simp*
  from §*3* **and** *this* **show** *?thesis*
    by (*rule rule-R*[**where** *p* = [»,»,«,«,»]]) *force*+

```
  qed
  then show ?thesis
  proof −
    have ⊢ (λx_β. B) ▪ x_β =_α B
      using prop-5208[where vs = [(x, β)]] and assms(2) by simp
    from §4 and this show ?thesis
      by (rule rule-R[where p = [»,»,«,»]]) force+
  qed
qed

proposition prop-5238:
  assumes vs ≠ [] and A ∈ wffs_α and B ∈ wffs_α
  shows ⊢ λ^Q_⋆ vs A =_foldr (→) (map var-type vs) α λ^Q_⋆ vs B ≡^Q ∀^Q_⋆ vs (A =_α B)
using assms proof (induction vs arbitrary: A B α rule: rev-nonempty-induct)
  case (single v)
  obtain x and β where v = (x, β)
    by fastforce
  from single.prems have
    λ^Q_⋆ vs A =_foldr (→) (map var-type vs) α λ^Q_⋆ vs B ≡^Q ∀^Q_⋆ vs (A =_α B) ∈ wffs_o
    by blast
  with single.prems and ‹v = (x, β)› show ?case
    using prop-5238-aux by simp
next
  case (snoc v vs)
  obtain x and β where v = (x, β)
    by fastforce
  from snoc.prems have λx_β. A ∈ wffs_{β→α} and λx_β. B ∈ wffs_{β→α}
    by auto
  then have
    ⊢
      λ^Q_⋆ vs (λx_β. A) =_foldr (→) (map var-type vs) (β→α) λ^Q_⋆ vs (λx_β. B)
      ≡^Q
      ∀^Q_⋆ vs ((λx_β. A) =_{β→α} (λx_β. B))
    by (fact snoc.IH)
  moreover from snoc.prems have ⊢ λx_β. A =_{β→α} λx_β. B ≡^Q ∀x_β. (A =_α B)
    by (fact prop-5238-aux)
  ultimately have
    ⊢
      λ^Q_⋆ vs (λx_β. A) =_foldr (→) (map var-type vs) (β→α) λ^Q_⋆ vs (λx_β. B)
      ≡^Q
      ∀^Q_⋆ vs ∀x_β. (A =_α B)
  unfolding equivalence-def proof (induction rule: rule-R[where p = » # foldr (λ-. (@) [»,«]) vs []])
    case occ-subform
    then show ?case
      using innermost-subform-in-generalized-forall[OF snoc.hyps] and is-subform-at.simps(3)
      by fastforce
  next
    case replacement
```

then show *?case*
using *innermost-replacement-in-generalized-forall*[*OF snoc.hyps*]
and *is-replacement-at-implies-in-positions* **and** *replace-right-app* **by** *force*
qed
with ‹$v = (x, \beta)$› **show** *?case*
by *simp*
qed

end

6.39 Proposition 5239

lemma *replacement-derivability*:
assumes $C \in wffs_\beta$
and $A \preceq_p C$
and $\vdash A =_\alpha B$
and $C\langle\!|p \leftarrow B|\!\rangle \rhd D$
shows $\vdash C =_\beta D$
using *assms* **proof** (*induction arbitrary*: D p)
case (*var-is-wff* γ x)
from *var-is-wff.prems*(*1*) **have** $p = []$ **and** $A = x_\gamma$
by (*auto elim*: *is-subform-at.elims*(*2*))
with *var-is-wff.prems*(*2*) **have** $\alpha = \gamma$
using *hyp-derivable-form-is-wffso* **and** *wff-has-unique-type* **and** *wffs-from-equality* **by** *blast*
moreover from ‹$p = []$› **and** *var-is-wff.prems*(*3*) **have** $D = B$
using *is-replacement-at-minimal-change*(*1*) **and** *is-subform-at.simps*(*1*) **by** *iprover*
ultimately show *?case*
using ‹$A = x_\gamma$› **and** *var-is-wff.prems*(*2*) **by** (*simp only*:)
next
case (*con-is-wff* γ c)
from *con-is-wff.prems*(*1*) **have** $p = []$ **and** $A = \{\!|c|\!\}_\gamma$
by (*auto elim*: *is-subform-at.elims*(*2*))
with *con-is-wff.prems*(*2*) **have** $\alpha = \gamma$
using *hyp-derivable-form-is-wffso* **and** *wff-has-unique-type*
by (*meson wffs-from-equality wffs-of-type-intros*(*2*))
moreover from ‹$p = []$› **and** *con-is-wff.prems*(*3*) **have** $D = B$
using *is-replacement-at-minimal-change*(*1*) **and** *is-subform-at.simps*(*1*) **by** *iprover*
ultimately show *?case*
using ‹$A = \{\!|c|\!\}_\gamma$› **and** *con-is-wff.prems*(*2*) **by** (*simp only*:)
next
case (*app-is-wff* γ δ C_1 C_2)
from *app-is-wff.prems*(*1*) **consider**
(a) $p = []$
| (b) $\exists p'.\ p = « \# p' \wedge A \preceq_{p'} C_1$
| (c) $\exists p'.\ p = » \# p' \wedge A \preceq_{p'} C_2$
using *subforms-from-app* **by** *blast*
then show *?case*
proof *cases*
case *a*

```
    with app-is-wff.prems(1) have A = C₁ ▪ C₂
      by simp
    moreover from a and app-is-wff.prems(3) have D = B
      using is-replacement-at-minimal-change(1) and at-top-is-self-subform by blast
    moreover from ‹A = C₁ ▪ C₂› and ‹D = B› and app-is-wff.hyps(1,2) and assms(3) have α =
δ
      using hyp-derivable-form-is-wffso and wff-has-unique-type
      by (blast dest: wffs-from-equality)
    ultimately show ?thesis
      using assms(3) by (simp only:)
  next
    case b
    then obtain p' where p = « # p' and A ≼_p' C₁
      by blast
    moreover obtain D₁ where D = D₁ ▪ C₂ and C₁⟨p' ← B⟩ ▷ D₁
      using app-is-wff.prems(3) and ‹p = « # p'› by (force dest: is-replacement-at.cases)
    ultimately have ⊢ C₁ =_γ→δ D₁
      using app-is-wff.IH(1) and assms(3) by blast
    moreover have ⊢ C₂ =_γ C₂
      by (fact prop-5200[OF app-is-wff.hyps(2)])
    ultimately have ⊢ C₁ ▪ C₂ =_δ D₁ ▪ C₂
      using Equality-Rules(4) by (simp only:)
    with ‹D = D₁ ▪ C₂› show ?thesis
      by (simp only:)
  next
    case c
    then obtain p' where p = » # p' and A ≼_p' C₂
      by blast
    moreover obtain D₂ where D = C₁ ▪ D₂ and C₂⟨p' ← B⟩ ▷ D₂
      using app-is-wff.prems(3) and ‹p = » # p'› by (force dest: is-replacement-at.cases)
    ultimately have ⊢ C₂ =_γ D₂
      using app-is-wff.IH(2) and assms(3) by blast
    moreover have ⊢ C₁ =_γ→δ C₁
      by (fact prop-5200[OF app-is-wff.hyps(1)])
    ultimately have ⊢ C₁ ▪ C₂ =_δ C₁ ▪ D₂
      using Equality-Rules(4) by (simp only:)
    with ‹D = C₁ ▪ D₂› show ?thesis
      by (simp only:)
  qed
next
  case (abs-is-wff δ C' γ x)
  from abs-is-wff.prems(1) consider (a) p = [] | (b) ∃ p'. p = « # p' ∧ A ≼_p' C'
    using subforms-from-abs by blast
  then show ?case
  proof cases
    case a
    with abs-is-wff.prems(1) have A = λx_γ. C'
      by simp
    moreover from a and abs-is-wff.prems(3) have D = B
```

```
    using is-replacement-at-minimal-change(1) and at-top-is-self-subform by blast
   moreover from ‹A = λx_γ. C'› and ‹D = B› and abs-is-wff.hyps(1) and assms(3) have α =
γ→δ
    using hyp-derivable-form-is-wffso and wff-has-unique-type
    by (blast dest: wffs-from-abs wffs-from-equality)
   ultimately show ?thesis
    using assms(3) by (simp only:)
  next
   case b
   then obtain p' where p = « # p' and A ≼_p' C'
    by blast
   moreover obtain D' where D = λx_γ. D' and C'⟪p' ← B⟫ ▷ D'
    using abs-is-wff.prems(3) and ‹p = « # p'› by (force dest: is-replacement-at.cases)
   ultimately have ⊢ C' =_δ D'
    using abs-is-wff.IH and assms(3) by blast
   then have ⊢ λx_γ. C' =_γ→δ λx_γ. D'
   proof −
    from ‹⊢ C' =_δ D'› have ⊢ ∀ x_γ. (C' =_δ D')
     using Gen by simp
    moreover from ‹⊢ C' =_δ D'› and abs-is-wff.hyps have D' ∈ wffs_δ
     using hyp-derivable-form-is-wffso by (blast dest: wffs-from-equality)
    with abs-is-wff.hyps have ⊢ (λx_γ. C' =_γ→δ λx_γ. D') ≡^Q ∀ x_γ. (C' =_δ D')
     using prop-5238[where vs = [(x, γ)]] by simp
    ultimately show ?thesis
     using Equality-Rules(1,2) unfolding equivalence-def by blast
   qed
   with ‹D = λx_γ. D'› show ?thesis
    by (simp only:)
  qed
qed

context
begin

private lemma prop-5239-aux-1:
  assumes p ∈ positions (•^Q_⋆ (FVar v) (map FVar vs))
  and p ≠ replicate (length vs) «
  shows
   (∃ A B. A • B ≼_p (•^Q_⋆ (FVar v) (map FVar vs)))
   ∨
   (∃ v ∈ lset vs. occurs-at v p (•^Q_⋆ (FVar v) (map FVar vs)))
using assms proof (induction vs arbitrary: p rule: rev-induct)
  case Nil
  then show ?case
   using surj-pair[of v] by fastforce
next
  case (snoc v' vs)
  from snoc.prems(1) consider
   (a) p = []
```

```
  | (b) p = [»]
  | (c) ∃ p' ∈ positions (·^Q_⋆ (FVar v) (map FVar vs)). p = « # p'
    using surj-pair[of v'] by fastforce
  then show ?case
  proof cases
    case c
    then obtain p' where p' ∈ positions (·^Q_⋆ (FVar v) (map FVar vs)) and p = « # p'
      by blast
    from ‹p = « # p'› and snoc.prems(2) have p' ≠ replicate (length vs) «
      by force
    then have
      (∃ A B. A · B ≼_p' ·^Q_⋆ (FVar v) (map FVar vs))
      ∨
      (∃ v ∈ lset vs. occurs-at v p' (·^Q_⋆ (FVar v) (map FVar vs)))
      using ‹p' ∈ positions (·^Q_⋆ (FVar v) (map FVar vs))› and snoc.IH by simp
    with ‹p = « # p'› show ?thesis
      by auto
  qed simp-all
qed

private lemma prop-5239-aux-2:
  assumes t ∉ lset vs ∪ vars C
  and C⟨|p ← (·^Q_⋆ (FVar t) (map FVar vs))|⟩ ▷ G
  and C⟨|p ← (·^Q_⋆ (λ^Q_⋆ vs A) (map FVar vs))|⟩ ▷ G'
  shows S {t ↣ λ^Q_⋆ vs A} G = G' (is ‹S ?ϑ G = G'›)
proof −
  have S ?ϑ (·^Q_⋆ (FVar t) (map FVar vs)) = ·^Q_⋆ (S ?ϑ (FVar t)) (map (λv'. S ?ϑ v') (map FVar
vs))
    using generalized-app-substitution by blast
  moreover have S ?ϑ (FVar t) = λ^Q_⋆ vs A
    using surj-pair[of t] by fastforce
  moreover from assms(1) have map (λv'. S ?ϑ v') (map FVar vs) = map FVar vs
    by (induction vs) auto
  ultimately show ?thesis
  using assms proof (induction C arbitrary: G G' p)
    case (FVar v)
    from FVar.prems(5) have p = [] and G = ·^Q_⋆ (FVar t) (map FVar vs)
      by (blast dest: is-replacement-at.cases)+
    moreover from FVar.prems(6) and ‹p = []› have G' = ·^Q_⋆ (λ^Q_⋆ vs A) (map FVar vs)
      by (blast dest: is-replacement-at.cases)
    ultimately show ?case
      using FVar.prems(1−3) by (simp only:)
  next
    case (FCon k)
    from FCon.prems(5) have p = [] and G = ·^Q_⋆ (FVar t) (map FVar vs)
      by (blast dest: is-replacement-at.cases)+
    moreover from FCon.prems(6) and ‹p = []› have G' = ·^Q_⋆ (λ^Q_⋆ vs A) (map FVar vs)
      by (blast dest: is-replacement-at.cases)
    ultimately show ?case
```

```
    using FCon.prems(1−3) by (simp only:)
  next
    case (FApp C₁ C₂)
    from FApp.prems(4) have t ∉ lset vs ∪ vars C₁ and t ∉ lset vs ∪ vars C₂
      by auto
    consider (a) p = [] | (b) ∃p'. p = « # p' | (c) ∃p'. p = » # p'
      by (metis direction.exhaust list.exhaust)
    then show ?case
    proof cases
      case a
      with FApp.prems(5) have G = ·^Q⋆ (FVar t) (map FVar vs)
        by (blast dest: is-replacement-at.cases)
      moreover from FApp.prems(6) and ‹p = []› have G' = ·^Q⋆ (λ^Q⋆ vs A) (map FVar vs)
        by (blast dest: is-replacement-at.cases)
      ultimately show ?thesis
        using FApp.prems(1−3) by (simp only:)
    next
      case b
      then obtain p' where p = « # p'
        by blast
      with FApp.prems(5) obtain G₁ where G = G₁ · C₂ and C₁⟨p' ← (·^Q⋆ (FVar t) (map FVar
vs))⟩ ▷ G₁
        by (blast elim: is-replacement-at.cases)
      moreover from ‹p = « # p'› and FApp.prems(6)
      obtain G'₁ where G' = G'₁ · C₂ and C₁⟨p' ← (·^Q⋆ (λ^Q⋆ vs A) (map FVar vs))⟩ ▷ G'₁
        by (blast elim: is-replacement-at.cases)
      moreover from ‹t ∉ lset vs ∪ vars C₂› have S {t ⇀ λ^Q⋆ vs A} C₂ = C₂
        using surj-pair[of t] and free-var-singleton-substitution-neutrality
        by (simp add: vars-is-free-and-bound-vars)
      ultimately show ?thesis
        using FApp.IH(1)[OF FApp.prems(1−3) ‹t ∉ lset vs ∪ vars C₁›] by simp
    next
      case c
      then obtain p' where p = » # p'
        by blast
      with FApp.prems(5) obtain G₂ where G = C₁ · G₂ and C₂⟨p' ← (·^Q⋆ (FVar t) (map FVar
vs))⟩ ▷ G₂
        by (blast elim: is-replacement-at.cases)
      moreover from ‹p = » # p'› and FApp.prems(6)
      obtain G'₂ where G' = C₁ · G'₂ and C₂⟨p' ← (·^Q⋆ (λ^Q⋆ vs A) (map FVar vs))⟩ ▷ G'₂
        by (blast elim: is-replacement-at.cases)
      moreover from ‹t ∉ lset vs ∪ vars C₁› have S {t ⇀ λ^Q⋆ vs A} C₁ = C₁
        using surj-pair[of t] and free-var-singleton-substitution-neutrality
        by (simp add: vars-is-free-and-bound-vars)
      ultimately show ?thesis
        using FApp.IH(2)[OF FApp.prems(1−3) ‹t ∉ lset vs ∪ vars C₂›] by simp
    qed
  next
    case (FAbs v C')
```

```
    from FAbs.prems(4) have t ∉ lset vs ∪ vars C' and t ≠ v
      using vars-form.elims by blast+
    from FAbs.prems(5) consider (a) p = [] | (b) ∃ p'. p = « # p'
      using is-replacement-at.simps by blast
    then show ?case
    proof cases
      case a
      with FAbs.prems(5) have G = ▪^Ω⋆ (FVar t) (map FVar vs)
        by (blast dest: is-replacement-at.cases)
      moreover from FAbs.prems(6) and ‹p = []› have G' = ▪^Ω⋆ (λ^Ω⋆ vs A) (map FVar vs)
        by (blast dest: is-replacement-at.cases)
      ultimately show ?thesis
        using FAbs.prems(1−3) by (simp only:)
    next
      case b
      then obtain p' where p = « # p'
        by blast
      then obtain G₁ where G = FAbs v G₁ and C'⟨|p' ← (▪^Ω⋆ (FVar t) (map FVar vs))|⟩ ▷ G₁
        using FAbs.prems(5) by (blast elim: is-replacement-at.cases)
      moreover from ‹p = « # p'› and FAbs.prems(6)
      obtain G'₁ where G' = FAbs v G'₁ and C'⟨|p' ← (▪^Ω⋆ (λ^Ω⋆ vs A) (map FVar vs))|⟩ ▷ G'₁
        by (blast elim: is-replacement-at.cases)
      ultimately have S {t ⇀ λ^Ω⋆ vs A} G₁ = G'₁
        using FAbs.IH[OF FAbs.prems(1−3) ‹t ∉ lset vs ∪ vars C'›] by simp
      with ‹G = FAbs v G₁› and ‹G' = FAbs v G'₁› and ‹t ≠ v› show ?thesis
        using surj-pair[of v] by fastforce
    qed
  qed
qed

private lemma prop-5239-aux-3:
  assumes t ∉ lset vs ∪ vars {A, C}
  and C⟨|p ← (▪^Ω⋆ (FVar t) (map FVar vs))|⟩ ▷ G
  and occurs-at t p' G
  shows p' = p @ replicate (length vs) « (is ‹p' = ?p_t›)
proof (cases vs = [])
  case True
  then have t ∉ vars C
    using assms(1) by auto
  moreover from True and assms(2) have C⟨|p ← FVar t|⟩ ▷ G
    by force
  ultimately show ?thesis
    using assms(3) and True and fresh-var-replacement-position-uniqueness by simp
next
  case False
  show ?thesis
  proof (rule ccontr)
    assume p' ≠ ?p_t
    have ¬ prefix ?p_t p
```

by (*simp add*: *False*)
from *assms*(*3*) **have** $p' \in$ *positions* G
using *is-subform-implies-in-positions* **by** *fastforce*
from *assms*(*2*) **have** $?p_t \in$ *positions* G
using *is-replacement-at-minimal-change*(*1*) **and** *is-subform-at-transitivity*
and *is-subform-implies-in-positions* **and** *leftmost-subform-in-generalized-app*
by (*metis length-map*)
from *assms*(*2*) **have** *occurs-at* t $?p_t$ G
unfolding *occurs-at-def* **using** *is-replacement-at-minimal-change*(*1*) **and** *is-subform-at-transitivity*
and *leftmost-subform-in-generalized-app*
by (*metis length-map*)
moreover from *assms*(*2*) **and** ‹$p' \in$ *positions* G› **have** $*$:
subform-at C p' = *subform-at* G p' **if** ¬ *prefix* p' p **and** ¬ *prefix* p p'
using *is-replacement-at-minimal-change*(*2*) **by** (*simp add*: *that*(*1*,*2*))
ultimately show *False*
proof (*cases* ¬ *prefix* p' p ∧ ¬ *prefix* p p')
case *True*
with *assms*(*3*) **and** $*$ **have** *occurs-at* t p' C
using *is-replacement-at-occurs*[*OF assms*(*2*)] **by** *blast*
then have $t \in$ *vars* C
using *is-subform-implies-in-positions* **and** *occurs-in-vars* **by** *fastforce*
with *assms*(*1*) **show** *?thesis*
by *simp*
next
case *False*
then consider (*a*) *prefix* p' p | (*b*) *prefix* p p'
by *blast*
then show *?thesis*
proof *cases*
case *a*
with ‹*occurs-at* t $?p_t$ G› **and** ‹$p' \neq ?p_t$› **and** *assms*(*3*) **show** *?thesis*
unfolding *occurs-at-def* **using** *loop-subform-impossibility*
by (*metis prefix-order.dual-order.order-iff-strict prefix-prefix*)
next
case *b*
have *strict-prefix* p' $?p_t$
proof (*rule ccontr*)
assume ¬ *strict-prefix* p' $?p_t$
then consider
(b_1) $p' = ?p_t$
| (b_2) *strict-prefix* $?p_t$ p'
| (b_3) ¬ *prefix* p' $?p_t$ **and** ¬ *prefix* $?p_t$ p'
by *fastforce*
then show *False*
proof *cases*
case b_1
with ‹$p' \neq ?p_t$› **show** *?thesis*
by *contradiction*
next

```
      case b2
      with ‹occurs-at t ?pt G› and assms(3) show ?thesis
        using loop-subform-impossibility by blast
    next
      case b3
      from b obtain p'' where p' = p @ p'' and p'' ∈ positions (•^Q⋆ (FVar t) (map FVar vs))
        using is-replacement-at-new-positions and ‹p' ∈ positions G› and assms(2) by blast
      moreover have p'' ≠ replicate (length vs) «
        using ‹p' = p @ p''› and ‹p' ≠ ?pt› by blast
      ultimately consider
        (b3-1) ∃ F1 F2. F1 • F2 ≼p'' (•^Q⋆ (FVar t) (map FVar vs))
      | (b3-2) ∃ v ∈ lset vs. occurs-at v p'' (•^Q⋆ (FVar t) (map FVar vs))
        using prop-5239-aux-1 and b3(1,2) and is-replacement-at-occurs
        and leftmost-subform-in-generalized-app-replacement
        by (metis (no-types, opaque-lifting) length-map prefix-append)
      then show ?thesis
      proof cases
        case b3-1
        with assms(2) and ‹p' = p @ p''› have ∃ F1 F2. F1 • F2 ≼p' G
          using is-replacement-at-minimal-change(1) and is-subform-at-transitivity by meson
        with ‹occurs-at t p' G› show ?thesis
          using is-subform-at-uniqueness by fastforce
      next
        case b3-2
        with assms(2) and ‹p' = p @ p''› have ∃ v ∈ lset vs. occurs-at v p' G
          unfolding occurs-at-def
          using is-replacement-at-minimal-change(1) and is-subform-at-transitivity by meson
        with assms(1,3) show ?thesis
          using is-subform-at-uniqueness by fastforce
      qed
    qed
  qed
  with ‹occurs-at t ?pt G› and assms(3) show ?thesis
    using loop-subform-impossibility by blast
 qed
qed
qed

private lemma prop-5239-aux-4:
  assumes t ∉ lset vs ∪ vars {A, C}
  and A ≼p C
  and lset vs ⊇ capture-exposed-vars-at p C A
  and C⟨p ← (•^Q⋆ (FVar t) (map FVar vs))⟩ ▷ G
  shows is-free-for (λ^Q⋆ vs A) t G
unfolding is-free-for-def proof (intro ballI impI)
  let ?pt = p @ replicate (length vs) «
  from assms(4) have FVar t ≼?pt G
    using is-replacement-at-minimal-change(1) and is-subform-at-transitivity
```

```
    and leftmost-subform-in-generalized-app by (metis length-map)
  fix v' and p'
  assume v' ∈ free-vars (λ^Q_⋆ vs A) and p' ∈ positions G and is-free-at t p' G
  have v' ∉ binders-at G ?p_t
  proof −
    have free-vars (λ^Q_⋆ vs A) = free-vars A − lset vs
      by (fact free-vars-of-generalized-abs)
    also from assms(2,3) have ... ⊆ free-vars A − (binders-at C p ∩ free-vars A)
      using capture-exposed-vars-at-alt-def and is-subform-implies-in-positions by fastforce
    also have ... = free-vars A − (binders-at G p ∩ free-vars A)
      using assms(2,4) is-replacement-at-binders is-subform-implies-in-positions by blast
    finally have free-vars (λ^Q_⋆ vs A) ⊆ free-vars A − (binders-at G p ∩ free-vars A) .
    moreover have binders-at (·^Q_⋆ (FVar t) (map FVar vs)) (replicate (length vs) «) = {}
      by (induction vs rule: rev-induct) simp-all
    with assms(4) have binders-at G ?p_t = binders-at G p
      using binders-at-concat and is-replacement-at-minimal-change(1) by blast
    ultimately show ?thesis
      using ‹v' ∈ free-vars (λ^Q_⋆ vs A)› by blast
  qed
  moreover have p' = ?p_t
    by
      (
        fact prop-5239-aux-3
          [OF assms(1,4) ‹is-free-at t p' G›[unfolded is-free-at-def, THEN conjunct1]]
      )
  ultimately show ¬ in-scope-of-abs v' p' G
    using binders-at-alt-def[OF ‹p' ∈ positions G›] and in-scope-of-abs-alt-def by auto
qed

proposition prop-5239:
  assumes is-rule-R-app p D C (A =_α B)
  and lset vs =
    {(x, β) | x β p' E. strict-prefix p' p ∧ λx_β. E ⪯_p' C ∧ (x, β) ∈ free-vars (A =_α B)}
  shows ⊢ ∀^Q_⋆ vs (A =_α B) ⊃^Q (C ≡^Q D)
proof −
  let ?γ = foldr (→) (map var-type vs) α
  obtain t where (t, ?γ) ∉ lset vs ∪ vars {A,B,C,D}
    using fresh-var-existence and vars-form-set-finiteness
    by (metis List.finite-set finite.simps finite-UnI)
  from assms(1) have A ∈ wffs_α and B ∈ wffs_α and A ⪯_p C
    using wffs-from-equality[OF equality-wff] by simp-all
  from assms(1) have C ∈ wffs_o and D ∈ wffs_o
    using replacement-preserves-typing by fastforce+
  have ·^Q_⋆ t_?γ (map FVar vs) ∈ wffs_α
    using generalized-app-wff[where As = map FVar vs and ts = map var-type vs]
    by (metis eq-snd-iff length-map nth-map wffs-of-type-intros(1))
  from assms(1) have p ∈ positions C
    using is-subform-implies-in-positions by fastforce
  then obtain G where C⟨⟨p ← (·^Q_⋆ t_?γ (map FVar vs))⟩⟩ ▷ G
```

using *is-replacement-at-existence* **by** *blast*
with ‹$A \preceq_p C$› **and** ‹$\cdot^{\mathfrak{Q}}_\star\ t_{?\gamma}\ (\mathit{map}\ \mathit{FVar}\ \mathit{vs}) \in \mathit{wffs}_\alpha$› **have** $G \in \mathit{wffs}_o$
using ‹$A \in \mathit{wffs}_\alpha$› **and** ‹$C \in \mathit{wffs}_o$› **and** *replacement-preserves-typing* **by** *blast*
let $?\vartheta = \{(\mathfrak{h},\ ?\gamma{\to}o) \rightarrowtail \lambda t_{?\gamma}.\ G,\ (\mathfrak{x},\ ?\gamma) \rightarrowtail \lambda^{\mathfrak{Q}}_\star\ \mathit{vs}\ A,\ (\mathfrak{y},\ ?\gamma) \rightarrowtail \lambda^{\mathfrak{Q}}_\star\ \mathit{vs}\ B\}$
and $?A = (\mathfrak{x}_{?\gamma} =_{?\gamma} \mathfrak{y}_{?\gamma}) \supset^{\mathfrak{Q}} (\mathfrak{h}_{?\gamma\to o} \cdot \mathfrak{x}_{?\gamma} \equiv^{\mathfrak{Q}} \mathfrak{h}_{?\gamma\to o} \cdot \mathfrak{y}_{?\gamma})$
have $\vdash ?A$
by (*fact axiom-is-derivable-from-no-hyps*[*OF axiom-2*])
moreover have $\lambda t_{?\gamma}.\ G \in \mathit{wffs}_{?\gamma\to o}$ **and** $\lambda^{\mathfrak{Q}}_\star\ \mathit{vs}\ A \in \mathit{wffs}_{?\gamma}$ **and** $\lambda^{\mathfrak{Q}}_\star\ \mathit{vs}\ B \in \mathit{wffs}_{?\gamma}$
by (*blast intro*: ‹$G \in \mathit{wffs}_o$› ‹$A \in \mathit{wffs}_\alpha$› ‹$B \in \mathit{wffs}_\alpha$›)+
then have *is-substitution* $?\vartheta$
by *simp*
moreover have
$\forall\, v \in \mathit{fmdom'}\ ?\vartheta.\ \mathit{var\text{-}name}\ v \notin \mathit{free\text{-}var\text{-}names}\ (\{\}{::}\mathit{form\ set}) \wedge \mathit{is\text{-}free\text{-}for}\ (?\vartheta\ \$\$!\ v)\ v\ ?A$
by
(
(
code-simp, *unfold atomize-conj*[*symmetric*], *simp*,
use is-free-for-in-equality is-free-for-in-equivalence is-free-for-in-imp is-free-for-in-var
is-free-for-to-app **in** *presburger*
)+,
blast
)
moreover have $?\vartheta \neq \{\$\$\}$
by *simp*
ultimately have $\vdash$ **S** $?\vartheta\ ?A$
by (*rule Sub*)
moreover have
S $?\vartheta\ ?A = (\lambda^{\mathfrak{Q}}_\star\ \mathit{vs}\ A =_{?\gamma} \lambda^{\mathfrak{Q}}_\star\ \mathit{vs}\ B) \supset^{\mathfrak{Q}} ((\lambda t_{?\gamma}.\ G) \cdot (\lambda^{\mathfrak{Q}}_\star\ \mathit{vs}\ A) \equiv^{\mathfrak{Q}} (\lambda t_{?\gamma}.\ G) \cdot (\lambda^{\mathfrak{Q}}_\star\ \mathit{vs}\ B))$
by *simp*
ultimately have §*1*:
$\vdash (\lambda^{\mathfrak{Q}}_\star\ \mathit{vs}\ A =_{?\gamma} \lambda^{\mathfrak{Q}}_\star\ \mathit{vs}\ B) \supset^{\mathfrak{Q}} ((\lambda t_{?\gamma}.\ G) \cdot (\lambda^{\mathfrak{Q}}_\star\ \mathit{vs}\ A) \equiv^{\mathfrak{Q}} (\lambda t_{?\gamma}.\ G) \cdot (\lambda^{\mathfrak{Q}}_\star\ \mathit{vs}\ B))$
by (*simp only*:)
then have §*2*: $\vdash (\forall^{\mathfrak{Q}}_\star\ \mathit{vs}\ (A =_\alpha B)) \supset^{\mathfrak{Q}} ((\lambda t_{?\gamma}.\ G) \cdot (\lambda^{\mathfrak{Q}}_\star\ \mathit{vs}\ A) \equiv^{\mathfrak{Q}} (\lambda t_{?\gamma}.\ G) \cdot (\lambda^{\mathfrak{Q}}_\star\ \mathit{vs}\ B))$
proof (*cases vs* $= []$)
case *True*
with §*1* **show** *?thesis*
by *simp*
next
case *False*
from §*1* **and** *prop-5238*[*OF False* ‹$A \in \mathit{wffs}_\alpha$› ‹$B \in \mathit{wffs}_\alpha$›] **show** *?thesis*
unfolding *equivalence-def* **by** (*rule rule-R*[**where** $p = [«,»]$]) *force*+
qed
moreover have $\vdash (\lambda t_{?\gamma}.\ G) \cdot (\lambda^{\mathfrak{Q}}_\star\ \mathit{vs}\ A) =_o C$ **and** $\vdash (\lambda t_{?\gamma}.\ G) \cdot (\lambda^{\mathfrak{Q}}_\star\ \mathit{vs}\ B) =_o D$
proof −
from *assms*(*1*) **have** $B \preceq_p D$
using *is-replacement-at-minimal-change*(*1*) **by** *force*
from *assms*(*1*) **have** $D\langle\!|p \leftarrow (\cdot^{\mathfrak{Q}}_\star\ t_{?\gamma}\ (\mathit{map}\ \mathit{FVar}\ \mathit{vs}))|\!\rangle \rhd G$
using ‹$C\langle\!|p \leftarrow (\cdot^{\mathfrak{Q}}_\star\ t_{?\gamma}\ (\mathit{map}\ \mathit{FVar}\ \mathit{vs}))|\!\rangle \rhd G$› **and** *replacement-override*
by (*meson is-rule-R-app-def*)

from ‹$B \preceq_p D$› **have** $p \in$ *positions* D
using *is-subform-implies-in-positions* **by** *auto*
from *assms*(*1*) **have** *binders-at* $D\ p =$ *binders-at* $C\ p$
using *is-replacement-at-binders* **by** *fastforce*
then have *binders-at* $D\ p \cap$ *free-vars* $B =$ *binders-at* $C\ p \cap$ *free-vars* B
by *simp*
with *assms*(*2*)
[
folded capture-exposed-vars-at-def,
unfolded capture-exposed-vars-at-alt-def[*OF* ‹$p \in$ *positions* C›]
] **have** *lset vs* $\supseteq$ *capture-exposed-vars-at* $p\ D\ B$
unfolding *capture-exposed-vars-at-alt-def*[*OF* ‹$p \in$ *positions* D›] **by** *auto*
have *is-free-for* $(\lambda^{\mathcal{Q}}_\star$ *vs* $A)\ (t,\ ?\gamma)\ G$ **and** *is-free-for* $(\lambda^{\mathcal{Q}}_\star$ *vs* $B)\ (t,\ ?\gamma)\ G$
proof −
have $(t,\ ?\gamma) \notin$ *lset vs* $\cup$ *vars* $\{A,\ C\}$ **and** $(t,\ ?\gamma) \notin$ *lset vs* $\cup$ *vars* $\{B,\ D\}$
using ‹$(t,\ ?\gamma) \notin$ *lset vs* $\cup$ *vars* $\{A,\ B,\ C,\ D\}$› **by** *simp-all*
moreover from *assms*(*2*) **have**
lset vs $\supseteq$ *capture-exposed-vars-at* $p\ C\ A$ **and** *lset vs* $\supseteq$ *capture-exposed-vars-at* $p\ D\ B$
by *fastforce fact*
ultimately show *is-free-for* $(\lambda^{\mathcal{Q}}_\star$ *vs* $A)\ (t,\ ?\gamma)\ G$ **and** *is-free-for* $(\lambda^{\mathcal{Q}}_\star$ *vs* $B)\ (t,\ ?\gamma)\ G$
using *prop-5239-aux-4* **and** ‹$B \preceq_p D$› **and** ‹$A \preceq_p C$› **and** ‹$C\langle\!|p \leftarrow (\cdot^{\mathcal{Q}}_\star\ t_{?\gamma}\ (map\ FVar\ vs))|\!\rangle \rhd G$›
and ‹$D\langle\!|p \leftarrow (\cdot^{\mathcal{Q}}_\star\ t_{?\gamma}\ (map\ FVar\ vs))|\!\rangle \rhd G$› **by** *meson*+
qed
then have $\vdash (\lambda t_{?\gamma}.\ G) \cdot (\lambda^{\mathcal{Q}}_\star$ *vs* $A) =_o \mathbf{S}\ \{(t,\ ?\gamma) \rightarrowtail \lambda^{\mathcal{Q}}_\star$ *vs* $A\}\ G$
and $\vdash (\lambda t_{?\gamma}.\ G) \cdot (\lambda^{\mathcal{Q}}_\star$ *vs* $B) =_o \mathbf{S}\ \{(t,\ ?\gamma) \rightarrowtail \lambda^{\mathcal{Q}}_\star$ *vs* $B\}\ G$
using *prop-5207*[*OF* ‹$\lambda^{\mathcal{Q}}_\star$ *vs* $A \in wffs_{?\gamma}$› ‹$G \in wffs_o$›]
and *prop-5207*[*OF* ‹$\lambda^{\mathcal{Q}}_\star$ *vs* $B \in wffs_{?\gamma}$› ‹$G \in wffs_o$›] **by** *auto*
moreover obtain G'_1 **and** G'_2
where $C\langle\!|p \leftarrow (\cdot^{\mathcal{Q}}_\star\ (\lambda^{\mathcal{Q}}_\star$ *vs* $A)\ (map\ FVar\ vs))|\!\rangle \rhd G'_1$
and $D\langle\!|p \leftarrow (\cdot^{\mathcal{Q}}_\star\ (\lambda^{\mathcal{Q}}_\star$ *vs* $B)\ (map\ FVar\ vs))|\!\rangle \rhd G'_2$
using *is-replacement-at-existence*[*OF* ‹$p \in$ *positions* C›]
and *is-replacement-at-existence*[*OF* ‹$p \in$ *positions* D›] **by** *metis*
then have S $\{(t,\ ?\gamma) \rightarrowtail \lambda^{\mathcal{Q}}_\star$ *vs* $A\}\ G = G'_1$ **and S** $\{(t,\ ?\gamma) \rightarrowtail \lambda^{\mathcal{Q}}_\star$ *vs* $B\}\ G = G'_2$
proof −
have $(t,\ ?\gamma) \notin$ *lset vs* $\cup$ *vars* C **and** $(t,\ ?\gamma) \notin$ *lset vs* $\cup$ *vars* D
using ‹$(t,\ ?\gamma) \notin$ *lset vs* $\cup$ *vars* $\{A,\ B,\ C,\ D\}$› **by** *simp-all*
then show S $\{(t,\ ?\gamma) \rightarrowtail \lambda^{\mathcal{Q}}_\star$ *vs* $A\}\ G = G'_1$ **and S** $\{(t,\ ?\gamma) \rightarrowtail \lambda^{\mathcal{Q}}_\star$ *vs* $B\}\ G = G'_2$
using ‹$C\langle\!|p \leftarrow (\cdot^{\mathcal{Q}}_\star\ t_{?\gamma}\ (map\ FVar\ vs))|\!\rangle \rhd G$› **and** ‹$D\langle\!|p \leftarrow (\cdot^{\mathcal{Q}}_\star\ t_{?\gamma}\ map\ FVar\ vs)|\!\rangle \rhd G$›
and ‹$C\langle\!|p \leftarrow (\cdot^{\mathcal{Q}}_\star\ (\lambda^{\mathcal{Q}}_\star$ *vs* $A)\ (map\ FVar\ vs))|\!\rangle \rhd G'_1$›
and ‹$D\langle\!|p \leftarrow (\cdot^{\mathcal{Q}}_\star\ (\lambda^{\mathcal{Q}}_\star$ *vs* $B)\ (map\ FVar\ vs))|\!\rangle \rhd G'_2$› **and** *prop-5239-aux-2* **by** *blast*+
qed
ultimately have $\vdash (\lambda t_{?\gamma}.\ G) \cdot (\lambda^{\mathcal{Q}}_\star$ *vs* $A) =_o G'_1$ **and** $\vdash (\lambda t_{?\gamma}.\ G) \cdot (\lambda^{\mathcal{Q}}_\star$ *vs* $B) =_o G'_2$
by (*simp-all only*:)
moreover
have $\vdash A =_\alpha (\cdot^{\mathcal{Q}}_\star\ (\lambda^{\mathcal{Q}}_\star$ *vs* $A)\ (map\ FVar\ vs))$ **and** $\vdash B =_\alpha (\cdot^{\mathcal{Q}}_\star\ (\lambda^{\mathcal{Q}}_\star$ *vs* $B)\ (map\ FVar\ vs))$
unfolding *atomize-conj* **proof** (*cases vs* $= []$)
assume *vs* $= []$

show $\vdash A =_\alpha \cdot^{\mathfrak{Q}}_\star\ (\lambda^{\mathfrak{Q}}_\star\ vs\ A)\ (map\ FVar\ vs) \wedge \vdash B =_\alpha \cdot^{\mathfrak{Q}}_\star\ (\lambda^{\mathfrak{Q}}_\star\ vs\ B)\ (map\ FVar\ vs)$
unfolding ‹$vs = []$› **using** *prop-5200* **and** ‹$A \in wffs_\alpha$› **and** ‹$B \in wffs_\alpha$› **by** *simp*
next
assume $vs \neq []$
show $\vdash A =_\alpha \cdot^{\mathfrak{Q}}_\star\ (\lambda^{\mathfrak{Q}}_\star\ vs\ A)\ (map\ FVar\ vs) \wedge \vdash B =_\alpha \cdot^{\mathfrak{Q}}_\star\ (\lambda^{\mathfrak{Q}}_\star\ vs\ B)\ (map\ FVar\ vs)$
using *Equality-Rules(2)*[*OF prop-5208*[*OF* ‹$vs \neq []$›]] **and** ‹$A \in wffs_\alpha$› **and** ‹$B \in wffs_\alpha$›
by *blast+*
qed
with
‹$C\langle\!|p \leftarrow (\cdot^{\mathfrak{Q}}_\star\ (\lambda^{\mathfrak{Q}}_\star\ vs\ A)\ (map\ FVar\ vs))|\!\rangle \rhd G'_1$›
and
‹$D\langle\!|p \leftarrow (\cdot^{\mathfrak{Q}}_\star\ (\lambda^{\mathfrak{Q}}_\star\ vs\ B)\ (map\ FVar\ vs))|\!\rangle \rhd G'_2$›
have $\vdash G'_1 =_o C$ **and** $\vdash G'_2 =_o D$
using *Equality-Rules(2)*[*OF replacement-derivability*] **and** ‹$C \in wffs_o$› **and** ‹$D \in wffs_o$›
and ‹$A \preceq_p C$› **and** ‹$B \preceq_p D$› **by** *blast+*
ultimately show $\vdash (\lambda t_{?\gamma}.\ G) \cdot (\lambda^{\mathfrak{Q}}_\star\ vs\ A) =_o C$ **and** $\vdash (\lambda t_{?\gamma}.\ G) \cdot (\lambda^{\mathfrak{Q}}_\star\ vs\ B) =_o D$
using *Equality-Rules(3)* **by** *blast+*
qed
ultimately show *?thesis*
proof −
from §*2* **and** ‹$\vdash (\lambda t_{?\gamma}.\ G) \cdot (\lambda^{\mathfrak{Q}}_\star\ vs\ A) =_o C$› **have**
$\vdash (\forall^{\mathfrak{Q}}_\star\ vs\ (A =_\alpha B)) \supset^{\mathfrak{Q}} (C \equiv^{\mathfrak{Q}} (\lambda t_{?\gamma}.\ G) \cdot (\lambda^{\mathfrak{Q}}_\star\ vs\ B))$
by (*rule rule-R*[**where** $p = [»,«,»]$]) *force+*
from *this* **and** ‹$\vdash (\lambda t_{?\gamma}.\ G) \cdot (\lambda^{\mathfrak{Q}}_\star\ vs\ B) =_o D$› **show** *?thesis*
by (*rule rule-R*[**where** $p = [»,»]$]) *force+*
qed
qed

end

6.40 Theorem 5240 (Deduction Theorem)

lemma *pseudo-rule-R-is-tautologous*:
assumes $C \in wffs_o$ **and** $D \in wffs_o$ **and** $E \in wffs_o$ **and** $H \in wffs_o$
shows *is-tautologous* $(((H \supset^{\mathfrak{Q}} C) \supset^{\mathfrak{Q}} ((H \supset^{\mathfrak{Q}} E) \supset^{\mathfrak{Q}} ((E \supset^{\mathfrak{Q}} (C \equiv^{\mathfrak{Q}} D)) \supset^{\mathfrak{Q}} (H \supset^{\mathfrak{Q}} D)))))$
proof −
let *?ϑ* $= \{(\mathfrak{x},\ o) \rightarrowtail C,\ (\mathfrak{y},\ o) \rightarrowtail D,\ (\mathfrak{z},\ o) \rightarrowtail E,\ (\mathfrak{h},\ o) \rightarrowtail H\}$
have
is-tautology
$(((\mathfrak{h}_o \supset^{\mathfrak{Q}} \mathfrak{x}_o) \supset^{\mathfrak{Q}} ((\mathfrak{h}_o \supset^{\mathfrak{Q}} \mathfrak{z}_o) \supset^{\mathfrak{Q}} ((\mathfrak{z}_o \supset^{\mathfrak{Q}} (\mathfrak{x}_o \equiv^{\mathfrak{Q}} \mathfrak{y}_o)) \supset^{\mathfrak{Q}} (\mathfrak{h}_o \supset^{\mathfrak{Q}} \mathfrak{y}_o)))))$
using $\mathcal{V}_B$*-simps* **by** *simp*
moreover have *is-substitution ?ϑ*
using *assms* **by** *auto*
moreover have $\forall (x,\ \alpha) \in fmdom'\ ?\vartheta.\ \alpha = o$
by *simp*
moreover have
$((H \supset^{\mathfrak{Q}} C) \supset^{\mathfrak{Q}} ((H \supset^{\mathfrak{Q}} E) \supset^{\mathfrak{Q}} ((E \supset^{\mathfrak{Q}} (C \equiv^{\mathfrak{Q}} D)) \supset^{\mathfrak{Q}} (H \supset^{\mathfrak{Q}} D))))$
=
S *?ϑ* $(((\mathfrak{h}_o \supset^{\mathfrak{Q}} \mathfrak{x}_o) \supset^{\mathfrak{Q}} ((\mathfrak{h}_o \supset^{\mathfrak{Q}} \mathfrak{z}_o) \supset^{\mathfrak{Q}} ((\mathfrak{z}_o \supset^{\mathfrak{Q}} (\mathfrak{x}_o \equiv^{\mathfrak{Q}} \mathfrak{y}_o)) \supset^{\mathfrak{Q}} (\mathfrak{h}_o \supset^{\mathfrak{Q}} \mathfrak{y}_o)))))$

```
    by simp
  ultimately show ?thesis
    by blast
qed

syntax
  -HypDer :: form ⇒ form set ⇒ form ⇒ bool (-,- ⊢ - [50, 50, 50] 50)
translations
  ℋ, H ⊢ P ⇀ ℋ ∪ {H} ⊢ P

theorem thm-5240:
  assumes finite ℋ
  and ℋ, H ⊢ P
  shows ℋ ⊢ H ⊃^Ω P
proof −
  from ‹ℋ, H ⊢ P› obtain 𝒮1 and 𝒮2 where ∗: is-hyp-proof-of (ℋ ∪ {H}) 𝒮1 𝒮2 P
    using hyp-derivability-implies-hyp-proof-existence by blast
  have ℋ ⊢ H ⊃^Ω (𝒮2 ! i') if i' <length 𝒮2 for i'
  using that proof (induction i' rule: less-induct)
    case (less i')
    let ?R = 𝒮2 ! i'
    from less.prems(1) and ∗ have is-hyps ℋ
      by fastforce
    from less.prems and ∗ have ?R ∈ wffs_o
      using elem-of-proof-is-wffo[simplified] by auto
    from less.prems and ∗ have is-hyp-proof-step (ℋ ∪ {H}) 𝒮1 𝒮2 i'
      by simp
    then consider
      (hyp) ?R ∈ ℋ ∪ {H}
    | (seq) ?R ∈ lset 𝒮1
    | (rule-R') ∃j k p. {j, k} ⊆ {0..<i'} ∧ is-rule-R'-app (ℋ ∪ {H}) p ?R (𝒮2 ! j) (𝒮2 ! k)
      by force
    then show ?case
    proof cases
      case hyp
      then show ?thesis
      proof (cases ?R = H)
        case True
        with ‹?R ∈ wffs_o› have is-tautologous (H ⊃^Ω ?R)
          using implication-reflexivity-is-tautologous by (simp only:)
        with ‹is-hyps ℋ› show ?thesis
          by (rule rule-P(2))
      next
        case False
        with hyp have ?R ∈ ℋ
          by blast
        with ‹is-hyps ℋ› have ℋ ⊢ ?R
          by (intro dv-hyp)
        moreover from less.prems(1) and ∗ have is-tautologous (?R ⊃^Ω (H ⊃^Ω ?R))
```

using *principle-of-simplification-is-tautologous*[*OF* ‹*?R* ∈ *wffs*$_o$›] **by** *force*
moreover from ‹*?R* ∈ *wffs*$_o$› **have** *is-hyps* {*?R*}
by *simp*
ultimately show *?thesis*
using *rule-P*(*1*)[**where** $\mathcal{G} = \{?R\}$ **and** *hs* = [*?R*], *OF* ‹*is-hyps* $\mathcal{H}$›] **by** *simp*
qed
next
case *seq*
then have $\mathcal{S}_1 \neq []$
by *force*
moreover from *less.prems*(*1*) **and** $*$ **have** *is-proof* $\mathcal{S}_1$
by *fastforce*
moreover from *seq* **obtain** i'' **where** $i'' <$ *length* $\mathcal{S}_1$ **and** $?R = \mathcal{S}_1 \,!\, i''$
by (*metis in-set-conv-nth*)
ultimately have *is-theorem ?R*
using *proof-form-is-theorem* **by** *fastforce*
with ‹*is-hyps* $\mathcal{H}$› **have** $\mathcal{H} \vdash ?R$
by (*intro dv-thm*)
moreover from ‹*?R* ∈ *wffs*$_o$› **and** *less.prems*(*1*) **and** $*$ **have** *is-tautologous* (*?R* $\supset^{\mathfrak{Q}}$ (*H* $\supset^{\mathfrak{Q}}$ *?R*))
using *principle-of-simplification-is-tautologous* **by** *force*
moreover from ‹*?R* ∈ *wffs*$_o$› **have** *is-hyps* {*?R*}
by *simp*
ultimately show *?thesis*
using *rule-P*(*1*)[**where** $\mathcal{G} = \{?R\}$ **and** *hs* = [*?R*], *OF* ‹*is-hyps* $\mathcal{H}$›] **by** *simp*
next
case *rule-R′*
then obtain *j* **and** *k* **and** *p*
where $\{j, k\} \subseteq \{0..<i'\}$ **and** *rule-R′-app*: *is-rule-R′-app* ($\mathcal{H} \cup \{H\}$) *p ?R* ($\mathcal{S}_2$! *j*) ($\mathcal{S}_2$! *k*)
by *auto*
then obtain *A* **and** *B* **and** *C* **and** α **where** $C = \mathcal{S}_2 \,!\, j$ **and** $\mathcal{S}_2 \,!\, k = A =_\alpha B$
by *fastforce*
with ‹$\{j, k\} \subseteq \{0..<i'\}$› **have** $\mathcal{H} \vdash H \supset^{\mathfrak{Q}} C$ **and** $\mathcal{H} \vdash H \supset^{\mathfrak{Q}} (A =_\alpha B)$
using *less.IH* **and** *less.prems*(*1*) **by** (*simp*, *force*)
define *S* **where** *S* ≡
$\{(x, \beta) \mid x\ \beta\ p'\ E.$ *strict-prefix* p' *p* ∧ λx_β. $E \preceq_{p'} C \wedge (x, \beta) \in$ *free-vars* ($A =_\alpha B$)$\}$
with ‹$C = \mathcal{S}_2 \,!\, j$› **and** ‹$\mathcal{S}_2 \,!\, k = A =_\alpha B$› **have** $\forall\, v \in S.\ v \notin$ *free-vars* ($\mathcal{H} \cup \{H\}$)
using *rule-R′-app* **by** *fastforce*
moreover have $S \subseteq$ *free-vars* ($A =_\alpha B$)
unfolding *S-def* **by** *blast*
then have *finite S*
by (*fact rev-finite-subset*[*OF free-vars-form-finiteness*])
then obtain *vs* **where** *lset vs* = *S*
using *finite-list* **by** *blast*
ultimately have $\mathcal{H} \vdash H \supset^{\mathfrak{Q}} \forall^{\mathfrak{Q}}_\star$ *vs* ($A =_\alpha B$)
using *generalized-prop-5237*[*OF* ‹*is-hyps* $\mathcal{H}$› ‹$\mathcal{H} \vdash H \supset^{\mathfrak{Q}} (A =_\alpha B)$›] **by** *simp*
moreover have *rule-R-app*: *is-rule-R-app p ?R* ($\mathcal{S}_2$! *j*) ($\mathcal{S}_2$! *k*)
using *rule-R′-app* **by** *fastforce*
with *S-def* **and** ‹*lset vs* = *S*› **have** $\vdash \forall^{\mathfrak{Q}}_\star$ *vs* ($A =_\alpha B$) $\supset^{\mathfrak{Q}}$ ($C \equiv^{\mathfrak{Q}}$ *?R*)

unfolding ‹$C = \mathcal{S}_2\ !\ j$› **and** ‹$\mathcal{S}_2\ !\ k = A =_\alpha B$› **using** *prop-5239* **by** (*simp only*:)
with ‹*is-hyps* $\mathcal{H}$› **have** $\mathcal{H} \vdash \forall^{\mathcal{Q}}{}_\star\ vs\ (A =_\alpha B) \supset^{\mathcal{Q}} (C \equiv^{\mathcal{Q}} ?R)$
by (*elim derivability-implies-hyp-derivability*)
ultimately show *?thesis*
proof −
let $?A_1 = H \supset^{\mathcal{Q}} C$ **and** $?A_2 = H \supset^{\mathcal{Q}} \forall^{\mathcal{Q}}{}_\star\ vs\ (A =_\alpha B)$
and $?A_3 = \forall^{\mathcal{Q}}{}_\star\ vs\ (A =_\alpha B) \supset^{\mathcal{Q}} (C \equiv^{\mathcal{Q}} ?R)$
let $?hs = [?A_1, ?A_2, ?A_3]$
let $?\mathcal{G} = lset\ ?hs$
from ‹$\mathcal{H} \vdash ?A_1$› **have** $H \in wffs_o$
using *hyp-derivable-form-is-wffso* **by** (*blast dest*: *wffs-from-imp-op*(*1*))
moreover from ‹$\mathcal{H} \vdash ?A_2$› **have** $\forall^{\mathcal{Q}}{}_\star\ vs\ (A =_\alpha B) \in wffs_o$
using *hyp-derivable-form-is-wffso* **by** (*blast dest*: *wffs-from-imp-op*(*2*))
moreover from ‹$C = \mathcal{S}_2\ !\ j$› **and** *rule-R-app* **have** $C \in wffs_o$
using *replacement-preserves-typing* **by** *fastforce*
ultimately have ∗: *is-tautologous* $(?A_1 \supset^{\mathcal{Q}} (?A_2 \supset^{\mathcal{Q}} (?A_3 \supset^{\mathcal{Q}} (H \supset^{\mathcal{Q}} ?R))))$
using ‹$?R \in wffs_o$› **by** (*intro pseudo-rule-R-is-tautologous*)
moreover from ‹$\mathcal{H} \vdash ?A_1$› **and** ‹$\mathcal{H} \vdash ?A_2$› **and** ‹$\mathcal{H} \vdash ?A_3$› **have** *is-hyps* $?\mathcal{G}$
using *hyp-derivable-form-is-wffso* **by** *simp*
moreover from ‹$\mathcal{H} \vdash ?A_1$› **and** ‹$\mathcal{H} \vdash ?A_2$› **and** ‹$\mathcal{H} \vdash ?A_3$› **have** $\forall A \in ?\mathcal{G}.\ \mathcal{H} \vdash A$
by *force*
ultimately show *?thesis*
using *rule-P*(*1*)[**where** $\mathcal{G} = ?\mathcal{G}$ **and** $hs = ?hs$ **and** $B = H \supset^{\mathcal{Q}} ?R$, *OF* ‹*is-hyps* $\mathcal{H}$›] **by** *simp*
qed
qed
qed
moreover from ‹*is-hyp-proof-of* $(\mathcal{H} \cup \{H\})\ \mathcal{S}_1\ \mathcal{S}_2\ P$› **have** $\mathcal{S}_2\ !\ (length\ \mathcal{S}_2 - 1) = P$
using *last-conv-nth* **by** *fastforce*
ultimately show *?thesis*
using ‹*is-hyp-proof-of* $(\mathcal{H} \cup \{H\})\ \mathcal{S}_1\ \mathcal{S}_2\ P$› **by** *force*
qed

lemmas *Deduction-Theorem* = *thm-5240*

We prove a generalization of the Deduction Theorem, namely that if $\mathcal{H} \cup \{H_1, \ldots, H_n\} \vdash P$ then $\mathcal{H} \vdash H_1 \supset^{\mathcal{Q}} (\cdots \supset^{\mathcal{Q}} (H_n \supset^{\mathcal{Q}} P) \cdots)$:

corollary *generalized-deduction-theorem*:
assumes *finite* $\mathcal{H}$ **and** *finite* $\mathcal{H}'$
and $\mathcal{H} \cup \mathcal{H}' \vdash P$
and $lset\ hs = \mathcal{H}'$
shows $\mathcal{H} \vdash hs \supset^{\mathcal{Q}}{}_\star\ P$
using *assms* **proof** (*induction hs arbitrary*: $\mathcal{H}'\ P$ *rule*: *rev-induct*)
case *Nil*
then show *?case*
by *simp*
next
case (*snoc H hs*)
from ‹$lset\ (hs\ @\ [H]) = \mathcal{H}'$› **have** $H \in \mathcal{H}'$
by *fastforce*

from ‹*lset* (*hs* @ [*H*]) = $\mathcal{H}'$› **obtain** $\mathcal{H}''$ **where** $\mathcal{H}'' \cup \{H\} = \mathcal{H}'$ **and** $\mathcal{H}'' = lset\ hs$
by *simp*
from ‹$\mathcal{H}'' \cup \{H\} = \mathcal{H}'$› **and** ‹$\mathcal{H} \cup \mathcal{H}' \vdash P$› **have** $\mathcal{H} \cup \mathcal{H}'' \cup \{H\} \vdash P$
by *fastforce*
with ‹*finite* $\mathcal{H}$› **and** ‹*finite* $\mathcal{H}'$› **and** ‹$\mathcal{H}'' = lset\ hs$› **have** $\mathcal{H} \cup \mathcal{H}'' \vdash H \supset^{\mathfrak{Q}} P$
using *Deduction-Theorem* **by** *simp*
with ‹$\mathcal{H}'' = lset\ hs$› **and** ‹*finite* $\mathcal{H}$› **have** $\mathcal{H} \vdash foldr\ (\supset^{\mathfrak{Q}})\ hs\ (H \supset^{\mathfrak{Q}} P)$
using *snoc.IH* **by** *fastforce*
moreover have $(hs\ @\ [H]) \supset^{\mathfrak{Q}}_{\star} P = hs \supset^{\mathfrak{Q}}_{\star} (H \supset^{\mathfrak{Q}} P)$
by *simp*
ultimately show *?case*
by *auto*
qed

6.41 Proposition 5241

proposition *prop-5241*:
assumes *is-hyps* $\mathcal{G}$
and $\mathcal{H} \vdash A$ **and** $\mathcal{H} \subseteq \mathcal{G}$
shows $\mathcal{G} \vdash A$
proof (*cases* $\mathcal{H} = \{\}$)
case *True*
show *?thesis*
by (*fact derivability-implies-hyp-derivability*[*OF assms*(*2*)[*unfolded True*] *assms*(*1*)])
next
case *False*
then obtain *hs* **where** *lset hs* = $\mathcal{H}$ **and** *hs* ≠ []
using *hyp-derivability-implies-hyp-proof-existence*[*OF assms*(*2*)] **unfolding** *is-hyp-proof-of-def*
by (*metis empty-set finite-list*)
with *assms*(*2*) **have** $\vdash hs \supset^{\mathfrak{Q}}_{\star} A$
using *generalized-deduction-theorem* **by** *force*
moreover from ‹*lset hs* = $\mathcal{H}$› **and** *assms*(*1*,*3*) **have** $\mathcal{G} \vdash H$ **if** $H \in lset\ hs$ **for** *H*
using *that* **by** (*blast intro*: *dv-hyp*)
ultimately show *?thesis*
using *assms*(*1*) **and** *generalized-modus-ponens* **and** *derivability-implies-hyp-derivability* **by** *meson*
qed

6.42 Proposition 5242 (Rule of Existential Generalization)

proposition *prop-5242*:
assumes $A \in wffs_\alpha$ **and** $B \in wffs_o$
and $\mathcal{H} \vdash \mathbf{S}\ \{(x, \alpha) \rightarrowtail A\}\ B$
and *is-free-for A* (x, α) *B*
shows $\mathcal{H} \vdash \exists\, x_\alpha.\ B$
proof −
from *assms*(*3*) **have** *is-hyps* $\mathcal{H}$
by (*blast dest*: *is-derivable-from-hyps.cases*)
then have $\mathcal{H} \vdash \forall\, x_\alpha.\ \sim^{\mathfrak{Q}} B \supset^{\mathfrak{Q}} \sim^{\mathfrak{Q}} \mathbf{S}\ \{(x, \alpha) \rightarrowtail A\}\ B$ (**is** ‹$\mathcal{H} \vdash$ *?C* $\supset^{\mathfrak{Q}} \sim^{\mathfrak{Q}}$ *?D*›)
using *prop-5226*[*OF assms*(*1*) *neg-wff*[*OF assms*(*2*)] *is-free-for-in-neg*[*OF assms*(*4*)]]
unfolding *derived-substitution-simps*(*4*) **using** *derivability-implies-hyp-derivability* **by** (*simp only*:)

moreover have $*$: *is-tautologous* $((?C \supset^{\mathfrak{Q}} \sim^{\mathfrak{Q}} ?D) \supset^{\mathfrak{Q}} (?D \supset^{\mathfrak{Q}} \sim^{\mathfrak{Q}} ?C))$
proof −
have $?C \in wffs_o$ **and** $?D \in wffs_o$
using *assms*(2) **and** *hyp-derivable-form-is-wffso*[*OF assms*(3)] **by** *auto*
then show *?thesis*
by (*fact pseudo-modus-tollens-is-tautologous*)
qed
moreover from *assms*(3) **and** $\langle \mathcal{H} \vdash ?C \supset^{\mathfrak{Q}} \sim^{\mathfrak{Q}} ?D \rangle$ **have** *is-hyps* $\{?C \supset^{\mathfrak{Q}} \sim^{\mathfrak{Q}} ?D,\ ?D\}$
using *hyp-derivable-form-is-wffso* **by** *force*
ultimately show *?thesis*
unfolding *exists-def* **using** *assms*(3)
and *rule-P*(1)
[
where $\mathcal{G} = \{?C \supset^{\mathfrak{Q}} \sim^{\mathfrak{Q}} ?D,\ ?D\}$ **and** $hs = [?C \supset^{\mathfrak{Q}} \sim^{\mathfrak{Q}} ?D,\ ?D]$ **and** $B = \sim^{\mathfrak{Q}} ?C$,
OF ⟨*is-hyps* $\mathcal{H}$⟩
]
by *simp*
qed

lemmas $\exists$ *Gen* = *prop-5242*

6.43 Proposition 5243 (Comprehension Theorem)

context
begin

private lemma *prop-5243-aux*:
assumes $\cdot^{\mathfrak{Q}}_{\star}\ B\ (map\ FVar\ vs) \in wffs_\gamma$
and $B \in wffs_\beta$
and $k < length\ vs$
shows $\beta \neq var\text{-}type\ (vs\ !\ k)$
proof −
from *assms*(1) **obtain** *ts*
where $length\ ts = length\ (map\ FVar\ vs)$
and $*$: $\forall k < length\ (map\ FVar\ vs).\ (map\ FVar\ vs)\ !\ k \in wffs_{ts\ !\ k}$
and $B \in wffs_{foldr\ (\rightarrow)\ ts\ \gamma}$
using *wffs-from-generalized-app* **by** *force*
have $\beta = foldr\ (\rightarrow)\ ts\ \gamma$
by (*fact wff-has-unique-type*[*OF assms*(2) ⟨$B \in wffs_{foldr\ (\rightarrow)\ ts\ \gamma}$⟩])
have $ts = map\ var\text{-}type\ vs$
proof −
have $length\ ts = length\ (map\ var\text{-}type\ vs)$
by (*simp add*: ⟨$length\ ts = length\ (map\ FVar\ vs)$⟩)
moreover have $\forall k < length\ ts.\ ts\ !\ k = (map\ var\text{-}type\ vs)\ !\ k$
proof (*intro allI impI*)
fix *k*
assume $k < length\ ts$
with $*$ **have** $(map\ FVar\ vs)\ !\ k \in wffs_{ts\ !\ k}$
by (*simp add*: ⟨$length\ ts = length\ (map\ FVar\ vs)$⟩)

with ‹k < *length ts*› **and** ‹*length ts* = *length* (*map var-type vs*)›
show *ts* ! k = (*map var-type vs*) ! k
using *surj-pair*[*of vs* ! k] **and** *wff-has-unique-type* **and** *wffs-of-type-intros*(*1*) **by** *force*
qed
ultimately show *?thesis*
using *list-eq-iff-nth-eq* **by** *blast*
qed
with ‹β = *foldr* ($\rightarrow$) *ts* γ› **and** *assms*(*3*) **show** *?thesis*
using *fun-type-atoms-neq-fun-type* **by** (*metis length-map nth-map*)
qed

proposition *prop-5243*:
assumes $B \in wffs_\beta$
and γ = *foldr* ($\rightarrow$) (*map var-type vs*) β
and (u, γ) $\notin$ *free-vars* B
shows $\vdash \exists\, u_\gamma.\ \forall^{\mathfrak{Q}}_\star$ *vs* (($\cdot^{\mathfrak{Q}}_\star$ u_γ (*map FVar vs*)) $=_\beta$ B)
proof (*cases vs* = [])
case *True*
with *assms*(*2*) **have** $\gamma = \beta$
by *simp*
from *assms*(*1*) **have** $u_\beta =_\beta B \in wffs_o$
by *blast*
moreover have $\vdash B =_\beta B$
by (*fact prop-5200*[*OF assms*(*1*)])
then have $\vdash$ **S** {(u, β) $\rightarrowtail$ B} ($u_\beta =_\beta B$)
using *free-var-singleton-substitution-neutrality*[*OF assms*(*3*)] **unfolding** ‹$\gamma = \beta$› **by** *simp*
moreover from *assms*(*3*)[*unfolded* ‹$\gamma = \beta$›] **have** *is-free-for* B (u, β) ($u_\beta =_\beta B$)
by (*intro is-free-for-in-equality*) (*use is-free-at-in-free-vars* **in** *auto*)
ultimately have $\vdash \exists\, u_\beta.\ (u_\beta =_\beta B)$
by (*rule* $\exists$*Gen*[*OF assms*(*1*)])
with ‹$\gamma = \beta$› **and** *True* **show** *?thesis*
by *simp*
next
case *False*
let *?*ϑ = {(u, γ) $\rightarrowtail$ $\lambda^{\mathfrak{Q}}_\star$ *vs* B}
from *assms*(*2*) **have** $*$: (u, γ) $\neq$ v **if** $v \in$ *lset vs* **for** v
using *that* **and** *fun-type-atoms-neq-fun-type* **by** (*metis in-set-conv-nth length-map nth-map snd-conv*)
from *False* **and** *assms*(*1*) **have** $\vdash$ $\cdot^{\mathfrak{Q}}_\star$ ($\lambda^{\mathfrak{Q}}_\star$ *vs* B) (*map FVar vs*) $=_\beta$ B
by (*fact prop-5208*)
then have $\vdash \forall^{\mathfrak{Q}}_\star$ *vs* ($\cdot^{\mathfrak{Q}}_\star$ ($\lambda^{\mathfrak{Q}}_\star$ *vs* B) (*map FVar vs*) $=_\beta$ B)
using *generalized-Gen* **by** *simp*
moreover
have S *?*ϑ ($\forall^{\mathfrak{Q}}_\star$ *vs* (($\cdot^{\mathfrak{Q}}_\star$ u_γ (*map FVar vs*)) $=_\beta$ B)) = $\forall^{\mathfrak{Q}}_\star$ *vs* ($\cdot^{\mathfrak{Q}}_\star$ ($\lambda^{\mathfrak{Q}}_\star$ *vs* B) (*map FVar vs*) $=_\beta$ B)
proof −
from $*$ **have** $**$: *map* (λA. **S** {(u, γ) $\rightarrowtail$ B} A) (*map FVar vs*) = *map FVar vs* **for** B
by (*induction vs*) *fastforce*+
from $*$ **have**
S *?*ϑ ($\forall^{\mathfrak{Q}}_\star$ *vs* (($\cdot^{\mathfrak{Q}}_\star$ u_γ (*map FVar vs*)) $=_\beta$ B)) = $\forall^{\mathfrak{Q}}_\star$ *vs* (**S** *?*ϑ (($\cdot^{\mathfrak{Q}}_\star$ u_γ (*map FVar vs*)) $=_\beta$

```
B))
      using generalized-forall-substitution by force
    also have ... = ∀Q⋆ vs ((S ?ϑ (·Q⋆ uγ (map FVar vs))) =β S {(u, γ) ⇀ λQ⋆ vs B} B)
      by simp
    also from assms(3) have ... = ∀Q⋆ vs ((S ?ϑ (·Q⋆ uγ (map FVar vs))) =β B)
      using free-var-singleton-substitution-neutrality by simp
    also have ... = ∀Q⋆ vs (·Q⋆ S ?ϑ (uγ) (map (λA. S ?ϑ A) (map FVar vs)) =β B)
      using generalized-app-substitution by simp
    also have ... = ∀Q⋆ vs (·Q⋆ (λQ⋆ vs B) (map (λA. S ?ϑ A) (map FVar vs)) =β B)
      by simp
    also from ∗∗ have ... = ∀Q⋆ vs (·Q⋆ (λQ⋆ vs B) (map FVar vs) =β B)
      by presburger
    finally show ?thesis .
  qed
  ultimately have ⊢ S ?ϑ (∀Q⋆ vs (·Q⋆ uγ (map FVar vs) =β B))
    by simp
  moreover from assms(3) have is-free-for (λQ⋆ vs B) (u, γ) (∀Q⋆ vs (·Q⋆ uγ (map FVar vs) =β
B))
    by
      (intro is-free-for-in-generalized-forall is-free-for-in-equality is-free-for-in-generalized-app)
      (use free-vars-of-generalized-abs is-free-at-in-free-vars in ‹fastforce+›)
  moreover have λQ⋆ vs B ∈ wffsγ and ∀Q⋆ vs (·Q⋆ uγ (map FVar vs) =β B) ∈ wffso
  proof −
    have FVar (vs ! k) ∈ wffsvar-type (vs ! k) if k < length vs for k
      using that and surj-pair[of vs ! k] by fastforce
    with assms(2) have ·Q⋆ uγ (map FVar vs) ∈ wffsβ
      using generalized-app-wff[where ts = map var-type vs] by force
    with assms(1) show ∀Q⋆ vs (·Q⋆ uγ (map FVar vs) =β B) ∈ wffso
      by (auto simp only:)
  qed (use assms(1,2) in blast)
  ultimately show ?thesis
    using ∃ Gen by (simp only:)
qed

end
```

6.44 Proposition 5244 (Existential Rule)

The proof in [2] uses the pseudo-rule Q and 2123 of $\mathcal{F}$. Therefore, we instead base our proof on the proof of Theorem 170 in [1]:

```
lemma prop-5244-aux:
  assumes A ∈ wffso and B ∈ wffso
  and (x, α) ∉ free-vars A
  shows ⊢ ∀ xα. (B ⊃Q A) ⊃Q (∃ xα. B ⊃Q A)
proof −
  have B ⊃Q A ∈ wffso
    using assms by blast
  moreover have is-free-for (xα) (x, α) (B ⊃Q A)
```

by *simp*
ultimately have $\vdash \forall\, x_\alpha.\ (B \supset^{\mathfrak{Q}} A) \supset^{\mathfrak{Q}} (B \supset^{\mathfrak{Q}} A)$
using *prop-5226*[**where** $A = x_\alpha$ **and** $B = B \supset^{\mathfrak{Q}} A$, *OF wffs-of-type-intros*(*1*)]
and *identity-singleton-substitution-neutrality* **by** *metis*
moreover have *is-hyps* $\{\forall\, x_\alpha.\ (B \supset^{\mathfrak{Q}} A)\}$
using ‹$B \supset^{\mathfrak{Q}} A \in \mathit{wffs}_o$› **by** *blast*
ultimately have §*1*: $\{\forall\, x_\alpha.\ (B \supset^{\mathfrak{Q}} A)\} \vdash \forall\, x_\alpha.\ (B \supset^{\mathfrak{Q}} A) \supset^{\mathfrak{Q}} (B \supset^{\mathfrak{Q}} A)$
by (*fact derivability-implies-hyp-derivability*)
have §*2*: $\{\forall\, x_\alpha.\ (B \supset^{\mathfrak{Q}} A)\} \vdash \forall\, x_\alpha.\ (B \supset^{\mathfrak{Q}} A)$
using ‹$B \supset^{\mathfrak{Q}} A \in \mathit{wffs}_o$› **by** (*blast intro*: *dv-hyp*)
have §*3*: $\{\forall\, x_\alpha.\ (B \supset^{\mathfrak{Q}} A)\} \vdash {\sim}^{\mathfrak{Q}} A \supset^{\mathfrak{Q}} {\sim}^{\mathfrak{Q}} B$
proof (*intro rule-P*(*1*)
[**where** $\mathcal{H} = \{\forall\, x_\alpha.\ (B \supset^{\mathfrak{Q}} A)\}$ **and** $\mathcal{G} = \{\forall\, x_\alpha.\ (B \supset^{\mathfrak{Q}} A) \supset^{\mathfrak{Q}} (B \supset^{\mathfrak{Q}} A), \forall\, x_\alpha.\ (B \supset^{\mathfrak{Q}} A)\}$])
have *is-tautologous* ($[C \supset^{\mathfrak{Q}} (B \supset^{\mathfrak{Q}} A),\ C] \supset^{\mathfrak{Q}}_{\star} ({\sim}^{\mathfrak{Q}} A \supset^{\mathfrak{Q}} {\sim}^{\mathfrak{Q}} B)$) **if** $C \in \mathit{wffs}_o$ **for** C
proof −
let *?*$\vartheta = \{(\mathfrak{x},\ o) \rightarrowtail A,\ (\mathfrak{y},\ o) \rightarrowtail B,\ (\mathfrak{z},\ o) \rightarrowtail C\}$
have *is-tautology* $((\mathfrak{z}_o \supset^{\mathfrak{Q}} (\mathfrak{y}_o \supset^{\mathfrak{Q}} \mathfrak{x}_o)) \supset^{\mathfrak{Q}} (\mathfrak{z}_o \supset^{\mathfrak{Q}} ({\sim}^{\mathfrak{Q}} \mathfrak{x}_o \supset^{\mathfrak{Q}} {\sim}^{\mathfrak{Q}} \mathfrak{y}_o)))$
(**is** *is-tautology ?A*)
using $\mathcal{V}_B$*-simps* **by** (*auto simp add*: *inj-eq*)
moreover have *is-pwff-substitution ?*ϑ
using *assms*(*1*,*2*) **and** *that* **by** *auto*
moreover have $[C \supset^{\mathfrak{Q}} (B \supset^{\mathfrak{Q}} A),\ C] \supset^{\mathfrak{Q}}_{\star} ({\sim}^{\mathfrak{Q}} A \supset^{\mathfrak{Q}} {\sim}^{\mathfrak{Q}} B) = \mathbf{S}$ *?*ϑ *?A*
by *simp*
ultimately show *?thesis*
by *blast*
qed
then show *is-tautologous* ($[\forall\, x_\alpha.\ (B \supset^{\mathfrak{Q}} A) \supset^{\mathfrak{Q}} (B \supset^{\mathfrak{Q}} A), \forall\, x_\alpha.\ (B \supset^{\mathfrak{Q}} A)] \supset^{\mathfrak{Q}}_{\star} ({\sim}^{\mathfrak{Q}} A \supset^{\mathfrak{Q}} {\sim}^{\mathfrak{Q}} B)$)
using ‹$B \supset^{\mathfrak{Q}} A \in \mathit{wffs}_o$› **and** *forall-wff* **by** *simp*
qed (*use* §*1* §*2* ‹*is-hyps* $\{\forall\, x_\alpha.\ (B \supset^{\mathfrak{Q}} A)\}$› *hyp-derivable-form-is-wffso*[*OF* §*1*] **in** *force*)+
have §*4*: $\{\forall\, x_\alpha.\ (B \supset^{\mathfrak{Q}} A)\} \vdash {\sim}^{\mathfrak{Q}} A \supset^{\mathfrak{Q}} \forall\, x_\alpha.\ {\sim}^{\mathfrak{Q}} B$
using *prop-5237*[*OF* ‹*is-hyps* $\{\forall\, x_\alpha.\ (B \supset^{\mathfrak{Q}} A)\}$› §*3*] **and** *assms*(*3*) **by** *auto*
have §*5*: $\{\forall\, x_\alpha.\ (B \supset^{\mathfrak{Q}} A)\} \vdash \exists\, x_\alpha.\ B \supset^{\mathfrak{Q}} A$
unfolding *exists-def*
proof (*intro rule-P*(*1*)[**where** $\mathcal{H} = \{\forall\, x_\alpha.\ (B \supset^{\mathfrak{Q}} A)\}$ **and** $\mathcal{G} = \{{\sim}^{\mathfrak{Q}} A \supset^{\mathfrak{Q}} \forall\, x_\alpha.\ {\sim}^{\mathfrak{Q}} B\}$])
have *is-tautologous* ($[{\sim}^{\mathfrak{Q}} A \supset^{\mathfrak{Q}} C] \supset^{\mathfrak{Q}}_{\star} ({\sim}^{\mathfrak{Q}} C \supset^{\mathfrak{Q}} A)$) **if** $C \in \mathit{wffs}_o$ **for** C
proof −
let *?*$\vartheta = \{(\mathfrak{x},\ o) \rightarrowtail A,\ (\mathfrak{y},\ o) \rightarrowtail C\}$
have *is-tautology* $(({\sim}^{\mathfrak{Q}} \mathfrak{x}_o \supset^{\mathfrak{Q}} \mathfrak{y}_o) \supset^{\mathfrak{Q}} ({\sim}^{\mathfrak{Q}} \mathfrak{y}_o \supset^{\mathfrak{Q}} \mathfrak{x}_o))$ (**is** *is-tautology ?A*)
using $\mathcal{V}_B$*-simps* **by** (*auto simp add*: *inj-eq*)
moreover have *is-pwff-substitution ?*ϑ
using *assms*(*1*) **and** *that* **by** *auto*
moreover have $[{\sim}^{\mathfrak{Q}} A \supset^{\mathfrak{Q}} C] \supset^{\mathfrak{Q}}_{\star} ({\sim}^{\mathfrak{Q}} C \supset^{\mathfrak{Q}} A) = \mathbf{S}$ *?*ϑ *?A*
by *simp*
ultimately show *?thesis*
by *blast*
qed
then show *is-tautologous* ($[{\sim}^{\mathfrak{Q}} A \supset^{\mathfrak{Q}} \forall\, x_\alpha.\ {\sim}^{\mathfrak{Q}} B] \supset^{\mathfrak{Q}}_{\star} ({\sim}^{\mathfrak{Q}} \forall\, x_\alpha.\ {\sim}^{\mathfrak{Q}} B \supset^{\mathfrak{Q}} A)$)
using *forall-wff*[*OF neg-wff*[*OF assms*(*2*)]] **by** (*simp only*:)

qed (*use* §4 ‹*is-hyps* {∀ x_α. ($B \supset^{\mathfrak{Q}} A$)}› *hyp-derivable-form-is-wffso*[*OF* §4] **in** *force*)+
then show *?thesis*
using *Deduction-Theorem* **by** *simp*
qed

proposition *prop-5244*:
assumes $\mathcal{H}$, $B \vdash A$
and (x, α) ∉ *free-vars* ($\mathcal{H} \cup \{A\}$)
shows $\mathcal{H}$, ∃ x_α. $B \vdash A$
proof −
from *assms*(*1*) **have** *is-hyps* $\mathcal{H}$
using *hyp-derivability-implies-hyp-proof-existence* **by** *force*
then have $\mathcal{H} \vdash B \supset^{\mathfrak{Q}} A$
using *assms*(*1*) **and** *Deduction-Theorem* **by** *simp*
then have $\mathcal{H} \vdash \forall\, x_\alpha.\ (B \supset^{\mathfrak{Q}} A)$
using *Gen* **and** *assms*(*2*) **by** *simp*
moreover have $A \in$ *wffs*$_o$ **and** $B \in$ *wffs*$_o$
by
(
fact hyp-derivable-form-is-wffso[*OF assms*(*1*)],
fact hyp-derivable-form-is-wffso[*OF* ‹$\mathcal{H} \vdash B \supset^{\mathfrak{Q}} A$›, *THEN wffs-from-imp-op*(*1*)]
)
with *assms*(*2*) **and** ‹*is-hyps* $\mathcal{H}$› **have** $\mathcal{H} \vdash \forall\, x_\alpha.\ (B \supset^{\mathfrak{Q}} A) \supset^{\mathfrak{Q}} (\exists\, x_\alpha.\ B \supset^{\mathfrak{Q}} A)$
using *prop-5244-aux*[*THEN derivability-implies-hyp-derivability*] **by** *simp*
ultimately have $\mathcal{H} \vdash \exists\, x_\alpha.\ B \supset^{\mathfrak{Q}} A$
by (*rule MP*)
then have $\mathcal{H}$, ∃ x_α. $B \vdash \exists\, x_\alpha.\ B \supset^{\mathfrak{Q}} A$
using *prop-5241* **and** *exists-wff*[*OF* ‹$B \in$ *wffs*$_o$›] **and** ‹*is-hyps* $\mathcal{H}$›
by (*meson Un-subset-iff empty-subsetI finite.simps finite-Un inf-sup-ord*(*3*) *insert-subsetI*)
moreover from ‹*is-hyps* $\mathcal{H}$› **and** ‹$B \in$ *wffs*$_o$› **have** *is-hyps* ($\mathcal{H} \cup \{\exists\, x_\alpha.\ B\}$)
by *auto*
then have $\mathcal{H}$, ∃ x_α. $B \vdash \exists\, x_\alpha.\ B$
using *dv-hyp* **by** *simp*
ultimately show *?thesis*
using *MP* **by** *blast*
qed

lemmas ∃ *-Rule* = *prop-5244*

6.45 Proposition 5245 (Rule C)

lemma *prop-5245-aux*:
assumes $x \neq y$
and (y, α) ∉ *free-vars* (∃ x_α. B)
and *is-free-for* (y_α) (x, α) B
shows *is-free-for* (x_α) (y, α) **S** {(x, α) ⇀ y_α} B
using *assms*(*2*,*3*) **proof** (*induction B*)
case (*FVar v*)
then show *?case*

```
    using surj-pair[of v] by fastforce
next
  case (FCon k)
  then show ?case
    using surj-pair[of k] by fastforce
next
  case (FApp B₁ B₂)
  from FApp.prems(1) have (y, α) ∉ free-vars (∃ xα. B₁) and (y, α) ∉ free-vars (∃ xα. B₂)
    by force+
  moreover from FApp.prems(2) have is-free-for (yα) (x, α) B₁ and is-free-for (yα) (x, α) B₂
    using is-free-for-from-app by iprover+
  ultimately have is-free-for (xα) (y, α) S {(x, α) ⇀ yα} B₁
    and is-free-for (xα) (y, α) S {(x, α) ⇀ yα} B₂
    using FApp.IH by simp-all
  then have is-free-for (xα) (y, α) ((S {(x, α) ⇀ yα} B₁) ∙ (S {(x, α) ⇀ yα} B₂))
    by (intro is-free-for-to-app)
  then show ?case
    unfolding singleton-substitution-simps(3) .
next
  case (FAbs v B′)
  obtain z and β where v = (z, β)
    by fastforce
  then show ?case
  proof (cases v = (x, α))
    case True
    with FAbs.prems(1) have (y, α) ∉ free-vars (∃ xα. B′)
      by simp
    moreover from assms(1) have (y, α) ≠ (x, α)
      by blast
    ultimately have (y, α) ∉ free-vars B′
      using FAbs.prems(1) by simp
    with ‹(y, α) ≠ (x, α)› have (y, α) ∉ free-vars (λxα. B′)
      by simp
    then have is-free-for (xα) (y, α) (λxα. B′)
      unfolding is-free-for-def using is-free-at-in-free-vars by blast
    then have is-free-for (xα) (y, α) S {(x, α) ⇀ yα} (λxα. B′)
      using singleton-substitution-simps(4) by presburger
    then show ?thesis
      unfolding True .
  next
    case False
    from assms(1) have (y, α) ≠ (x, α)
      by blast
    with FAbs.prems(1) have ∗: (y, α) ∉ free-vars (∃ xα. (λzβ. B′))
      using ‹v = (z, β)› by fastforce
    then show ?thesis
    proof (cases (y, α) ≠ v)
      case True
      from True[unfolded ‹v = (z, β)›] and ∗ have (y, α) ∉ free-vars (∃ xα. B′)
```

```
      by simp
    moreover from False[unfolded ‹v = (z, β)›] have is-free-for (y_α) (x, α) B'
      using is-free-for-from-abs[OF FAbs.prems(2)[unfolded ‹v = (z, β)›]] by blast
    ultimately have is-free-for (x_α) (y, α) (S {(x, α) ⇀ y_α} B')
      by (fact FAbs.IH)
    then have is-free-for (x_α) (y, α) (λz_β. (S {(x, α) ⇀ y_α} B'))
      using False[unfolded ‹v = (z, β)›] by (intro is-free-for-to-abs, fastforce+)
    then show ?thesis
      unfolding singleton-substitution-simps(4) and ‹v = (z, β)› using ‹(z, β) ≠ (x, α)› by auto
  next
    case False
    then have v = (y, α)
      by simp
    have is-free-for (x_α) (y, α) (λy_α. S {(x, α) ⇀ y_α} B')
    proof−
      have (y, α) ∉ free-vars (λy_α. S {(x, α) ⇀ y_α} B')
        by simp
      then show ?thesis
        using is-free-at-in-free-vars by blast
    qed
    with‹v = (y, α)› and ‹(y, α) ≠ (x, α)› show ?thesis
     using singleton-substitution-simps(4) by presburger
  qed
 qed
qed

proposition prop-5245:
 assumes H ⊢ ∃ x_α. B
 and H, S {(x, α) ⇀ y_α} B ⊢ A
 and is-free-for (y_α) (x, α) B
 and (y, α) ∉ free-vars (H ∪ {∃ x_α. B, A})
 shows H ⊢ A
proof −
 from assms(1) have is-hyps H
   by (blast elim: is-derivable-from-hyps.cases)
 from assms(2,4) have H, ∃ y_α. S {(x, α) ⇀ y_α} B ⊢ A
   using ∃-Rule by simp
 then have ∗: H ⊢ (∃ y_α. S {(x, α) ⇀ y_α} B) ⊃^Ω A (is ‹- ⊢ ?F›)
   using Deduction-Theorem and ‹is-hyps H› by blast
 then have H ⊢ ∃ x_α. B ⊃^Ω A
 proof (cases x = y)
   case True
   with ∗ show ?thesis
     using identity-singleton-substitution-neutrality by force
 next
   case False
   from assms(4) have (y, α) ∉ free-vars (∃ x_α. B)
     using free-vars-in-all-vars by auto
   have ∼^Ω S {(x, α) ⇀ y_α} B ∈ wffs_o
```

by
(
fact hyp-derivable-form-is-wffso
[*OF* ∗, *THEN wffs-from-imp-op*(*1*), *THEN wffs-from-exists*, *THEN neg-wff*]
)
moreover from *False* **have** $(x, \alpha) \notin$ *free-vars* $(\sim^{\mathfrak{Q}}$ **S** $\{(x, \alpha) \rightarrowtail y_\alpha\}\ B)$
using *free-var-in-renaming-substitution* **by** *simp*
moreover have *is-free-for* $(x_\alpha)\ (y, \alpha)\ (\sim^{\mathfrak{Q}}$ **S** $\{(x, \alpha) \rightarrowtail y_\alpha\}\ B)$
by (*intro is-free-for-in-neg prop-5245-aux*[*OF False* ‹$(y, \alpha) \notin$ *free-vars* $(\exists\, x_\alpha.\ B)$› *assms*(*3*)])
moreover from *assms*(*3*,*4*) **have** **S** $\{(y, \alpha) \rightarrowtail x_\alpha\}$ **S** $\{(x, \alpha) \rightarrowtail y_\alpha\}\ B = B$
using *identity-singleton-substitution-neutrality* **and** *renaming-substitution-composability*
by *force*
ultimately have $\vdash (\lambda y_\alpha.\ \sim^{\mathfrak{Q}}$ **S** $\{(x, \alpha) \rightarrowtail y_\alpha\}\ B) =_{\alpha\to o} (\lambda x_\alpha.\ \sim^{\mathfrak{Q}} B)$
using α[**where** $A = \sim^{\mathfrak{Q}}$ **S** $\{(x, \alpha) \rightarrowtail y_\alpha\}\ B$] **by** (*metis derived-substitution-simps*(*4*))
then show *?thesis*
by (*rule rule-RR*[*OF disjI1*, **where** $p = [«,»,»,»]$ **and** $C =$ *?F*]) (*use* ∗ **in** *force*)+
qed
with *assms*(*1*) **show** *?thesis*
by (*rule MP*)
qed

lemmas *Rule-C* = *prop-5245*

end

7 Semantics

theory *Semantics*
imports
ZFC-in-HOL.ZFC-Typeclasses
Syntax
Boolean-Algebra
begin

no-notation *funcset* (**infixr** $\to$ *60*)
notation *funcset* (**infixr** $\nrightarrow$ *60*)

abbreviation *vfuncset* :: $V \Rightarrow V \Rightarrow V$ (**infixr** $\longmapsto$ *60*) **where**
$A \longmapsto B \equiv$ *VPi* A (λ-. B)

notation *app* (**infixl** · *300*)

syntax
-vlambda :: *pttrn* ⇒ $V \Rightarrow (V \Rightarrow V) \Rightarrow V$ ((*3***λ-:-** **./** -) [*0*, *0*, *3*] *3*)
translations
λx **:** A**.** $f \rightleftharpoons$ *CONST VLambda* A (λx. f)

lemma *vlambda-extensionality*:
assumes $\bigwedge x.\ x \in$ *elts* $A \Longrightarrow f\ x = g\ x$

shows $(\boldsymbol{\lambda} x : A.\ f\ x) = (\boldsymbol{\lambda} x : A.\ g\ x)$
unfolding *VLambda-def* **using** *assms* **by** *auto*

7.1 Frames

locale *frame* =
fixes $\mathcal{D} :: type \Rightarrow V$
assumes *truth-values-domain-def*: $\mathcal{D}\ o = \mathbb{B}$
and *function-domain-def*: $\forall \alpha\ \beta.\ \mathcal{D}\ (\alpha \to \beta) \leq \mathcal{D}\ \alpha \longmapsto \mathcal{D}\ \beta$
and *domain-nonemptiness*: $\forall \alpha.\ \mathcal{D}\ \alpha \neq 0$
begin

lemma *function-domainD*:
assumes $f \in elts\ (\mathcal{D}\ (\alpha \to \beta))$
shows $f \in elts\ (\mathcal{D}\ \alpha \longmapsto \mathcal{D}\ \beta)$
using *assms* **and** *function-domain-def* **by** *blast*

lemma *vlambda-from-function-domain*:
assumes $f \in elts\ (\mathcal{D}\ (\alpha \to \beta))$
obtains b **where** $f = (\boldsymbol{\lambda} x : \mathcal{D}\ \alpha.\ b\ x)$ **and** $\forall x \in elts\ (\mathcal{D}\ \alpha).\ b\ x \in elts\ (\mathcal{D}\ \beta)$
using *function-domainD*[*OF assms*] **by** (*metis VPi-D eta*)

lemma *app-is-domain-respecting*:
assumes $f \in elts\ (\mathcal{D}\ (\alpha \to \beta))$ **and** $x \in elts\ (\mathcal{D}\ \alpha)$
shows $f \cdot x \in elts\ (\mathcal{D}\ \beta)$
by (*fact VPi-D*[*OF function-domainD*[*OF assms*(*1*)] *assms*(*2*)])

One-element function on $\mathcal{D}\ \alpha$:

definition *one-element-function* :: $V \Rightarrow type \Rightarrow V$ $(\{\text{-}\}_{\text{-}}\ [901,\ 0]\ 900)$ **where**
[*simp*]: $\{x\}_\alpha = (\boldsymbol{\lambda} y : \mathcal{D}\ \alpha.\ bool\text{-}to\text{-}V\ (y = x))$

lemma *one-element-function-is-domain-respecting*:
shows $\{x\}_\alpha \in elts\ (\mathcal{D}\ \alpha \longmapsto \mathcal{D}\ o)$
unfolding *one-element-function-def* **and** *truth-values-domain-def* **by** (*intro VPi-I*) (*simp*, *metis*)

lemma *one-element-function-simps*:
shows $x \in elts\ (\mathcal{D}\ \alpha) \Longrightarrow \{x\}_\alpha \cdot x = \mathbf{T}$
and $[\![\{x,\ y\} \subseteq elts\ (\mathcal{D}\ \alpha);\ y \neq x]\!] \Longrightarrow \{x\}_\alpha \cdot y = \mathbf{F}$
by *simp-all*

lemma *one-element-function-injectivity*:
assumes $\{x,\ x'\} \subseteq elts\ (\mathcal{D}\ i)$ **and** $\{x\}_i = \{x'\}_i$
shows $x = x'$
using *assms*(*1*) **and** *VLambda-eq-D2*[*OF assms*(*2*)[*unfolded one-element-function-def*]]
and *injD*[*OF bool-to-V-injectivity*] **by** *blast*

lemma *one-element-function-uniqueness*:
assumes $x \in elts\ (\mathcal{D}\ i)$
shows $(SOME\ x'.\ x' \in elts\ (\mathcal{D}\ i) \wedge \{x\}_i = \{x'\}_i) = x$

by (*auto simp add*: *assms one-element-function-injectivity*)

Identity relation on $\mathcal{D}\ \alpha$:

definition *identity-relation* :: *type* $\Rightarrow$ *V* (*q*_ [*0*] *100*) **where**
[*simp*]: $q_\alpha = (\boldsymbol{\lambda} x : \mathcal{D}\ \alpha .\ \{x\}_\alpha)$

lemma *identity-relation-is-domain-respecting*:
shows $q_\alpha \in$ *elts* $(\mathcal{D}\ \alpha \longmapsto \mathcal{D}\ \alpha \longmapsto \mathcal{D}\ o)$
using *VPi-I* **and** *one-element-function-is-domain-respecting* **by** *simp*

lemma *q-is-equality*:
assumes $\{x, y\} \subseteq$ *elts* $(\mathcal{D}\ \alpha)$
shows $(q_\alpha) \cdot x \cdot y = \mathbf{T} \longleftrightarrow x = y$
unfolding *identity-relation-def*
using *assms* **and** *injD*[*OF bool-to-V-injectivity*] **by** *fastforce*

Unique member selector:

definition *is-unique-member-selector* :: *V* $\Rightarrow$ *bool* **where**
[*iff*]: *is-unique-member-selector f* $\longleftrightarrow$ $(\forall x \in$ *elts* $(\mathcal{D}\ i).\ f \cdot \{x\}_i = x)$

Assignment:

definition *is-assignment* :: (*var* $\Rightarrow$ *V*) $\Rightarrow$ *bool* **where**
[*iff*]: *is-assignment* $\varphi \longleftrightarrow (\forall x\ \alpha.\ \varphi\ (x, \alpha) \in$ *elts* $(\mathcal{D}\ \alpha))$

end

abbreviation *one-element-function-in* ({-}_^ [*901*, *0*, *0*] *900*) **where**
$\{x\}_\alpha{}^{\mathcal{D}} \equiv$ *frame.one-element-function* $\mathcal{D}\ x\ \alpha$

abbreviation *identity-relation-in* (*q*_^ [*0*, *0*] *100*) **where**
$q_\alpha{}^{\mathcal{D}} \equiv$ *frame.identity-relation* $\mathcal{D}\ \alpha$

ψ is a "v-variant" of φ if ψ is an assignment that agrees with φ except possibly on v:

definition *is-variant-of* :: (*var* $\Rightarrow$ *V*) $\Rightarrow$ *var* $\Rightarrow$ (*var* $\Rightarrow$ *V*) $\Rightarrow$ *bool* (- $\sim$_ - [*51*, *0*, *51*] *50*) **where**
[*iff*]: $\psi \sim_v \varphi \longleftrightarrow (\forall v'.\ v' \neq v \longrightarrow \psi\ v' = \varphi\ v')$

7.2 Pre-models (interpretations)

We use the term "pre-model" instead of "interpretation" since the latter is already a keyword:

locale *premodel* = *frame* +
fixes $\mathcal{J}$ :: *con* $\Rightarrow$ *V*
assumes *Q-denotation*: $\forall \alpha.\ \mathcal{J}$ (*Q-constant-of-type* α) = q_α
and ι*-denotation*: *is-unique-member-selector* ($\mathcal{J}$ *iota-constant*)
and *non-logical-constant-denotation*: $\forall c\ \alpha.\ \neg$ *is-logical-constant* $(c, \alpha) \longrightarrow \mathcal{J}\ (c, \alpha) \in$ *elts* $(\mathcal{D}\ \alpha)$
begin

Wff denotation function:

definition *is-wff-denotation-function* :: $((var \Rightarrow V) \Rightarrow form \Rightarrow V) \Rightarrow bool$ **where**
[*iff*]: *is-wff-denotation-function* $\mathcal{V} \longleftrightarrow$
(
$\forall \varphi.$ *is-assignment* $\varphi \longrightarrow$
$(\forall A\ \alpha.\ A \in wffs_\alpha \longrightarrow \mathcal{V}\ \varphi\ A \in elts\ (\mathcal{D}\ \alpha)) \wedge$ — closure condition, see note in page 186
$(\forall x\ \alpha.\ \mathcal{V}\ \varphi\ (x_\alpha) = \varphi\ (x,\ \alpha)) \wedge$
$(\forall c\ \alpha.\ \mathcal{V}\ \varphi\ (\{\!|c|\!\}_\alpha) = \mathcal{J}\ (c,\ \alpha)) \wedge$
$(\forall A\ B\ \alpha\ \beta.\ A \in wffs_{\beta\to\alpha} \wedge B \in wffs_\beta \longrightarrow \mathcal{V}\ \varphi\ (A \centerdot B) = (\mathcal{V}\ \varphi\ A) \cdot (\mathcal{V}\ \varphi\ B)) \wedge$
$(\forall x\ B\ \alpha\ \beta.\ B \in wffs_\beta \longrightarrow \mathcal{V}\ \varphi\ (\lambda x_\alpha.\ B) = (\boldsymbol{\lambda} z :: \mathcal{D}\ \alpha.\ \mathcal{V}\ (\varphi((x,\ \alpha) := z))\ B))$
)

lemma *wff-denotation-function-is-domain-respecting*:
assumes *is-wff-denotation-function* $\mathcal{V}$
and $A \in wffs_\alpha$
and *is-assignment* φ
shows $\mathcal{V}\ \varphi\ A \in elts\ (\mathcal{D}\ \alpha)$
using *assms* **by** *force*

lemma *wff-var-denotation*:
assumes *is-wff-denotation-function* $\mathcal{V}$
and *is-assignment* φ
shows $\mathcal{V}\ \varphi\ (x_\alpha) = \varphi\ (x,\ \alpha)$
using *assms* **by** *force*

lemma *wff-Q-denotation*:
assumes *is-wff-denotation-function* $\mathcal{V}$
and *is-assignment* φ
shows $\mathcal{V}\ \varphi\ (Q_\alpha) = q_\alpha$
using *assms* **and** *Q-denotation* **by** *force*

lemma *wff-iota-denotation*:
assumes *is-wff-denotation-function* $\mathcal{V}$
and *is-assignment* φ
shows *is-unique-member-selector* $(\mathcal{V}\ \varphi\ \iota)$
using *assms* **and** *ι-denotation* **by** *fastforce*

lemma *wff-non-logical-constant-denotation*:
assumes *is-wff-denotation-function* $\mathcal{V}$
and *is-assignment* φ
and $\neg$ *is-logical-constant* $(c,\ \alpha)$
shows $\mathcal{V}\ \varphi\ (\{\!|c|\!\}_\alpha) = \mathcal{J}\ (c,\ \alpha)$
using *assms* **by** *auto*

lemma *wff-app-denotation*:
assumes *is-wff-denotation-function* $\mathcal{V}$
and *is-assignment* φ
and $A \in wffs_{\beta\to\alpha}$
and $B \in wffs_\beta$
shows $\mathcal{V}\ \varphi\ (A \centerdot B) = \mathcal{V}\ \varphi\ A \cdot \mathcal{V}\ \varphi\ B$

using *assms* **by** *blast*

lemma *wff-abs-denotation*:
assumes *is-wff-denotation-function* $\mathcal{V}$
and *is-assignment* φ
and $B \in wffs_\beta$
shows $\mathcal{V}\ \varphi\ (\lambda x_\alpha.\ B) = (\boldsymbol{\lambda} z : \mathcal{D}\ \alpha.\ \mathcal{V}\ (\varphi((x,\ \alpha) := z))\ B)$
using *assms* **unfolding** *is-wff-denotation-function-def* **by** *metis*

lemma *wff-denotation-function-is-uniquely-determined*:
assumes *is-wff-denotation-function* $\mathcal{V}$
and *is-wff-denotation-function* $\mathcal{V}'$
and *is-assignment* φ
and $A \in wffs$
shows $\mathcal{V}\ \varphi\ A = \mathcal{V}'\ \varphi\ A$
proof −
obtain α **where** $A \in wffs_\alpha$
using *assms*(*4*) **by** *blast*
then show *?thesis*
using *assms*(*3*) **proof** (*induction A arbitrary*: φ)
case *var-is-wff*
with *assms*(*1*,*2*) **show** *?case*
by *auto*
next
case *con-is-wff*
with *assms*(*1*,*2*) **show** *?case*
by *auto*
next
case *app-is-wff*
with *assms*(*1*,*2*) **show** *?case*
using *wff-app-denotation* **by** *metis*
next
case (*abs-is-wff* β *A* α *x*)
have *is-assignment* $(\varphi((x,\ \alpha) := z))$ **if** $z \in elts\ (\mathcal{D}\ \alpha)$ **for** *z*
using *that* **and** *abs-is-wff.prems* **by** *simp*
then have $*$: $\mathcal{V}\ (\varphi((x,\ \alpha) := z))\ A = \mathcal{V}'\ (\varphi((x,\ \alpha) := z))\ A$ **if** $z \in elts\ (\mathcal{D}\ \alpha)$ **for** *z*
using *abs-is-wff.IH* **and** *that* **by** *blast*
have $\mathcal{V}\ \varphi\ (\lambda x_\alpha.\ A) = (\boldsymbol{\lambda} z : \mathcal{D}\ \alpha.\ \mathcal{V}\ (\varphi((x,\ \alpha) := z))\ A)$
by (*fact wff-abs-denotation*[*OF assms*(*1*) *abs-is-wff.prems abs-is-wff.hyps*])
also have $\ldots = (\boldsymbol{\lambda} z : \mathcal{D}\ \alpha.\ \mathcal{V}'\ (\varphi((x,\ \alpha) := z))\ A)$
using $*$ **and** *vlambda-extensionality* **by** *fastforce*
also have $\ldots = \mathcal{V}'\ \varphi\ (\lambda x_\alpha.\ A)$
by (*fact wff-abs-denotation*[*OF assms*(*2*) *abs-is-wff.prems abs-is-wff.hyps*, *symmetric*])
finally show *?case* **.**
qed
qed

end

7.3 General models

type-synonym *model-structure* = (*type* ⇒ *V*) × (*con* ⇒ *V*) × ((*var* ⇒ *V*) ⇒ *form* ⇒ *V*)

The assumption in the following locale implies that there must exist a function that is a wff denotation function for the pre-model, which is a requirement in the definition of general model in [2]:

locale *general-model* = *premodel* +
  fixes $\mathcal{V}$ :: (*var* ⇒ *V*) ⇒ *form* ⇒ *V*
  assumes $\mathcal{V}$*-is-wff-denotation-function*: *is-wff-denotation-function* $\mathcal{V}$
begin

lemma *mixed-beta-conversion*:
  assumes *is-assignment* φ
  and $y \in$ *elts* ($\mathcal{D}$ α)
  and $B \in$ *wffs*$_\beta$
  shows $\mathcal{V}$ φ (λx_α. *B*) · *y* = $\mathcal{V}$ (φ((*x*, α) := *y*)) *B*
  using *wff-abs-denotation*[*OF* $\mathcal{V}$*-is-wff-denotation-function assms*(*1*,*3*)] **and** *beta*[*OF assms*(*2*)] **by** *simp*

lemma *conj-fun-is-domain-respecting*:
  assumes *is-assignment* φ
  shows $\mathcal{V}$ φ ($\wedge_{o\to o\to o}$) ∈ *elts* ($\mathcal{D}$ ($o\to o\to o$))
  using *assms* **and** *conj-fun-wff* **and** $\mathcal{V}$*-is-wff-denotation-function* **by** *auto*

lemma *fully-applied-conj-fun-is-domain-respecting*:
  assumes *is-assignment* φ
  and {*x*, *y*} ⊆ *elts* ($\mathcal{D}$ *o*)
  shows $\mathcal{V}$ φ ($\wedge_{o\to o\to o}$) · *x* · *y* ∈ *elts* ($\mathcal{D}$ *o*)
 using *assms* **and** *conj-fun-is-domain-respecting* **and** *app-is-domain-respecting* **by** (*meson insert-subset*)

lemma *imp-fun-denotation-is-domain-respecting*:
  assumes *is-assignment* φ
  shows $\mathcal{V}$ φ ($\supset_{o\to o\to o}$) ∈ *elts* ($\mathcal{D}$ ($o\to o\to o$))
  using *assms* **and** *imp-fun-wff* **and** $\mathcal{V}$*-is-wff-denotation-function* **by** *simp*

lemma *fully-applied-imp-fun-denotation-is-domain-respecting*:
  assumes *is-assignment* φ
  and {*x*, *y*} ⊆ *elts* ($\mathcal{D}$ *o*)
  shows $\mathcal{V}$ φ ($\supset_{o\to o\to o}$) · *x* · *y* ∈ *elts* ($\mathcal{D}$ *o*)
  using *assms* **and** *imp-fun-denotation-is-domain-respecting* **and** *app-is-domain-respecting*
  by (*meson insert-subset*)

end

abbreviation *is-general-model* :: *model-structure* ⇒ *bool* **where**
  is-general-model $\mathcal{M}$ ≡ *case* $\mathcal{M}$ *of* ($\mathcal{D}$, $\mathcal{J}$, $\mathcal{V}$) ⇒ *general-model* $\mathcal{D}$ $\mathcal{J}$ $\mathcal{V}$

7.4 Standard models

locale *standard-model* = *general-model* +
  assumes *full-function-domain-def*: $\forall\,\alpha\ \beta.\ \mathcal{D}\ (\alpha{\rightarrow}\beta) = \mathcal{D}\ \alpha \longmapsto \mathcal{D}\ \beta$

abbreviation *is-standard-model* :: *model-structure* ⇒ *bool* **where**
  is-standard-model $\mathcal{M} \equiv$ *case* $\mathcal{M}$ *of* $(\mathcal{D}, \mathcal{J}, \mathcal{V})$ ⇒ *standard-model* $\mathcal{D}\ \mathcal{J}\ \mathcal{V}$

lemma *standard-model-is-general-model*:
  assumes *is-standard-model* $\mathcal{M}$
  shows *is-general-model* $\mathcal{M}$
  using *assms* **and** *standard-model.axioms*(*1*) **by** *force*

7.5 Validity

abbreviation *is-assignment-into-frame* (- $\rightsquigarrow$ - [*51*, *51*] *50*) **where**
  $\varphi \rightsquigarrow \mathcal{D} \equiv$ *frame.is-assignment* $\mathcal{D}\ \varphi$

abbreviation *is-assignment-into-model* (- $\rightsquigarrow_M$ - [*51*, *51*] *50*) **where**
  $\varphi \rightsquigarrow_M \mathcal{M} \equiv$ (*case* $\mathcal{M}$ *of* $(\mathcal{D}, \mathcal{J}, \mathcal{V})$ ⇒ $\varphi \rightsquigarrow \mathcal{D}$)

abbreviation *satisfies* (- $\models_-$ - [*50*, *50*, *50*] *50*) **where**
  $\mathcal{M} \models_\varphi A \equiv$ *case* $\mathcal{M}$ *of* $(\mathcal{D}, \mathcal{J}, \mathcal{V})$ ⇒ $\mathcal{V}\ \varphi\ A = \mathbf{T}$

abbreviation *is-satisfiable-in* **where**
  is-satisfiable-in $A\ \mathcal{M} \equiv \exists\,\varphi.\ \varphi \rightsquigarrow_M \mathcal{M} \wedge \mathcal{M} \models_\varphi A$

abbreviation *is-valid-in* (- $\models$ - [*50*, *50*] *50*) **where**
  $\mathcal{M} \models A \equiv \forall\,\varphi.\ \varphi \rightsquigarrow_M \mathcal{M} \longrightarrow \mathcal{M} \models_\varphi A$

abbreviation *is-valid-in-the-general-sense* ($\models$ - [*50*] *50*) **where**
  $\models A \equiv \forall\,\mathcal{M}.$ *is-general-model* $\mathcal{M} \longrightarrow \mathcal{M} \models A$

abbreviation *is-valid-in-the-standard-sense* ($\models_S$ - [*50*] *50*) **where**
  $\models_S A \equiv \forall\,\mathcal{M}.$ *is-standard-model* $\mathcal{M} \longrightarrow \mathcal{M} \models A$

abbreviation *is-true-sentence-in* **where**
  is-true-sentence-in $A\ \mathcal{M} \equiv$ *is-sentence* $A \wedge \mathcal{M} \models_{undefined} A$ — assignments are not meaningful

abbreviation *is-false-sentence-in* **where**
  is-false-sentence-in $A\ \mathcal{M} \equiv$ *is-sentence* $A \wedge \neg\ \mathcal{M} \models_{undefined} A$ — assignments are not meaningful

abbreviation *is-model-for* **where**
  is-model-for $\mathcal{M}\ \mathcal{G} \equiv \forall\,A \in \mathcal{G}.\ \mathcal{M} \models A$

lemma *general-validity-in-standard-validity*:
  assumes $\models A$
  shows $\models_S A$
  using *assms* **and** *standard-model-is-general-model* **by** *blast*

end

8 Soundness

theory *Soundness*
imports
Elementary-Logic
Semantics
begin

no-notation *funcset* (**infixr** $\rightarrow$ *60*)
notation *funcset* (**infixr** $\nrightarrow$ *60*)

8.1 Proposition 5400

proposition (**in** *general-model*) *prop-5400*:
assumes $A \in \mathit{wffs}_\alpha$
and $\varphi \rightsquigarrow \mathcal{D}$ **and** $\psi \rightsquigarrow \mathcal{D}$
and $\forall\, v \in \mathit{free\text{-}vars}\ A.\ \varphi\ v = \psi\ v$
shows $\mathcal{V}\ \varphi\ A = \mathcal{V}\ \psi\ A$
proof −
from *assms*(*1*) **show** *?thesis*
using *assms*(*2*,*3*,*4*) **proof** (*induction A arbitrary*: $\varphi\ \psi$)
case (*var-is-wff* $\alpha\ x$)
have $(x,\ \alpha) \in \mathit{free\text{-}vars}\ (x_\alpha)$
by *simp*
from *assms*(*1*) **and** *var-is-wff.prems*(*1*) **have** $\mathcal{V}\ \varphi\ (x_\alpha) = \varphi\ (x,\ \alpha)$
using $\mathcal{V}$*-is-wff-denotation-function* **by** *fastforce*
also from $\langle (x,\ \alpha) \in \mathit{free\text{-}vars}\ (x_\alpha) \rangle$ **and** *var-is-wff.prems*(*3*) **have** $\ldots = \psi\ (x,\ \alpha)$
by (*simp only*:)
also from *assms*(*1*) **and** *var-is-wff.prems*(*2*) **have** $\ldots = \mathcal{V}\ \psi\ (x_\alpha)$
using $\mathcal{V}$*-is-wff-denotation-function* **by** *fastforce*
finally show *?case* **.**
next
case (*con-is-wff* $\alpha\ c$)
from *assms*(*1*) **and** *con-is-wff.prems*(*1*) **have** $\mathcal{V}\ \varphi\ (\{\!|c|\!\}_\alpha) = \mathcal{J}\ (c,\ \alpha)$
using $\mathcal{V}$*-is-wff-denotation-function* **by** *fastforce*
also from *assms*(*1*) **and** *con-is-wff.prems*(*2*) **have** $\ldots = \mathcal{V}\ \psi\ (\{\!|c|\!\}_\alpha)$
using $\mathcal{V}$*-is-wff-denotation-function* **by** *fastforce*
finally show *?case* **.**
next
case (*app-is-wff* $\alpha\ \beta\ A\ B$)
have *free-vars* $(A \cdot B) =$ *free-vars* $A \cup$ *free-vars* B
by *simp*
with *app-is-wff.prems*(*3*)
have $\forall\, v \in \mathit{free\text{-}vars}\ A.\ \varphi\ v = \psi\ v$ **and** $\forall\, v \in \mathit{free\text{-}vars}\ B.\ \varphi\ v = \psi\ v$
by *blast*+
with *app-is-wff.IH* **and** *app-is-wff.prems*(*1*,*2*) **have** $\mathcal{V}\ \varphi\ A = \mathcal{V}\ \psi\ A$ **and** $\mathcal{V}\ \varphi\ B = \mathcal{V}\ \psi\ B$
by *blast*+

from *assms(1)* **and** *app-is-wff.prems(1)* **and** *app-is-wff.hyps* **have** $\mathcal{V}\ \varphi\ (A \cdot B) = \mathcal{V}\ \varphi\ A \cdot \mathcal{V}\ \varphi\ B$
using *$\mathcal{V}$-is-wff-denotation-function* **by** *fastforce*
also from $\langle \mathcal{V}\ \varphi\ A = \mathcal{V}\ \psi\ A \rangle$ **and** $\langle \mathcal{V}\ \varphi\ B = \mathcal{V}\ \psi\ B \rangle$ **have** $\ldots = \mathcal{V}\ \psi\ A \cdot \mathcal{V}\ \psi\ B$
by (*simp only*:)
also from *assms(1)* **and** *app-is-wff.prems(2)* **and** *app-is-wff.hyps* **have** $\ldots = \mathcal{V}\ \psi\ (A \cdot B)$
using *$\mathcal{V}$-is-wff-denotation-function* **by** *fastforce*
finally show *?case* **.**
next
case (*abs-is-wff* $\beta\ A\ \alpha\ x$)
have *free-vars* $(\lambda x_\alpha.\ A)$ = *free-vars* $A - \{(x,\ \alpha)\}$
by *simp*
with *abs-is-wff.prems(3)* **have** $\forall\, v \in$ *free-vars* $A.\ v \neq (x,\ \alpha) \longrightarrow \varphi\ v = \psi\ v$
by *blast*
then have $\forall\, v \in$ *free-vars* $A.\ (\varphi((x,\ \alpha) := z))\ v = (\psi((x,\ \alpha) := z))\ v$ **if** $z \in$ *elts* $(\mathcal{D}\ \alpha)$ **for** z
by *simp*
moreover from *abs-is-wff.prems(1,2)*
have $\forall\, x'\ \alpha'.\ (\varphi((x,\ \alpha) := z))\ (x',\ \alpha') \in$ *elts* $(\mathcal{D}\ \alpha')$
and $\forall\, x'\ \alpha'.\ (\psi((x,\ \alpha) := z))\ (x',\ \alpha') \in$ *elts* $(\mathcal{D}\ \alpha')$
if $z \in$ *elts* $(\mathcal{D}\ \alpha)$ **for** z
using *that* **by** *force+*
ultimately have *$\mathcal{V}$-φ-ψ-eq*: $\mathcal{V}\ (\varphi((x,\ \alpha) := z))\ A = \mathcal{V}\ (\psi((x,\ \alpha) := z))\ A$ **if** $z \in$ *elts* $(\mathcal{D}\ \alpha)$ **for** z
using *abs-is-wff.IH* **and** *that* **by** *simp*
from *assms(1)* **and** *abs-is-wff.prems(1)* **and** *abs-is-wff.hyps*
have $\mathcal{V}\ \varphi\ (\lambda x_\alpha.\ A) = (\boldsymbol{\lambda} z : \mathcal{D}\ \alpha.\ \mathcal{V}\ (\varphi((x,\ \alpha) := z))\ A)$
using *wff-abs-denotation*[*OF $\mathcal{V}$-is-wff-denotation-function*] **by** *simp*
also from *$\mathcal{V}$-φ-ψ-eq* **have** $\ldots = (\boldsymbol{\lambda} z : \mathcal{D}\ \alpha.\ \mathcal{V}\ (\psi((x,\ \alpha) := z))\ A)$
by (*fact vlambda-extensionality*)
also from *assms(1)* **and** *abs-is-wff.hyps* **have** $\ldots = \mathcal{V}\ \psi\ (\lambda x_\alpha.\ A)$
using *wff-abs-denotation*[*OF $\mathcal{V}$-is-wff-denotation-function abs-is-wff.prems(2)*] **by** *simp*
finally show *?case* **.**
qed
qed

corollary (**in** *general-model*) *closed-wff-is-meaningful-regardless-of-assignment*:
assumes *is-closed-wff-of-type* $A\ \alpha$
and $\varphi \rightsquigarrow \mathcal{D}$ **and** $\psi \rightsquigarrow \mathcal{D}$
shows $\mathcal{V}\ \varphi\ A = \mathcal{V}\ \psi\ A$
using *assms* **and** *prop-5400* **by** *blast*

8.2 Proposition 5401

lemma (**in** *general-model*) *prop-5401-a*:
assumes $\varphi \rightsquigarrow \mathcal{D}$
and $A \in \mathit{wffs}_\alpha$
and $B \in \mathit{wffs}_\beta$
shows $\mathcal{V}\ \varphi\ ((\lambda x_\alpha.\ B) \cdot A) = \mathcal{V}\ (\varphi((x,\ \alpha) := \mathcal{V}\ \varphi\ A))\ B$
proof −
from *assms(2,3)* **have** $\lambda x_\alpha.\ B \in \mathit{wffs}_{\alpha \to \beta}$
by *blast*

with *assms*(*1*,*2*) **have** $\mathcal{V}\ \varphi\ ((\lambda x_\alpha.\ B) \cdot A) = \mathcal{V}\ \varphi\ (\lambda x_\alpha.\ B) \cdot \mathcal{V}\ \varphi\ A$
using $\mathcal{V}$*-is-wff-denotation-function* **by** *blast*
also from *assms*(*1*,*3*) **have** $\ldots = app\ (\boldsymbol{\lambda} z : \mathcal{D}\ \alpha.\ \mathcal{V}\ (\varphi((x,\ \alpha) := z))\ B)\ (\mathcal{V}\ \varphi\ A)$
using *wff-abs-denotation*[*OF* $\mathcal{V}$*-is-wff-denotation-function*] **by** *simp*
also from *assms*(*1*,*2*) **have** $\ldots = \mathcal{V}\ (\varphi((x,\ \alpha) := \mathcal{V}\ \varphi\ A))\ B$
using $\mathcal{V}$*-is-wff-denotation-function* **by** *auto*
finally show *?thesis* **.**
qed

lemma (**in** *general-model*) *prop-5401-b*:
assumes $\varphi \rightsquigarrow \mathcal{D}$
and $A \in wffs_\alpha$
and $B \in wffs_\alpha$
shows $\mathcal{V}\ \varphi\ (A =_\alpha B) = \mathbf{T} \longleftrightarrow \mathcal{V}\ \varphi\ A = \mathcal{V}\ \varphi\ B$
proof −
from *assms* **have** $\{\mathcal{V}\ \varphi\ A,\ \mathcal{V}\ \varphi\ B\} \subseteq elts\ (\mathcal{D}\ \alpha)$
using $\mathcal{V}$*-is-wff-denotation-function* **by** *auto*
have $\mathcal{V}\ \varphi\ (A =_\alpha B) = \mathcal{V}\ \varphi\ (Q_\alpha \cdot A \cdot B)$
by *simp*
also from *assms* **have** $\ldots = \mathcal{V}\ \varphi\ (Q_\alpha \cdot A) \cdot \mathcal{V}\ \varphi\ B$
using $\mathcal{V}$*-is-wff-denotation-function* **by** *blast*
also from *assms* **have** $\ldots = \mathcal{V}\ \varphi\ (Q_\alpha) \cdot \mathcal{V}\ \varphi\ A \cdot \mathcal{V}\ \varphi\ B$
using *Q-wff* **and** *wff-app-denotation*[*OF* $\mathcal{V}$*-is-wff-denotation-function*] **by** *fastforce*
also from *assms*(*1*) **have** $\ldots = (q_\alpha) \cdot \mathcal{V}\ \varphi\ A \cdot \mathcal{V}\ \varphi\ B$
using *Q-denotation* **and** $\mathcal{V}$*-is-wff-denotation-function* **by** *fastforce*
also from $\langle\{\mathcal{V}\ \varphi\ A,\ \mathcal{V}\ \varphi\ B\} \subseteq elts\ (\mathcal{D}\ \alpha)\rangle$ **have** $\ldots = \mathbf{T} \longleftrightarrow \mathcal{V}\ \varphi\ A = \mathcal{V}\ \varphi\ B$
using *q-is-equality* **by** *simp*
finally show *?thesis* **.**
qed

corollary (**in** *general-model*) *prop-5401-b′*:
assumes $\varphi \rightsquigarrow \mathcal{D}$
and $A \in wffs_o$
and $B \in wffs_o$
shows $\mathcal{V}\ \varphi\ (A \equiv^{\mathcal{Q}} B) = \mathbf{T} \longleftrightarrow \mathcal{V}\ \varphi\ A = \mathcal{V}\ \varphi\ B$
using *assms* **and** *prop-5401-b* **by** *auto*

lemma (**in** *general-model*) *prop-5401-c*:
assumes $\varphi \rightsquigarrow \mathcal{D}$
shows $\mathcal{V}\ \varphi\ T_o = \mathbf{T}$
proof −
have $Q_o \in wffs_{o\to o\to o}$
by *blast*
moreover have $\mathcal{V}\ \varphi\ T_o = \mathcal{V}\ \varphi\ (Q_o =_{o\to o\to o} Q_o)$
unfolding *true-def* **..**
ultimately have $\ldots = \mathbf{T} \longleftrightarrow \mathcal{V}\ \varphi\ (Q_o) = \mathcal{V}\ \varphi\ (Q_o)$
using *prop-5401-b* **and** *assms* **by** *blast*
then show *?thesis*
by *simp*

qed

lemma (**in** *general-model*) *prop-5401-d*:
assumes $\varphi \rightsquigarrow \mathcal{D}$
shows $\mathcal{V}\ \varphi\ F_o = \mathbf{F}$
proof −
have $\lambda \mathfrak{x}_o.\ T_o \in \mathit{wffs}_{o \to o}$ **and** $\lambda \mathfrak{x}_o.\ \mathfrak{x}_o \in \mathit{wffs}_{o \to o}$
by *blast+*
moreover have $\mathcal{V}\ \varphi\ F_o = \mathcal{V}\ \varphi\ (\lambda \mathfrak{x}_o.\ T_o =_{o \to o} \lambda \mathfrak{x}_o.\ \mathfrak{x}_o)$
unfolding *false-def* **..**
ultimately have $\mathcal{V}\ \varphi\ F_o = \mathbf{T} \longleftrightarrow \mathcal{V}\ \varphi\ (\lambda \mathfrak{x}_o.\ T_o) = \mathcal{V}\ \varphi\ (\lambda \mathfrak{x}_o.\ \mathfrak{x}_o)$
using *prop-5401-b* **and** *assms* **by** *simp*
moreover have $\mathcal{V}\ \varphi\ (\lambda \mathfrak{x}_o.\ T_o) \neq \mathcal{V}\ \varphi\ (\lambda \mathfrak{x}_o.\ \mathfrak{x}_o)$
proof −
have $\mathcal{V}\ \varphi\ (\lambda \mathfrak{x}_o.\ T_o) = (\boldsymbol{\lambda} z : \mathcal{D}\ o.\ \mathbf{T})$
proof −
from *assms* **have** *T-denotation*: $\mathcal{V}\ (\varphi((\mathfrak{x},\ o) := z))\ T_o = \mathbf{T}$ **if** $z \in \mathit{elts}\ (\mathcal{D}\ o)$ **for** z
using *prop-5401-c* **and** *that* **by** *simp*
from *assms* **have** $\mathcal{V}\ \varphi\ (\lambda \mathfrak{x}_o.\ T_o) = (\boldsymbol{\lambda} z : \mathcal{D}\ o.\ \mathcal{V}\ (\varphi((\mathfrak{x},\ o) := z))\ T_o)$
using *wff-abs-denotation*[*OF* $\mathcal{V}$*-is-wff-denotation-function*] **by** *blast*
also from *assms* **and** *T-denotation* **have** $\ldots = (\boldsymbol{\lambda} z : \mathcal{D}\ o.\ \mathbf{T})$
using *vlambda-extensionality* **by** *fastforce*
finally show *?thesis* **.**
qed
moreover have $\mathcal{V}\ \varphi\ (\lambda \mathfrak{x}_o.\ \mathfrak{x}_o) = (\boldsymbol{\lambda} z : \mathcal{D}\ o.\ z)$
proof −
from *assms* **have** $\mathfrak{x}$*-denotation*: $\mathcal{V}\ (\varphi((\mathfrak{x},\ o) := z))\ (\mathfrak{x}_o) = z$ **if** $z \in \mathit{elts}\ (\mathcal{D}\ o)$ **for** z
using *that* **and** $\mathcal{V}$*-is-wff-denotation-function* **by** *auto*
from *assms* **have** $\mathcal{V}\ \varphi\ (\lambda \mathfrak{x}_o.\ \mathfrak{x}_o) = (\boldsymbol{\lambda} z : \mathcal{D}\ o.\ \mathcal{V}\ (\varphi((\mathfrak{x},\ o) := z))\ (\mathfrak{x}_o))$
using *wff-abs-denotation*[*OF* $\mathcal{V}$*-is-wff-denotation-function*] **by** *blast*
also from $\mathfrak{x}$*-denotation* **have** $\ldots = (\boldsymbol{\lambda} z : (\mathcal{D}\ o).\ z)$
using *vlambda-extensionality* **by** *fastforce*
finally show *?thesis* **.**
qed
moreover have $(\boldsymbol{\lambda} z : \mathcal{D}\ o.\ \mathbf{T}) \neq (\boldsymbol{\lambda} z : \mathcal{D}\ o.\ z)$
proof −
from *assms*(*1*) **have** $(\boldsymbol{\lambda} z : \mathcal{D}\ o.\ \mathbf{T}) \cdot \mathbf{F} = \mathbf{T}$
by (*simp add*: *truth-values-domain-def*)
moreover from *assms*(*1*) **have** $(\boldsymbol{\lambda} z : \mathcal{D}\ o.\ z) \cdot \mathbf{F} = \mathbf{F}$
by (*simp add*: *truth-values-domain-def*)
ultimately show *?thesis*
by (*auto simp add*: *inj-eq*)
qed
ultimately show *?thesis*
by *simp*
qed
moreover from *assms* **have** $\mathcal{V}\ \varphi\ F_o \in \mathit{elts}\ (\mathcal{D}\ o)$
using *false-wff* **and** $\mathcal{V}$*-is-wff-denotation-function* **by** *fast*
ultimately show *?thesis*

using *assms(1)* **by** (*simp add*: *truth-values-domain-def*)
qed

lemma (**in** *general-model*) *prop-5401-e*:
assumes $\varphi \rightsquigarrow \mathcal{D}$
and $\{x, y\} \subseteq \mathit{elts}\ (\mathcal{D}\ o)$
shows $\mathcal{V}\ \varphi\ (\wedge_{o\to o\to o}) \cdot x \cdot y = (\mathit{if}\ x = \mathbf{T} \wedge y = \mathbf{T}\ \mathit{then}\ \mathbf{T}\ \mathit{else}\ \mathbf{F})$
proof −
let $?B_{leq} = \lambda\mathfrak{g}_{o\to o\to o}.\ \mathfrak{g}_{o\to o\to o} \cdot T_o \cdot T_o$
let $?B_{req} = \lambda\mathfrak{g}_{o\to o\to o}.\ \mathfrak{g}_{o\to o\to o} \cdot \mathfrak{x}_o \cdot \mathfrak{y}_o$
let $?B_{eq} = ?B_{leq} =_{(o\to o\to o)\to o} ?B_{req}$
let $?B_{\mathfrak{y}} = \lambda\mathfrak{y}_o.\ ?B_{eq}$
let $?B_{\mathfrak{x}} = \lambda\mathfrak{x}_o.\ ?B_{\mathfrak{y}}$
let $?\varphi' = \varphi((\mathfrak{x},\ o) := x,\ (\mathfrak{y},\ o) := y)$
let $?\varphi'' = \lambda g.\ ?\varphi'((\mathfrak{g},\ o\to o\to o) := g)$
have $\mathfrak{g}_{o\to o\to o} \cdot T_o \in \mathit{wffs}_{o\to o}$
by *blast*
have $\mathfrak{g}_{o\to o\to o} \cdot T_o \cdot T_o \in \mathit{wffs}_o$ **and** $\mathfrak{g}_{o\to o\to o} \cdot \mathfrak{x}_o \cdot \mathfrak{y}_o \in \mathit{wffs}_o$
by *blast+*
have $?B_{leq} \in \mathit{wffs}_{(o\to o\to o)\to o}$ **and** $?B_{req} \in \mathit{wffs}_{(o\to o\to o)\to o}$
by *blast+*
then have $?B_{eq} \in \mathit{wffs}_o$ **and** $?B_{\mathfrak{y}} \in \mathit{wffs}_{o\to o}$ **and** $?B_{\mathfrak{x}} \in \mathit{wffs}_{o\to o\to o}$
by *blast+*
have $\mathcal{V}\ \varphi\ (\wedge_{o\to o\to o}) \cdot x \cdot y = \mathcal{V}\ \varphi\ ?B_{\mathfrak{x}} \cdot x \cdot y$
by *simp*
also from *assms* **and** ‹$?B_{\mathfrak{y}} \in \mathit{wffs}_{o\to o}$› **have** $\ldots = \mathcal{V}\ (\varphi((\mathfrak{x},\ o) := x))\ ?B_{\mathfrak{y}} \cdot y$
using *mixed-beta-conversion* **by** *simp*
also from *assms* **and** ‹$?B_{eq} \in \mathit{wffs}_o$› **have** $\ldots = \mathcal{V}\ ?\varphi'\ ?B_{eq}$
using *mixed-beta-conversion* **by** *simp*
finally have $\mathcal{V}\ \varphi\ (\wedge_{o\to o\to o}) \cdot x \cdot y = \mathbf{T} \longleftrightarrow \mathcal{V}\ ?\varphi'\ ?B_{leq} = \mathcal{V}\ ?\varphi'\ ?B_{req}$
using *assms* **and** ‹$?B_{leq} \in \mathit{wffs}_{(o\to o\to o)\to o}$› **and** ‹$?B_{req} \in \mathit{wffs}_{(o\to o\to o)\to o}$› **and** *prop-5401-b*
by *simp*
also have $\ldots \longleftrightarrow (\boldsymbol{\lambda} g : \mathcal{D}\ (o\to o\to o).\ g \cdot \mathbf{T} \cdot \mathbf{T}) = (\boldsymbol{\lambda} g : \mathcal{D}\ (o\to o\to o).\ g \cdot x \cdot y)$
proof −
have *leq*: $\mathcal{V}\ ?\varphi'\ ?B_{leq} = (\boldsymbol{\lambda} g : \mathcal{D}\ (o\to o\to o).\ g \cdot \mathbf{T} \cdot \mathbf{T})$
and *req*: $\mathcal{V}\ ?\varphi'\ ?B_{req} = (\boldsymbol{\lambda} g : \mathcal{D}\ (o\to o\to o).\ g \cdot x \cdot y)$
proof −
from *assms(1,2)* **have** *is-assg-*φ'': $?\varphi''\ g \rightsquigarrow \mathcal{D}$ **if** $g \in \mathit{elts}\ (\mathcal{D}\ (o\to o\to o))$ **for** g
using *that* **by** *auto*
have *side-eq-denotation*:
$\mathcal{V}\ ?\varphi'\ (\lambda\mathfrak{g}_{o\to o\to o}.\ \mathfrak{g}_{o\to o\to o} \cdot A \cdot B) = (\boldsymbol{\lambda} g : \mathcal{D}\ (o\to o\to o).\ g \cdot \mathcal{V}\ (?\varphi''\ g)\ A \cdot \mathcal{V}\ (?\varphi''\ g)\ B)$
if $A \in \mathit{wffs}_o$ **and** $B \in \mathit{wffs}_o$ **for** A **and** B
proof −
from *that* **have** $\mathfrak{g}_{o\to o\to o} \cdot A \cdot B \in \mathit{wffs}_o$
by *blast*
have $\mathcal{V}\ (?\varphi''\ g)\ (\mathfrak{g}_{o\to o\to o} \cdot A \cdot B) = g \cdot \mathcal{V}\ (?\varphi''\ g)\ A \cdot \mathcal{V}\ (?\varphi''\ g)\ B$
if $g \in \mathit{elts}\ (\mathcal{D}\ (o\to o\to o))$ **for** g
proof −
from ‹$A \in \mathit{wffs}_o$› **have** $\mathfrak{g}_{o\to o\to o} \cdot A \in \mathit{wffs}_{o\to o}$

by *blast*
with *that* **and** *is-assg-φ″* **and** ‹$B \in wffs_o$› **have**
$\mathcal{V}\ (?\varphi''\ g)\ (\mathfrak{g}_{o\to o\to o} \cdot A \cdot B) = \mathcal{V}\ (?\varphi''\ g)\ (\mathfrak{g}_{o\to o\to o} \cdot A) \cdot \mathcal{V}\ (?\varphi''\ g)\ B$
using *wff-app-denotation*[*OF* $\mathcal{V}$*-is-wff-denotation-function*] **by** *simp*
also from *that* **and** ‹$A \in wffs_o$› **and** *is-assg-φ″* **have**
$\ldots = \mathcal{V}\ (?\varphi''\ g)\ (\mathfrak{g}_{o\to o\to o}) \cdot \mathcal{V}\ (?\varphi''\ g)\ A \cdot \mathcal{V}\ (?\varphi''\ g)\ B$
by (*metis* $\mathcal{V}$*-is-wff-denotation-function wff-app-denotation wffs-of-type-intros*(*1*))
finally show *?thesis*
using *that* **and** *is-assg-φ″* **and** $\mathcal{V}$*-is-wff-denotation-function* **by** *auto*
qed
moreover from *assms* **have** *is-assignment ?φ′*
by *auto*
with ‹$\mathfrak{g}_{o\to o\to o} \cdot A \cdot B \in wffs_o$› **have**
$\mathcal{V}\ ?\varphi'\ (\lambda\mathfrak{g}_{o\to o\to o}.\ \mathfrak{g}_{o\to o\to o} \cdot A \cdot B) = (\boldsymbol{\lambda} g : \mathcal{D}\ (o\to o\to o).\ \mathcal{V}\ (?\varphi''\ g)\ (\mathfrak{g}_{o\to o\to o} \cdot A \cdot B))$
using *wff-abs-denotation*[*OF* $\mathcal{V}$*-is-wff-denotation-function*] **by** *simp*
ultimately show *?thesis*
using *vlambda-extensionality* **by** *fastforce*
qed
— Proof of *leq*:
show $\mathcal{V}\ ?\varphi'\ ?B_{leq} = (\boldsymbol{\lambda} g : \mathcal{D}\ (o\to o\to o).\ g \cdot \mathbf{T} \cdot \mathbf{T})$
proof −
have $\mathcal{V}\ (?\varphi''\ g)\ T_o = \mathbf{T}$ **if** $g \in elts\ (\mathcal{D}\ (o\to o\to o))$ **for** g
using *that* **and** *is-assg-φ″* **and** *prop-5401-c* **by** *simp*
then show *?thesis*
using *side-eq-denotation* **and** *true-wff* **and** *vlambda-extensionality* **by** *fastforce*
qed
— Proof of *req*:
show $\mathcal{V}\ ?\varphi'\ ?B_{req} = (\boldsymbol{\lambda} g : \mathcal{D}\ (o\to o\to o).\ g \cdot x \cdot y)$
proof −
from *is-assg-φ″* **have** $\mathcal{V}\ (?\varphi''\ g)\ (\mathfrak{x}_o) = x$ **and** $\mathcal{V}\ (?\varphi''\ g)\ (\mathfrak{y}_o) = y$
if $g \in elts\ (\mathcal{D}\ (o\to o\to o))$ **for** g
using *that* **and** $\mathcal{V}$*-is-wff-denotation-function* **by** *auto*
with *side-eq-denotation* **show** *?thesis*
using *wffs-of-type-intros*(*1*) **and** *vlambda-extensionality* **by** *fastforce*
qed
qed
then show *?thesis*
by *auto*
qed
also have $\ldots \longleftrightarrow (\forall\, g \in elts\ (\mathcal{D}\ (o\to o\to o)).\ g \cdot \mathbf{T} \cdot \mathbf{T} = g \cdot x \cdot y)$
using *vlambda-extensionality* **and** *VLambda-eq-D2* **by** *fastforce*
finally have *and-eqv*:
$\mathcal{V}\ \varphi\ (\wedge_{o\to o\to o}) \cdot x \cdot y = \mathbf{T} \longleftrightarrow (\forall\, g \in elts\ (\mathcal{D}\ (o\to o\to o)).\ g \cdot \mathbf{T} \cdot \mathbf{T} = g \cdot x \cdot y)$
by *blast*
then show *?thesis*
proof −
from *assms*(*1*,*2*) **have** *is-assg-1*: $\varphi((\mathfrak{x},\ o) := \mathbf{T}) \rightsquigarrow \mathcal{D}$
by (*simp add*: *truth-values-domain-def*)
then have *is-assg-2*: $\varphi((\mathfrak{x},\ o) := \mathbf{T},\ (\mathfrak{y},\ o) := \mathbf{T}) \rightsquigarrow \mathcal{D}$

unfolding *is-assignment-def* **by** (*metis fun-upd-apply prod.sel*(*2*))
from *assms* **consider** (*a*) $x = \mathbf{T} \wedge y = \mathbf{T}$ | (*b*) $x \neq \mathbf{T}$ | (*c*) $y \neq \mathbf{T}$
by *blast*
then show *?thesis*
proof *cases*
case *a*
then have $g \cdot \mathbf{T} \cdot \mathbf{T} = g \cdot x \cdot y$ **if** $g \in$ *elts* ($\mathcal{D}$ ($o \to o \to o$)) **for** g
by *simp*
with *a* **and** *and-eqv* **show** *?thesis*
by *simp*
next
case *b*
let *?g-witness* = $\lambda\mathfrak{x}_o.\ \lambda\mathfrak{y}_o.\ \mathfrak{x}_o$
have $\lambda\mathfrak{y}_o.\ \mathfrak{x}_o \in$ *wffs*$_{o \to o}$
by *blast*
then have *is-closed-wff-of-type ?g-witness* ($o \to o \to o$)
by *force*
moreover from *assms* **have** *is-assg-*φ': $?\varphi' \rightsquigarrow \mathcal{D}$
by *simp*
ultimately have $\mathcal{V}\ \varphi$ *?g-witness* $\cdot\ \mathbf{T} \cdot \mathbf{T} = \mathcal{V}\ ?\varphi'$ *?g-witness* $\cdot\ \mathbf{T} \cdot \mathbf{T}$
using *assms*(*1*) **and** *closed-wff-is-meaningful-regardless-of-assignment* **by** *metis*
also from *assms* **and** ‹$\lambda\mathfrak{y}_o.\ \mathfrak{x}_o \in$ *wffs*$_{o \to o}$› **have**
$\mathcal{V}\ ?\varphi'$ *?g-witness* $\cdot\ \mathbf{T} \cdot \mathbf{T} = \mathcal{V}\ (?\varphi'((\mathfrak{x},\ o) := \mathbf{T}))\ (\lambda\mathfrak{y}_o.\ \mathfrak{x}_o) \cdot \mathbf{T}$
using *mixed-beta-conversion* **and** *truth-values-domain-def* **by** *auto*
also from *assms*(*1*) **and** ‹$\lambda\mathfrak{y}_o.\ \mathfrak{x}_o \in$ *wffs*$_{o \to o}$› **and** *is-assg-1* **and** *calculation* **have**
$\ldots = \mathcal{V}\ (?\varphi'((\mathfrak{x},\ o) := \mathbf{T},\ (\mathfrak{y},\ o) := \mathbf{T}))\ (\mathfrak{x}_o)$
using *mixed-beta-conversion* **and** *is-assignment-def*
by (*metis fun-upd-same fun-upd-twist fun-upd-upd wffs-of-type-intros*(*1*))
also have $\ldots = \mathbf{T}$
using *is-assg-2* **and** $\mathcal{V}$*-is-wff-denotation-function* **by** *fastforce*
finally have $\mathcal{V}\ \varphi$ *?g-witness* $\cdot\ \mathbf{T} \cdot \mathbf{T} = \mathbf{T}$ **.**
with *b* **have** $\mathcal{V}\ \varphi$ *?g-witness* $\cdot\ \mathbf{T} \cdot \mathbf{T} \neq x$
by *blast*
moreover have $x = \mathcal{V}\ \varphi$ *?g-witness* $\cdot\ x \cdot y$
proof −
from *is-assg-*φ' **have** $x = \mathcal{V}\ ?\varphi'\ (\mathfrak{x}_o)$
using $\mathcal{V}$*-is-wff-denotation-function* **by** *auto*
also from *assms*(*2*) **and** *is-assg-*φ' **have** $\ldots = \mathcal{V}\ ?\varphi'\ (\lambda\mathfrak{y}_o.\ \mathfrak{x}_o) \cdot y$
using *wffs-of-type-intros*(*1*)[**where** $x = \mathfrak{x}$ **and** $\alpha = o$]
by (*simp add*: *mixed-beta-conversion* $\mathcal{V}$*-is-wff-denotation-function*)
also from *assms*(*2*) **have** $\ldots = \mathcal{V}\ ?\varphi'$ *?g-witness* $\cdot\ x \cdot y$
using *is-assg-*φ' **and** ‹$\lambda\mathfrak{y}_o.\ \mathfrak{x}_o \in$ *wffs*$_{o \to o}$›
by (*simp add*: *mixed-beta-conversion fun-upd-twist*)
also from *assms*(*1*,*2*) **have** $\ldots = \mathcal{V}\ \varphi$ *?g-witness* $\cdot\ x \cdot y$
using *is-assg-*φ' **and** ‹*is-closed-wff-of-type ?g-witness* ($o \to o \to o$)›
and *closed-wff-is-meaningful-regardless-of-assignment* **by** *metis*
finally show *?thesis* **.**
qed
moreover from *assms*(*1*,*2*) **have** $\mathcal{V}\ \varphi$ *?g-witness* $\in$ *elts* ($\mathcal{D}$ ($o \to o \to o$))

using ‹*is-closed-wff-of-type ?g-witness* ($o \to o \to o$)› **and** $\mathcal{V}$*-is-wff-denotation-function* **by** *simp*
ultimately have $\exists\, g \in \mathit{elts}\ (\mathcal{D}\ (o \to o \to o)).\ g \cdot \mathbf{T} \cdot \mathbf{T} \neq g \cdot x \cdot y$
by *auto*
moreover from *assms* **have** $\mathcal{V}\ \varphi\ (\wedge_{o \to o \to o}) \cdot x \cdot y \in \mathit{elts}\ (\mathcal{D}\ o)$
by (*rule fully-applied-conj-fun-is-domain-respecting*)
ultimately have $\mathcal{V}\ \varphi\ (\wedge_{o \to o \to o}) \cdot x \cdot y = \mathbf{F}$
using *and-eqv* **and** *truth-values-domain-def* **by** *fastforce*
with *b* **show** *?thesis*
by *simp*
next
case *c*
let *?g-witness* $= \lambda \mathfrak{x}_o.\ \lambda \mathfrak{y}_o.\ \mathfrak{y}_o$
have $\lambda \mathfrak{y}_o.\ \mathfrak{y}_o \in \mathit{wffs}_{o \to o}$
by *blast*
then have *is-closed-wff-of-type ?g-witness* ($o \to o \to o$)
by *force*
moreover from *assms*(*1*,*2*) **have** *is-assg-*φ': $?\varphi' \rightsquigarrow \mathcal{D}$
by *simp*
ultimately have $\mathcal{V}\ \varphi$ *?g-witness* $\cdot\ \mathbf{T} \cdot \mathbf{T} = \mathcal{V}\ ?\varphi'$ *?g-witness* $\cdot\ \mathbf{T} \cdot \mathbf{T}$
using *assms*(*1*) **and** *closed-wff-is-meaningful-regardless-of-assignment* **by** *metis*
also from *is-assg-1* **and** *is-assg-*φ' **have** $\ldots = \mathcal{V}\ (?\varphi'((\mathfrak{x},\ o) := \mathbf{T}))\ (\lambda \mathfrak{y}_o.\ \mathfrak{y}_o) \cdot \mathbf{T}$
using ‹$\lambda \mathfrak{y}_o.\ \mathfrak{y}_o \in \mathit{wffs}_{o \to o}$› **and** *mixed-beta-conversion* **and** *truth-values-domain-def* **by** *auto*
also from *assms*(*1*) **and** ‹$\lambda \mathfrak{y}_o.\ \mathfrak{y}_o \in \mathit{wffs}_{o \to o}$› **and** *is-assg-1* **and** *calculation* **have**
$\ldots = \mathcal{V}\ (?\varphi'((\mathfrak{x},\ o) := \mathbf{T},\ (\mathfrak{y},\ o) := \mathbf{T}))\ (\mathfrak{y}_o)$
using *mixed-beta-conversion* **and** *is-assignment-def*
by (*metis fun-upd-same fun-upd-twist fun-upd-upd wffs-of-type-intros*(*1*))
also have $\ldots = \mathbf{T}$
using *is-assg-2* **and** $\mathcal{V}$*-is-wff-denotation-function* **by** *force*
finally have $\mathcal{V}\ \varphi$ *?g-witness* $\cdot\ \mathbf{T} \cdot \mathbf{T} = \mathbf{T}$ **.**
with *c* **have** $\mathcal{V}\ \varphi$ *?g-witness* $\cdot\ \mathbf{T} \cdot \mathbf{T} \neq y$
by *blast*
moreover have $y = \mathcal{V}\ \varphi$ *?g-witness* $\cdot\ x \cdot y$
proof −
from *assms*(*2*) **and** *is-assg-*φ' **have** $y = \mathcal{V}\ ?\varphi'\ (\lambda \mathfrak{y}_o.\ \mathfrak{y}_o) \cdot y$
using *wffs-of-type-intros*(*1*)[**where** $x = \mathfrak{y}$ **and** $\alpha = o$]
and $\mathcal{V}$*-is-wff-denotation-function* **and** *mixed-beta-conversion* **by** *auto*
also from *assms*(*2*) **and** ‹$\lambda \mathfrak{y}_o.\ \mathfrak{y}_o \in \mathit{wffs}_{o \to o}$› **have** $\ldots = \mathcal{V}\ ?\varphi'$ *?g-witness* $\cdot\ x \cdot y$
using *is-assg-*φ' **by** (*simp add*: *mixed-beta-conversion fun-upd-twist*)
also from *assms*(*1*,*2*) **have** $\ldots = \mathcal{V}\ \varphi$ *?g-witness* $\cdot\ x \cdot y$
using *is-assg-*φ' **and** ‹*is-closed-wff-of-type ?g-witness* ($o \to o \to o$)›
and *closed-wff-is-meaningful-regardless-of-assignment* **by** *metis*
finally show *?thesis* **.**
qed
moreover from *assms*(*1*) **have** $\mathcal{V}\ \varphi$ *?g-witness* $\in \mathit{elts}\ (\mathcal{D}\ (o \to o \to o))$
using ‹*is-closed-wff-of-type ?g-witness* ($o \to o \to o$)› **and** $\mathcal{V}$*-is-wff-denotation-function* **by** *auto*
ultimately have $\exists\, g \in \mathit{elts}\ (\mathcal{D}\ (o \to o \to o)).\ g \cdot \mathbf{T} \cdot \mathbf{T} \neq g \cdot x \cdot y$
by *auto*
moreover from *assms* **have** $\mathcal{V}\ \varphi\ (\wedge_{o \to o \to o}) \cdot x \cdot y \in \mathit{elts}\ (\mathcal{D}\ o)$
by (*rule fully-applied-conj-fun-is-domain-respecting*)

ultimately have $\mathcal{V}\ \varphi\ (\wedge_{o\to o\to o})\cdot x\cdot y = \mathbf{F}$
using *and-eqv* **and** *truth-values-domain-def* **by** *fastforce*
with *c* **show** *?thesis*
by *simp*
qed
qed
qed

corollary (**in** *general-model*) *prop-5401-e'*:
assumes $\varphi \rightsquigarrow \mathcal{D}$
and $A \in \mathit{wffs}_o$ **and** $B \in \mathit{wffs}_o$
shows $\mathcal{V}\ \varphi\ (A \wedge^{\mathcal{Q}} B) = \mathcal{V}\ \varphi\ A \wedge \mathcal{V}\ \varphi\ B$
proof −
from *assms* **have** $\{\mathcal{V}\ \varphi\ A,\ \mathcal{V}\ \varphi\ B\} \subseteq \mathit{elts}\ (\mathcal{D}\ o)$
using *$\mathcal{V}$-is-wff-denotation-function* **by** *simp*
from *assms(2)* **have** $\wedge_{o\to o\to o}\centerdot A \in \mathit{wffs}_{o\to o}$
by *blast*
have $\mathcal{V}\ \varphi\ (A \wedge^{\mathcal{Q}} B) = \mathcal{V}\ \varphi\ (\wedge_{o\to o\to o}\centerdot A\centerdot B)$
by *simp*
also from *assms* **have** $\ldots = \mathcal{V}\ \varphi\ (\wedge_{o\to o\to o}\centerdot A)\cdot \mathcal{V}\ \varphi\ B$
using *$\mathcal{V}$-is-wff-denotation-function* **and** $\langle \wedge_{o\to o\to o}\centerdot A \in \mathit{wffs}_{o\to o}\rangle$ **by** *blast*
also from *assms* **have** $\ldots = \mathcal{V}\ \varphi\ (\wedge_{o\to o\to o})\cdot \mathcal{V}\ \varphi\ A\cdot \mathcal{V}\ \varphi\ B$
using *$\mathcal{V}$-is-wff-denotation-function* **and** *conj-fun-wff* **by** *fastforce*
also from *assms(1,2)* **have** $\ldots = (\mathit{if}\ \mathcal{V}\ \varphi\ A = \mathbf{T} \wedge \mathcal{V}\ \varphi\ B = \mathbf{T}\ \mathit{then}\ \mathbf{T}\ \mathit{else}\ \mathbf{F})$
using $\langle\{\mathcal{V}\ \varphi\ A,\ \mathcal{V}\ \varphi\ B\} \subseteq \mathit{elts}\ (\mathcal{D}\ o)\rangle$ **and** *prop-5401-e* **by** *simp*
also have $\ldots = \mathcal{V}\ \varphi\ A \wedge \mathcal{V}\ \varphi\ B$
using *truth-values-domain-def* **and** $\langle\{\mathcal{V}\ \varphi\ A,\ \mathcal{V}\ \varphi\ B\} \subseteq \mathit{elts}\ (\mathcal{D}\ o)\rangle$ **by** *fastforce*
finally show *?thesis* **.**
qed

lemma (**in** *general-model*) *prop-5401-f*:
assumes $\varphi \rightsquigarrow \mathcal{D}$
and $\{x,\ y\} \subseteq \mathit{elts}\ (\mathcal{D}\ o)$
shows $\mathcal{V}\ \varphi\ (\supset_{o\to o\to o})\cdot x\cdot y = (\mathit{if}\ x = \mathbf{T} \wedge y = \mathbf{F}\ \mathit{then}\ \mathbf{F}\ \mathit{else}\ \mathbf{T})$
proof −
let $?\varphi' = \varphi((\mathfrak{x},\ o) := x,\ (\mathfrak{y},\ o) := y)$
from *assms(2)* **have** $\{x,\ y\} \subseteq \mathit{elts}\ \mathbb{B}$
unfolding *truth-values-domain-def* **.**
have $(\mathfrak{x}_o \equiv^{\mathcal{Q}} \mathfrak{x}_o \wedge^{\mathcal{Q}} \mathfrak{y}_o) \in \mathit{wffs}_o$
by *blast*
then have $\lambda\mathfrak{y}_o.\ (\mathfrak{x}_o \equiv^{\mathcal{Q}} \mathfrak{x}_o \wedge^{\mathcal{Q}} \mathfrak{y}_o) \in \mathit{wffs}_{o\to o}$
by *blast*
from *assms* **have** *is-assg-φ'*: $?\varphi' \rightsquigarrow \mathcal{D}$
by *simp*
from *assms(1)* **have** $\mathcal{V}\ ?\varphi'\ (\mathfrak{x}_o) = x$ **and** $\mathcal{V}\ ?\varphi'\ (\mathfrak{y}_o) = y$
using *is-assg-φ'* **and** *$\mathcal{V}$-is-wff-denotation-function* **by** *force+*
have $\mathcal{V}\ \varphi\ (\supset_{o\to o\to o})\cdot x\cdot y = \mathcal{V}\ \varphi\ (\lambda\mathfrak{x}_o.\ \lambda\mathfrak{y}_o.\ (\mathfrak{x}_o \equiv^{\mathcal{Q}} \mathfrak{x}_o \wedge^{\mathcal{Q}} \mathfrak{y}_o))\cdot x\cdot y$
by *simp*
also from *assms* **have** $\ldots = \mathcal{V}\ (\varphi((\mathfrak{x},\ o) := x))\ (\lambda\mathfrak{y}_o.\ (\mathfrak{x}_o \equiv^{\mathcal{Q}} \mathfrak{x}_o \wedge^{\mathcal{Q}} \mathfrak{y}_o))\cdot y$

using ‹$\lambda\mathfrak{y}_o.\ (\mathfrak{x}_o \equiv^{\mathcal{Q}} \mathfrak{x}_o \wedge^{\mathcal{Q}} \mathfrak{y}_o) \in \mathit{wffs}_{o\to o}$› **and** *mixed-beta-conversion* **by** *simp*
also from *assms* **have** $\ldots = \mathcal{V}\ ?\varphi'\ (\mathfrak{x}_o \equiv^{\mathcal{Q}} \mathfrak{x}_o \wedge^{\mathcal{Q}} \mathfrak{y}_o)$
using *mixed-beta-conversion* **and** ‹$(\mathfrak{x}_o \equiv^{\mathcal{Q}} \mathfrak{x}_o \wedge^{\mathcal{Q}} \mathfrak{y}_o) \in \mathit{wffs}_o$› **by** *simp*
finally have $\mathcal{V}\ \varphi\ (\supset_{o\to o\to o}) \cdot x \cdot y = \mathbf{T} \longleftrightarrow \mathcal{V}\ ?\varphi'\ (\mathfrak{x}_o) = \mathcal{V}\ ?\varphi'\ (\mathfrak{x}_o \wedge^{\mathcal{Q}} \mathfrak{y}_o)$
using *prop-5401-b'*[*OF is-assg-*φ'] **and** *conj-op-wff* **and** *wffs-of-type-intros*(*1*) **by** *simp*
also have $\ldots \longleftrightarrow x = x \wedge y$
unfolding *prop-5401-e'*[*OF is-assg-*φ' *wffs-of-type-intros*(*1*) *wffs-of-type-intros*(*1*)]
and ‹$\mathcal{V}\ ?\varphi'\ (\mathfrak{x}_o) = x$› **and** ‹$\mathcal{V}\ ?\varphi'\ (\mathfrak{y}_o) = y$› **..**
also have $\ldots \longleftrightarrow x = (\mathit{if}\ x = \mathbf{T} \wedge y = \mathbf{T}\ \mathit{then}\ \mathbf{T}\ \mathit{else}\ \mathbf{F})$
using ‹$\{x,\ y\} \subseteq \mathit{elts}\ \mathbb{B}$› **by** *auto*
also have $\ldots \longleftrightarrow \mathbf{T} = (\mathit{if}\ x = \mathbf{T} \wedge y = \mathbf{F}\ \mathit{then}\ \mathbf{F}\ \mathit{else}\ \mathbf{T})$
using ‹$\{x,\ y\} \subseteq \mathit{elts}\ \mathbb{B}$› **by** *auto*
finally show *?thesis*
using *assms* **and** *fully-applied-imp-fun-denotation-is-domain-respecting* **and** *tv-cases*
and *truth-values-domain-def* **by** *metis*
qed

corollary (**in** *general-model*) *prop-5401-f'*:
assumes $\varphi \rightsquigarrow \mathcal{D}$
and $A \in \mathit{wffs}_o$ **and** $B \in \mathit{wffs}_o$
shows $\mathcal{V}\ \varphi\ (A \supset^{\mathcal{Q}} B) = \mathcal{V}\ \varphi\ A \supset \mathcal{V}\ \varphi\ B$
proof −
from *assms* **have** $\{\mathcal{V}\ \varphi\ A,\ \mathcal{V}\ \varphi\ B\} \subseteq \mathit{elts}\ (\mathcal{D}\ o)$
using $\mathcal{V}$*-is-wff-denotation-function* **by** *simp*
from *assms*(*2*) **have** $\supset_{o\to o\to o} \blacktriangleright A \in \mathit{wffs}_{o\to o}$
by *blast*
have $\mathcal{V}\ \varphi\ (A \supset^{\mathcal{Q}} B) = \mathcal{V}\ \varphi\ (\supset_{o\to o\to o} \blacktriangleright A \blacktriangleright B)$
by *simp*
also from *assms*(*1*,*3*) **have** $\ldots = \mathcal{V}\ \varphi\ (\supset_{o\to o\to o} \blacktriangleright A) \cdot \mathcal{V}\ \varphi\ B$
using $\mathcal{V}$*-is-wff-denotation-function* **and** ‹$\supset_{o\to o\to o} \blacktriangleright A \in \mathit{wffs}_{o\to o}$› **by** *blast*
also from *assms* **have** $\ldots = \mathcal{V}\ \varphi\ (\supset_{o\to o\to o}) \cdot \mathcal{V}\ \varphi\ A \cdot \mathcal{V}\ \varphi\ B$
using $\mathcal{V}$*-is-wff-denotation-function* **and** *imp-fun-wff* **by** *fastforce*
also from *assms* **have** $\ldots = (\mathit{if}\ \mathcal{V}\ \varphi\ A = \mathbf{T} \wedge \mathcal{V}\ \varphi\ B = \mathbf{F}\ \mathit{then}\ \mathbf{F}\ \mathit{else}\ \mathbf{T})$
using ‹$\{\mathcal{V}\ \varphi\ A,\ \mathcal{V}\ \varphi\ B\} \subseteq \mathit{elts}\ (\mathcal{D}\ o)$› **and** *prop-5401-f* **by** *simp*
also have $\ldots = \mathcal{V}\ \varphi\ A \supset \mathcal{V}\ \varphi\ B$
using *truth-values-domain-def* **and** ‹$\{\mathcal{V}\ \varphi\ A,\ \mathcal{V}\ \varphi\ B\} \subseteq \mathit{elts}\ (\mathcal{D}\ o)$› **by** *auto*
finally show *?thesis* **.**
qed

lemma (**in** *general-model*) *forall-denotation*:
assumes $\varphi \rightsquigarrow \mathcal{D}$
and $A \in \mathit{wffs}_o$
shows $\mathcal{V}\ \varphi\ (\forall\, x_\alpha.\ A) = \mathbf{T} \longleftrightarrow (\forall\, z \in \mathit{elts}\ (\mathcal{D}\ \alpha).\ \mathcal{V}\ (\varphi((x,\ \alpha) := z))\ A = \mathbf{T})$
proof −
from *assms*(*1*) **have** *lhs*: $\mathcal{V}\ \varphi\ (\lambda\mathfrak{x}_\alpha.\ T_o) \cdot z = \mathbf{T}$ **if** $z \in \mathit{elts}\ (\mathcal{D}\ \alpha)$ **for** z
using *prop-5401-c* **and** *mixed-beta-conversion* **and** *that* **and** *true-wff* **by** *simp*
from *assms* **have** *rhs*: $\mathcal{V}\ \varphi\ (\lambda x_\alpha.\ A) \cdot z = \mathcal{V}\ (\varphi((x,\ \alpha) := z))\ A$ **if** $z \in \mathit{elts}\ (\mathcal{D}\ \alpha)$ **for** z
using *that* **by** (*simp add*: *mixed-beta-conversion*)
from *assms*(*2*) **have** $\lambda\mathfrak{x}_\alpha.\ T_o \in \mathit{wffs}_{\alpha\to o}$ **and** $\lambda x_\alpha.\ A \in \mathit{wffs}_{\alpha\to o}$

by *auto*
have $\mathcal{V}\ \varphi\ (\forall\, x_\alpha.\ A) = \mathcal{V}\ \varphi\ (\prod \alpha \cdot (\lambda x_\alpha.\ A))$
unfolding *forall-def* **..**
also have $\ldots = \mathcal{V}\ \varphi\ (Q_{\alpha\to o} \cdot (\lambda \mathfrak{x}_\alpha.\ T_o) \cdot (\lambda x_\alpha.\ A))$
unfolding *PI-def* **..**
also have $\ldots = \mathcal{V}\ \varphi\ ((\lambda \mathfrak{x}_\alpha.\ T_o) =_{\alpha\to o} (\lambda x_\alpha.\ A))$
unfolding *equality-of-type-def* **..**
finally have $\mathcal{V}\ \varphi\ (\forall\, x_\alpha.\ A) = \mathcal{V}\ \varphi\ ((\lambda \mathfrak{x}_\alpha.\ T_o) =_{\alpha\to o} (\lambda x_\alpha.\ A))$ **.**
moreover from *assms*(*1*,*2*) **have**
$\mathcal{V}\ \varphi\ ((\lambda \mathfrak{x}_\alpha.\ T_o) =_{\alpha\to o} (\lambda x_\alpha.\ A)) = \mathbf{T} \longleftrightarrow \mathcal{V}\ \varphi\ (\lambda \mathfrak{x}_\alpha.\ T_o) = \mathcal{V}\ \varphi\ (\lambda x_\alpha.\ A)$
using $\langle \lambda \mathfrak{x}_\alpha.\ T_o \in \mathit{wffs}_{\alpha\to o}\rangle$ **and** $\langle \lambda x_\alpha.\ A \in \mathit{wffs}_{\alpha\to o}\rangle$ **and** *prop-5401-b* **by** *blast*
moreover
have $(\mathcal{V}\ \varphi\ (\lambda \mathfrak{x}_\alpha.\ T_o) = \mathcal{V}\ \varphi\ (\lambda x_\alpha.\ A)) \longleftrightarrow (\forall\, z \in \mathit{elts}\ (\mathcal{D}\ \alpha).\ \mathcal{V}\ (\varphi((x,\ \alpha) := z))\ A = \mathbf{T})$
proof
assume $\mathcal{V}\ \varphi\ (\lambda \mathfrak{x}_\alpha.\ T_o) = \mathcal{V}\ \varphi\ (\lambda x_\alpha.\ A)$
with *lhs* **and** *rhs* **show** $\forall\, z \in \mathit{elts}\ (\mathcal{D}\ \alpha).\ \mathcal{V}\ (\varphi((x,\ \alpha) := z))\ A = \mathbf{T}$
by *auto*
next
assume $\forall\, z \in \mathit{elts}\ (\mathcal{D}\ \alpha).\ \mathcal{V}\ (\varphi((x,\ \alpha) := z))\ A = \mathbf{T}$
moreover from *assms* **have** $\mathcal{V}\ \varphi\ (\lambda \mathfrak{x}_\alpha.\ T_o) = (\boldsymbol{\lambda} z : \mathcal{D}\ \alpha.\ \mathcal{V}\ (\varphi((\mathfrak{x},\ \alpha) := z))\ T_o)$
using *wff-abs-denotation*[*OF* $\mathcal{V}$*-is-wff-denotation-function*] **by** *blast*
moreover from *assms* **have** $\mathcal{V}\ \varphi\ (\lambda x_\alpha.\ A) = (\boldsymbol{\lambda} z : \mathcal{D}\ \alpha.\ \mathcal{V}\ (\varphi((x,\ \alpha) := z))\ A)$
using *wff-abs-denotation*[*OF* $\mathcal{V}$*-is-wff-denotation-function*] **by** *blast*
ultimately show $\mathcal{V}\ \varphi\ (\lambda \mathfrak{x}_\alpha.\ T_o) = \mathcal{V}\ \varphi\ (\lambda x_\alpha.\ A)$
using *lhs* **and** *vlambda-extensionality* **by** *fastforce*
qed
ultimately show *?thesis*
by (*simp only*:)
qed

lemma *prop-5401-g*:
assumes *is-general-model* $\mathcal{M}$
and $\varphi \rightsquigarrow_M \mathcal{M}$
and $A \in \mathit{wffs}_o$
shows $\mathcal{M} \models_\varphi \forall\, x_\alpha.\ A \longleftrightarrow (\forall\, \psi.\ \psi \rightsquigarrow_M \mathcal{M} \wedge \psi \sim_{(x,\ \alpha)} \varphi \longrightarrow \mathcal{M} \models_\psi A)$
proof −
obtain $\mathcal{D}$ **and** $\mathcal{J}$ **and** $\mathcal{V}$ **where** $\mathcal{M} = (\mathcal{D},\ \mathcal{J},\ \mathcal{V})$
using *prod-cases3* **by** *blast*
with *assms* **have**
$\mathcal{M} \models_\varphi \forall\, x_\alpha.\ A$
$\longleftrightarrow$
$\forall\, x_\alpha.\ A \in \mathit{wffs}_o \wedge \mathit{is\text{-}general\text{-}model}\ (\mathcal{D},\ \mathcal{J},\ \mathcal{V}) \wedge \varphi \rightsquigarrow \mathcal{D} \wedge \mathcal{V}\ \varphi\ (\forall\, x_\alpha.\ A) = \mathbf{T}$
by *fastforce*
also from *assms* **and** $\langle \mathcal{M} = (\mathcal{D},\ \mathcal{J},\ \mathcal{V})\rangle$ **have** $\ldots \longleftrightarrow (\forall\, z \in \mathit{elts}\ (\mathcal{D}\ \alpha).\ \mathcal{V}\ (\varphi((x,\ \alpha) := z))\ A = \mathbf{T})$
using *general-model.forall-denotation* **by** *fastforce*
also have $\ldots \longleftrightarrow (\forall\, \psi.\ \psi \rightsquigarrow \mathcal{D} \wedge \psi \sim_{(x,\ \alpha)} \varphi \longrightarrow \mathcal{M} \models_\psi A)$
proof
assume $*$: $\forall\, z \in \mathit{elts}\ (\mathcal{D}\ \alpha).\ \mathcal{V}\ (\varphi((x,\ \alpha) := z))\ A = \mathbf{T}$

```
    {
      fix ψ
      assume ψ ⇝ 𝒟 and ψ ∼(x, α) φ
      have 𝒱 ψ A = T
      proof −
        have ∃ z ∈ elts (𝒟 α). ψ = φ((x, α) := z)
        proof (rule ccontr)
          assume ¬ (∃ z ∈ elts (𝒟 α). ψ = φ((x, α) := z))
          with ‹ψ ∼(x, α) φ› have ∀ z ∈ elts (𝒟 α). ψ (x, α) ≠ z
            by fastforce
          then have ψ (x, α) ∉ elts (𝒟 α)
            by blast
          moreover from assms(1) and ‹𝓜 = (𝒟, 𝒥, 𝒱)› and ‹ψ ⇝ 𝒟› have ψ (x, α) ∈ elts (𝒟 α)
            using general-model-def and premodel-def and frame.is-assignment-def by auto
          ultimately show False
            by simp
        qed
        with ∗ show ?thesis
          by fastforce
      qed
      with assms(1) and ‹𝓜 = (𝒟, 𝒥, 𝒱)› have 𝓜 ⊨ψ A
        by simp
    }
    then show ∀ ψ. ψ ⇝ 𝒟 ∧ ψ ∼(x, α) φ ⟶ 𝓜 ⊨ψ A
      by blast
  next
    assume ∗: ∀ ψ. ψ ⇝ 𝒟 ∧ ψ ∼(x, α) φ ⟶ 𝓜 ⊨ψ A
    show ∀ z ∈ elts (𝒟 α). 𝒱 (φ((x, α) := z)) A = T
    proof
      fix z
      assume z ∈ elts (𝒟 α)
      with assms(1,2) and ‹𝓜 = (𝒟, 𝒥, 𝒱)› have φ((x, α) := z) ⇝ 𝒟
        using general-model-def and premodel-def and frame.is-assignment-def by auto
      moreover have φ((x, α) := z) ∼(x, α) φ
        by simp
      ultimately have 𝓜 ⊨φ((x, α) := z) A
        using ∗ by blast
      with assms(1) and ‹𝓜 = (𝒟, 𝒥, 𝒱)› and ‹φ((x, α) := z) ⇝ 𝒟› show 𝒱 (φ((x, α) := z)) A =
T
        by simp
    qed
  qed
  finally show ?thesis
    using ‹𝓜 = (𝒟, 𝒥, 𝒱)›
    by simp
qed

lemma (in general-model) axiom-1-validity-aux:
```

assumes $\varphi \rightsquigarrow \mathcal{D}$
shows $\mathcal{V}\ \varphi\ (\mathfrak{g}_{o\to o} \cdot T_o \wedge^{\mathcal{Q}} \mathfrak{g}_{o\to o} \cdot F_o \equiv^{\mathcal{Q}} \forall \mathfrak{x}_o.\ \mathfrak{g}_{o\to o} \cdot \mathfrak{x}_o) = \mathbf{T}$ (**is** $\mathcal{V}\ \varphi\ (?A \equiv^{\mathcal{Q}} ?B) = \mathbf{T}$)
proof −
let $?\mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V})$
from *assms* **have** $*$: *is-general-model* $?\mathcal{M}\ \varphi \rightsquigarrow_M ?\mathcal{M}$
using *general-model-axioms* **by** *blast+*
have $?A \equiv^{\mathcal{Q}} ?B \in \mathit{wffs}_o$
using *axioms.axiom-1* **and** *axioms-are-wffs-of-type-o* **by** *blast*
have *lhs*: $\mathcal{V}\ \varphi\ ?A = \varphi\ (\mathfrak{g},\ o\to o) \cdot \mathbf{T} \wedge \varphi\ (\mathfrak{g},\ o\to o) \cdot \mathbf{F}$
proof −
have $\mathfrak{g}_{o\to o} \cdot T_o \in \mathit{wffs}_o$ **and** $\mathfrak{g}_{o\to o} \cdot F_o \in \mathit{wffs}_o$
by *blast+*
with *assms* **have** $\mathcal{V}\ \varphi\ ?A = \mathcal{V}\ \varphi\ (\mathfrak{g}_{o\to o} \cdot T_o) \wedge \mathcal{V}\ \varphi\ (\mathfrak{g}_{o\to o} \cdot F_o)$
using *prop-5401-e′* **by** *simp*
also from *assms* **have** $\ldots = \varphi\ (\mathfrak{g},\ o\to o) \cdot \mathcal{V}\ \varphi\ (T_o) \wedge \varphi\ (\mathfrak{g},\ o\to o) \cdot \mathcal{V}\ \varphi\ (F_o)$
using *wff-app-denotation*[*OF* $\mathcal{V}$*-is-wff-denotation-function*]
and *wff-var-denotation*[*OF* $\mathcal{V}$*-is-wff-denotation-function*]
by (*metis false-wff true-wff wffs-of-type-intros*(*1*))
finally show *?thesis*
using *assms* **and** *prop-5401-c* **and** *prop-5401-d* **by** *simp*
qed
have $\mathcal{V}\ \varphi\ (?A \equiv^{\mathcal{Q}} ?B) = \mathbf{T}$
proof (*cases* $\forall z \in \mathit{elts}\ (\mathcal{D}\ o).\ \varphi\ (\mathfrak{g},\ o\to o) \cdot z = \mathbf{T}$)
case *True*
with *assms* **have** $\varphi\ (\mathfrak{g},\ o\to o) \cdot \mathbf{T} = \mathbf{T}$ **and** $\varphi\ (\mathfrak{g},\ o\to o) \cdot \mathbf{F} = \mathbf{T}$
using *truth-values-domain-def* **by** *auto*
with *lhs* **have** $\mathcal{V}\ \varphi\ ?A = \mathbf{T} \wedge \mathbf{T}$
by (*simp only*:)
also have $\ldots = \mathbf{T}$
by *simp*
finally have $\mathcal{V}\ \varphi\ ?A = \mathbf{T}$ **.**
moreover have $\mathcal{V}\ \varphi\ ?B = \mathbf{T}$
proof −
have $\mathfrak{g}_{o\to o} \cdot \mathfrak{x}_o \in \mathit{wffs}_o$
by *blast*
moreover
{
fix ψ
assume $\psi \rightsquigarrow \mathcal{D}$ **and** $\psi \sim_{(\mathfrak{x},\ o)} \varphi$
with *assms* **have** $\mathcal{V}\ \psi\ (\mathfrak{g}_{o\to o} \cdot \mathfrak{x}_o) = \mathcal{V}\ \psi\ (\mathfrak{g}_{o\to o}) \cdot \mathcal{V}\ \psi\ (\mathfrak{x}_o)$
using $\mathcal{V}$*-is-wff-denotation-function* **by** *blast*
also from $\langle\psi \rightsquigarrow \mathcal{D}\rangle$ **have** $\ldots = \psi\ (\mathfrak{g},\ o\to o) \cdot \psi\ (\mathfrak{x},\ o)$
using $\mathcal{V}$*-is-wff-denotation-function* **by** *auto*
also from $\langle\psi \sim_{(\mathfrak{x},\ o)} \varphi\rangle$ **have** $\ldots = \varphi\ (\mathfrak{g},\ o\to o) \cdot \psi\ (\mathfrak{x},\ o)$
by *simp*
also from *True* **and** $\langle\psi \rightsquigarrow \mathcal{D}\rangle$ **have** $\ldots = \mathbf{T}$
by *blast*
finally have $\mathcal{V}\ \psi\ (\mathfrak{g}_{o\to o} \cdot \mathfrak{x}_o) = \mathbf{T}$ **.**
with *assms* **and** $\langle\mathfrak{g}_{o\to o} \cdot \mathfrak{x}_o \in \mathit{wffs}_o\rangle$ **have** $?\mathcal{M} \models_\psi \mathfrak{g}_{o\to o} \cdot \mathfrak{x}_o$

by *simp*
}
ultimately have $?\mathcal{M} \models_\varphi ?B$
using *assms* **and** $*$ **and** *prop-5401-g* **by** *auto*
with $*(2)$ **show** *?thesis*
by *simp*
qed
ultimately show *?thesis*
using *assms* **and** *prop-5401-b′* **and** *wffs-from-equivalence*[*OF* ‹$?A \equiv^{\mathfrak{Q}} ?B \in wffs_o$›] **by** *simp*
next
case *False*
then have $\exists z \in elts\ (\mathcal{D}\ o).\ \varphi\ (\mathfrak{g},\ o{\to}o) \cdot z \neq \mathbf{T}$
by *blast*
moreover from $*$ **have** $\forall z \in elts\ (\mathcal{D}\ o).\ \varphi\ (\mathfrak{g},\ o{\to}o) \cdot z \in elts\ (\mathcal{D}\ o)$
using *app-is-domain-respecting* **by** *blast*
ultimately obtain z **where** $z \in elts\ (\mathcal{D}\ o)$ **and** $\varphi\ (\mathfrak{g},\ o{\to}o) \cdot z = \mathbf{F}$
using *truth-values-domain-def* **by** *auto*
define ψ **where** *ψ-def*: $\psi = \varphi((\mathfrak{x},\ o) := z)$
with $*$ **and** ‹$z \in elts\ (\mathcal{D}\ o)$› **have** $\psi \rightsquigarrow \mathcal{D}$
by *simp*
then have $\mathcal{V}\ \psi\ (\mathfrak{g}_{o\to o} \cdot \mathfrak{x}_o) = \mathcal{V}\ \psi\ (\mathfrak{g}_{o\to o}) \cdot \mathcal{V}\ \psi\ (\mathfrak{x}_o)$
using *𝒱-is-wff-denotation-function* **by** *blast*
also from ‹$\psi \rightsquigarrow \mathcal{D}$› **have** $\ldots = \psi\ (\mathfrak{g},\ o{\to}o) \cdot \psi\ (\mathfrak{x},\ o)$
using *𝒱-is-wff-denotation-function* **by** *auto*
also from *ψ-def* **have** $\ldots = \varphi\ (\mathfrak{g},\ o{\to}o) \cdot z$
by *simp*
also have $\ldots = \mathbf{F}$
unfolding ‹$\varphi\ (\mathfrak{g},\ o{\to}o) \cdot z = \mathbf{F}$› **..**
finally have $\mathcal{V}\ \psi\ (\mathfrak{g}_{o\to o} \cdot \mathfrak{x}_o) = \mathbf{F}$ **.**
with ‹$\psi \rightsquigarrow \mathcal{D}$› **have** $\neg\ ?\mathcal{M} \models_\psi \mathfrak{g}_{o\to o} \cdot \mathfrak{x}_o$
by (*auto simp add*: *inj-eq*)
with ‹$\psi \rightsquigarrow \mathcal{D}$› **and** *ψ-def* **have** $\neg\ (\forall \psi.\ \psi \rightsquigarrow \mathcal{D} \wedge \psi \sim_{(\mathfrak{x},\ o)} \varphi \longrightarrow ?\mathcal{M} \models_\psi \mathfrak{g}_{o\to o} \cdot \mathfrak{x}_o)$
using *fun-upd-other* **by** *fastforce*
with ‹$\neg\ ?\mathcal{M} \models_\psi \mathfrak{g}_{o\to o} \cdot \mathfrak{x}_o$› **have** $\neg\ ?\mathcal{M} \models_\varphi ?B$
using *prop-5401-g*[*OF* $*$ *wffs-from-forall*[*OF wffs-from-equivalence*(*2*)[*OF* ‹$?A \equiv^{\mathfrak{Q}} ?B \in wffs_o$›]]]
by *blast*
then have $\mathcal{V}\ \varphi\ (\forall \mathfrak{x}_o.\ \mathfrak{g}_{o\to o} \cdot \mathfrak{x}_o) \neq \mathbf{T}$
by *simp*
moreover from *assms* **have** $\mathcal{V}\ \varphi\ ?B \in elts\ (\mathcal{D}\ o)$
using *wffs-from-equivalence*[*OF* ‹$?A \equiv^{\mathfrak{Q}} ?B \in wffs_o$›] **and** *𝒱-is-wff-denotation-function* **by** *auto*
ultimately have $\mathcal{V}\ \varphi\ ?B = \mathbf{F}$
by (*simp add*: *truth-values-domain-def*)
moreover have $\mathcal{V}\ \varphi\ (\mathfrak{g}_{o\to o} \cdot T_o \wedge^{\mathfrak{Q}} \mathfrak{g}_{o\to o} \cdot F_o) = \mathbf{F}$
proof −
from ‹$z \in elts\ (\mathcal{D}\ o)$› **and** ‹$\varphi\ (\mathfrak{g},\ o{\to}o) \cdot z = \mathbf{F}$›
have $((\varphi\ (\mathfrak{g},\ o{\to}o)) \cdot \mathbf{T}) = \mathbf{F} \vee ((\varphi\ (\mathfrak{g},\ o{\to}o)) \cdot \mathbf{F}) = \mathbf{F}$
using *truth-values-domain-def* **by** *fastforce*
moreover from ‹$z \in elts\ (\mathcal{D}\ o)$› **and** ‹$\varphi\ (\mathfrak{g},\ o{\to}o) \cdot z = \mathbf{F}$›
and ‹$\forall z \in elts\ (\mathcal{D}\ o).\ \varphi\ (\mathfrak{g},\ o{\to}o) \cdot z \in elts\ (\mathcal{D}\ o)$›

```
    have {(φ (𝔤, o→o)) · T, (φ (𝔤, o→o)) · F} ⊆ elts (𝒟 o)
      by (simp add: truth-values-domain-def)
    ultimately have ((φ (𝔤, o→o)) · T) ∧ ((φ (𝔤, o→o)) · F) = F
      by auto
    with lhs show ?thesis
      by (simp only:)
  qed
  ultimately show ?thesis
    using assms and prop-5401-b′ and wffs-from-equivalence[OF ‹?A ≡^Q ?B ∈ wffs_o›] by simp
 qed
 then show ?thesis .
qed

lemma axiom-1-validity:
 shows ⊨ 𝔤_{o→o} · T_o ∧^Q 𝔤_{o→o} · F_o ≡^Q ∀𝔵_o. 𝔤_{o→o} · 𝔵_o (is ⊨ ?A ≡^Q ?B)
proof (intro allI impI)
 fix 𝓜 and φ
 assume ∗: is-general-model 𝓜 φ ⇝_M 𝓜
 show 𝓜 ⊨_φ ?A ≡^Q ?B
 proof −
  obtain 𝒟 and 𝒥 and 𝒱 where 𝓜 = (𝒟, 𝒥, 𝒱)
    using prod-cases3 by blast
  moreover from ∗ and ‹𝓜 = (𝒟, 𝒥, 𝒱)› have 𝒱 φ (?A ≡^Q ?B) = T
    using general-model.axiom-1-validity-aux by simp
  ultimately show ?thesis
    by simp
 qed
qed

lemma (in general-model) axiom-2-validity-aux:
 assumes φ ⇝ 𝒟
 shows 𝒱 φ ((𝔵_α =_α 𝔶_α) ⊃^Q (𝔥_{α→o} · 𝔵_α ≡^Q 𝔥_{α→o} · 𝔶_α)) = T (is 𝒱 φ (?A ⊃^Q ?B) = T)
proof −
 have ?A ⊃^Q ?B ∈ wffs_o
   using axioms.axiom-2 and axioms-are-wffs-of-type-o by blast
 from ‹?A ⊃^Q ?B ∈ wffs_o› have ?A ∈ wffs_o and ?B ∈ wffs_o
   using wffs-from-imp-op by blast+
 with assms have 𝒱 φ (?A ⊃^Q ?B) = 𝒱 φ ?A ⊃ 𝒱 φ ?B
   using prop-5401-f′ by simp
 moreover from assms and ‹?A ∈ wffs_o› and ‹?B ∈ wffs_o› have {𝒱 φ ?A, 𝒱 φ ?B} ⊆ elts (𝒟 o)
   using 𝒱-is-wff-denotation-function by simp
 then have {𝒱 φ ?A, 𝒱 φ ?B} ⊆ elts 𝔹
   by (simp add: truth-values-domain-def)
 ultimately have 𝒱-imp-T: 𝒱 φ (?A ⊃^Q ?B) = T ⟷ 𝒱 φ ?A = F ∨ 𝒱 φ ?B = T
   by fastforce
 then show ?thesis
 proof (cases φ (𝔵, α) = φ (𝔶, α))
   case True
   from assms and ‹?B ∈ wffs_o› have 𝒱 φ ?B = T ⟷ 𝒱 φ (𝔥_{α→o} · 𝔵_α) = 𝒱 φ (𝔥_{α→o} · 𝔶_α)
```

using *wffs-from-equivalence* **and** *prop-5401-b′* **by** *metis*
moreover have $\mathcal{V}\ \varphi\ (\mathfrak{h}_{\alpha\to o} \centerdot \mathfrak{x}_\alpha) = \mathcal{V}\ \varphi\ (\mathfrak{h}_{\alpha\to o} \centerdot \mathfrak{y}_\alpha)$
proof −
from *assms* **and** ‹*?B* ∈ *wffs*$_o$› **have** $\mathcal{V}\ \varphi\ (\mathfrak{h}_{\alpha\to o} \centerdot \mathfrak{x}_\alpha) = \mathcal{V}\ \varphi\ (\mathfrak{h}_{\alpha\to o}) \centerdot \mathcal{V}\ \varphi\ (\mathfrak{x}_\alpha)$
using $\mathcal{V}$*-is-wff-denotation-function* **by** *blast*
also from *assms* **have** ... $= \varphi\ (\mathfrak{h},\ \alpha{\to}o) \centerdot \varphi\ (\mathfrak{x},\ \alpha)$
using $\mathcal{V}$*-is-wff-denotation-function* **by** *auto*
also from *True* **have** ... $= \varphi\ (\mathfrak{h},\ \alpha{\to}o) \centerdot \varphi\ (\mathfrak{y},\ \alpha)$
by (*simp only*:)
also from *assms* **have** ... $= \mathcal{V}\ \varphi\ (\mathfrak{h}_{\alpha\to o}) \centerdot \mathcal{V}\ \varphi\ (\mathfrak{y}_\alpha)$
using $\mathcal{V}$*-is-wff-denotation-function* **by** *auto*
also from *assms* **and** ‹*?B* ∈ *wffs*$_o$› **have** ... $= \mathcal{V}\ \varphi\ (\mathfrak{h}_{\alpha\to o} \centerdot \mathfrak{y}_\alpha)$
using *wff-app-denotation*[*OF* $\mathcal{V}$*-is-wff-denotation-function*] **by** (*metis wffs-of-type-intros*(*1*))
finally show *?thesis* **.**
qed
ultimately show *?thesis*
using $\mathcal{V}$*-imp-T* **by** *simp*
next
case *False*
from *assms* **have** $\mathcal{V}\ \varphi\ ?A = \mathbf{T} \longleftrightarrow \mathcal{V}\ \varphi\ (\mathfrak{x}_\alpha) = \mathcal{V}\ \varphi\ (\mathfrak{y}_\alpha)$
using *prop-5401-b* **by** *blast*
moreover from *False* **and** *assms* **have** $\mathcal{V}\ \varphi\ (\mathfrak{x}_\alpha) \neq \mathcal{V}\ \varphi\ (\mathfrak{y}_\alpha)$
using $\mathcal{V}$*-is-wff-denotation-function* **by** *auto*
ultimately have $\mathcal{V}\ \varphi\ ?A = \mathbf{F}$
using *assms* **and** ‹$\{\mathcal{V}\ \varphi\ ?A,\ \mathcal{V}\ \varphi\ ?B\} \subseteq$ *elts* $\mathbb{B}$›
by *simp*
then show *?thesis*
using $\mathcal{V}$*-imp-T* **by** *simp*
qed
qed

lemma *axiom-2-validity*:
shows $\models (\mathfrak{x}_\alpha =_\alpha \mathfrak{y}_\alpha) \supset^{\mathfrak{Q}} (\mathfrak{h}_{\alpha\to o} \centerdot \mathfrak{x}_\alpha \equiv^{\mathfrak{Q}} \mathfrak{h}_{\alpha\to o} \centerdot \mathfrak{y}_\alpha)$ (**is** $\models$ *?A* $\supset^{\mathfrak{Q}}$ *?B*)
proof (*intro allI impI*)
fix $\mathcal{M}$ **and** φ
assume ∗: *is-general-model* $\mathcal{M}\ \varphi \rightsquigarrow_M \mathcal{M}$
show $\mathcal{M} \models_\varphi$ *?A* $\supset^{\mathfrak{Q}}$ *?B*
proof −
obtain $\mathcal{D}$ **and** $\mathcal{J}$ **and** $\mathcal{V}$ **where** $\mathcal{M} = (\mathcal{D},\ \mathcal{J},\ \mathcal{V})$
using *prod-cases3* **by** *blast*
moreover from ∗ **and** ‹$\mathcal{M} = (\mathcal{D},\ \mathcal{J},\ \mathcal{V})$› **have** $\mathcal{V}\ \varphi$ (*?A* $\supset^{\mathfrak{Q}}$ *?B*) = $\mathbf{T}$
using *general-model.axiom-2-validity-aux* **by** *simp*
ultimately show *?thesis*
by *force*
qed
qed

lemma (**in** *general-model*) *axiom-3-validity-aux*:
assumes $\varphi \rightsquigarrow \mathcal{D}$
shows $\mathcal{V}\ \varphi\ ((\mathfrak{f}_{\alpha\to\beta} =_{\alpha\to\beta} \mathfrak{g}_{\alpha\to\beta}) \equiv^{\mathfrak{Q}} \forall \mathfrak{x}_\alpha.\ (\mathfrak{f}_{\alpha\to\beta} \centerdot \mathfrak{x}_\alpha =_\beta \mathfrak{g}_{\alpha\to\beta} \centerdot \mathfrak{x}_\alpha)) = \mathbf{T}$

```
  (is 𝒱 φ (?A ≡^Q ?B) = T)
proof −
  let ?M = (𝒟, 𝒥, 𝒱)
  from assms have ∗: is-general-model ?M φ ⇝_M ?M
    using general-model-axioms by blast+
  have B′-wffo: 𝔣_{α→β} ▪ 𝔵α =_β 𝔤_{α→β} ▪ 𝔵α ∈ wffs_o
    by blast
  have ?A ≡^Q ?B ∈ wffs_o and ?A ∈ wffs_o and ?B ∈ wffs_o
  proof −
    show ?A ≡^Q ?B ∈ wffs_o
      using axioms.axiom-3 and axioms-are-wffs-of-type-o
      by blast
    then show ?A ∈ wffs_o and ?B ∈ wffs_o
      by (blast dest: wffs-from-equivalence)+
  qed
  have 𝒱 φ ?A = 𝒱 φ ?B
  proof (cases φ (𝔣, α→β) = φ (𝔤, α→β))
    case True
    have 𝒱 φ ?A = T
    proof −
      from assms have 𝒱 φ (𝔣_{α→β}) = φ (𝔣, α→β)
        using 𝒱-is-wff-denotation-function by auto
      also from True have ... = φ (𝔤, α→β)
        by (simp only:)
      also from assms have ... = 𝒱 φ (𝔤_{α→β})
        using 𝒱-is-wff-denotation-function by auto
      finally have 𝒱 φ (𝔣_{α→β}) = 𝒱 φ (𝔤_{α→β}) ▪
      with assms show ?thesis
        using prop-5401-b by blast
    qed
    moreover have 𝒱 φ ?B = T
    proof −
      {
        fix ψ
        assume ψ ⇝ 𝒟 and ψ ∼_(𝔵, α) φ
        from assms and ‹ψ ⇝ 𝒟› have 𝒱 ψ (𝔣_{α→β} ▪ 𝔵α) = 𝒱 ψ (𝔣_{α→β}) ▪ 𝒱 ψ (𝔵α)
          using 𝒱-is-wff-denotation-function by blast
        also from assms and ‹ψ ⇝ 𝒟› have ... = ψ (𝔣, α→β) ▪ ψ (𝔵, α)
          using 𝒱-is-wff-denotation-function by auto
        also from ‹ψ ∼_(𝔵, α) φ› have ... = φ (𝔣, α→β) ▪ ψ (𝔵, α)
          by simp
        also from True have ... = φ (𝔤, α→β) ▪ ψ (𝔵, α)
          by (simp only:)
        also from ‹ψ ∼_(𝔵, α) φ› have ... = ψ (𝔤, α→β) ▪ ψ (𝔵, α)
          by simp
        also from assms and ‹ψ ⇝ 𝒟› have ... = 𝒱 ψ (𝔤_{α→β}) ▪ 𝒱 ψ (𝔵α)
          using 𝒱-is-wff-denotation-function by auto
        also from assms and ‹ψ ⇝ 𝒟› have ... = 𝒱 ψ (𝔤_{α→β} ▪ 𝔵α)
          using wff-app-denotation[OF 𝒱-is-wff-denotation-function] by (metis wffs-of-type-intros(1))
```

finally have $\mathcal{V}\ \psi\ (\mathfrak{f}_{\alpha\to\beta} \cdot \mathfrak{x}\alpha) = \mathcal{V}\ \psi\ (\mathfrak{g}_{\alpha\to\beta} \cdot \mathfrak{x}\alpha)$.
with *B′-wffo* **and** *assms* **and** $\langle\psi \rightsquigarrow \mathcal{D}\rangle$ **have** $\mathcal{V}\ \psi\ (\mathfrak{f}_{\alpha\to\beta} \cdot \mathfrak{x}\alpha =_\beta \mathfrak{g}_{\alpha\to\beta} \cdot \mathfrak{x}\alpha) = \mathbf{T}$
using *prop-5401-b* **and** *wffs-from-equality* **by** *blast*
with $*(2)$ **have** $?\mathcal{M} \models_\psi \mathfrak{f}_{\alpha\to\beta} \cdot \mathfrak{x}\alpha =_\beta \mathfrak{g}_{\alpha\to\beta} \cdot \mathfrak{x}\alpha$
by *simp*
}
with $*$ **and** *B′-wffo* **have** $?\mathcal{M} \models_\varphi ?B$
using *prop-5401-g* **by** *force*
with $*(2)$ **show** *?thesis*
by *auto*
qed
ultimately show *?thesis* ..
next
case *False*
from $*$ **have** $\varphi\ (\mathfrak{f}, \alpha\to\beta) \in \mathit{elts}\ (\mathcal{D}\ \alpha \longmapsto \mathcal{D}\ \beta)$ **and** $\varphi\ (\mathfrak{g}, \alpha\to\beta) \in \mathit{elts}\ (\mathcal{D}\ \alpha \longmapsto \mathcal{D}\ \beta)$
by (*simp-all add*: *function-domainD*)
with *False* **obtain** z **where** $z \in \mathit{elts}\ (\mathcal{D}\ \alpha)$ **and** $\varphi\ (\mathfrak{f}, \alpha\to\beta) \cdot z \neq \varphi\ (\mathfrak{g}, \alpha\to\beta) \cdot z$
by (*blast dest*: *fun-ext*)
define ψ **where** $\psi = \varphi((\mathfrak{x}, \alpha) := z)$
from $*$ **and** $\langle z \in \mathit{elts}\ (\mathcal{D}\ \alpha)\rangle$ **have** $\psi \rightsquigarrow \mathcal{D}$ **and** $\psi \sim_{(\mathfrak{x},\ \alpha)} \varphi$
unfolding ψ*-def* **by** *fastforce+*
have $\mathcal{V}\ \psi\ (f_{\alpha\to\beta} \cdot \mathfrak{x}\alpha) = \varphi\ (f, \alpha\to\beta) \cdot z$ **for** f
proof −
from $\langle\psi \rightsquigarrow \mathcal{D}\rangle$ **have** $\mathcal{V}\ \psi\ (f_{\alpha\to\beta} \cdot \mathfrak{x}\alpha) = \mathcal{V}\ \psi\ (f_{\alpha\to\beta}) \cdot \mathcal{V}\ \psi\ (\mathfrak{x}\alpha)$
using $\mathcal{V}$*-is-wff-denotation-function* **by** *blast*
also from $\langle\psi \rightsquigarrow \mathcal{D}\rangle$ **have** $\ldots = \psi\ (f, \alpha\to\beta) \cdot \psi\ (\mathfrak{x}, \alpha)$
using $\mathcal{V}$*-is-wff-denotation-function* **by** *auto*
finally show *?thesis*
unfolding ψ*-def* **by** *simp*
qed
then have $\mathcal{V}\ \psi\ (\mathfrak{f}_{\alpha\to\beta} \cdot \mathfrak{x}\alpha) = \varphi\ (\mathfrak{f}, \alpha\to\beta) \cdot z$ **and** $\mathcal{V}\ \psi\ (\mathfrak{g}_{\alpha\to\beta} \cdot \mathfrak{x}\alpha) = \varphi\ (\mathfrak{g}, \alpha\to\beta) \cdot z$
by (*simp-all only*:)
with $\langle\varphi\ (\mathfrak{f}, \alpha\to\beta) \cdot z \neq \varphi\ (\mathfrak{g}, \alpha\to\beta) \cdot z\rangle$ **have** $\mathcal{V}\ \psi\ (\mathfrak{f}_{\alpha\to\beta} \cdot \mathfrak{x}\alpha) \neq \mathcal{V}\ \psi\ (\mathfrak{g}_{\alpha\to\beta} \cdot \mathfrak{x}\alpha)$
by *simp*
then have $\mathcal{V}\ \psi\ (\mathfrak{f}_{\alpha\to\beta} \cdot \mathfrak{x}\alpha =_\beta \mathfrak{g}_{\alpha\to\beta} \cdot \mathfrak{x}\alpha) = \mathbf{F}$
proof −
from *B′-wffo* **and** $\langle\psi \rightsquigarrow \mathcal{D}\rangle$ **and** $*$ **have** $\mathcal{V}\ \psi\ (\mathfrak{f}_{\alpha\to\beta} \cdot \mathfrak{x}\alpha =_\beta \mathfrak{g}_{\alpha\to\beta} \cdot \mathfrak{x}\alpha) \in \mathit{elts}\ (\mathcal{D}\ o)$
using $\mathcal{V}$*-is-wff-denotation-function* **by** *auto*
moreover from *B′-wffo* **have** $\{\mathfrak{f}_{\alpha\to\beta} \cdot \mathfrak{x}\alpha, \mathfrak{g}_{\alpha\to\beta} \cdot \mathfrak{x}\alpha\} \subseteq \mathit{wffs}_\beta$
by *blast*
with $\langle\psi \rightsquigarrow \mathcal{D}\rangle$ **and** $\langle\mathcal{V}\ \psi\ (\mathfrak{f}_{\alpha\to\beta} \cdot \mathfrak{x}\alpha) \neq \mathcal{V}\ \psi\ (\mathfrak{g}_{\alpha\to\beta} \cdot \mathfrak{x}\alpha)\rangle$ **and** *B′-wffo*
have $\mathcal{V}\ \psi\ (\mathfrak{f}_{\alpha\to\beta} \cdot \mathfrak{x}\alpha =_\beta \mathfrak{g}_{\alpha\to\beta} \cdot \mathfrak{x}\alpha) \neq \mathbf{T}$
using *prop-5401-b* **by** *simp*
ultimately show *?thesis*
by (*simp add*: *truth-values-domain-def*)
qed
with $\langle\psi \rightsquigarrow \mathcal{D}\rangle$ **have** $\neg\ ?\mathcal{M} \models_\psi \mathfrak{f}_{\alpha\to\beta} \cdot \mathfrak{x}\alpha =_\beta \mathfrak{g}_{\alpha\to\beta} \cdot \mathfrak{x}\alpha$
by (*auto simp add*: *inj-eq*)
with $\langle\psi \rightsquigarrow \mathcal{D}\rangle$ **and** $\langle\psi \sim_{(\mathfrak{x},\ \alpha)} \varphi\rangle$

```
    have ∃ψ. ψ ⇝ 𝒟 ∧ ψ ∼(𝔯, α) φ ∧ ¬ ?𝓜 ⊨ψ 𝔣α→β ▪ 𝔯α =β 𝔤α→β ▪ 𝔯α
      by blast
    with ∗ and B′-wffo have ¬ ?𝓜 ⊨φ ?B
      using prop-5401-g by blast
    then have 𝒱 φ ?B = 𝐅
    proof −
      from ‹?B ∈ wffs_o› and ∗ have 𝒱 φ ?B ∈ elts (𝒟 o)
        using 𝒱-is-wff-denotation-function by auto
      with ‹¬ ?𝓜 ⊨φ ?B› and ‹?B ∈ wffs_o› show ?thesis
        using truth-values-domain-def by fastforce
    qed
    moreover have 𝒱 φ (𝔣α→β =α→β 𝔤α→β) = 𝐅
    proof −
      from ∗ have 𝒱 φ (𝔣α→β) = φ (𝔣, α→β) and 𝒱 φ (𝔤α→β) = φ (𝔤, α→β)
        using 𝒱-is-wff-denotation-function by auto
      with False have 𝒱 φ (𝔣α→β) ≠ 𝒱 φ (𝔤α→β)
        by simp
      with ∗ have 𝒱 φ (𝔣α→β =α→β 𝔤α→β) ≠ 𝐓
        using prop-5401-b by blast
      moreover from ∗ and ‹?A ∈ wffs_o› have 𝒱 φ (𝔣α→β =α→β 𝔤α→β) ∈ elts (𝒟 o)
        using 𝒱-is-wff-denotation-function by auto
      ultimately show ?thesis
        by (simp add: truth-values-domain-def)
    qed
    ultimately show ?thesis
      by (simp only:)
  qed
  with ∗ and ‹?A ∈ wffs_o› and ‹?B ∈ wffs_o› show ?thesis
    using prop-5401-b′ by simp
qed

lemma axiom-3-validity:
  shows ⊨ (𝔣α→β =α→β 𝔤α→β) ≡𝛺 ∀𝔯α. (𝔣α→β ▪ 𝔯α =β 𝔤α→β ▪ 𝔯α) (is ⊨ ?A ≡𝛺 ?B)
proof (intro allI impI)
  fix 𝓜 and φ
  assume ∗: is-general-model 𝓜 φ ⇝M 𝓜
  show 𝓜 ⊨φ ?A ≡𝛺 ?B
  proof −
    obtain 𝒟 and 𝒥 and 𝒱 where 𝓜 = (𝒟, 𝒥, 𝒱)
      using prod-cases3 by blast
    moreover from ∗ and ‹𝓜 = (𝒟, 𝒥, 𝒱)› have 𝒱 φ (?A ≡𝛺 ?B) = 𝐓
      using general-model.axiom-3-validity-aux by simp
    ultimately show ?thesis
      by simp
  qed
qed

lemma (in general-model) axiom-4-1-con-validity-aux:
  assumes φ ⇝ 𝒟
```

and $A \in \mathit{wffs}_\alpha$
shows $\mathcal{V}\ \varphi\ ((\lambda x_\alpha.\ \{\!|c|\!\}_\beta) \cdot A =_\beta \{\!|c|\!\}_\beta) = \mathbf{T}$
proof −
from *assms(2)* **have** $(\lambda x_\alpha.\ \{\!|c|\!\}_\beta) \cdot A =_\beta \{\!|c|\!\}_\beta \in \mathit{wffs}_o$
using *axioms.axiom-4-1-con* **and** *axioms-are-wffs-of-type-o* **by** *blast*
define ψ **where** $\psi = \varphi((x,\ \alpha) := \mathcal{V}\ \varphi\ A)$
from *assms* **have** $\mathcal{V}\ \varphi\ ((\lambda x_\alpha.\ \{\!|c|\!\}_\beta) \cdot A) = \mathcal{V}\ (\varphi((x,\ \alpha) := \mathcal{V}\ \varphi\ A))\ (\{\!|c|\!\}_\beta)$
using *prop-5401-a* **by** *blast*
also have $\ldots = \mathcal{V}\ \psi\ (\{\!|c|\!\}_\beta)$
unfolding ψ*-def* **..**
also from *assms* **and** ψ*-def* **have** $\ldots = \mathcal{V}\ \varphi\ (\{\!|c|\!\}_\beta)$
using $\mathcal{V}$*-is-wff-denotation-function* **by** *auto*
finally have $\mathcal{V}\ \varphi\ ((\lambda x_\alpha.\ \{\!|c|\!\}_\beta) \cdot A) = \mathcal{V}\ \varphi\ (\{\!|c|\!\}_\beta)$.
with *assms(1)* **and** $\langle(\lambda x_\alpha.\ \{\!|c|\!\}_\beta) \cdot A =_\beta \{\!|c|\!\}_\beta \in \mathit{wffs}_o\rangle$ **show** *?thesis*
using *wffs-from-equality(1)* **and** *prop-5401-b* **by** *blast*
qed

lemma *axiom-4-1-con-validity*:
assumes $A \in \mathit{wffs}_\alpha$
shows $\models (\lambda x_\alpha.\ \{\!|c|\!\}_\beta) \cdot A =_\beta \{\!|c|\!\}_\beta$
proof (*intro allI impI*)
fix $\mathcal{M}$ **and** φ
assume $*$: *is-general-model* $\mathcal{M}\ \varphi \rightsquigarrow_M \mathcal{M}$
show $\mathcal{M} \models_\varphi (\lambda x_\alpha.\ \{\!|c|\!\}_\beta) \cdot A =_\beta \{\!|c|\!\}_\beta$
proof −
obtain $\mathcal{D}$ **and** $\mathcal{J}$ **and** $\mathcal{V}$ **where** $\mathcal{M} = (\mathcal{D},\ \mathcal{J},\ \mathcal{V})$
using *prod-cases3* **by** *blast*
moreover from *assms* **and** $*$ **and** $\langle\mathcal{M} = (\mathcal{D},\ \mathcal{J},\ \mathcal{V})\rangle$ **have** $\mathcal{V}\ \varphi\ ((\lambda x_\alpha.\ \{\!|c|\!\}_\beta) \cdot A =_\beta \{\!|c|\!\}_\beta) = \mathbf{T}$
using *general-model.axiom-4-1-con-validity-aux* **by** *simp*
ultimately show *?thesis*
by *simp*
qed
qed

lemma (**in** *general-model*) *axiom-4-1-var-validity-aux*:
assumes $\varphi \rightsquigarrow \mathcal{D}$
and $A \in \mathit{wffs}_\alpha$
and $(y,\ \beta) \neq (x,\ \alpha)$
shows $\mathcal{V}\ \varphi\ ((\lambda x_\alpha.\ y_\beta) \cdot A =_\beta y_\beta) = \mathbf{T}$
proof −
from *assms(2)* **have** $(\lambda x_\alpha.\ y_\beta) \cdot A =_\beta y_\beta \in \mathit{wffs}_o$
using *axioms.axiom-4-1-var* **and** *axioms-are-wffs-of-type-o* **by** *blast*
define ψ **where** $\psi = \varphi((x,\ \alpha) := \mathcal{V}\ \varphi\ A)$
with *assms(1,2)* **have** $\mathcal{V}\ \varphi\ ((\lambda x_\alpha.\ y_\beta) \cdot A) = \mathcal{V}\ (\varphi((x,\ \alpha) := \mathcal{V}\ \varphi\ A))\ (y_\beta)$
using *prop-5401-a* **by** *blast*
also have $\ldots = \mathcal{V}\ \psi\ (y_\beta)$
unfolding ψ*-def* **..**
also have $\ldots = \mathcal{V}\ \varphi\ (y_\beta)$
proof −

```
    from assms(1,2) have 𝒱 φ A ∈ elts (𝒟 α)
      using 𝒱-is-wff-denotation-function by auto
    with ψ-def and assms(1) have ψ ⇝ 𝒟
      by simp
    moreover have free-vars (y_β) = {(y, β)}
      by simp
    with ψ-def and assms(3) have ∀ v ∈ free-vars (y_β). φ v = ψ v
      by auto
    ultimately show ?thesis
      using prop-5400[OF wffs-of-type-intros(1) assms(1)] by simp
  qed
  finally have 𝒱 φ ((λx_α. y_β) ▪ A) = 𝒱 φ (y_β) .
  with ‹(λx_α. y_β) ▪ A =_β y_β ∈ wffs_o› show ?thesis
    using wffs-from-equality(1) and prop-5401-b[OF assms(1)] by blast
qed

lemma axiom-4-1-var-validity:
  assumes A ∈ wffs_α
  and (y, β) ≠ (x, α)
  shows ⊨ (λx_α. y_β) ▪ A =_β y_β
proof (intro allI impI)
  fix 𝓜 and φ
  assume ∗: is-general-model 𝓜 φ ⇝_M 𝓜
  show 𝓜 ⊨_φ (λx_α. y_β) ▪ A =_β y_β
  proof −
    obtain 𝒟 and 𝒥 and 𝒱 where 𝓜 = (𝒟, 𝒥, 𝒱)
      using prod-cases3 by blast
    moreover from assms and ∗ and ‹𝓜 = (𝒟, 𝒥, 𝒱)› have 𝒱 φ ((λx_α. y_β) ▪ A =_β y_β) = T
      using general-model.axiom-4-1-var-validity-aux by auto
    ultimately show ?thesis
      by simp
  qed
qed

lemma (in general-model) axiom-4-2-validity-aux:
  assumes φ ⇝ 𝒟
  and A ∈ wffs_α
  shows 𝒱 φ ((λx_α. x_α) ▪ A =_α A) = T
proof −
  from assms(2) have (λx_α. x_α) ▪ A =_α A ∈ wffs_o
    using axioms.axiom-4-2 and axioms-are-wffs-of-type-o by blast
  define ψ where ψ = φ((x, α) := 𝒱 φ A)
  with assms have 𝒱 φ ((λx_α. x_α) ▪ A) = 𝒱 ψ (x_α)
    using prop-5401-a by blast
  also from assms and ψ-def have ... = ψ (x, α)
    using 𝒱-is-wff-denotation-function by force
  also from ψ-def have ... = 𝒱 φ A
    by simp
  finally have 𝒱 φ ((λx_α. x_α) ▪ A) = 𝒱 φ A .
```

with *assms(1)* **and** ‹$(\lambda x_\alpha.\ x_\alpha) \cdot A =_\alpha A \in \mathit{wffs}_o$› **show** *?thesis*
using *wffs-from-equality* **and** *prop-5401-b* **by** *meson*
qed

lemma *axiom-4-2-validity*:
assumes $A \in \mathit{wffs}_\alpha$
shows $\models (\lambda x_\alpha.\ x_\alpha) \cdot A =_\alpha A$
proof (*intro allI impI*)
fix $\mathcal{M}$ **and** φ
assume $*$: *is-general-model* $\mathcal{M}\ \varphi \rightsquigarrow_M \mathcal{M}$
show $\mathcal{M} \models_\varphi (\lambda x_\alpha.\ x_\alpha) \cdot A =_\alpha A$
proof −
obtain $\mathcal{D}$ **and** $\mathcal{J}$ **and** $\mathcal{V}$ **where** $\mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V})$
using *prod-cases3* **by** *blast*
moreover from *assms* **and** $*$ **and** ‹$\mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V})$› **have** $\mathcal{V}\ \varphi\ ((\lambda x_\alpha.\ x_\alpha) \cdot A =_\alpha A) = \mathbf{T}$
using *general-model.axiom-4-2-validity-aux* **by** *simp*
ultimately show *?thesis*
by *simp*
qed
qed

lemma (**in** *general-model*) *axiom-4-3-validity-aux*:
assumes $\varphi \rightsquigarrow \mathcal{D}$
and $A \in \mathit{wffs}_\alpha$ **and** $B \in \mathit{wffs}_{\gamma\to\beta}$ **and** $C \in \mathit{wffs}_\gamma$
shows $\mathcal{V}\ \varphi\ ((\lambda x_\alpha.\ B \cdot C) \cdot A =_\beta ((\lambda x_\alpha.\ B) \cdot A) \cdot ((\lambda x_\alpha.\ C) \cdot A)) = \mathbf{T}$
(**is** $\mathcal{V}\ \varphi\ (?A =_\beta ?B) = \mathbf{T}$)
proof −
from *assms(2−4)* **have** $?A =_\beta ?B \in \mathit{wffs}_o$
using *axioms.axiom-4-3* **and** *axioms-are-wffs-of-type-o* **by** *blast*
define ψ **where** $\psi = \varphi((x, \alpha) := \mathcal{V}\ \varphi\ A)$
with *assms(1,2)* **have** $\psi \rightsquigarrow \mathcal{D}$
using $\mathcal{V}$*-is-wff-denotation-function* **by** *auto*
from *assms* **and** ψ*-def* **have** $\mathcal{V}\ \varphi\ ?A = \mathcal{V}\ \psi\ (B \cdot C)$
using *prop-5401-a* **by** *blast*
also from *assms(3,4)* **and** ψ*-def* **and** ‹$\psi \rightsquigarrow \mathcal{D}$› **have** $\ldots = \mathcal{V}\ \psi\ B \cdot \mathcal{V}\ \psi\ C$
using $\mathcal{V}$*-is-wff-denotation-function* **by** *blast*
also from *assms(1−3)* **and** ψ*-def* **have** $\ldots = \mathcal{V}\ \varphi\ ((\lambda x_\alpha.\ B) \cdot A) \cdot \mathcal{V}\ \psi\ C$
using *prop-5401-a* **by** *simp*
also from *assms(1,2,4)* **and** ψ*-def* **have** $\ldots = \mathcal{V}\ \varphi\ ((\lambda x_\alpha.\ B) \cdot A) \cdot \mathcal{V}\ \varphi\ ((\lambda x_\alpha.\ C) \cdot A)$
using *prop-5401-a* **by** *simp*
also have $\ldots = \mathcal{V}\ \varphi\ ?B$
proof −
have $(\lambda x_\alpha.\ B) \cdot A \in \mathit{wffs}_{\gamma\to\beta}$ **and** $(\lambda x_\alpha.\ C) \cdot A \in \mathit{wffs}_\gamma$
using *assms(2−4)* **by** *blast+*
with *assms(1)* **show** *?thesis*
using *wff-app-denotation*[*OF* $\mathcal{V}$*-is-wff-denotation-function*] **by** *simp*
qed
finally have $\mathcal{V}\ \varphi\ ?A = \mathcal{V}\ \varphi\ ?B$.
with *assms(1)* **and** ‹$?A =_\beta ?B \in \mathit{wffs}_o$› **show** *?thesis*

using *prop-5401-b* **and** *wffs-from-equality* **by** *meson*
qed

lemma *axiom-4-3-validity*:
assumes $A \in \mathit{wffs}_\alpha$ **and** $B \in \mathit{wffs}_{\gamma\to\beta}$ **and** $C \in \mathit{wffs}_\gamma$
shows $\models (\lambda x_\alpha.\ B \cdot C) \cdot A =_\beta ((\lambda x_\alpha.\ B) \cdot A) \cdot ((\lambda x_\alpha.\ C) \cdot A)$ (**is** $\models$ *?A* $=_\beta$ *?B*)
proof (*intro allI impI*)
fix $\mathcal{M}$ **and** φ
assume $*$: *is-general-model* $\mathcal{M}$ φ $\rightsquigarrow_M$ $\mathcal{M}$
show $\mathcal{M} \models_\varphi$ *?A* $=_\beta$ *?B*
proof −
obtain $\mathcal{D}$ **and** $\mathcal{J}$ **and** $\mathcal{V}$ **where** $\mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V})$
using *prod-cases3* **by** *blast*
moreover from *assms* **and** $*$ **and** ‹$\mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V})$› **have** $\mathcal{V}\ \varphi$ (*?A* $=_\beta$ *?B*) = **T**
using *general-model.axiom-4-3-validity-aux* **by** *simp*
ultimately show *?thesis*
by *simp*
qed
qed

lemma (**in** *general-model*) *axiom-4-4-validity-aux*:
assumes $\varphi \rightsquigarrow \mathcal{D}$
and $A \in \mathit{wffs}_\alpha$ **and** $B \in \mathit{wffs}_\delta$
and $(y, \gamma) \notin \{(x, \alpha)\} \cup \mathit{vars}\ A$
shows $\mathcal{V}\ \varphi\ ((\lambda x_\alpha.\ \lambda y_\gamma.\ B) \cdot A =_{\gamma\to\delta} (\lambda y_\gamma.\ (\lambda x_\alpha.\ B) \cdot A)) = \mathbf{T}$
(**is** $\mathcal{V}\ \varphi$ (*?A* $=_{\gamma\to\delta}$ *?B*) = **T**)
proof −
from *assms*(*2*,*3*) **have** *?A* $=_{\gamma\to\delta}$ *?B* $\in \mathit{wffs}_o$
using *axioms.axiom-4-4* **and** *axioms-are-wffs-of-type-o* **by** *blast*
let *?D* = $\lambda y_\gamma.\ B$
define ψ **where** $\psi = \varphi((x, \alpha) := \mathcal{V}\ \varphi\ A)$
from *assms*(*1*,*2*) **and** ψ-*def* **have** $\psi \rightsquigarrow \mathcal{D}$
using $\mathcal{V}$-*is-wff-denotation-function* **by** *simp*
{
fix *z*
assume *z* $\in$ *elts* ($\mathcal{D}$ γ)
define φ' **where** $\varphi' = \varphi((y, \gamma) := z)$
from *assms*(*1*) **and** ‹*z* $\in$ *elts* ($\mathcal{D}$ γ)› **and** φ'-*def* **have** $\varphi' \rightsquigarrow \mathcal{D}$
by *simp*
moreover from φ'-*def* **and** *assms*(*4*) **have** $\forall\, v \in \mathit{free\text{-}vars}\ A.\ \varphi\ v = \varphi'\ v$
using *free-vars-in-all-vars* **by** *auto*
ultimately have $\mathcal{V}\ \varphi\ A = \mathcal{V}\ \varphi'\ A$
using *assms*(*1*,*2*) **and** *prop-5400* **by** *blast*
with ψ-*def* **and** φ'-*def* **and** *assms*(*4*) **have** $\varphi'((x, \alpha) := \mathcal{V}\ \varphi'\ A) = \psi((y, \gamma) := z)$
by *auto*
with ‹$\psi \rightsquigarrow \mathcal{D}$› **and** ‹*z* $\in$ *elts* ($\mathcal{D}$ γ)› **and** *assms*(*3*) **have** $\mathcal{V}\ \psi$ *?D* $\cdot$ *z* = $\mathcal{V}\ (\psi((y, \gamma) := z))\ B$
by (*simp add*: *mixed-beta-conversion*)
also from ‹$\varphi' \rightsquigarrow \mathcal{D}$› **and** *assms*(*2*,*3*) **have** $\ldots = \mathcal{V}\ \varphi'\ ((\lambda x_\alpha.\ B) \cdot A)$
using *prop-5401-a* **and** ‹$\varphi'((x, \alpha) := \mathcal{V}\ \varphi'\ A) = \psi((y, \gamma) := z)$› **by** *simp*

```
    also from φ'-def and assms(1) and ‹z ∈ elts (𝒟 γ)› and ‹?A =_{γ→δ} ?B ∈ wffs_o›
    have ... = 𝒱 φ ?B • z
      by (metis mixed-beta-conversion wffs-from-abs wffs-from-equality(2))
    finally have 𝒱 ψ ?D • z = 𝒱 φ ?B • z .
  }
  note ∗ = this
  then have 𝒱 ψ ?D = 𝒱 φ ?B
  proof −
    from ‹ψ ⇝ 𝒟› and assms(3) have 𝒱 ψ ?D = (λz : 𝒟 γ. 𝒱 (ψ((y, γ) := z)) B)
      using wff-abs-denotation[OF 𝒱-is-wff-denotation-function] by simp
    moreover from assms(1) have 𝒱 φ ?B = (λz : 𝒟 γ. 𝒱 (φ((y, γ) := z)) ((λx_α. B) • A))
      using wffs-from-abs[OF wffs-from-equality(2)[OF ‹?A =_{γ→δ} ?B ∈ wffs_o›]]
      and wff-abs-denotation[OF 𝒱-is-wff-denotation-function] by meson
    ultimately show ?thesis
      using vlambda-extensionality and ∗ by fastforce
  qed
  with assms(1−3) and ψ-def have 𝒱 φ ?A = 𝒱 φ ?B
    using prop-5401-a and wffs-of-type-intros(4) by metis
  with assms(1) show ?thesis
    using prop-5401-b and wffs-from-equality[OF ‹?A =_{γ→δ} ?B ∈ wffs_o›] by blast
qed

lemma axiom-4-4-validity:
  assumes A ∈ wffs_α and B ∈ wffs_δ
  and (y, γ) ∉ {(x, α)} ∪ vars A
  shows ⊨ (λx_α. λy_γ. B) • A =_{γ→δ} (λy_γ. (λx_α. B) • A) (is ⊨ ?A =_{γ→δ} ?B)
proof (intro allI impI)
  fix 𝓜 and φ
  assume ∗: is-general-model 𝓜 φ ⇝_M 𝓜
  show 𝓜 ⊨_φ ?A =_{γ→δ} ?B
  proof −
    obtain 𝒟 and 𝒥 and 𝒱 where 𝓜 = (𝒟, 𝒥, 𝒱)
      using prod-cases3 by blast
    moreover from assms and ∗ and ‹𝓜 = (𝒟, 𝒥, 𝒱)› have 𝒱 φ (?A =_{γ→δ} ?B) = T
      using general-model.axiom-4-4-validity-aux by blast
    ultimately show ?thesis
      by simp
  qed
qed

lemma (in general-model) axiom-4-5-validity-aux:
  assumes φ ⇝ 𝒟
  and A ∈ wffs_α and B ∈ wffs_δ
  shows 𝒱 φ ((λx_α. λx_α. B) • A =_{α→δ} (λx_α. B)) = T
proof −
  define ψ where ψ = φ((x, α) := 𝒱 φ A)
  from assms have wff: (λx_α. λx_α. B) • A =_{α→δ} (λx_α. B) ∈ wffs_o
    using axioms.axiom-4-5 and axioms-are-wffs-of-type-o by blast
  with assms(1,2) and ψ-def have 𝒱 φ ((λx_α. λx_α. B) • A) = 𝒱 ψ (λx_α. B)
```

```
    using prop-5401-a and wffs-from-equality(2) by blast
  also have ... = 𝒱 φ (λx_α. B)
  proof −
    have (x, α) ∉ free-vars (λx_α. B)
      by simp
    with ψ-def have ∀ v ∈ free-vars (λx_α. B). φ v = ψ v
      by simp
    moreover from ψ-def and assms(1,2) have ψ ⇝ 𝒟
      using 𝒱-is-wff-denotation-function by simp
    moreover from assms(2,3) have (λx_α. B) ∈ wffs_{α→δ}
      by fastforce
    ultimately show ?thesis
      using assms(1) and prop-5400 by metis
  qed
  finally have 𝒱 φ ((λx_α. λx_α. B) ▪ A) = 𝒱 φ (λx_α. B) .
  with wff and assms(1) show ?thesis
    using prop-5401-b and wffs-from-equality by meson
qed

lemma axiom-4-5-validity:
  assumes A ∈ wffs_α and B ∈ wffs_δ
  shows ⊨ (λx_α. λx_α. B) ▪ A =_{α→δ} (λx_α. B)
proof (intro allI impI)
  fix 𝓜 and φ
  assume ∗: is-general-model 𝓜 φ ⇝_M 𝓜
  show 𝓜 ⊨_φ (λx_α. λx_α. B) ▪ A =_{α→δ} (λx_α. B)
  proof −
    obtain 𝒟 and 𝒥 and 𝒱 where 𝓜 = (𝒟, 𝒥, 𝒱)
      using prod-cases3 by blast
    moreover
    from assms and ∗ and ‹𝓜 = (𝒟, 𝒥, 𝒱)› have 𝒱 φ ((λx_α. λx_α. B) ▪ A =_{α→δ} (λx_α. B)) = T
      using general-model.axiom-4-5-validity-aux by blast
    ultimately show ?thesis
      by simp
  qed
qed

lemma (in general-model) axiom-5-validity-aux:
  assumes φ ⇝ 𝒟
  shows 𝒱 φ (ι ▪ (Q_i ▪ 𝔶_i) =_i 𝔶_i) = T
proof −
  have ι ▪ (Q_i ▪ 𝔶_i) =_i 𝔶_i ∈ wffs_o
    using axioms.axiom-5 and axioms-are-wffs-of-type-o by blast
  have Q_i ▪ 𝔶_i ∈ wffs_{i→o}
    by blast
  with assms have 𝒱 φ (ι ▪ (Q_i ▪ 𝔶_i)) = 𝒱 φ ι ∙ 𝒱 φ (Q_i ▪ 𝔶_i)
    using 𝒱-is-wff-denotation-function by blast
  also from assms have ... = 𝒱 φ ι ∙ (𝒱 φ (Q_i) ∙ 𝒱 φ (𝔶_i))
    using wff-app-denotation[OF 𝒱-is-wff-denotation-function] by (metis Q-wff wffs-of-type-intros(1))
```

also from *assms* **have** $\ldots = \mathcal{J}\ (\mathfrak{c}_\iota,\ (i{\to}o){\to}i) \cdot (\mathcal{J}\ (\mathfrak{c}_Q,\ i{\to}i{\to}o) \cdot \mathcal{V}\ \varphi\ (\mathfrak{y}_i))$
using *$\mathcal{V}$-is-wff-denotation-function* **by** *auto*
also from *assms* **have** $\ldots = \mathcal{J}\ (\mathfrak{c}_\iota,\ (i{\to}o){\to}i) \cdot ((q_i{}^{\mathcal{D}}) \cdot \mathcal{V}\ \varphi\ (\mathfrak{y}_i))$
using *Q-constant-of-type-def* **and** *Q-denotation* **by** *simp*
also from *assms* **have** $\ldots = \mathcal{J}\ (\mathfrak{c}_\iota,\ (i{\to}o){\to}i) \cdot \{\mathcal{V}\ \varphi\ (\mathfrak{y}_i)\}_i{}^{\mathcal{D}}$
using *$\mathcal{V}$-is-wff-denotation-function* **by** *auto*
finally have $\mathcal{V}\ \varphi\ (\iota \cdot (Q_i \cdot \mathfrak{y}_i)) = \mathcal{J}\ (\mathfrak{c}_\iota,\ (i{\to}o){\to}i) \cdot \{\mathcal{V}\ \varphi\ (\mathfrak{y}_i)\}_i{}^{\mathcal{D}}$.
moreover from *assms* **have** $\mathcal{J}\ (\mathfrak{c}_\iota,\ (i{\to}o){\to}i) \cdot \{\mathcal{V}\ \varphi\ (\mathfrak{y}_i)\}_i{}^{\mathcal{D}} = \mathcal{V}\ \varphi\ (\mathfrak{y}_i)$
using *$\mathcal{V}$-is-wff-denotation-function* **and** *ι-denotation* **by** *force*
ultimately have $\mathcal{V}\ \varphi\ (\iota \cdot (Q_i \cdot \mathfrak{y}_i)) = \mathcal{V}\ \varphi\ (\mathfrak{y}_i)$
by (*simp only*:)
with *assms* **and** $\langle Q_i \cdot \mathfrak{y}_i \in \mathit{wffs}_{i\to o}\rangle$ **show** *?thesis*
using *prop-5401-b* **by** *blast*
qed

lemma *axiom-5-validity*:
shows $\models \iota \cdot (Q_i \cdot \mathfrak{y}_i) =_i \mathfrak{y}_i$
proof (*intro allI impI*)
fix $\mathcal{M}$ **and** φ
assume $*$: *is-general-model* $\mathcal{M}\ \varphi \rightsquigarrow_M \mathcal{M}$
show $\mathcal{M} \models_\varphi \iota \cdot (Q_i \cdot \mathfrak{y}_i) =_i \mathfrak{y}_i$
proof −
obtain $\mathcal{D}$ **and** $\mathcal{J}$ **and** $\mathcal{V}$ **where** $\mathcal{M} = (\mathcal{D},\ \mathcal{J},\ \mathcal{V})$
using *prod-cases3* **by** *blast*
moreover from $*$ **and** $\langle \mathcal{M} = (\mathcal{D},\ \mathcal{J},\ \mathcal{V})\rangle$ **have** $\mathcal{V}\ \varphi\ (\iota \cdot (Q_i \cdot \mathfrak{y}_i) =_i \mathfrak{y}_i) = \mathbf{T}$
using *general-model.axiom-5-validity-aux* **by** *simp*
ultimately show *?thesis*
by *simp*
qed
qed

lemma *axioms-validity*:
assumes $A \in \mathit{axioms}$
shows $\models A$
using *assms*
and *axiom-1-validity*
and *axiom-2-validity*
and *axiom-3-validity*
and *axiom-4-1-con-validity*
and *axiom-4-1-var-validity*
and *axiom-4-2-validity*
and *axiom-4-3-validity*
and *axiom-4-4-validity*
and *axiom-4-5-validity*
and *axiom-5-validity*
by *cases auto*

lemma (**in** *general-model*) *rule-R-validity-aux*:
assumes $A \in \mathit{wffs}_\alpha$ **and** $B \in \mathit{wffs}_\alpha$

```
  and ∀φ. φ ⇝ 𝒟 ⟶ 𝒱 φ A = 𝒱 φ B
  and C ∈ wffs_β and C' ∈ wffs_β
  and p ∈ positions C and A ≼_p C and C⟨p ← B⟩ ▷ C'
  shows ∀φ. φ ⇝ 𝒟 ⟶ 𝒱 φ C = 𝒱 φ C'
proof −
  from assms(8,3−5,7) show ?thesis
  proof (induction arbitrary: β)
    case pos-found
    then show ?case
      by simp
  next
    case (replace-left-app p G B' G' H)
    show ?case
    proof (intro allI impI)
      fix φ
      assume φ ⇝ 𝒟
      from ‹G ∙ H ∈ wffs_β› obtain γ where G ∈ wffs_{γ→β} and H ∈ wffs_γ
        by (rule wffs-from-app)
      with ‹G' ∙ H ∈ wffs_β› have G' ∈ wffs_{γ→β}
        by (metis wff-has-unique-type wffs-from-app)
      from assms(1) and ‹φ ⇝ 𝒟› and ‹G ∈ wffs_{γ→β}› and ‹H ∈ wffs_γ›
      have 𝒱 φ (G ∙ H) = 𝒱 φ G ∙ 𝒱 φ H
        using 𝒱-is-wff-denotation-function by blast
      also from ‹φ ⇝ 𝒟› and ‹G ∈ wffs_{γ→β}› and ‹G' ∈ wffs_{γ→β}› have ... = 𝒱 φ G' ∙ 𝒱 φ H
        using replace-left-app.IH and replace-left-app.prems(1,4) by simp
      also from assms(1) and ‹φ ⇝ 𝒟› and ‹G' ∈ wffs_{γ→β}› and ‹H ∈ wffs_γ›
      have ... = 𝒱 φ (G' ∙ H)
        using 𝒱-is-wff-denotation-function by fastforce
      finally show 𝒱 φ (G ∙ H) = 𝒱 φ (G' ∙ H) .
    qed
  next
    case (replace-right-app p H B' H' G)
    show ?case
    proof (intro allI impI)
      fix φ
      assume φ ⇝ 𝒟
      from ‹G ∙ H ∈ wffs_β› obtain γ where G ∈ wffs_{γ→β} and H ∈ wffs_γ
        by (rule wffs-from-app)
      with ‹G ∙ H' ∈ wffs_β› have H' ∈ wffs_γ
        using wff-has-unique-type and wffs-from-app by (metis type.inject)
      from assms(1) and ‹φ ⇝ 𝒟› and ‹G ∈ wffs_{γ→β}› and ‹H ∈ wffs_γ›
      have 𝒱 φ (G ∙ H) = 𝒱 φ G ∙ 𝒱 φ H
        using 𝒱-is-wff-denotation-function by blast
      also from ‹φ ⇝ 𝒟› and ‹H ∈ wffs_γ› and ‹H' ∈ wffs_γ› have ... = 𝒱 φ G ∙ 𝒱 φ H'
        using replace-right-app.IH and replace-right-app.prems(1,4) by force
      also from assms(1) and ‹φ ⇝ 𝒟› and ‹G ∈ wffs_{γ→β}› and ‹H' ∈ wffs_γ›
      have ... = 𝒱 φ (G ∙ H')
        using 𝒱-is-wff-denotation-function by fastforce
```

```
    finally show 𝒱 φ (G ⬝ H) = 𝒱 φ (G ⬝ H′) .
  qed
 next
  case (replace-abs p E B′ E′ x γ)
  show ?case
  proof (intro allI impI)
   fix φ
   assume φ ⇝ 𝒟
   define ψ where ψ z = φ((x, γ) := z) for z
   with ‹φ ⇝ 𝒟› have ψ-assg: ψ z ⇝ 𝒟 if z ∈ elts (𝒟 γ) for z
    by (simp add: that)
   from ‹λx_γ. E ∈ wffs_β› obtain δ where β = γ→δ and E ∈ wffs_δ
    by (rule wffs-from-abs)
   with ‹λx_γ. E′ ∈ wffs_β› have E′ ∈ wffs_δ
    using wffs-from-abs by blast
   from assms(1) and ‹φ ⇝ 𝒟› and ‹E ∈ wffs_δ› and ψ-def
   have 𝒱 φ (λx_γ. E) = (λz : 𝒟 γ. 𝒱 (ψ z) E)
    using wff-abs-denotation[OF 𝒱-is-wff-denotation-function] by simp
   also have ... = (λz : 𝒟 γ. 𝒱 (ψ z) E′)
   proof (intro vlambda-extensionality)
    fix z
    assume z ∈ elts (𝒟 γ)
    from ‹E ∈ wffs_δ› and ‹E′ ∈ wffs_δ› have ∀φ. φ ⇝ 𝒟 ⟶ 𝒱 φ E = 𝒱 φ E′
     using replace-abs.prems(1,4) and replace-abs.IH by simp
    with ψ-assg and ‹z ∈ elts (𝒟 γ)› show 𝒱 (ψ z) E = 𝒱 (ψ z) E′
     by simp
   qed
   also from assms(1) and ‹φ ⇝ 𝒟› and ‹E′ ∈ wffs_δ› and ψ-def
   have ... = 𝒱 φ (λx_γ. E′)
    using wff-abs-denotation[OF 𝒱-is-wff-denotation-function] by simp
   finally show 𝒱 φ (λx_γ. E) = 𝒱 φ (λx_γ. E′) .
  qed
 qed
qed

lemma rule-R-validity:
 assumes C ∈ wffs_o and C′ ∈ wffs_o and E ∈ wffs_o
 and ⊨ C and ⊨ E
 and is-rule-R-app p C′ C E
 shows ⊨ C′
proof (intro allI impI)
 fix 𝓜 and φ
 assume is-general-model 𝓜 and φ ⇝_M 𝓜
 show 𝓜 ⊨_φ C′
 proof −
  have 𝓜 ⊨ C′
  proof −
   obtain 𝒟 and 𝒥 and 𝒱 where 𝓜 = (𝒟, 𝒥, 𝒱)
    using prod-cases3 by blast
```

```
    from assms(6) obtain A and B and α where A ∈ wffs_α and B ∈ wffs_α and E = A =_α B
      using wffs-from-equality by (meson is-rule-R-app-def)
    note ∗ = ‹is-general-model M› ‹M = (D, J, V)› ‹φ ⇝_M M›
    have V φ' C = V φ' C' if φ' ⇝ D for φ'
    proof −
      from assms(5) and ∗(1,2) and ‹A ∈ wffs_α› and ‹B ∈ wffs_α› and ‹E = A =_α B› and that
      have ∀φ'. φ' ⇝ D ⟶ V φ' A = V φ' B
        using general-model.prop-5401-b by blast
      moreover
      from ‹E = A =_α B› and assms(6) have p ∈ positions C and A ≼_p C and C⟨p ← B⟩ ▷ C'
        using is-subform-implies-in-positions by auto
      ultimately show ?thesis
        using ‹A ∈ wffs_α› and ‹B ∈ wffs_α› and ‹C ∈ wffs_o› and assms(2) and that and ∗(1,2)
        and general-model.rule-R-validity-aux by blast
    qed
    with assms(4) and ∗(1,2) show ?thesis
      by simp
  qed
  with ‹φ ⇝_M M› show ?thesis
    by blast
 qed
qed

lemma individual-proof-step-validity:
 assumes is-proof S and A ∈ lset S
 shows ⊨ A
using assms proof (induction length S arbitrary: S A rule: less-induct)
 case less
 from ‹A ∈ lset S› obtain i' where S ! i' = A and S ≠ [] and i' < length S
   by (metis empty-iff empty-set in-set-conv-nth)
 with ‹is-proof S› have is-proof (take (Suc i') S) and take (Suc i') S ≠ []
   using proof-prefix-is-proof[where S₁ = take (Suc i') S and S₂ = drop (Suc i') S]
   and append-take-drop-id by simp-all
 from ‹i' < length S› consider (a) i' < length S − 1 | (b) i' = length S − 1
   by fastforce
 then show ?case
 proof cases
   case a
   then have length (take (Suc i') S) < length S
     by simp
   with ‹S ! i' = A› and ‹take (Suc i') S ≠ []› have A ∈ lset (take (Suc i') S)
     by (simp add: take-Suc-conv-app-nth)
   with ‹length (take (Suc i') S) < length S› and ‹is-proof (take (Suc i') S)› show ?thesis
     using less(1) by blast
 next
   case b
   with ‹S ! i' = A› and ‹S ≠ []› have last S = A
     using last-conv-nth by blast
   with ‹is-proof S› and ‹S ≠ []› and b have is-proof-step S i'
```

```
    using added-suffix-proof-preservation[where 𝒮′ = []] by simp
  then consider
    (axiom) 𝒮 ! i′ ∈ axioms
  | (rule-R) ∃ p j k. {j, k} ⊆ {0..<i′} ∧ is-rule-R-app p (𝒮 ! i′) (𝒮 ! j) (𝒮 ! k)
    by fastforce
  then show ?thesis
  proof cases
    case axiom
    with ‹𝒮 ! i′ = A› show ?thesis
      by (blast dest: axioms-validity)
  next
    case rule-R
    then obtain p and j and k
      where {j, k} ⊆ {0..<i′} and is-rule-R-app p (𝒮 ! i′) (𝒮 ! j) (𝒮 ! k)
      by blast
    let ?𝒮_j = take (Suc j) 𝒮 and ?𝒮_k = take (Suc k) 𝒮
    obtain 𝒮_j′ and 𝒮_k′ where 𝒮 = ?𝒮_j @ 𝒮_j′ and 𝒮 = ?𝒮_k @ 𝒮_k′
      by (metis append-take-drop-id)
    with ‹is-proof 𝒮› have is-proof (?𝒮_j @ 𝒮_j′) and is-proof (?𝒮_k @ 𝒮_k′)
      by (simp-all only:)
    moreover from ‹𝒮 ≠ []› have ?𝒮_j ≠ [] and ?𝒮_k ≠ []
      by simp-all
    ultimately have is-proof-of ?𝒮_j (last ?𝒮_j) and is-proof-of ?𝒮_k (last ?𝒮_k)
      using proof-prefix-is-proof-of-last[where 𝒮 = ?𝒮_j and 𝒮′ = 𝒮_j′]
      and proof-prefix-is-proof-of-last[where 𝒮 = ?𝒮_k and 𝒮′ = 𝒮_k′]
      by fastforce+
    moreover
    from ‹{j, k} ⊆ {0..<i′}› and b have length ?𝒮_j < length 𝒮 and length ?𝒮_k < length 𝒮
      by force+
    moreover from calculation(3,4) have 𝒮 ! j ∈ lset ?𝒮_j and 𝒮 ! k ∈ lset ?𝒮_k
      by (simp-all add: take-Suc-conv-app-nth)
    ultimately have ⊨ 𝒮 ! j and ⊨ 𝒮 ! k
      using ‹?𝒮_j ≠ []› and ‹?𝒮_k ≠ []› and less(1) unfolding is-proof-of-def by presburger+
    moreover have 𝒮 ! i′ ∈ wffs_o and 𝒮 ! j ∈ wffs_o and 𝒮 ! k ∈ wffs_o
      using ‹is-rule-R-app p (𝒮 ! i′) (𝒮 ! j) (𝒮 ! k)› and replacement-preserves-typing
      by force+
    ultimately show ?thesis
      using ‹is-rule-R-app p (𝒮 ! i′) (𝒮 ! j) (𝒮 ! k)› and ‹𝒮 ! i′ = A›
      and rule-R-validity[where C′ = A] by blast
  qed
 qed
qed

lemma semantic-modus-ponens:
 assumes is-general-model 𝓜
 and A ∈ wffs_o and B ∈ wffs_o
 and 𝓜 ⊨ A ⊃^Q B
 and 𝓜 ⊨ A
 shows 𝓜 ⊨ B
```

proof (*intro allI impI*)
fix φ
assume $\varphi \rightsquigarrow_M \mathcal{M}$
moreover obtain $\mathcal{D}$ **and** $\mathcal{J}$ **and** $\mathcal{V}$ **where** $\mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V})$
using *prod-cases3* **by** *blast*
ultimately have $\varphi \rightsquigarrow \mathcal{D}$
by *simp*
show $\mathcal{M} \models_\varphi B$
proof −
from *assms*(*4*) **have** $\mathcal{V}\ \varphi\ (A \supset^{\mathfrak{Q}} B) = \mathbf{T}$
using ‹$\mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V})$› **and** ‹$\varphi \rightsquigarrow_M \mathcal{M}$› **by** *auto*
with *assms*(*1*−*3*) **have** $\mathcal{V}\ \varphi\ A \supset \mathcal{V}\ \varphi\ B = \mathbf{T}$
using ‹$\mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V})$› **and** ‹$\varphi \rightsquigarrow_M \mathcal{M}$› **and** *general-model.prop-5401-f'* **by** *simp*
moreover from *assms*(*5*) **have** $\mathcal{V}\ \varphi\ A = \mathbf{T}$
using ‹$\mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V})$› **and** ‹$\varphi \rightsquigarrow \mathcal{D}$› **by** *auto*
moreover from ‹$\mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V})$› **and** *assms*(*1*) **have** *elts* $(\mathcal{D}\ o)$ = *elts* $\mathbb{B}$
using *frame.truth-values-domain-def* **and** *general-model-def* **and** *premodel-def* **by** *fastforce*
with *assms* **and** ‹$\mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V})$› **and** ‹$\varphi \rightsquigarrow \mathcal{D}$› **and** ‹$\mathcal{V}\ \varphi\ A = \mathbf{T}$› **have** $\{\mathcal{V}\ \varphi\ A, \mathcal{V}\ \varphi\ B\} \subseteq$ *elts* $\mathbb{B}$
using *general-model.𝒱-is-wff-denotation-function*
and *premodel.wff-denotation-function-is-domain-respecting* **and** *general-model.axioms*(*1*) **by** *blast*
ultimately have $\mathcal{V}\ \varphi\ B = \mathbf{T}$
by *fastforce*
with ‹$\mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V})$› **and** *assms*(*1*) **and** ‹$\varphi \rightsquigarrow \mathcal{D}$› **show** *?thesis*
by *simp*
qed
qed

lemma *generalized-semantic-modus-ponens*:
assumes *is-general-model* $\mathcal{M}$
and *lset hs* $\subseteq$ *wffs*$_o$
and $\forall H \in$ *lset hs*. $\mathcal{M} \models H$
and $P \in$ *wffs*$_o$
and $\mathcal{M} \models hs \supset^{\mathfrak{Q}}_\star P$
shows $\mathcal{M} \models P$
using *assms*(*2*−*5*) **proof** (*induction hs arbitrary*: *P rule*: *rev-induct*)
case *Nil*
then show *?case* **by** *simp*
next
case (*snoc H' hs*)
from ‹$\mathcal{M} \models (hs\ @\ [H']) \supset^{\mathfrak{Q}}_\star P$› **have** $\mathcal{M} \models hs \supset^{\mathfrak{Q}}_\star (H' \supset^{\mathfrak{Q}} P)$
by *simp*
moreover from ‹$\forall H \in$ *lset* $(hs\ @\ [H'])$. $\mathcal{M} \models H$› **and** ‹*lset* $(hs\ @\ [H']) \subseteq$ *wffs*$_o$›
have $\forall H \in$ *lset hs*. $\mathcal{M} \models H$ **and** *lset hs* $\subseteq$ *wffs*$_o$
by *simp-all*
moreover from ‹*lset* $(hs\ @\ [H']) \subseteq$ *wffs*$_o$› **and** ‹$P \in$ *wffs*$_o$› **have** $H' \supset^{\mathfrak{Q}} P \in$ *wffs*$_o$
by *auto*
ultimately have $\mathcal{M} \models H' \supset^{\mathfrak{Q}} P$
by (*elim snoc.IH*)

moreover from ‹$\forall\, H \in \mathit{lset}\ (\mathit{hs}\ @\ [H']).\ \mathcal{M} \models H$› **have** $\mathcal{M} \models H'$
by *simp*
moreover from ‹$H' \supset^{\mathcal{Q}} P \in \mathit{wffs}_o$› **have** $H' \in \mathit{wffs}_o$
using *wffs-from-imp-op*(*1*) **by** *blast*
ultimately show *?case*
using *assms*(*1*) **and** ‹$P \in \mathit{wffs}_o$› **and** *semantic-modus-ponens* **by** *simp*
qed

8.3 Proposition 5402(a)

proposition *theoremhood-implies-validity*:
assumes *is-theorem* A
shows $\models A$
using *assms* **and** *individual-proof-step-validity* **by** *force*

8.4 Proposition 5402(b)

proposition *hyp-derivability-implies-validity*:
assumes *is-hyps* $\mathcal{G}$
and *is-model-for* $\mathcal{M}\ \mathcal{G}$
and $\mathcal{G} \vdash A$
and *is-general-model* $\mathcal{M}$
shows $\mathcal{M} \models A$
proof −
from *assms*(*3*) **have** $A \in \mathit{wffs}_o$
by (*fact hyp-derivable-form-is-wffso*)
from ‹$\mathcal{G} \vdash A$› **and** ‹*is-hyps* $\mathcal{G}$› **obtain** $\mathcal{H}$ **where** *finite* $\mathcal{H}$ **and** $\mathcal{H} \subseteq \mathcal{G}$ **and** $\mathcal{H} \vdash A$
by *blast*
moreover from ‹*finite* $\mathcal{H}$› **obtain** *hs* **where** *lset hs* $= \mathcal{H}$
using *finite-list* **by** *blast*
ultimately have $\vdash \mathit{hs} \supset^{\mathcal{Q}}_{\star} A$
using *generalized-deduction-theorem* **by** *simp*
with *assms*(*4*) **have** $\mathcal{M} \models \mathit{hs} \supset^{\mathcal{Q}}_{\star} A$
using *derivability-from-no-hyps-theoremhood-equivalence* **and** *theoremhood-implies-validity*
by *blast*
moreover from ‹$\mathcal{H} \subseteq \mathcal{G}$› **and** *assms*(*2*) **have** $\mathcal{M} \models H$ **if** $H \in \mathcal{H}$ **for** H
using *that* **by** *blast*
moreover from ‹$\mathcal{H} \subseteq \mathcal{G}$› **and** ‹*lset hs* $= \mathcal{H}$› **and** *assms*(*1*) **have** *lset hs* $\subseteq \mathit{wffs}_o$
by *blast*
ultimately show *?thesis*
using *assms*(*1*,*4*) **and** ‹$A \in \mathit{wffs}_o$› **and** ‹*lset hs* $= \mathcal{H}$› **and** *generalized-semantic-modus-ponens*
by *auto*
qed

8.5 Theorem 5402 (Soundness Theorem)

lemmas *thm-5402* = *theoremhood-implies-validity hyp-derivability-implies-validity*

end

9 Consistency

theory *Consistency*
imports
Soundness
begin

definition *is-inconsistent-set* :: *form set* ⇒ *bool* **where**
[*iff*]: *is-inconsistent-set* $\mathcal{G}$ ⟷ $\mathcal{G} \vdash F_o$

definition $\mathcal{Q}_0$*-is-inconsistent* :: *bool* **where**
[*iff*]: $\mathcal{Q}_0$*-is-inconsistent* ⟷ $\vdash F_o$

definition *is-wffo-consistent-with* :: *form* ⇒ *form set* ⇒ *bool* **where**
[*iff*]: *is-wffo-consistent-with* B $\mathcal{G}$ ⟷ ¬ *is-inconsistent-set* ($\mathcal{G} \cup \{B\}$)

9.1 Existence of a standard model

We construct a standard model in which $\mathcal{D}$ i is the set $\{0\}$:

primrec *singleton-standard-domain-family* ($\mathcal{D}^S$) **where**
$\mathcal{D}^S$ $i = 1$ — i.e., $\mathcal{D}^S$ i = *ZFC-in-HOL.set* $\{0\}$
| $\mathcal{D}^S$ $o = \mathbb{B}$
| $\mathcal{D}^S$ ($\alpha \to \beta$) = $\mathcal{D}^S$ $\alpha \longmapsto \mathcal{D}^S$ β

interpretation *singleton-standard-frame*: *frame* $\mathcal{D}^S$
proof *unfold-locales*
{
fix α
have $\mathcal{D}^S$ $\alpha \neq 0$
proof (*induction* α)
case (*TFun* β γ)
from ‹$\mathcal{D}^S$ $\gamma \neq 0$› **obtain** y **where** $y \in$ *elts* ($\mathcal{D}^S$ γ)
by *fastforce*
then have (**λ**z **:** $\mathcal{D}^S$ β**.** y) ∈ *elts* ($\mathcal{D}^S$ $\beta \longmapsto \mathcal{D}^S$ γ)
by (*intro VPi-I*)
then show *?case*
by *force*
qed *simp-all*
}
then show ∀ α. $\mathcal{D}^S$ $\alpha \neq 0$
by (*intro allI*)
qed *simp-all*

definition *singleton-standard-constant-denotation-function* ($\mathcal{J}^S$) **where**
[*simp*]: $\mathcal{J}^S$ k =
(
if
∃ β. *is-Q-constant-of-type* k β
then

```
    let β = type-of-Q-constant k in q_β^{D^S}
  else
  if
    is-iota-constant k
  then
    λz : D^S (i→o). 0
  else
    case k of (c, α) ⇒ SOME z. z ∈ elts (D^S α)
)

interpretation singleton-standard-premodel: premodel D^S J^S
proof (unfold-locales)
  show ∀α. J^S (Q-constant-of-type α) = q_α^{D^S}
    by simp
next
  show singleton-standard-frame.is-unique-member-selector (J^S iota-constant)
  unfolding singleton-standard-frame.is-unique-member-selector-def proof
    fix x
    assume x ∈ elts (D^S i)
    then have x = 0
      by simp
    moreover have (λz : D^S (i→o). 0) · {0}_i^{D^S} = 0
      using beta[OF singleton-standard-frame.one-element-function-is-domain-respecting]
      unfolding singleton-standard-domain-family.simps(3) by blast
    ultimately show (J^S iota-constant) · {x}_i^{D^S} = x
      by fastforce
  qed
next
  show ∀c α. ¬ is-logical-constant (c, α) ⟶ J^S (c, α) ∈ elts (D^S α)
  proof (intro allI impI)
    fix c and α
    assume ¬ is-logical-constant (c, α)
    then have J^S (c, α) = (SOME z. z ∈ elts (D^S α))
      by auto
    moreover have ∃z. z ∈ elts (D^S α)
      using eq0-iff and singleton-standard-frame.domain-nonemptiness by presburger
    then have (SOME z. z ∈ elts (D^S α)) ∈ elts (D^S α)
      using some-in-eq by auto
    ultimately show J^S (c, α) ∈ elts (D^S α)
      by auto
  qed
qed

fun singleton-standard-wff-denotation-function (V^S) where
  V^S φ (x_α) = φ (x, α)
| V^S φ (⦃c⦄α) = J^S (c, α)
| V^S φ (A ▪ B) = (V^S φ A) · (V^S φ B)
| V^S φ (λx_α. A) = (λz : D^S α. V^S (φ((x, α) := z)) A)
```

lemma *singleton-standard-wff-denotation-function-closure*:
assumes *frame.is-assignment* $\mathcal{D}^S$ φ
and $A \in wffs_\alpha$
shows $\mathcal{V}^S$ φ $A \in$ *elts* ($\mathcal{D}^S$ α)
using *assms*(*2*,*1*) **proof** (*induction A arbitrary*: φ)
case (*var-is-wff* α x)
then show *?case*
by *simp*
next
case (*con-is-wff* α c)
then show *?case*
proof (*cases* (c, α) *rule*: *constant-cases*)
case *non-logical*
then show *?thesis*
using *singleton-standard-premodel.non-logical-constant-denotation*
and *singleton-standard-wff-denotation-function.simps*(*2*) **by** *presburger*
next
case (*Q-constant* β)
then have $\mathcal{V}^S$ φ ($\{\!|c|\!\}_\alpha$) = ${q_\beta}^{\mathcal{D}^S}$
by *simp*
moreover have ${q_\beta}^{\mathcal{D}^S} \in$ *elts* ($\mathcal{D}^S$ ($\beta \to \beta \to o$))
using *singleton-standard-domain-family.simps*(*3*)
and *singleton-standard-frame.identity-relation-is-domain-respecting* **by** *presburger*
ultimately show *?thesis*
using *Q-constant* **by** *simp*
next
case ι*-constant*
then have $\mathcal{V}^S$ φ ($\{\!|c|\!\}_\alpha$) = ($\boldsymbol{\lambda} z : \mathcal{D}^S$ ($i \to o$)**.** *0*)
by *simp*
moreover have ($\boldsymbol{\lambda} z : \mathcal{D}^S$ ($i \to o$)**.** *0*) $\in$ *elts* ($\mathcal{D}^S$ (($i \to o$)$\to i$))
by (*simp add*: *VPi-I*)
ultimately show *?thesis*
using ι*-constant* **by** *simp*
qed
next
case (*app-is-wff* α β A B)
have $\mathcal{V}^S$ φ (A ▪ B) = ($\mathcal{V}^S$ φ A) ⋅ ($\mathcal{V}^S$ φ B)
using *singleton-standard-wff-denotation-function.simps*(*3*) **.**
moreover have $\mathcal{V}^S$ φ $A \in$ *elts* ($\mathcal{D}^S$ ($\alpha \to \beta$)) **and** $\mathcal{V}^S$ φ $B \in$ *elts* ($\mathcal{D}^S$ α)
using *app-is-wff.IH* **and** *app-is-wff.prems* **by** *simp-all*
ultimately show *?case*
by (*simp only*: *singleton-standard-frame.app-is-domain-respecting*)
next
case (*abs-is-wff* β A α x)
have $\mathcal{V}^S$ φ (λx_α. A) = ($\boldsymbol{\lambda} z : \mathcal{D}^S$ α**.** $\mathcal{V}^S$ ($\varphi((x, \alpha) := z)$) A)
using *singleton-standard-wff-denotation-function.simps*(*4*) **.**
moreover have $\mathcal{V}^S$ ($\varphi((x, \alpha) := z)$) $A \in$ *elts* ($\mathcal{D}^S$ β) **if** $z \in$ *elts* ($\mathcal{D}^S$ α) **for** z
using *that* **and** *abs-is-wff.IH* **and** *abs-is-wff.prems* **by** *simp*

ultimately show *?case*
by (*simp add*: *VPi-I*)
qed

interpretation *singleton-standard-model*: *standard-model* $\mathcal{D}^S$ $\mathcal{J}^S$ $\mathcal{V}^S$
proof (*unfold-locales*)
show *singleton-standard-premodel.is-wff-denotation-function* $\mathcal{V}^S$
by (*simp add*: *singleton-standard-wff-denotation-function-closure*)
next
show $\forall\, \alpha\ \beta.\ \mathcal{D}^S\ (\alpha{\to}\beta) = \mathcal{D}^S\ \alpha \longmapsto \mathcal{D}^S\ \beta$
using *singleton-standard-domain-family.simps*(*3*) **by** (*intro allI*)
qed

proposition *standard-model-existence*:
shows $\exists\, \mathcal{M}.$ *is-standard-model* $\mathcal{M}$
using *singleton-standard-model.standard-model-axioms* **by** *auto*

9.2 Theorem 5403 (Consistency Theorem)

proposition *model-existence-implies-set-consistency*:
assumes *is-hyps* $\mathcal{G}$
and $\exists\, \mathcal{M}.$ *is-general-model* $\mathcal{M} \wedge$ *is-model-for* $\mathcal{M}\ \mathcal{G}$
shows $\neg$ *is-inconsistent-set* $\mathcal{G}$
proof (*rule ccontr*)
from *assms*(*2*) **obtain** $\mathcal{D}$ **and** $\mathcal{J}$ **and** $\mathcal{V}$ **and** $\mathcal{M}$
where $\mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V})$ **and** *is-model-for* $\mathcal{M}\ \mathcal{G}$ **and** *is-general-model* $\mathcal{M}$ **by** *fastforce*
assume $\neg\ \neg$ *is-inconsistent-set* $\mathcal{G}$
then have $\mathcal{G} \vdash F_o$
by *simp*
with ‹*is-general-model* $\mathcal{M}$› **have** $\mathcal{M} \models F_o$
using *thm-5402*(*2*)[*OF assms*(*1*) ‹*is-model-for* $\mathcal{M}\ \mathcal{G}$›] **by** *simp*
then have $\mathcal{V}\ \varphi\ F_o = \mathbf{T}$ **if** $\varphi \rightsquigarrow \mathcal{D}$ **for** φ
using *that* **and** ‹$\mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V})$› **by** *force*
moreover have $\mathcal{V}\ \varphi\ F_o = \mathbf{F}$ **if** $\varphi \rightsquigarrow \mathcal{D}$ **for** φ
using ‹$\mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V})$› **and** ‹*is-general-model* $\mathcal{M}$› **and** *that* **and** *general-model.prop-5401-d*
by *simp*
ultimately have $\nexists\, \varphi.\ \varphi \rightsquigarrow \mathcal{D}$
by (*auto simp add*: *inj-eq*)
moreover have $\exists\, \varphi.\ \varphi \rightsquigarrow \mathcal{D}$
proof −
— Since by definition domains are not empty then, by using the Axiom of Choice, we can specify an assignment ψ that simply chooses some element in the respective domain for each variable. Nonetheless, as pointed out in Footnote 11, page 19 in [1], it is not necessary to use the Axiom of Choice to show that assignments exist since some assignments can be described explicitly.
let $?\psi = \lambda v.$ *case v of* (-, α) ⇒ *SOME z.* $z \in$ *elts* ($\mathcal{D}\ \alpha$)
from ‹$\mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V})$› **and** ‹*is-general-model* $\mathcal{M}$› **have** $\forall\, \alpha.$ *elts* ($\mathcal{D}\ \alpha$) $\neq \{\}$
using *frame.domain-nonemptiness* **and** *premodel-def* **and** *general-model.axioms*(*1*) **by** *auto*
with ‹$\mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V})$› **and** ‹*is-general-model* $\mathcal{M}$› **have** $?\psi \rightsquigarrow \mathcal{D}$
using *frame.is-assignment-def* **and** *premodel-def* **and** *general-model.axioms*(*1*)

```
    by (metis (mono-tags) case-prod-conv some-in-eq)
  then show ?thesis
    by (intro exI)
 qed
 ultimately show False ..
qed

proposition 𝒬₀-is-consistent:
 shows ¬ 𝒬₀-is-inconsistent
proof −
 have ∃ℳ. is-general-model ℳ ∧ is-model-for ℳ {}
  using standard-model-existence and standard-model.axioms(1) by blast
 then show ?thesis
  using model-existence-implies-set-consistency by simp
qed

lemmas thm-5403 = 𝒬₀-is-consistent model-existence-implies-set-consistency

proposition principle-of-explosion:
 assumes is-hyps 𝒢
 shows is-inconsistent-set 𝒢 ⟷ (∀ A ∈ (wffs_o). 𝒢 ⊢ A)
proof
 assume is-inconsistent-set 𝒢
 show ∀ A ∈ (wffs_o). 𝒢 ⊢ A
 proof
  fix A
  assume A ∈ wffs_o
  from ‹is-inconsistent-set 𝒢› have 𝒢 ⊢ F_o
   unfolding is-inconsistent-set-def .
  then have 𝒢 ⊢ ∀ 𝔯_o. 𝔯_o
   unfolding false-is-forall .
  with ‹A ∈ wffs_o› have 𝒢 ⊢ S {(𝔯, o) ⇀ A} (𝔯_o)
   using ∀ I by fastforce
  then show 𝒢 ⊢ A
   by simp
 qed
next
 assume ∀ A ∈ (wffs_o). 𝒢 ⊢ A
 then have 𝒢 ⊢ F_o
  using false-wff by (elim bspec)
 then show is-inconsistent-set 𝒢
  unfolding is-inconsistent-set-def .
qed

end
```

References

[1] P. B. Andrews. *A Transfinite Type Theory with Type Variables*, volume 36 of *Studies in Logic and the Foundations of Mathematics*. North-Holland Publishing Company, 1965.

[2] P. B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof*, volume 27 of *Applied Logic Series*. Springer Dordrecht, 2002.