Metatheory of $\mathcal{Q}_0$

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

April 18, 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** $lset \equiv List.set$

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 :: $['a,\ 'a] \Rightarrow fmaplet$ (- /$$:=/ -)
  -fmaplets :: $['a,\ 'a] \Rightarrow fmaplet$ (- /[$$:=]/ -)

syntax
  -fmaplet :: $['a,\ 'a] \Rightarrow fmaplet$ (- /$\rightarrowtail$/ -)
  -fmaplets :: $['a,\ 'a] \Rightarrow fmaplet$ (- /[$\rightarrowtail$]/ -)

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

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

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

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

lemma *fmran'-fmadd* [*simp*]:
assumes *fmdom'* $m \cap$ *fmdom'* $m' = \{\}$
shows *fmran'* $(m \mathbin{++_f} m') =$ *fmran'* $m \cup$ *fmran'* m'
using *assms* **proof** (*intro subset-antisym subsetI*)
fix x
assume *fmdom'* $m \cap$ *fmdom'* $m' = \{\}$ **and** $x \in$ *fmran'* $(m \mathbin{++_f} m')$
then show $x \in$ *fmran'* $m \cup$ *fmran'* m'
by (*auto simp add*: *fmlookup-ran'-iff*) *meson*
next
fix x
assume *fmdom'* $m \cap$ *fmdom'* $m' = \{\}$ **and** $x \in$ *fmran'* $m \cup$ *fmran'* m'
then show $x \in$ *fmran'* $(m \mathbin{++_f} m')$
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' $$ $k = Some\ v$
by *force*
from *False* **have** $\{k \rightarrowtail v\} +\!+_f$ *fmdrop* k $(m'(k' \rightarrowtail v')) = \{k \rightarrowtail v\} +\!+_f$ (*fmdrop* k $m')(k' \rightarrowtail v')$
by (*simp add*: *fmdrop-fmupd*)
also have $\ldots = (\{k \rightarrowtail v\} +\!+_f$ *fmdrop* k $m')(k' \rightarrowtail v')$
by *fastforce*
also from *fmupd.prems* **and** *fmupd.IH*[*OF* ‹m' $$ $k = Some\ v$›] **have** $\ldots = m'(k' \rightarrowtail v')$
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* $(\alpha \to \beta) =$ *Suc* (*type-size* α + *type-size* β)

primrec *subtypes* :: *type* ⇒ *type set* **where**
subtypes $i = \{\}$
| *subtypes* $o = \{\}$
| *subtypes* $(\alpha \to \beta) = \{\alpha, \beta\} \cup$ *subtypes* $\alpha \cup$ *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** $(x, \alpha) \notin$ *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** $\bullet$ *200*)
| *FAbs var form*

syntax
  -FVar :: *nat* $\Rightarrow$ *type* $\Rightarrow$ *form* (-$_{-}$ [*899*, *0*] *900*)
  -FCon :: *nat* $\Rightarrow$ *type* $\Rightarrow$ *form* ($\{\!|$-$|\!\}_{-}$ [*899*, *0*] *900*)
  -FAbs :: *nat* $\Rightarrow$ *type* $\Rightarrow$ *form* $\Rightarrow$ *form* ((4λ-$_{-}$./ -) [*0*, *0*, *104*] *104*)
translations
  $x_\alpha \rightleftharpoons$ *CONST FVar* (x, α)
  $\{\!|c|\!\}_\alpha \rightleftharpoons$ *CONST FCon* (c, α)
  $\lambda x_\alpha.\ A \rightleftharpoons$ *CONST FAbs* $(x, \alpha)\ A$

2.5 Generalized operators

Generalized application. We define $\bullet^{\mathcal{Q}}{}_{\star}\ A\ [B_1, B_2, \ldots, B_n]$ as $A \bullet B_1 \bullet B_2 \bullet \cdots \bullet B_n$:

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

Generalized abstraction. We define $\lambda^{\mathcal{Q}}{}_{\star}\ [x_1, \ldots, x_n]\ A$ as $\lambda x_1. \cdots\ \lambda x_n.\ A$:

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

fun *form-size* :: *form* $\Rightarrow$ *nat* **where**
  form-size $(x_\alpha) = 1$
| *form-size* $(\{\!|c|\!\}_\alpha) = 1$
| *form-size* $(A \bullet B)$ = *Suc* (*form-size A* + *form-size B*)
| *form-size* $(\lambda x_\alpha.\ A)$ = *Suc* (*form-size A*)

fun *form-depth* :: *form* $\Rightarrow$ *nat* **where**
  form-depth $(x_\alpha) = 0$
| *form-depth* $(\{\!|c|\!\}_\alpha) = 0$
| *form-depth* $(A \bullet B)$ = *Suc* (*max* (*form-depth A*) (*form-depth B*))
| *form-depth* $(\lambda x_\alpha.\ A)$ = *Suc* (*form-depth A*)

2.6 Subformulas

fun *subforms* :: *form* $\Rightarrow$ *form set* **where**
  subforms $(x_\alpha) = \{\}$
| *subforms* $(\{\!|c|\!\}_\alpha) = \{\}$
| *subforms* $(A \bullet B) = \{A, B\}$
| *subforms* $(\lambda x_\alpha.\ A) = \{A\}$

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

fun *positions* :: *form* $\Rightarrow$ *position set* **where**
positions $(x_\alpha) = \{[]\}$
| *positions* $(\{\!\!\{c\}\!\!\}_\alpha) = \{[]\}$
| *positions* $(A \cdot B) = \{[]\} \cup \{« \# p \mid p.\ p \in$ *positions* $A\} \cup \{» \# p \mid p.\ p \in$ *positions* $B\}$
| *positions* $(\lambda x_\alpha.\ A) = \{[]\} \cup \{« \# p \mid p.\ p \in$ *positions* $A\}$

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

fun *subform-at* :: *form* $\Rightarrow$ *position* $\rightharpoonup$ *form* **where**
subform-at A $[]$ = *Some* A
| *subform-at* $(A \cdot B)$ $(« \# p)$ = *subform-at* A p
| *subform-at* $(A \cdot B)$ $(» \# p)$ = *subform-at* B p
| *subform-at* $(\lambda x_\alpha.\ A)$ $(« \# p)$ = *subform-at* A p
| *subform-at* - - = *None*

fun *is-subform-at* :: *form* $\Rightarrow$ *position* $\Rightarrow$ *form* $\Rightarrow$ *bool* ((- $\preceq$-/ -) [*51*,*0*,*51*] *50*) **where**
is-subform-at A $[]$ $A' = (A = A')$
| *is-subform-at* C $(« \# p)$ $(A \cdot B)$ = *is-subform-at* C p A
| *is-subform-at* C $(» \# p)$ $(A \cdot B)$ = *is-subform-at* C p B
| *is-subform-at* C $(« \# p)$ $(\lambda x_\alpha.\ A)$ = *is-subform-at* C p A
| *is-subform-at* - - - = *False*

lemma *is-subform-at-alt-def*:
shows $A' \preceq_p A$ = (*case subform-at A p of Some B* $\Rightarrow B = A' \mid$ *None* $\Rightarrow$ *False*)
by (*induction* A' *p A rule*: *is-subform-at.induct*) *auto*

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

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

lemma *subform-at-subforms-var*:
assumes $x_\alpha \preceq_p C$
shows $\nexists A.\ A \preceq_{p\ @\ [d]} C$
using *assms* **by** (*induction* x_α *p C rule*: *is-subform-at.induct*) *auto*

lemma *subform-at-subforms-app*:
assumes $A \cdot B \preceq_p C$
shows $A \preceq_{p\ @\ [«]} C$ **and** $B \preceq_{p\ @\ [»]} C$
using *assms* **by** (*induction* $A \cdot B$ *p C rule*: *is-subform-at.induct*) *auto*

lemma *subform-at-subforms-abs*:
  assumes $\lambda x_\alpha.\ A \preceq_p C$
  shows $A \preceq_{p\ @\ [«]} C$
  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 $\nexists B.\ B \preceq_{»\ \#\ p} \lambda x_\alpha.\ A$
  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 $A \preceq_p B \bullet C$
  shows
    $(A = B \bullet C \wedge p = []) \vee$
    $(A \neq B \bullet C \wedge$
      $(\exists p' \in \textit{positions}\ B.\ p = «\ \#\ p' \wedge A \preceq_{p'} B) \vee (\exists p' \in \textit{positions}\ C.\ p = »\ \#\ p' \wedge A \preceq_{p'} C))$
  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 $(A = \lambda x_\alpha.\ B \wedge p = []) \vee (A \neq \lambda x_\alpha.\ B \wedge (\exists p' \in \textit{positions}\ B.\ p = «\ \#\ p' \wedge A \preceq_{p'} B))$
  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 $B \preceq_{\mathit{replicate}\ (\mathit{length}\ As)\ «} \bullet^{\mathcal{Q}}{}_{\star}\ B\ As$
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 $p \in \mathit{positions}\ A$
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 $C \preceq_{\mathit{drop}\ (\mathit{length}\ p')\ p} B$
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 ‹$A' \preceq_{p'} A$› **and** *assms*(*2*) **have** $A' = B$
by (*rule is-subform-at-uniqueness*)
with ‹$C \preceq_{[d]} A'$› **have** $C \preceq_{[d]} B$
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 $p'' \neq []$
by *force*
moreover from *snoc.prems*(*2*) **obtain** A' **where** $C \preceq_{[d]} A'$ **and** $A' \preceq_{p''} A$
using *superform-existence* **by** *blast*
ultimately have $A' \preceq_{drop\ (length\ p')\ p''} B$
using *snoc.IH* **and** ‹*strict-prefix* p' p''› **by** *blast*
with ‹$C \preceq_{[d]} A'$› **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 $B \preceq_p A$
and *strict-prefix* p' p
shows $\neg\ B \preceq_{p'} A$
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 $B \preceq_{p'} A$
and $C \preceq_p A$
shows *form-size* C < *form-size* B
proof −
from *assms*(*1*) **have** $p \neq []$
by *force*
have $C \preceq_{drop\ (length\ p')\ p} B$
by (*fact subform-nesting*[*OF assms*])
moreover have *drop* (*length* p') $p \neq []$
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*]: $A \preceq B = (\exists p.\ A \preceq_p B)$

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

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

end

abbreviation *free-var-names* :: *'a* $\Rightarrow$ *nat set* **where**
  free-var-names $\mathcal{X}$ $\equiv$ *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* $(\bullet^{\mathcal{Q}}{}_{\star}\ A\ Bs)$ = *free-vars* A $\cup$ *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 $\subseteq$ *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* $(A \bullet B)$ = *free-vars* A $\cup$ *free-vars* B
    using *free-vars-form.simps*(*3*) **.**
  also from *FApp.IH* **have** ... $\subseteq$ *vars* A $\cup$ *vars* B
    by *blast*
  also have ... = *vars* $(A \bullet B)$
    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* $(B \bullet C) =$ *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 $(A \bullet B)$ (« # p) $=$ *binders-at* $A\ p$
| *binders-at* $(A \bullet B)$ (» # p) $=$ *binders-at* $B\ p$
| *binders-at* $(\lambda x_\alpha.\ A)$ (« # 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 (« # p) $(A \bullet B) \longleftrightarrow$ *occurs-at* $v\ p\ A$
and *occurs-at* v (» # p) $(A \bullet B) \longleftrightarrow$ *occurs-at* $v\ p\ B$
and *occurs-at* v (« # p) $(\lambda x_\alpha.\ A) \longleftrightarrow$ *occurs-at* $v\ p\ A$

and *occurs-at* v $(d \mathbin{\#} p)$ $(FVar\ v')$ $\longleftrightarrow$ *False*
and *occurs-at* v $(» \mathbin{\#} p)$ $(\lambda x_\alpha.\ A)$ $\longleftrightarrow$ *False*
and *occurs-at* v [] $(A \cdot B)$ $\longleftrightarrow$ *False*
and *occurs-at* v [] $(\lambda x_\alpha.\ A)$ $\longleftrightarrow$ *False*
by (*fastforce elim*: *is-subform-at.elims*)+

definition *occurs* :: *var* $\Rightarrow$ *form* $\Rightarrow$ *bool* **where**
[*iff*]: *occurs* v B $\longleftrightarrow$ $(\exists\, p \in$ *positions* $B.$ *occurs-at* $v\ p\ B)$

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

abbreviation *strict-prefixes* **where**
strict-prefixes $xs \equiv [ys \leftarrow$ *prefixes* $xs.\ ys \neq xs]$

definition *in-scope-of-abs* :: *var* $\Rightarrow$ *position* $\Rightarrow$ *form* $\Rightarrow$ *bool* **where**
[*iff*]: *in-scope-of-abs* $v\ p\ B \longleftrightarrow$ (
$p \neq [] \wedge$
(
$\exists\, p' \in$ *lset* (*strict-prefixes* p).
case (*subform-at* $B\ p'$) *of*
Some (*FAbs* v' -) $\Rightarrow v = v'$
| - $\Rightarrow$ *False*
)
)

lemma *in-scope-of-abs-alt-def*:
shows
in-scope-of-abs $v\ p\ B$
$\longleftrightarrow$
$p \neq [] \wedge (\exists\, p' \in$ *positions* $B.\ \exists\, C.$ *strict-prefix* $p'\ p \wedge$ *FAbs* $v\ C \preceq_{p'} B)$
proof
assume *in-scope-of-abs* $v\ p\ B$
then show $p \neq [] \wedge (\exists\, p' \in$ *positions* $B.\ \exists\, C.$ *strict-prefix* $p'\ p \wedge$ *FAbs* $v\ C \preceq_{p'} B)$
by (*induction rule*: *subform-at.induct*) *force*+
next
assume $p \neq [] \wedge (\exists\, p' \in$ *positions* $B.\ \exists\, C.$ *strict-prefix* $p'\ p \wedge$ *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 $(« \mathbin{\#} p)$ $(A \cdot B) \longleftrightarrow$ *in-scope-of-abs* $v\ p\ A$
by *force*

lemma *in-scope-of-abs-in-right-app*:
shows *in-scope-of-abs* v $(» \mathbin{\#} p)$ $(A \cdot B) \longleftrightarrow$ *in-scope-of-abs* $v\ p\ B$

by *force*

lemma *in-scope-of-abs-in-app*:
assumes *in-scope-of-abs* v p $(A \cdot B)$
obtains p' **where** $(p = « \# p' \wedge$ *in-scope-of-abs* v p' $A) \vee (p = » \# p' \wedge$ *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
$\forall p'$.
$(p = « \# p' \longrightarrow \neg$ *in-scope-of-abs* v' p' $A)$
$\wedge$
$(p = » \# p' \longrightarrow \neg$ *in-scope-of-abs* v' p' $B)$
shows $\neg$ *in-scope-of-abs* v' p $(A \cdot B)$
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) $\longleftrightarrow v = v' \vee$ *in-scope-of-abs* v p B
proof
assume *in-scope-of-abs* v $(« \# p)$ (*FAbs* v' B)
then obtain p' **and** C
where $p' \in$ *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 $v = v' \vee$ *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 ¬ *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*) ⟷ *is-bound-at v p B*
  by *auto*

lemma *is-bound-at-in-right-app*:
  shows *is-bound-at v* (» # *p*) (*B* • *C*) ⟷ *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' ∧ *is-bound-at v* p' *B*) ∨ (p = » # p' ∧ *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 $v = v'$ ∨ *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** $v = v'$ ∨ *is-bound-at v* p' *B*
proof −
  obtain x **and** α **where** $v' = (x, \alpha)$
    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 ($v = v'$ ∧ *occurs-at v p B*) ∨ *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 $p \in$ *positions* A
and *is-bound-at* v p $A \lor v \in$ *binders-at* A p
shows $v \in$ *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 ($B \cdot C$) | (b) $v \in$ *binders-at* ($B \cdot C$) p
by *blast*
then show *?case*
proof *cases*
case a
then have $p \neq []$
using *occurs-at-alt-def*(*8*) **by** *blast*
then obtain d **and** p' **where** $p = d \# p'$
by (*meson list.exhaust*)
with ‹$p \in$ *positions* ($B \cdot C$)›
consider (a_1) $p =$ « # p' **and** $p' \in$ *positions* B | (a_2) $p =$ » # p' **and** $p' \in$ *positions* C
by *force*
then show *?thesis*
proof *cases*
case a_1
from a_1(*1*) **and** ‹*is-bound-at* v p ($B \cdot C$)› **have** *is-bound-at* v p' B
using *is-bound-at-in-left-app* **by** *blast*
with a_1(*2*) **have** $v \in$ *bound-vars* B
using *FApp.IH*(*1*) **by** *blast*
then show *?thesis*
by *simp*
next
case a_2
from a_2(*1*) **and** ‹*is-bound-at* v p ($B \cdot C$)› **have** *is-bound-at* v p' C
using *is-bound-at-in-right-app* **by** *blast*
with a_2(*2*) **have** $v \in$ *bound-vars* C
using *FApp.IH*(*2*) **by** *blast*
then show *?thesis*
by *simp*
qed
next
case b
then have $p \neq []$
by *force*
then obtain d **and** p' **where** $p = d \# p'$

by (*meson list.exhaust*)
with ‹$p \in$ *positions* $(B \cdot C)$›
consider (b_1) $p = « \# p'$ **and** $p' \in$ *positions* B | (b_2) $p = » \# p'$ **and** $p' \in$ *positions* C
by *force*
then show *?thesis*
proof *cases*
case b_1
from $b_1(1)$ **and** ‹$v \in$ *binders-at* $(B \cdot C)$ p› **have** $v \in$ *binders-at* B p'
by *force*
with $b_1(2)$ **have** $v \in$ *bound-vars* B
using *FApp.IH*(*1*) **by** *blast*
then show *?thesis*
by *simp*
next
case b_2
from $b_2(1)$ **and** ‹$v \in$ *binders-at* $(B \cdot C)$ p› **have** $v \in$ *binders-at* C p'
by *force*
with $b_2(2)$ **have** $v \in$ *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) $v \in$ *binders-at* (*FAbs* v' B) p
by *blast*
then show *?case*
proof *cases*
case a
then have $p \neq []$
using *occurs-at-alt-def*(*9*) **by** *force*
with ‹$p \in$ *positions* (*FAbs* v' B)› **obtain** p' **where** $p = « \# p'$ **and** $p' \in$ *positions* B
by (*cases FAbs* v' B *rule*: *positions.cases*) *fastforce+*
from ‹$p = « \# p'$› **and** ‹*is-bound-at* v p (*FAbs* v' B)› **have** $v = v' \vee$ *is-bound-at* v p' B
using *is-bound-at-from-abs* **by** *blast*
then consider (a_1) $v = v'$ | (a_2) *is-bound-at* v p' B
by *blast*
then show *?thesis*
proof *cases*
case a_1
then show *?thesis*
using *surj-pair*[*of* v'] **by** *fastforce*
next
case a_2
then have $v \in$ *bound-vars* B
using ‹$p' \in$ *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 (b_1) v = v' | (b_2) v ∈ binders-at B p'
      by (cases FAbs v' B rule: positions.cases) fastforce+
    then show ?thesis
    proof cases
      case b_1
      then show ?thesis
        using surj-pair[of v'] by fastforce
    next
      case b_2
      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 (a_1) is-bound-at v p B | (a_2) v ∈ binders-at B p
      by blast
    then show ?thesis
    proof cases
      case a_1
      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 a_2
then have $v \in$ *binders-at* $(B \cdot C)$ (« # p)
by *simp*
then show *?thesis*
using *3.prems*(*1*) **and** ‹« # $p \in$ *positions* $(B \cdot C)$› **by** *blast*
qed
next
case b
with *3.IH*(*2*) **obtain** p **where** $p \in$ *positions* C **and** *is-bound-at* v p $C \lor v \in$ *binders-at* C p
by *blast*
from ‹$p \in$ *positions* C› **have** » # $p \in$ *positions* $(B \cdot C)$
by *simp*
from ‹*is-bound-at* v p $C \lor v \in$ *binders-at* C p›
consider (b_1) *is-bound-at* v p C | (b_2) $v \in$ *binders-at* C p
by *blast*
then show *?thesis*
proof *cases*
case b_1
then have *is-bound-at* v (» # p) $(B \cdot C)$
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 b_2
then have $v \in$ *binders-at* $(B \cdot C)$ (» # p)
by *simp*
then show *?thesis*
using *3.prems*(*1*) **and** ‹» # $p \in$ *positions* $(B \cdot C)$› **by** *blast*
qed
qed
next
case (*4* x α B)
from ‹$v \in$ *bound-vars* $(\lambda x_\alpha.\ B)$› **consider** (a) $v = (x, \alpha)$ | (b) $v \in$ *bound-vars* B
by *force*
then show *?case*
proof *cases*
case a
then have $v \in$ *binders-at* $(\lambda x_\alpha.\ B)$ [«]
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** $p \in$ *positions* B **and** *is-bound-at* v p $B \lor v \in$ *binders-at* B p
by *blast*
from ‹$p \in$ *positions* B› **have** « # $p \in$ *positions* $(\lambda x_\alpha.\ B)$
by *simp*
from ‹*is-bound-at* v p $B \lor v \in$ *binders-at* B p›
consider (b_1) *is-bound-at* v p B | (b_2) $v \in$ *binders-at* B p

by *blast*
then show *?thesis*
proof *cases*
case b_1
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** ‹« # $p \in$ *positions* (λx_α. B)› **by** *blast*
next
case b_2
then have $v \in$ *binders-at* (λx_α. B) (« # p)
by *simp*
then show *?thesis*
using *4.prems(1)* **and** ‹« # $p \in$ *positions* (λx_α. B)› **by** *blast*
qed
qed
qed *simp-all*

lemma *bound-vars-alt-def*:
shows *bound-vars* $A = \{v \mid v\ p.\ p \in$ *positions* $A \wedge$ (*is-bound-at* v p $A \vee v \in$ *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* $\Rightarrow$ *position* $\Rightarrow$ *form* $\Rightarrow$ *bool* **where**
[*iff*]: *is-free-at* v p $B \longleftrightarrow$ *occurs-at* v p $B \wedge \neg$ *in-scope-of-abs* v p B

lemma *is-free-at-in-var*:
shows *is-free-at* v [] (*FVar* v') $\longleftrightarrow v = v'$
by *simp*

lemma *not-is-free-at-in-con*:
shows $\neg$ *is-free-at* v [] ($\{\!|c|\!\}_\alpha$)
by *simp*

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

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

lemma *is-free-at-from-app*:
assumes *is-free-at* v p ($B \bullet C$)
obtains p' **where** ($p =$ « # $p' \wedge$ *is-free-at* v p' B) $\vee$ ($p =$ » # $p' \wedge$ *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** $v' = (x, \alpha)$
    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** $v \neq v'$
  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 $p \in$ *positions A* **and** *is-free-at v p A*
  shows $v \in$ *free-vars A*
using *assms* **proof** (*induction A arbitrary*: *p*)
  case (*FApp B C*)
  from ‹*is-free-at v p* (*B* • *C*)› **have** $p \neq []$
    using *occurs-at-alt-def*(*8*) **by** *blast*
  then obtain d **and** p' **where** $p = d \# p'$
    by (*meson list.exhaust*)
  with ‹$p \in$ *positions* (*B* • *C*)›
  consider (*a*) $p =$ « # p' **and** $p' \in$ *positions B* | (*b*) $p =$ » # p' **and** $p' \in$ *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** $p \in$ *positions B* **and** *is-free-at v p B*
by *blast*
from ‹$p \in$ *positions B*› **have** » # $p \in$ *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 ‹$v \in$ *free-vars* (λx_α. *A*)› **have** $v \in$ *free-vars A* $- \{(x, \alpha)\}$ **and** $v \neq (x, \alpha)$
by *simp-all*
then have $v \in$ *free-vars A*
by *blast*
with *4.IH* **obtain** *p* **where** $p \in$ *positions A* **and** *is-free-at v p A*
by *blast*
from ‹$p \in$ *positions A*› **have** « # $p \in$ *positions* (λx_α. *A*)
by *simp*
moreover from ‹*is-free-at v p A*› **and** ‹$v \neq (x, \alpha)$› **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* = $\{v \mid v\ p.\ p \in$ *positions A* $\wedge$ *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* $\Rightarrow$ *form* $\Rightarrow$ *bool* **where**
[*iff*]: *is-bound v B* $\longleftrightarrow$ ($\exists\, p \in$ *positions B*. *is-bound-at v p B* $\vee$ $v \in$ *binders-at B p*)

lemma *is-bound-in-app-homomorphism*:
shows *is-bound v* (*A* • *B*) $\longleftrightarrow$ *is-bound v A* $\vee$ *is-bound v B*
proof
assume *is-bound v* (*A* • *B*)
then obtain *p* **where** $p \in$ *positions* (*A* • *B*) **and** *is-bound-at v p* (*A* • *B*) $\vee$ $v \in$ *binders-at* (*A* • *B*) *p*
by *auto*
then have $p \neq []$
by *fastforce*

```
with ‹p ∈ positions (A • B)› obtain p' and d where p = d # p'
  by auto
from ‹is-bound-at v p (A • B) ∨ v ∈ binders-at (A • B) p›
consider (a) is-bound-at v p (A • B) | (b) v ∈ binders-at (A • B) p
  by blast
then show is-bound v A ∨ is-bound v B
proof cases
  case a
  then show ?thesis
  proof (cases d)
    case Left
    then have p' ∈ positions A
      using ‹p = d # p'› and ‹p ∈ positions (A • B)› by fastforce
    moreover from ‹is-bound-at v p (A • 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 • 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' ∈ positions B
      using ‹p = d # p'› and ‹p ∈ positions (A • B)› by fastforce
    moreover from ‹is-bound-at v p (A • 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 • 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' ∈ positions A
      using ‹p = d # p'› and ‹p ∈ positions (A • B)› by fastforce
    moreover from ‹v ∈ binders-at (A • B) p› have v ∈ binders-at A p'
      using Left and ‹p = d # p'› by force
    ultimately show ?thesis
      by auto
  next
    case Right
    then have p' ∈ positions B
      using ‹p = d # p'› and ‹p ∈ positions (A • B)› by fastforce
    moreover from ‹v ∈ binders-at (A • B) p› have v ∈ 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_α)
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 $(B \bullet C) \longleftrightarrow$ *is-free-for* A v $B \land$ *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
$\longleftrightarrow$
(
$\nexists p.$
(
$p \in$ *positions* $B \wedge$ *is-free-at* v p $B \wedge p \neq [] \wedge$
$(\exists v' \in$ *free-vars* $A.\ \exists p'\ C.$ *strict-prefix* p' $p \wedge$ *FAbs* v' $C \preceq_{p'} B)$
)
)
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** $x \neq w$
shows $\neg$ *is-free-for* (*FVar* w) x (*FAbs* w B)
proof (*rule ccontr*)
assume $\neg\ \neg$ *is-free-for* (*FVar* w) x (*FAbs* w B)
then have
$\forall v' \in$ *free-vars* (*FVar* w). $\forall p \in$ *positions* (*FAbs* w B). *is-free-at* x p (*FAbs* w B)
$\longrightarrow \neg$ *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
$\forall p \in$ *positions* (*FAbs* w B). *is-free-at* x p (*FAbs* w B) $\longrightarrow \neg$ *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** « # $p \in$ *positions* (*FAbs* w B)
using *is-subform-implies-in-positions* **by** *force*
ultimately have $\neg$ *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-vars* A **and** $v \in$ *free-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* ⇒ *position* ⇒ *form* ⇒ *form* ⇒ *bool*
((*4*-⦇- ← -⦈ ⊳ -) [*1000*, *0*, *0*, *0*] *900*)
where
pos-found: $A⦇p \leftarrow C⦈ \rhd C'$ **if** $p = []$ **and** $C = C'$
| *replace-left-app*: $(G \cdot H)⦇« \# p \leftarrow C⦈ \rhd (G' \cdot H)$ **if** $p \in$ *positions* G **and** $G⦇p \leftarrow C⦈ \rhd G'$
| *replace-right-app*: $(G \cdot H)⦇» \# p \leftarrow C⦈ \rhd (G \cdot H')$ **if** $p \in$ *positions* H **and** $H⦇p \leftarrow C⦈ \rhd H'$
| *replace-abs*: $(\lambda x_\gamma.\ E)⦇« \# p \leftarrow C⦈ \rhd (\lambda x_\gamma.\ E')$ **if** $p \in$ *positions* E **and** $E⦇p \leftarrow C⦈ \rhd E'$

lemma *is-replacement-at-implies-in-positions*:
assumes $C⦇p \leftarrow A⦈ \rhd D$
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** $C⦇p \leftarrow A⦈ \rhd D$
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 $\forall\, p' \in positions\ D.\ \neg\ prefix\ p'\ p \wedge \neg\ prefix\ p\ p' \longrightarrow subform\text{-}at\ D\ p' = subform\text{-}at\ C\ p'$
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 $binders\text{-}at\ D\ p = binders\text{-}at\ C\ p$
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' $C \longleftrightarrow$ *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 $v \notin$ *vars* C
and $C\langle\!\langle p \leftarrow FVar\ v \rangle\!\rangle \rhd G$
and *occurs-at* v p' G
shows $p' = p$
proof (*rule ccontr*)
assume $p' \neq p$
from *assms*(*2*) **have** *occurs-at* v p G
by (*simp add*: *is-replacement-at-minimal-change*(*1*))
moreover have $*$: *occurs-at* v p' $C \longleftrightarrow$ *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 $v \in$ *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 $FVar\ v \preceq_p G$
by (*fact is-replacement-at-minimal-change*(*1*)[*OF assms*(*2*)])
moreover from *assms*(*3*) **have** $FVar\ v \preceq_{p'} G$
by *simp*
ultimately show *?thesis*
using ‹$p' \neq p$› **and** *False* **and** *loop-subform-impossibility*
by (*blast dest*: *prefix-order*.*antisym-conv2*)
qed
qed

lemma *is-replacement-at-new-positions*:
assumes $C\langle\!|p \leftarrow A|\!\rangle \rhd D$ **and** *prefix* $p\ p'$ **and** $p' \in$ *positions* D
obtains p'' **where** $p' = p\ @\ p''$ **and** $p'' \in$ *positions* A
using *assms* **by** (*induction arbitrary*: *thesis* p' *rule*: *is-replacement-at.induct*, *auto*) *blast*+

lemma *replacement-override*:
assumes $C\langle\!|p \leftarrow B|\!\rangle \rhd D$ **and** $C\langle\!|p \leftarrow A|\!\rangle \rhd F$
shows $D\langle\!|p \leftarrow A|\!\rangle \rhd F$
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 $p \in$ *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** $F = F' \cdot H$ **and** $G\langle\!|p \leftarrow A|\!\rangle \rhd F'$
by (*fastforce elim*: *is-replacement-at.cases*)
from ‹$G\langle\!|p \leftarrow A|\!\rangle \rhd F'$› **have** $G'\langle\!|p \leftarrow A|\!\rangle \rhd F'$
by (*fact replace-left-app.IH*)
with ‹$p \in$ *positions* G'› **show** *?case*
unfolding ‹$F = F' \cdot H$› **by** *blast*
next
case (*replace-right-app p H C* H' *G*)
have $p \in$ *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** $F = G \bullet F'$ **and** $H\langle\!|p \leftarrow A|\!\rangle \rhd F'$
by (*fastforce elim*: *is-replacement-at.cases*)
from ‹$H\langle\!|p \leftarrow A|\!\rangle \rhd F'$› **have** $H'\langle\!|p \leftarrow A|\!\rangle \rhd F'$
by (*fact replace-right-app.IH*)
with ‹$p \in$ *positions* H'› **show** *?case*
unfolding ‹$F = G \bullet F'$› **by** *blast*
next
case (*replace-abs* p E C E' x γ)
have $p \in$ *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** $F = \lambda x_\gamma.\ F'$ **and** $E\langle\!|p \leftarrow A|\!\rangle \rhd F'$
by (*fastforce elim*: *is-replacement-at.cases*)
from ‹$E\langle\!|p \leftarrow A|\!\rangle \rhd F'$› **have** $E'\langle\!|p \leftarrow A|\!\rangle \rhd F'$
by (*fact replace-abs.IH*)
with ‹$p \in$ *positions* E'› **show** *?case*
unfolding ‹$F = \lambda x_\gamma.\ F'$› **by** *blast*
qed

lemma *leftmost-subform-in-generalized-app-replacement*:
shows $(\bullet^{\mathcal{Q}}_{\star}\ C\ As)\langle\!|$*replicate* (*length As*) « $\leftarrow D|\!\rangle \rhd (\bullet^{\mathcal{Q}}_{\star}\ D\ As)$
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*) $\mathfrak{x}$ **where** $\mathfrak{x} \equiv 0$
abbreviation (*input*) $\mathfrak{y}$ **where** $\mathfrak{y} \equiv Suc\ \mathfrak{x}$
abbreviation (*input*) $\mathfrak{z}$ **where** $\mathfrak{z} \equiv Suc\ \mathfrak{y}$
abbreviation (*input*) $\mathfrak{f}$ **where** $\mathfrak{f} \equiv Suc\ \mathfrak{z}$
abbreviation (*input*) $\mathfrak{g}$ **where** $\mathfrak{g} \equiv Suc\ \mathfrak{f}$
abbreviation (*input*) $\mathfrak{h}$ **where** $\mathfrak{h} \equiv Suc\ \mathfrak{g}$
abbreviation (*input*) $\mathfrak{c}$ **where** $\mathfrak{c} \equiv Suc\ \mathfrak{h}$
abbreviation (*input*) $\mathfrak{c}_Q$ **where** $\mathfrak{c}_Q \equiv Suc\ \mathfrak{c}$
abbreviation (*input*) $\mathfrak{c}_\iota$ **where** $\mathfrak{c}_\iota \equiv Suc\ \mathfrak{c}_Q$

definition *Q-constant-of-type* :: *type* $\Rightarrow$ *con* **where**
[*simp*]: *Q-constant-of-type* $\alpha = (\mathfrak{c}_Q,\ \alpha{\to}\alpha{\to}o)$

definition *iota-constant* :: *con* **where**
[*simp*]: *iota-constant* $\equiv (\mathfrak{c}_\iota,\ (i{\to}o){\to}i)$

definition Q :: *type* $\Rightarrow$ *form* ($Q_{\text{-}}$) **where**
[*simp*]: Q_α = *FCon* (*Q-constant-of-type* α)

definition *iota* :: *form* (ι) **where**
[*simp*]: ι = *FCon iota-constant*

definition *is-Q-constant-of-type* :: *con* ⇒ *type* ⇒ *bool* **where**
[*iff*]: *is-Q-constant-of-type p* α ⟷ *p* = *Q-constant-of-type* α

definition *is-iota-constant* :: *con* ⇒ *bool* **where**
[*iff*]: *is-iota-constant p* ⟷ *p* = *iota-constant*

definition *is-logical-constant* :: *con* ⇒ *bool* **where**
[*iff*]: *is-logical-constant p* ⟷ (∃ β. *is-Q-constant-of-type p* β) ∨ *is-iota-constant p*

definition *type-of-Q-constant* :: *con* ⇒ *type* **where**
[*simp*]: *type-of-Q-constant p* = (*THE* α. *is-Q-constant-of-type p* α)

lemma *constant-cases*[*case-names non-logical Q-constant* ι*-constant*, *cases type*: *con*]:
assumes ¬ *is-logical-constant p* ⟹ *P*
and ⋀β. *is-Q-constant-of-type p* β ⟹ *P*
and *is-iota-constant p* ⟹ *P*
shows *P*
using *assms* **by** *blast*

2.12 Definitions and abbreviations

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

definition *equivalence* :: *form* ⇒ *form* ⇒ *form* (**infixl** $\equiv^\mathcal{Q}$ *102*) **where**
[*simp*]: $A \equiv^\mathcal{Q} B = A =_o B$ — 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* ⇒ *form* ($\prod$ -) **where**
[*simp*]: $\prod\alpha = Q_{\alpha\to o} \cdot (\lambda\mathfrak{x}_\alpha.\ T_o)$

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

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

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

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

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

lemma *innermost-replacement-in-generalized-forall*:
assumes *vs* $\neq$ []
shows $(\forall^{\mathcal{Q}}{}_{\star}\ vs\ C)\langle\!|$*foldr* ($\lambda$-. (@) [»,«]) *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* (λ-. (@) [»,«]) *vs* [] $\in$ *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* ($\lambda$-. (@) [»,«]) *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* ($\lambda$-. (@) [»,«]) *vs* [] $\leftarrow B|\!\rangle \rhd (\lambda x_{\alpha}.\ \forall^{\mathcal{Q}}{}_{\star}\ vs\ B)$
by (*rule replace-abs*)
moreover have « # *foldr* (λ-. (@) [»,«]) *vs* [] $\in$ *positions* $(\lambda x_{\alpha}.\ \forall^{\mathcal{Q}}{}_{\star}\ vs\ C)$
using ‹*foldr* (λ-. (@) [»,«]) *vs* [] $\in$ *positions* $(\forall^{\mathcal{Q}}{}_{\star}\ vs\ C)$› **by** *simp*
ultimately have
$(\prod_{\alpha} \bullet (\lambda x_{\alpha}.\ \forall^{\mathcal{Q}}{}_{\star}\ vs\ C))\langle\!|$» # « # *foldr* ($\lambda$-. (@) [»,«]) *vs* [] $\leftarrow B|\!\rangle \rhd (\prod_{\alpha} \bullet (\lambda x_{\alpha}.\ \forall^{\mathcal{Q}}{}_{\star}\ vs\ B))$
by *blast*
then have $(\forall x_{\alpha}.\ \forall^{\mathcal{Q}}{}_{\star}\ vs\ C)\langle\!|$[»,«] @ *foldr* ($\lambda$-. (@) [»,«]) *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*]: $\wedge_{o \to o \to o} =$
$\lambda \mathfrak{x}_o.\ \lambda \mathfrak{y}_o.$
(
$(\lambda \mathfrak{g}_{o \to o \to o}.\ \mathfrak{g}_{o \to o \to o} \bullet T_o \bullet T_o) =_{(o \to o \to o) \to o} (\lambda \mathfrak{g}_{o \to o \to o}.\ \mathfrak{g}_{o \to o \to o} \bullet \mathfrak{x}_o \bullet \mathfrak{y}_o)$
)

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-vars* $A - \{(x, \alpha)\}$
by *simp*

definition *inequality-of-type* :: *form* ⇒ *type* ⇒ *form* ⇒ *form* ((- ≠-/ -) [*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-wff-of-type* α *f*}

abbreviation *wffs* :: *form set* **where**
wffs $\equiv \bigcup\alpha.\ \mathit{wffs}_{\alpha}$

lemma *is-wff-of-type-wffs-of-type-eq* [*pred-set-conv*]:
shows *is-wff-of-type* $\alpha = (\lambda f.\ f \in \mathit{wffs}_{\alpha})$
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* ! $k \in \mathit{wffs}_{ts\ !\ k}$
and $B \in \mathit{wffs}_{foldr\ (\rightarrow)\ ts\ \beta}$
shows $\bullet^{\mathcal{Q}}_{\star}$ *B As* $\in \mathit{wffs}_{\beta}$
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** $A \in \mathit{wffs}_{t}$
by *fastforce*
moreover from *Cons.prems*(*2*) **have** $B \in \mathit{wffs}_{t \rightarrow foldr\ (\rightarrow)\ ts\ \beta}$
by *auto*
ultimately have $B \bullet A \in \mathit{wffs}_{foldr\ (\rightarrow)\ ts\ \beta}$
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* ! $k \in \mathit{wffs}_{ts\ !\ k}$
by *fastforce*
ultimately have $\bullet^{\mathcal{Q}}_{\star}$ $(B \bullet A)$ *As* $\in \mathit{wffs}_{\beta}$
using *Cons.IH* **by** (*simp only*:)
moreover have $\bullet^{\mathcal{Q}}_{\star}$ *B* (*A* # *As*) = $\bullet^{\mathcal{Q}}_{\star}$ $(B \bullet A)$ *As*
by *simp*
ultimately show *?case*
by (*simp only*:)
qed

lemma *generalized-abs-wff* [*intro*]:
assumes $B \in \mathit{wffs}_{\beta}$
shows $\lambda^{\mathcal{Q}}_{\star}$ *vs* $B \in \mathit{wffs}_{foldr\ (\rightarrow)\ (map\ snd\ vs)\ \beta}$
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_α
    by auto
  from Cons.prems have λ^Q_⋆ vs B ∈ wffs_?δ
    by (fact Cons.IH)
  with ‹v = (x, α)› have FAbs v (λ^Q_⋆ vs B) ∈ wffs_α→?δ
    by blast
  moreover from ‹v = (x, α)› have foldr (→) (map snd (v # vs)) β = α→?δ
    by simp
  moreover have λ^Q_⋆ (v # vs) B = FAbs v (λ^Q_⋆ vs B)
    by simp
  ultimately show ?case by (simp only:)
qed

lemma Q-wff [intro]:
  shows Q_α ∈ wffs_α→α→o
  by auto

lemma iota-wff [intro]:
  shows ι ∈ wffs_(i→o)→i
  by auto

lemma equality-wff [intro]:
  assumes A ∈ wffs_α and B ∈ wffs_α
  shows A =_α B ∈ wffs_o
  using assms by auto

lemma equivalence-wff [intro]:
  assumes A ∈ wffs_o and B ∈ wffs_o
  shows A ≡^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_(α→o)→o
  using PI-def by fastforce
```

lemma *forall-wff* [*intro*]:
  assumes $A \in wffs_o$
  shows $\forall x_\alpha.\ A \in wffs_o$
  using *assms* **and** *pi-wff* **unfolding** *forall-def* **by** *blast*

lemma *generalized-forall-wff* [*intro*]:
  assumes $B \in wffs_o$
  shows $\forall^{\mathcal{Q}}{}_\star\ vs\ B \in wffs_o$
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 $\wedge_{o \to o \to o} \in wffs_{o \to o \to o}$
  by *auto*

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

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

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

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

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

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

lemma *exists-wff* [*intro*]:
  assumes $A \in wffs_o$
  shows $\exists x_\alpha.\ A \in wffs_o$

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 $\wedge_{o\to o\to o} \bullet A \bullet B \in wffs_o$
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 $\supset_{o\to o\to o} \bullet A \bullet B \in wffs_o$
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 $\vee_{o\to o\to o} \bullet A \bullet B \in wffs_o$
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 $A \vee^{\mathcal{Q}} B \in \mathit{wffs}_o$
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 $\bigwedge A\ B\ \alpha.\ A \in wffs_{\alpha\to o} \Longrightarrow B \in wffs_\alpha \Longrightarrow \mathcal{P}\ (A \bullet B)$
  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* ($\bullet^{\mathcal{Q}}{}_\star$ *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* ($\bullet^{\mathcal{Q}}{}_\star$ *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* $(\forall^{\mathcal{Q}}{}_\star\ vs\ B)$
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* $(\forall^{\mathcal{Q}}{}_\star\ vs\ B)$
    by (*fact Cons.IH*)
  from *this* **and** ‹$v' \notin$ *free-vars A*›[*unfolded* ‹$v' = (x, \alpha)$›] **have** *is-free-for A v* $(\forall x_\alpha.\ \forall^{\mathcal{Q}}{}_\star\ vs\ B)$
    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* $(B \wedge^{\mathcal{Q}} C)$
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* $(\wedge_{o\to o\to o} \bullet B \bullet C)$
    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* $A\ \alpha \longleftrightarrow A \in wffs_\alpha \wedge$ *free-vars* $A = \{\}$

definition *is-sentence* :: *form* ⇒ *bool* **where**
[*iff*]: *is-sentence* $A \longleftrightarrow$ *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* $\vartheta \longleftrightarrow (\forall\, (x, \alpha) \in$ *fmdom'* $\vartheta.\ \vartheta$ \$\$! $(x, \alpha) \in wffs_\alpha)$

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 $(A \bullet B)$ = (**S** ϑ A) • (**S** ϑ B)
| **S** ϑ $(\lambda x_\alpha.\ A)$ = (*if* $(x, \alpha) \notin$ *fmdom'* ϑ *then* $\lambda x_\alpha.$ **S** ϑ A *else* $\lambda x_\alpha.$ **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 $A \in wffs_\alpha$
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* $(x, \alpha) \in$ *fmdom'* ϑ) (*use fmdom'-notI* **in** ‹*force+*›)
next
case (*abs-is-wff* β A α x)
then show *?case*
proof (*cases* $(x, \alpha) \in$ *fmdom'* ϑ)
case *True*
then have **S** ϑ $(\lambda x_\alpha.\ A) = \lambda x_\alpha.$ **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) $A \in wffs_\beta$
by *simp*
ultimately show *?thesis*
by *auto*
next

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

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

lemma *generalized-app-substitution*:
shows $\mathbf{S}\ \vartheta\ (\bullet^{\mathcal{Q}}{}_\star\ A\ Bs) = \bullet^{\mathcal{Q}}{}_\star\ (\mathbf{S}\ \vartheta\ A)\ (\mathit{map}\ (\lambda B.\ \mathbf{S}\ \vartheta\ B)\ Bs)$
by (*induction Bs arbitrary*: *A*) *simp-all*

lemma *generalized-abs-substitution*:
shows $\mathbf{S}\ \vartheta\ (\lambda^{\mathcal{Q}}{}_\star\ vs\ A) = \lambda^{\mathcal{Q}}{}_\star\ vs\ (\mathbf{S}\ (\mathit{fmdrop\text{-}set}\ (\mathit{fmdom'}\ \vartheta \cap \mathit{lset}\ vs)\ \vartheta)\ A)$
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* $v \notin \mathit{fmdom'}\ \vartheta$)
case *True*
then have $*$: $\mathit{fmdom'}\ \vartheta \cap \mathit{lset}\ (v\ \#\ vs) = \mathit{fmdom'}\ \vartheta \cap \mathit{lset}\ vs$
by *simp*
from *True* **have** $\mathbf{S}\ \vartheta\ (\lambda^{\mathcal{Q}}{}_\star\ (v\ \#\ vs)\ A) = \lambda x_\alpha.\ \mathbf{S}\ \vartheta\ (\lambda^{\mathcal{Q}}{}_\star\ vs\ A)$
using $\langle v = (x, \alpha)\rangle$ **by** *auto*
also have $\ldots = \lambda x_\alpha.\ \lambda^{\mathcal{Q}}{}_\star\ vs\ (\mathbf{S}\ (\mathit{fmdrop\text{-}set}\ (\mathit{fmdom'}\ \vartheta \cap \mathit{lset}\ vs)\ \vartheta)\ A)$
using *Cons.IH* **by** (*simp only*:)
also have $\ldots = \lambda^{\mathcal{Q}}{}_\star\ (v\ \#\ vs)\ (\mathbf{S}\ (\mathit{fmdrop\text{-}set}\ (\mathit{fmdom'}\ \vartheta \cap \mathit{lset}\ (v\ \#\ vs))\ \vartheta)\ A)$

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

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

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

lemma singleton-substitution-simps:
  shows S {(x, α) ↣ A} (y_β) = (if (x, α) ≠ (y, β) then y_β 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} (λy_β. B) = λy_β. (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 $\mathbf{S}$ $\{v \rightarrowtail \mathit{FVar}\ v\}\ A = A$
by
($\mathit{induction}\ A\ \mathit{rule}$: *free-vars-form.induct*)
(*simp-all add*: *empty-substitution-neutrality fmdrop-fmupd-same*)

lemma *free-var-in-renaming-substitution*:
assumes $x \neq y$
shows $(x, \alpha) \notin$ *free-vars* ($\mathbf{S}$ $\{(x, \alpha) \rightarrowtail y_\alpha\}\ B$)
using *assms* **by** (*induction B rule*: *free-vars-form.induct*) *simp-all*

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

The following lemma corresponds to X5100 in [2]:

lemma *substitution-composability*:
assumes $v' \notin$ *vars* B
shows $\mathbf{S}$ $\{v' \rightarrowtail A\}$ $\mathbf{S}$ $\{v \rightarrowtail \mathit{FVar}\ v'\}\ B = \mathbf{S}$ $\{v \rightarrowtail A\}\ B$
using *assms* **proof** (*induction B arbitrary*: v')
case (*FAbs w C*)
then show *?case*
proof (*cases* $v = w$)
case *True*
from ‹$v' \notin$ *vars* (*FAbs w C*)› **have** $v' \notin$ *free-vars* (*FAbs w C*)
using *free-vars-in-all-vars* **by** *blast*
then have $\mathbf{S}$ $\{v' \rightarrowtail A\}$ (*FAbs w C*) = *FAbs w C*
by (*rule free-var-singleton-substitution-neutrality*)
from ‹$v = w$› **have** $v \notin$ *free-vars* (*FAbs w C*)
using *surj-pair*[*of w*] **by** *fastforce*
then have $\mathbf{S}$ $\{v \rightarrowtail A\}$ (*FAbs w C*) = *FAbs w C*
by (*fact free-var-singleton-substitution-neutrality*)
also from ‹$\mathbf{S}$ $\{v' \rightarrowtail A\}$ (*FAbs w C*) = *FAbs w C*› **have** ... = $\mathbf{S}$ $\{v' \rightarrowtail A\}$ (*FAbs w C*)
by (*simp only*:)
also from ‹$v = w$› **have** ... = $\mathbf{S}$ $\{v' \rightarrowtail A\}$ $\mathbf{S}$ $\{v \rightarrowtail \mathit{FVar}\ v'\}$ (*FAbs w C*)
using *free-var-singleton-substitution-neutrality*[*OF* ‹$v \notin$ *free-vars* (*FAbs w C*)›] **by** (*simp only*:)
finally show *?thesis* **..**
next
case *False*
from *FAbs.prems* **have** $v' \notin$ *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 ‹$v \neq w$› **have** **S** $\{v \rightarrowtail A\}$ (*FAbs* w C) = *FAbs* w (**S** $\{v \rightarrowtail A\}$ C)
using *surj-pair*[*of* w] **by** *fastforce*
also from *FAbs.IH* **have** ... = *FAbs* w (**S** $\{v' \rightarrowtail A\}$ **S** $\{v \rightarrowtail$ *FVar* $v'\}$ C)
using ‹$v' \notin$ *vars* C› **by** *simp*
also from ‹$v' \neq w$› **have** ... = **S** $\{v' \rightarrowtail A\}$ (*FAbs* w (**S** $\{v \rightarrowtail$ *FVar* $v'\}$ C))
using *surj-pair*[*of* w] **by** *fastforce*
also from ‹$v \neq w$› **have** ... = **S** $\{v' \rightarrowtail A\}$ **S** $\{v \rightarrowtail$ *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 $z \notin$ *free-vars* A **and** *is-free-for* (*FVar* z) x A
shows **S** $\{z \rightarrowtail$ *FVar* $y\}$ **S** $\{x \rightarrowtail$ *FVar* $z\}$ A = **S** $\{x \rightarrowtail$ *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 $?\vartheta_{zy} = \{z \rightarrowtail$ *FVar* $y\}$ **and** $?\vartheta_{xz} = \{x \rightarrowtail$ *FVar* $z\}$ **and** $?\vartheta_{xy} = \{x \rightarrowtail$ *FVar* $y\}$
from ‹*is-free-for* (*FVar* z) x ($B \cdot C$)› **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 ‹$z \notin$ *free-vars* ($B \cdot C$)› **have** $z \notin$ *free-vars* B **and** $z \notin$ *free-vars* C
by *simp-all*
ultimately have $*$: **S** $?\vartheta_{zy}$ **S** $?\vartheta_{xz}$ B = **S** $?\vartheta_{xy}$ B **and** $**$: **S** $?\vartheta_{zy}$ **S** $?\vartheta_{xz}$ C = **S** $?\vartheta_{xy}$ C
using *FApp.IH* **by** *simp-all*
have **S** $?\vartheta_{zy}$ **S** $?\vartheta_{xz}$ ($B \cdot C$) = (**S** $?\vartheta_{zy}$ **S** $?\vartheta_{xz}$ B) $\cdot$ (**S** $?\vartheta_{zy}$ **S** $?\vartheta_{xz}$ C)
by *simp*
also from $*$ **and** $**$ **have** ... = (**S** $?\vartheta_{xy}$ B) $\cdot$ (**S** $?\vartheta_{xy}$ C)
by (*simp only*:)
also have ... = **S** $?\vartheta_{xy}$ ($B \cdot C$)
by *simp*
finally show *?case* **.**
next
case (*FAbs* w B)

```
let ?ϑ_zy = {z ↣ FVar y} and ?ϑ_xz = {x ↣ FVar z} and ?ϑ_xy = {x ↣ FVar y}
show ?case
proof (cases x = w)
  case True
  then show ?thesis
  proof (cases z = w)
    case True
    with ‹x = w› have x ∉ free-vars (FAbs w B) and z ∉ free-vars (FAbs w B)
      using surj-pair[of w] by fastforce+
    from ‹x ∉ free-vars (FAbs w B)› have S ?ϑ_xy (FAbs w B) = FAbs w B
      by (fact free-var-singleton-substitution-neutrality)
    also from ‹z ∉ free-vars (FAbs w B)› have ... = S ?ϑ_zy (FAbs w B)
      by (fact free-var-singleton-substitution-neutrality[symmetric])
    also from ‹x ∉ free-vars (FAbs w B)› have ... = S ?ϑ_zy S ?ϑ_xz (FAbs w B)
      using free-var-singleton-substitution-neutrality by simp
    finally show ?thesis ..
  next
    case False
    with ‹x = w› have z ∉ free-vars B and x ∉ free-vars (FAbs w B)
      using ‹z ∉ free-vars (FAbs w B)› and surj-pair[of w] by fastforce+
    from ‹z ∉ free-vars B› have S ?ϑ_zy B = B
      by (fact free-var-singleton-substitution-neutrality)
    from ‹x ∉ free-vars (FAbs w B)› have S ?ϑ_xy (FAbs w B) = FAbs w B
      by (fact free-var-singleton-substitution-neutrality)
    also from ‹S ?ϑ_zy B = B› have ... = FAbs w (S ?ϑ_zy B)
      by (simp only:)
    also from ‹z ∉ free-vars (FAbs w B)› have ... = S ?ϑ_zy (FAbs w B)
      by (simp add: ‹FAbs w B = FAbs w (S ?ϑ_zy B)› free-var-singleton-substitution-neutrality)
    also from ‹x ∉ free-vars (FAbs w B)› have ... = S ?ϑ_zy S ?ϑ_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 ∉ free-vars B
    proof (rule ccontr)
      assume ¬ x ∉ free-vars B
      with ‹x ≠ w› have x ∈ free-vars (FAbs w B)
        using surj-pair[of w] by fastforce
      then obtain p where p ∈ 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 = « # p'
        using is-free-at-from-absE[OF ‹is-free-at x p (FAbs w B)›] by blast
      ultimately have z ≠ w
```

using *in-scope-of-abs-in-abs* **by** *blast*
with ‹$z = w$› **show** *False*
by *contradiction*
qed
then have $*$: **S** $?\vartheta_{xy}$ B = **S** $?\vartheta_{xz}$ B
using *free-var-singleton-substitution-neutrality* **by** *auto*
from ‹$x \neq w$› **have S** $?\vartheta_{xy}$ (*FAbs w B*) = *FAbs w* (**S** $?\vartheta_{xy}$ B)
using *surj-pair*[*of w*] **by** *fastforce*
also from $*$ **have** ... = *FAbs w* (**S** $?\vartheta_{xz}$ B)
by (*simp only*:)
also from *FAbs.prems*(*1*) **have** ... = **S** $?\vartheta_{zy}$ (*FAbs w* (**S** $?\vartheta_{xz}$ B))
using ‹$x \notin$ *free-vars B*› **and** *free-var-singleton-substitution-neutrality* **by** *auto*
also from ‹$x \neq w$› **have** ... = **S** $?\vartheta_{zy}$ **S** $?\vartheta_{xz}$ (*FAbs w B*)
using *surj-pair*[*of w*] **by** *fastforce*
finally show *?thesis* **..**
next
case *False*
obtain v_w **and** α **where** $w = (v_w, \alpha)$
by *fastforce*
with ‹*is-free-for* (*FVar z*) *x* (*FAbs w B*)› **and** ‹$x \neq w$› **have** *is-free-for* (*FVar z*) *x B*
using *is-free-for-from-abs* **by** *iprover*
moreover from ‹$z \notin$ *free-vars* (*FAbs w B*)› **and** ‹$z \neq w$› **and** ‹$w = (v_w, \alpha)$› **have** $z \notin$ *free-vars*
B
by *simp*
ultimately have $*$: **S** $?\vartheta_{zy}$ **S** $?\vartheta_{xz}$ B = **S** $?\vartheta_{xy}$ B
using *FAbs.IH* **by** *simp*
from ‹$x \neq w$› **have S** $?\vartheta_{xy}$ (*FAbs w B*) = *FAbs w* (**S** $?\vartheta_{xy}$ B)
using ‹$w = (v_w, \alpha)$› **and** *free-var-singleton-substitution-neutrality* **by** *simp*
also from $*$ **have** ... = *FAbs w* (**S** $?\vartheta_{zy}$ **S** $?\vartheta_{xz}$ B)
by (*simp only*:)
also from ‹$z \neq w$› **have** ... = **S** $?\vartheta_{zy}$ (*FAbs w* (**S** $?\vartheta_{xz}$ B))
using ‹$w = (v_w, \alpha)$› **and** *free-var-singleton-substitution-neutrality* **by** *simp*
also from ‹$x \neq w$› **have** ... = **S** $?\vartheta_{zy}$ **S** $?\vartheta_{xz}$ (*FAbs w B*)
using ‹$w = (v_w, \alpha)$› **and** *free-var-singleton-substitution-neutrality* **by** *simp*
finally show *?thesis* **..**
qed
qed
qed

lemma *absent-vars-substitution-preservation*:
assumes $v \notin$ *vars A*
and $\forall v' \in$ *fmdom'* ϑ. $v \notin$ *vars* (ϑ \$\$! v')
shows $v \notin$ *vars* (**S** ϑ A)
using *assms* **proof** (*induction A arbitrary*: ϑ)
case (*FVar v'*)
then show *?case*
using *surj-pair*[*of v'*] **by** (*cases* $v' \in$ *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** $v \notin vars\ B$
using *vars-form.elims* **by** *auto*
then show *?case*
proof (*cases* $w \in fmdom'\ \vartheta$)
case *True*
from *FAbs.prems*(*2*) **have** $\forall v' \in fmdom'\ (fmdrop\ w\ \vartheta).\ v \notin vars\ ((fmdrop\ w\ \vartheta)\ \$\$!\ v')$
by *auto*
with ‹$v \notin vars\ B$› **have** $v \notin vars\ (\mathbf{S}\ (fmdrop\ w\ \vartheta)\ B)$
by (*fact FAbs.IH*)
with *FAbs.prems*(*1*) **have** $v \notin vars\ (FAbs\ w\ (\mathbf{S}\ (fmdrop\ w\ \vartheta)\ B))$
using *surj-pair*[*of w*] **by** *fastforce*
moreover from *True* **have** $\mathbf{S}\ \vartheta\ (FAbs\ w\ B) = FAbs\ w\ (\mathbf{S}\ (fmdrop\ w\ \vartheta)\ B)$
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 $\vartheta\ \$\$\ v = None$ **and** $v \notin free\text{-}vars\ B$
shows $\mathbf{S}\ (\{v \rightarrowtail A\} \mathbin{++_f} \vartheta)\ B = \mathbf{S}\ \vartheta\ B$
using *assms* **proof** (*induction B arbitrary*: ϑ)
case (*FAbs w B*)
show *?case*
proof (*cases* $v \neq w$)
case *True*
with *FAbs.prems*(*2*) **have** $v \notin free\text{-}vars\ B$
using *surj-pair*[*of w*] **by** *fastforce*
then show *?thesis*
proof (*cases* $w \in fmdom'\ \vartheta$)
case *True*
then have $\mathbf{S}\ (\{v \rightarrowtail A\} \mathbin{++_f} \vartheta)\ (FAbs\ w\ B) = FAbs\ w\ (\mathbf{S}\ (fmdrop\ w\ (\{v \rightarrowtail A\} \mathbin{++_f} \vartheta))\ B)$
using *surj-pair*[*of w*] **by** *fastforce*
also from ‹$v \neq w$› **and** *True* **have** $\ldots = FAbs\ w\ (\mathbf{S}\ (\{v \rightarrowtail A\} \mathbin{++_f} fmdrop\ w\ \vartheta)\ B)$
by (*simp add*: *fmdrop-fmupd*)
also from *FAbs.prems*(*1*) **and** ‹$v \notin free\text{-}vars\ B$› **have** $\ldots = FAbs\ w\ (\mathbf{S}\ (fmdrop\ w\ \vartheta)\ B)$
using *FAbs.IH* **by** *simp*

also from *True* **have** $\ldots = \mathbf{S}\ \vartheta\ (\mathit{FAbs}\ w\ B)$
using *surj-pair*[*of w*] **by** *fastforce*
finally show *?thesis* **.**
next
case *False*
with *FAbs.prems*(*1*) **have** $\mathbf{S}\ (\{v \rightarrowtail A\} \mathbin{++_f} \vartheta)\ (\mathit{FAbs}\ w\ B) = \mathit{FAbs}\ w\ (\mathbf{S}\ (\{v \rightarrowtail A\} \mathbin{++_f} \vartheta)\ B)$
using $\langle v \neq w\rangle$ **and** *surj-pair*[*of w*] **by** *fastforce*
also from *FAbs.prems*(*1*) **and** $\langle v \notin \mathit{free\text{-}vars}\ B\rangle$ **have** $\ldots = \mathit{FAbs}\ w\ (\mathbf{S}\ \vartheta\ B)$
using *FAbs.IH* **by** *simp*
also from *False* **have** $\ldots = \mathbf{S}\ \vartheta\ (\mathit{FAbs}\ w\ B)$
using *surj-pair*[*of w*] **by** *fastforce*
finally show *?thesis* **.**
qed
next
case *False*
then have $\mathit{fmdrop}\ w\ (\{v \rightarrowtail A\} \mathbin{++_f} \vartheta) = \mathit{fmdrop}\ w\ \vartheta$
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 $\vartheta\ \$\$\ v = \mathit{None}$ **and** $v \notin \mathit{vars}\ B$
shows $\mathbf{S}\ (\{v \rightarrowtail A\} \mathbin{++_f} \vartheta)\ B = \mathbf{S}\ \vartheta\ B$
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 $y \notin \mathit{vars}\ B$
and $x \notin \mathit{fmdom}'\ \vartheta$
and $\forall\, v \in \mathit{fmdom}'\ \vartheta.\ y \notin \mathit{vars}\ (\vartheta\ \$\$!\ v)$
and $\forall\, v \in \mathit{fmdom}'\ \vartheta.\ \mathit{is\text{-}free\text{-}for}\ (\vartheta\ \$\$!\ v)\ v\ B$
shows $\mathit{is\text{-}free\text{-}for}\ A\ y\ (\mathbf{S}\ (\{x \rightarrowtail \mathit{FVar}\ y\} \mathbin{++_f} \vartheta)\ B)$
using *assms* **proof** (*induction B arbitrary*: ϑ)
case (*FVar w*)
then show *?case*
proof (*cases* $w = x$)
case *True*
with *FVar.prems*(*3*) **have** $\mathbf{S}\ (\{x \rightarrowtail \mathit{FVar}\ y\} \mathbin{++_f} \vartheta)\ (\mathit{FVar}\ w) = \mathit{FVar}\ y$
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* $w \in \mathit{fmdom}'\ \vartheta$)
case *True*
from *False* **have** $\mathbf{S}\ (\{x \rightarrowtail \mathit{FVar}\ y\} \mathbin{++_f} \vartheta)\ (\mathit{FVar}\ w) = \mathbf{S}\ \vartheta\ (\mathit{FVar}\ w)$

using *substitution-absorption* **and** *surj-pair*[*of w*] **by** *force*
also from *True* **have** ... $= \vartheta$ \$\$! w
using *surj-pair*[*of w*] **by** (*metis fmdom′-notI option.case-eq-if substitute.simps*(*1*))
finally have S ($\{x \rightarrowtail FVar\ y\} ++_f \vartheta$) ($FVar\ w$) $= \vartheta$ \$\$! w **.**
moreover from *True* **and** *FVar.prems*(*4*) **have** $y \notin vars$ (ϑ \$\$! w)
by *blast*
ultimately show *?thesis*
using *form-is-free-for-absent-var* **by** *presburger*
next
case *False*
with *FVar.prems*(*3*) **and** ‹$w \neq x$› **have S** ($\{x \rightarrowtail FVar\ y\} ++_f \vartheta$) ($FVar\ w$) $= FVar\ w$
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** $y \notin vars\ C$ **and** $y \notin vars\ D$
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 $\forall v \in fmdom'\ \vartheta$. *is-free-for* ($\vartheta$ \$\$! v) v C $\wedge$ *is-free-for* (ϑ \$\$! v) v D
proof (*rule ballI*)
fix v
assume $v \in fmdom'\ \vartheta$
with *FApp.prems*(*5*) **have** *is-free-for* (ϑ \$\$! v) v ($C \bullet D$)
by *blast*
then show *is-free-for* (ϑ \$\$! v) v C $\wedge$ *is-free-for* (ϑ \$\$! v) v D
using *is-free-for-from-app* **by** *iprover+*
qed
then have
$*$: $\forall v \in fmdom'\ \vartheta$. *is-free-for* ($\vartheta$ \$\$! v) v C **and** $**$: $\forall v \in fmdom'\ \vartheta$. *is-free-for* ($\vartheta$ \$\$! v) v D
by *auto*
have S ($\{x \rightarrowtail FVar\ y\} ++_f \vartheta$) ($C \bullet D$) $=$ (**S** ($\{x \rightarrowtail FVar\ y\} ++_f \vartheta$) C) $\bullet$ (**S** ($\{x \rightarrowtail FVar\ y\}$ $++_f \vartheta$) D)
by *simp*
moreover have *is-free-for A y* (**S** ($\{x \rightarrowtail FVar\ y\} ++_f \vartheta$) C)
by (*rule FApp.IH*(*1*)[*OF* ‹*is-free-for A x C*› ‹$y \notin vars\ C$› *FApp.prems*(*3*,*4*) $*$])
moreover have *is-free-for A y* (**S** ($\{x \rightarrowtail FVar\ y\} ++_f \vartheta$) D)
by (*rule FApp.IH*(*2*)[*OF* ‹*is-free-for A x D*› ‹$y \notin vars\ D$› *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′* ϑ› **have** $w \notin$ *fmdom′* ϑ **and** $x \notin$ *free-vars* (*FAbs w B*)
using ‹$w = (x_w, \alpha_w)$› **by** *fastforce+*
with *True* **have** **S** ($\{x \rightarrowtail$ *FVar y*$\}$ $++_f$ ϑ) (*FAbs w B*) = **S** ϑ (*FAbs w B*)
using *substitution-free-absorption* **by** *blast*
also have ... = *FAbs w* (**S** ϑ *B*)
using ‹$w = (x_w, \alpha_w)$› ‹$w \notin$ *fmdom′* ϑ› *substitute.simps(4)* **by** *presburger*
finally have **S** ($\{x \rightarrowtail$ *FVar y*$\}$ $++_f$ ϑ) (*FAbs w B*) = *FAbs w* (**S** ϑ *B*) **.**
moreover from ‹**S** ϑ (*FAbs w B*) = *FAbs w* (**S** ϑ *B*)› **have** $y \notin$ *vars* (*FAbs w* (**S** ϑ *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′* ϑ)
case *True*
then have **S** ($\{x \rightarrowtail$ *FVar y*$\}$ $++_f$ ϑ) (*FAbs w B*) = *FAbs w* (**S** (*fmdrop w* ($\{x \rightarrowtail$ *FVar y*$\}$ $++_f$ ϑ)) *B*)
using ‹$w = (v_w, \alpha_w)$› **by** (*simp add*: *fmdrop-idle′*)
also from ‹$w \neq x$› **and** *True* **have** ... = *FAbs w* (**S** ($\{x \rightarrowtail$ *FVar y*$\}$ $++_f$ *fmdrop w* ϑ) *B*)
by (*simp add*: *fmdrop-fmupd*)
finally
have $*$: **S** ($\{x \rightarrowtail$ *FVar y*$\}$ $++_f$ ϑ) (*FAbs w B*) = *FAbs w* (**S** ($\{x \rightarrowtail$ *FVar y*$\}$ $++_f$ *fmdrop w* ϑ) *B*) **.**
have $\forall\, v \in$ *fmdom′* (*fmdrop w* ϑ). *is-free-for* (*fmdrop w* ϑ \$\$! *v*) *v B*
proof
fix *v*
assume $v \in$ *fmdom′* (*fmdrop w* ϑ)
with *FAbs.prems(5)* **have** *is-free-for* (*fmdrop w* ϑ \$\$! *v*) *v* (*FAbs w B*)
by *auto*
moreover from ‹$v \in$ *fmdom′* (*fmdrop w* ϑ)› **have** $v \neq w$
by *auto*
ultimately show *is-free-for* (*fmdrop w* ϑ \$\$! *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* ϑ)
by *simp*
moreover from *FAbs.prems(4)* **have** $\forall\, v \in$ *fmdom′* (*fmdrop w* ϑ). $y \notin$ *vars* (*fmdrop w* ϑ \$\$! *v*)

by *simp*
ultimately have *is-free-for A y* (**S** ($\{x \rightarrowtail FVar\ y\}$ $++_f$ *fmdrop w* ϑ) *B*)
using ‹*is-free-for A x B*› **and** ‹$y \notin$ *vars B*› **and** *FAbs.IH* **by** *iprover*
then show *?thesis*
proof (*cases* $x \notin$ *free-vars B*)
case *True*
have $y \notin$ *vars* (**S** ($\{x \rightarrowtail FVar\ y\}$ $++_f$ ϑ) (*FAbs w B*))
proof −
have S ($\{x \rightarrowtail FVar\ y\}$ $++_f$ ϑ) (*FAbs w B*) = *FAbs w* (**S** ($\{x \rightarrowtail FVar\ y\}$ $++_f$ *fmdrop w* ϑ) *B*)
using ∗ **.**
also from ‹$x \notin$ *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 ($\{x \rightarrowtail FVar\ y\}$ $++_f$ ϑ) (*FAbs w B*) = *FAbs w* (**S** (*fmdrop w* ϑ) *B*) **.**
with *FAbs.prems*(*2*) **and** ‹$w = (v_w, \alpha_w)$› **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 $w \notin$ *free-vars A*
proof (*rule ccontr*)
assume ¬ $w \notin$ *free-vars A*
with *False* **and** ‹$w \neq x$› **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** ($\{x \rightarrowtail FVar\ y\}$ $++_f$ *fmdrop w* ϑ) *B*)›
have *is-free-for A y* (*FAbs w* (**S** ($\{x \rightarrowtail FVar\ y\}$ $++_f$ *fmdrop w* ϑ) *B*))
unfolding ‹$w = (v_w, \alpha_w)$› **using** *is-free-for-to-abs* **by** *iprover*
with ∗ **show** *?thesis*
by (*simp only*:)
qed
next
case *False*
have $\forall v \in$ *fmdom′* ϑ. *is-free-for* (ϑ \$\$! *v*) *v B*
proof (*rule ballI*)
fix *v*
assume $v \in$ *fmdom′* ϑ
with *FAbs.prems*(*5*) **have** *is-free-for* (ϑ \$\$! *v*) *v* (*FAbs w B*)
by *blast*
moreover from ‹$v \in$ *fmdom′* ϑ› **and** ‹$w \notin$ *fmdom′* ϑ› **have** $v \neq w$
by *blast*
ultimately show *is-free-for* (ϑ \$\$! *v*) *v B*
unfolding ‹$w = (v_w, \alpha_w)$› **using** *is-free-for-from-abs* **by** *iprover*
qed
with ‹*is-free-for A x B*› **and** ‹$y \notin$ *vars B*› **and** *FAbs.prems*(*3*,*4*)

have *is-free-for* $A\ y$ (**S** $(\{x \rightarrowtail \mathit{FVar}\ y\} \mathbin{++_f} \vartheta)\ B$)
using *FAbs.IH* **by** *iprover*
then show *?thesis*
proof (*cases* $x \notin$ *free-vars* B)
case *True*
have $y \notin$ *vars* (**S** $(\{x \rightarrowtail \mathit{FVar}\ y\} \mathbin{++_f} \vartheta)$ $(\mathit{FAbs}\ w\ B)$)
proof −
from *False* **and** ‹$w = (v_w, \alpha_w)$› **and** ‹$w \neq x$›
have S $(\{x \rightarrowtail \mathit{FVar}\ y\} \mathbin{++_f} \vartheta)$ $(\mathit{FAbs}\ w\ B) = \mathit{FAbs}\ w$ (**S** $(\{x \rightarrowtail \mathit{FVar}\ y\} \mathbin{++_f} \vartheta)\ B$)
by *auto*
also from ‹$x \notin$ *free-vars* B› **and** *FAbs.prems*(*3*) **have** $\ldots = \mathit{FAbs}\ w$ (**S** $\vartheta\ B$)
using *substitution-free-absorption* **by** (*simp add*: *fmdom′-notD*)
finally have S $(\{x \rightarrowtail \mathit{FVar}\ y\} \mathbin{++_f} \vartheta)$ $(\mathit{FAbs}\ w\ B) = \mathit{FAbs}\ w$ (**S** $\vartheta\ B$) **.**
with *FAbs.prems*(*2*,*4*) **and** ‹$w = (v_w, \alpha_w)$› **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 $w \notin$ *free-vars* A
proof (*rule ccontr*)
assume $\neg\ w \notin$ *free-vars* A
with *False* **and** ‹$w \neq x$› **have** $\neg$ *is-free-for* $A\ x$ $(\mathit{FAbs}\ w\ B)$
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** $(\{x \rightarrowtail \mathit{FVar}\ y\} \mathbin{++_f} \vartheta)\ B$)›
have *is-free-for* $A\ y$ $(\mathit{FAbs}\ w$ (**S** $(\{x \rightarrowtail \mathit{FVar}\ y\} \mathbin{++_f} \vartheta)\ B$))
unfolding ‹$w = (v_w, \alpha_w)$› **using** *is-free-for-to-abs* **by** *iprover*
moreover from ‹$w \notin$ *fmdom′* ϑ› **and** ‹$w \neq x$› **and** *FAbs.prems*(*3*)
have S $(\{x \rightarrowtail \mathit{FVar}\ y\} \mathbin{++_f} \vartheta)$ $(\mathit{FAbs}\ w\ B) = \mathit{FAbs}\ w$ (**S** $(\{x \rightarrowtail \mathit{FVar}\ y\} \mathbin{++_f} \vartheta)\ B$)
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* $\{v \rightarrowtail A\}$
and ϑ \$\$ $v = \mathit{None}$ **and** $\forall\, v' \in$ *fmdom′* ϑ. $v' \notin$ *vars* (ϑ \$\$! v')
shows S $\{v \rightarrowtail A\}$ **S** $\vartheta\ B$ = **S** $(\{v \rightarrowtail A\} \mathbin{++_f} \vartheta)\ B$
using *assms*(*1*,*3*,*4*) **proof** (*induction* B *arbitrary*: ϑ)
case $(\mathit{FVar}\ v')$

then show *?case*
proof $(cases\ v' \notin fmdom'\ \vartheta)$
  case *True*
  then show *?thesis*
    using *surj-pair*[*of v'*] **by** *fastforce*
next
  case *False*
  then obtain A' **where** $\vartheta\ \$\$\ v' = Some\ A'$
    by (*meson fmlookup-dom'-iff*)
  with *False* **and** *FVar.prems*(*3*) **have** $v \notin vars\ A'$
    by *fastforce*
  then have $\mathbf{S}\ \{v \rightarrowtail A\}\ A' = A'$
    using *free-var-singleton-substitution-neutrality* **and** *free-vars-in-all-vars* **by** *blast*
  from ‹$\vartheta\ \$\$\ v' = Some\ A'$› **have** $\mathbf{S}\ \{v \rightarrowtail A\}\ \mathbf{S}\ \vartheta\ (FVar\ v') = \mathbf{S}\ \{v \rightarrowtail A\}\ A'$
    using *surj-pair*[*of v'*] **by** *fastforce*
  also from ‹$\mathbf{S}\ \{v \rightarrowtail A\}\ A' = A'$› **have** $\ldots = A'$
    by (*simp only*:)
  also from ‹$\vartheta\ \$\$\ v' = Some\ A'$› **and** ‹$\vartheta\ \$\$\ v = None$› **have** $\ldots = \mathbf{S}\ (\{v \rightarrowtail A\} ++_f \vartheta)\ (FVar\ v')$
    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 $\mathbf{S}\ \{v \rightarrowtail A\}\ \mathbf{S}\ \vartheta\ (C \cdot D) = \mathbf{S}\ \{v \rightarrowtail A\}\ ((\mathbf{S}\ \vartheta\ C) \cdot (\mathbf{S}\ \vartheta\ D))$
  by *auto*
also have $\ldots = (\mathbf{S}\ \{v \rightarrowtail A\}\ \mathbf{S}\ \vartheta\ C) \cdot (\mathbf{S}\ \{v \rightarrowtail A\}\ \mathbf{S}\ \vartheta\ D)$
  by *simp*
also from *FApp.IH* **have** $\ldots = (\mathbf{S}\ (\{v \rightarrowtail A\} ++_f \vartheta)\ C) \cdot (\mathbf{S}\ (\{v \rightarrowtail A\} ++_f \vartheta)\ D)$
  using *FApp.prems*(*1*,*2*,*3*) **by** *presburger*
also have $\ldots = \mathbf{S}\ (\{v \rightarrowtail A\} ++_f \vartheta)\ (C \cdot D)$
  by *simp*
finally show *?case* **.**
next
case (*FAbs w C*)
obtain v_w **and** α **where** $w = (v_w, \alpha)$
  by *fastforce*
then show *?case*
proof $(cases\ v \neq w)$
  case *True*
  show *?thesis*
  proof $(cases\ w \notin fmdom'\ \vartheta)$
    case *True*
    then have $\mathbf{S}\ \{v \rightarrowtail A\}\ \mathbf{S}\ \vartheta\ (FAbs\ w\ C) = \mathbf{S}\ \{v \rightarrowtail A\}\ (FAbs\ w\ (\mathbf{S}\ \vartheta\ C))$
      by (*simp add*: ‹$w = (v_w, \alpha)$›)
    also from ‹$v \neq w$› **have** $\ldots = FAbs\ w\ (\mathbf{S}\ \{v \rightarrowtail A\}\ \mathbf{S}\ \vartheta\ C)$

by (*simp add*: ‹$w = (v_w, \alpha)$›)
also from *FAbs.IH* **have** ... $=$ *FAbs* w (**S** ({$v \rightarrowtail A$} $++_f$ ϑ) C)
using *FAbs.prems*(*1*,*2*,*3*) **by** *blast*
also from ‹$v \neq w$› **and** *True* **have** ... $=$ **S** ({$v \rightarrowtail A$} $++_f$ ϑ) (*FAbs* w C)
by (*simp add*: ‹$w = (v_w, \alpha)$›)
finally show *?thesis* **.**
next
case *False*
then have S {$v \rightarrowtail A$} **S** ϑ (*FAbs* w C) $=$ **S** {$v \rightarrowtail A$} (*FAbs* w (**S** (*fmdrop* w ϑ) C))
by (*simp add*: ‹$w = (v_w, \alpha)$›)
also from ‹$v \neq w$› **have** ... $=$ *FAbs* w (**S** {$v \rightarrowtail A$} **S** (*fmdrop* w ϑ) C)
by (*simp add*: ‹$w = (v_w, \alpha)$›)
also have ... $=$ *FAbs* w (**S** ({$v \rightarrowtail A$} $++_f$ *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** $\forall\, v' \in$ *fmdom'* (*fmdrop* w ϑ). $v \notin$ *vars* ((*fmdrop* w ϑ) \$\$!
v')
by *force*
ultimately show *?thesis*
using *FAbs.IH* **by** *blast*
qed
also from ‹$v \neq w$› **have** ... $=$ **S** ({$v \rightarrowtail A$} $++_f$ ϑ) (*FAbs* w C)
by (*simp add*: ‹$w = (v_w, \alpha)$› *fmdrop-idle'*)
finally show *?thesis* **.**
qed
next
case *False*
then show *?thesis*
proof (*cases* $w \notin$ *fmdom'* ϑ)
case *True*
then have S {$v \rightarrowtail A$} **S** ϑ (*FAbs* w C) $=$ **S** {$v \rightarrowtail A$} (*FAbs* w (**S** ϑ C))
by (*simp add*: ‹$w = (v_w, \alpha)$›)
also from ‹$\neg\, v \neq w$› **have** ... $=$ *FAbs* w (**S** ϑ C)
using ‹$w = (v_w, \alpha)$› **and** *singleton-substitution-simps*(*4*) **by** *presburger*
also from ‹$\neg\, v \neq w$› **and** *True* **have** ... $=$ *FAbs* w (**S** (*fmdrop* w ({$v \rightarrowtail A$} $++_f$ ϑ)) C)
by (*simp add*: *fmdrop-fmupd-same fmdrop-idle'*)
also from ‹$\neg\, v \neq w$› **have** ... $=$ **S** ({$v \rightarrowtail A$} $++_f$ ϑ) (*FAbs* w C)
by (*simp add*: ‹$w = (v_w, \alpha)$›)
finally show *?thesis* **.**
next
case *False*
then have S {$v \rightarrowtail A$} **S** ϑ (*FAbs* w C) $=$ **S** {$v \rightarrowtail A$} (*FAbs* w (**S** (*fmdrop* w ϑ) C))
by (*simp add*: ‹$w = (v_w, \alpha)$›)
also from ‹$\neg\, v \neq w$› **have** ... $=$ *FAbs* w (**S** (*fmdrop* w ϑ) C)
using ‹ϑ \$\$ $v =$ *None*› **and** *False* **by** (*simp add*: *fmdom'-notI*)
also from ‹$\neg\, v \neq w$› **have** ... $=$ *FAbs* w (**S** (*fmdrop* w ({$v \rightarrowtail A$} $++_f$ ϑ)) C)

by (*simp add*: *fmdrop-fmupd-same*)
also from ‹¬ $v \neq w$› **and** *False* **and** ‹ϑ $$ v = *None*› **have** ... = **S** ($\{v \rightarrowtail A\} ++_f \vartheta$) (*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. $\exists v$. A = *FVar* v) ϑ

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

- $x^1_{\alpha_1} \ \dots \ x^n_{\alpha_n}$ are distinct variables
- $y^1_{\alpha_1} \ \dots \ y^n_{\alpha_n}$ are distinct variables, distinct from $x^1_{\alpha_1} \ \dots \ x^n_{\alpha_n}$ 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* $\times$ *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) $\cap$ (*fmdom′* $\vartheta \cup$ *vars B*) = {}
shows S ϑ B = *foldr* ($\lambda(x, y)$ C. **S** $\{x \rightarrowtail y\}$ C) *ps B*
using *assms* **proof** (*induction ps arbitrary*: ϑ)
case *Nil*
then have ϑ = {$$}

by *simp*
then have $\mathbf{S}\ \vartheta\ B = B$
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 $?\vartheta' = $ *fmap-of-list ps*
from *Cons.prems*(*1*) **and** ‹$p = (x,\ FVar\ y)$› **have** $\vartheta = $ *fmupd* x (*FVar y*) $?\vartheta'$
by *simp*
moreover from *Cons.prems*(*3*) **and** ‹$p = (x,\ FVar\ y)$› **have** $x \notin$ *fmdom'* $?\vartheta'$
by *simp*
ultimately have $\vartheta = \{x \rightarrowtail FVar\ y\} +\!+_f\ ?\vartheta'$
using *fmap-singleton-comm* **by** *fastforce*
with *Cons.prems*(*2*) **and** ‹$x \notin$ *fmdom'* $?\vartheta'$› **have** *is-renaming-substitution* $?\vartheta'$
unfolding *is-renaming-substitution-def* **and** ‹$\vartheta = $ *fmupd* x (*FVar y*) $?\vartheta'$›
using *updated-substitution-is-substitution* **by** (*metis fmdiff-fmupd fmdom'-notD fmpred-filter*)
from *Cons.prems*(*2*) **and** ‹$\vartheta = $ *fmupd* x (*FVar y*) $?\vartheta'$› **have** *is-renaming-substitution* $\{x \rightarrowtail FVar$
$y\}$
by *auto*
have
foldr ($\lambda(x,\ y)\ C.\ \mathbf{S}\ \{x \rightarrowtail y\}\ C$) ($p\ \#\ ps$) B
$=$
$\mathbf{S}\ \{x \rightarrowtail FVar\ y\}$ (*foldr* ($\lambda(x,\ y)\ C.\ \mathbf{S}\ \{x \rightarrowtail y\}\ C$) *ps B*)
by (*simp add*: ‹$p = (x,\ FVar\ y)$›)
also have $\ldots = \mathbf{S}\ \{x \rightarrowtail FVar\ y\}\ \mathbf{S}\ ?\vartheta'\ B$
proof −
from *Cons.prems*(*3*,*4*) **have** *distinct* (*map fst ps*) **and** *distinct* (*map snd ps*)
by *fastforce*+
moreover have *vars* (*fmran'* $?\vartheta'$) $\cap$ (*fmdom'* $?\vartheta' \cup$ *vars* B) $= \{\}$
proof −
have *vars* (*fmran'* ϑ) = *vars* ($\{FVar\ y\} \cup$ *fmran'* $?\vartheta'$)
using ‹$\vartheta = $ *fmupd* x (*FVar y*) $?\vartheta'$› **and** ‹$x \notin$ *fmdom'* $?\vartheta'$› **by** (*metis fmdom'-notD fmran'-fmupd*)
then have *vars* (*fmran'* ϑ) $= \{y\} \cup$ *vars* (*fmran'* $?\vartheta'$)
using *singleton-form-set-vars* **by** *auto*
moreover have *fmdom'* $\vartheta = \{x\} \cup$ *fmdom'* $?\vartheta'$
by (*simp add*: ‹$\vartheta = \{x \rightarrowtail FVar\ y\} +\!+_f\ ?\vartheta'$›)
ultimately show *?thesis*
using *Cons.prems*(*5*) **by** *auto*
qed
ultimately show *?thesis*
using *Cons.IH* **and** ‹*is-renaming-substitution* $?\vartheta'$› **by** *simp*
qed
also have $\ldots = \mathbf{S}\ (\{x \rightarrowtail FVar\ y\} +\!+_f\ ?\vartheta')\ B$
proof (*rule substitution-fusion*)
show *is-substitution* $?\vartheta'$
using ‹*is-renaming-substitution* $?\vartheta'$› **by** *simp*

show *is-substitution* $\{x \rightarrowtail FVar\ y\}$
using ‹*is-renaming-substitution* $\{x \rightarrowtail FVar\ y\}$› **by** *simp*
show $?\vartheta'$ \$\$ $x = None$
using ‹$x \notin fmdom'\ ?\vartheta'$› **by** *blast*
show $\forall\, v' \in fmdom'\ ?\vartheta'.\ x \notin vars\ (?\vartheta'$ \$\$! $v')$
proof −
have $x \in fmdom'\ \vartheta$
using ‹$\vartheta = \{x \rightarrowtail FVar\ y\} ++_f\ ?\vartheta'$› **by** *simp*
then have $x \notin vars\ (fmran'\ \vartheta)$
using *Cons.prems*(*5*) **by** *blast*
moreover have $\{?\vartheta'$ \$\$! $v' \mid v'.\ v' \in fmdom'\ ?\vartheta'\} \subseteq fmran'\ \vartheta$
unfolding ‹$\vartheta = ?\vartheta'(x \rightarrowtail FVar\ y)$› **using** ‹$?\vartheta'$ \$\$ $x = None$›
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 ‹$\vartheta = \{x \rightarrowtail FVar\ y\} ++_f\ ?\vartheta'$› **have** $\ldots = \mathbf{S}\ \vartheta\ B$
by (*simp only*:)
finally show *?case* **..**
qed

lemma *free-vars-agreement-substitution-equality*:
assumes $fmdom'\ \vartheta = fmdom'\ \vartheta'$
and $\forall\, v \in free\text{-}vars\ A \cap fmdom'\ \vartheta.\ \vartheta$ \$\$! $v = \vartheta'$ \$\$! v
shows $\mathbf{S}\ \vartheta\ A = \mathbf{S}\ \vartheta'\ A$
using *assms* **proof** (*induction A arbitrary*: $\vartheta\ \vartheta'$)
case (*FVar v*)
have $free\text{-}vars\ (FVar\ v) = \{v\}$
using *surj-pair*[*of v*] **by** *fastforce*
with *FVar* **have** ϑ \$\$! $v = \vartheta'$ \$\$! 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 $\mathbf{S}\ \vartheta\ (B \cdot C) = (\mathbf{S}\ \vartheta\ B) \cdot (\mathbf{S}\ \vartheta\ C)$
by *simp*
also have $\ldots = (\mathbf{S}\ \vartheta'\ B) \cdot (\mathbf{S}\ \vartheta'\ C)$
proof −
have $\forall\, v \in free\text{-}vars\ B \cap fmdom'\ \vartheta.\ \vartheta$ \$\$! $v = \vartheta'$ \$\$! v
and $\forall\, v \in free\text{-}vars\ C \cap fmdom'\ \vartheta.\ \vartheta$ \$\$! $v = \vartheta'$ \$\$! 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 $\mathsf{S}\,{}^{x_\alpha}_{A_\alpha}\ \mathsf{S}\,{}^{x^1_{\alpha_1}\ \dots\ x^n_{\alpha_n}}_{A^1_{\alpha_1}\ \dots\ A^n_{\alpha_n}}\ B = \mathsf{S}\,{}^{x_\alpha \qquad x^1_{\alpha_1} \qquad \dots \qquad x^n_{\alpha_n}}_{A_\alpha\ \mathsf{S}{}^{x_\alpha}_{A_\alpha} A^1_{\alpha_1}\ \dots\ \mathsf{S}{}^{x_\alpha}_{A_\alpha} A^n_{\alpha_n}}\ B$ provided that x_α is distinct from $x^1_{\alpha_1}, \dots, x^n_{\alpha_n}$ 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 $\forall\, v' \in \mathit{fmdom'}\ \vartheta.\ \textit{is-free-for}\ (\vartheta\ \$\$!\ v')\ v'\ B \wedge \textit{is-free-for}\ (\vartheta\ \$\$!\ v')\ v'\ C$
proof
fix v'
assume $v' \in \mathit{fmdom'}\ \vartheta$
with $\mathit{FApp.prems}(2)$ **have** $\textit{is-free-for}\ (\vartheta\ \$\$!\ v')\ v'\ (B \bullet C)$
by *blast*
then show $\textit{is-free-for}\ (\vartheta\ \$\$!\ v')\ v'\ B \wedge \textit{is-free-for}\ (\vartheta\ \$\$!\ v')\ v'\ C$
using *is-free-for-from-app* **by** *iprover*
qed
with *FApp.IH* **and** $\mathit{FApp.prems}(1)$ **show** *?case*
by *simp*
next
case $(\mathit{FAbs}\ w\ B)$
let $?\vartheta' = \mathit{fmmap}\ (\lambda A'.\ \mathbf{S}\ \{v \rightarrowtail A\}\ A')\ \vartheta$
show *?case*
proof $(\mathit{cases}\ w \in \mathit{fmdom'}\ \vartheta)$
case *True*
then have $w \in \mathit{fmdom'}\ ?\vartheta'$
by *simp*
with *True* **and** *FAbs.prems* **have** $v \neq w$
by *blast*
from *True* **have** $\mathbf{S}\ \{v \rightarrowtail A\}\ \mathbf{S}\ \vartheta\ (\mathit{FAbs}\ w\ B) = \mathbf{S}\ \{v \rightarrowtail A\}\ (\mathit{FAbs}\ w\ (\mathbf{S}\ (\mathit{fmdrop}\ w\ \vartheta)\ B))$
using *surj-pair*[*of w*] **by** *fastforce*
also from ‹$v \neq w$› **have** $\ldots = \mathit{FAbs}\ w\ (\mathbf{S}\ \{v \rightarrowtail A\}\ \mathbf{S}\ (\mathit{fmdrop}\ w\ \vartheta)\ B)$
using *surj-pair*[*of w*] **by** *fastforce*
also have $\ldots = \mathit{FAbs}\ w\ (\mathbf{S}\ (\mathit{fmdrop}\ w\ (\{v \rightarrowtail A\} \mathbin{++_f} ?\vartheta'))\ B)$
proof −
obtain x_w **and** α_w **where** $w = (x_w, \alpha_w)$
by *fastforce*
have $\forall\, v' \in \mathit{fmdom'}\ (\mathit{fmdrop}\ w\ \vartheta).\ \textit{is-free-for}\ ((\mathit{fmdrop}\ w\ \vartheta)\ \$\$!\ v')\ v'\ B$
proof
fix v'
assume $v' \in \mathit{fmdom'}\ (\mathit{fmdrop}\ w\ \vartheta)$
with $\mathit{FAbs.prems}(2)$ **have** $\textit{is-free-for}\ (\vartheta\ \$\$!\ v')\ v'\ (\mathit{FAbs}\ w\ B)$
by *auto*
with ‹$w = (x_w, \alpha_w)$› **and** ‹$v' \in \mathit{fmdom'}\ (\mathit{fmdrop}\ w\ \vartheta)$›
have $\textit{is-free-for}\ (\vartheta\ \$\$!\ v')\ v'\ (\lambda x_{w}{\alpha_w}.\ B)$ **and** $v' \neq (x_w, \alpha_w)$
by *auto*
then have $\textit{is-free-for}\ (\vartheta\ \$\$!\ v')\ v'\ B$
using *is-free-for-from-abs* **by** *presburger*
with ‹$v' \neq (x_w, \alpha_w)$› **and** ‹$w = (x_w, \alpha_w)$› **show** $\textit{is-free-for}\ (\mathit{fmdrop}\ w\ \vartheta\ \$\$!\ v')\ v'\ B$
by *simp*
qed
moreover have $v \notin \mathit{fmdom'}\ (\mathit{fmdrop}\ w\ \vartheta)$
by $(\mathit{simp\ add}\colon \mathit{FAbs.prems}(1))$
ultimately show *?thesis*
using *FAbs.IH* **and** ‹$v \neq w$› **by** $(\mathit{simp\ add}\colon \textit{fmdrop-fmupd})$
qed
finally show *?thesis*

using ‹$w \in$ *fmdom′* $?\vartheta'$› **and** *surj-pair*[*of w*] **by** *fastforce*
next
case *False*
then have $w \notin$ *fmdom′* $?\vartheta'$
by *simp*
from *FAbs.prems* **have** $v \notin$ *fmdom′* $?\vartheta'$
by *simp*
from *False* **have** $*$: $\mathbf{S}\ \{v \rightarrowtail A\}\ \mathbf{S}\ \vartheta\ (\mathit{FAbs}\ w\ B) = \mathbf{S}\ \{v \rightarrowtail A\}\ (\mathit{FAbs}\ w\ (\mathbf{S}\ \vartheta\ B))$
using *surj-pair*[*of w*] **by** *fastforce*
then show *?thesis*
proof (*cases* $v \neq w$)
case *True*
then have $\mathbf{S}\ \{v \rightarrowtail A\}\ (\mathit{FAbs}\ w\ (\mathbf{S}\ \vartheta\ B)) = \mathit{FAbs}\ w\ (\mathbf{S}\ \{v \rightarrowtail A\}\ (\mathbf{S}\ \vartheta\ B))$
using *surj-pair*[*of w*] **by** *fastforce*
also have $\ldots = \mathit{FAbs}\ w\ (\mathbf{S}\ (\{v \rightarrowtail A\} \mathbin{++_f} ?\vartheta')\ B)$
proof −
obtain x_w **and** α_w **where** $w = (x_w, \alpha_w)$
by *fastforce*
have $\forall\, v' \in$ *fmdom′* ϑ. *is-free-for* (ϑ \$\$! v') v' B
proof
fix v'
assume $v' \in$ *fmdom′* ϑ
with *FAbs.prems*(*2*) **have** *is-free-for* (ϑ \$\$! v') v' (*FAbs w B*)
by *auto*
with ‹$w = (x_w, \alpha_w)$› **and** ‹$v' \in$ *fmdom′* ϑ› **and** *False*
have *is-free-for* (ϑ \$\$! v') v' ($\lambda x_{w\alpha_w}.\ B$) **and** $v' \neq (x_w, \alpha_w)$
by *fastforce+*
then have *is-free-for* (ϑ \$\$! v') v' B
using *is-free-for-from-abs* **by** *presburger*
with ‹$v' \neq (x_w, \alpha_w)$› **and** ‹$w = (x_w, \alpha_w)$› **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
$\mathbf{S}\ \{v \rightarrowtail A\}\ (\mathit{FAbs}\ w\ (\mathbf{S}\ \vartheta\ B)) = \mathit{FAbs}\ w\ (\mathbf{S}\ (\{v \rightarrowtail A\} \mathbin{++_f} \mathit{fmmap}\ (\mathit{substitute}\ \{v \rightarrowtail A\})\ \vartheta)$
$B)$
moreover have $w \notin$ *fmdom′* $(\{v \rightarrowtail A\} \mathbin{++_f} \mathit{fmmap}\ (\mathit{substitute}\ \{v \rightarrowtail A\})\ \vartheta)$
using *False* **and** *True* **by** *auto*
ultimately show *?thesis*
using $*$ **and** *surj-pair*[*of w*] **by** *fastforce*
qed
next
case *False*
then have $v \notin$ *free-vars* $(\mathit{FAbs}\ w\ (\mathbf{S}\ \vartheta\ B))$
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 $v \notin$ *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 $v \notin$ *fmdom′* ϑ
and $v \notin$ *vars* (*fmran′* ϑ)
and $v \notin$ *vars* A
shows $v \notin$ *vars* (**S** ϑ A)
using *assms* **proof** (*induction* A *arbitrary*: ϑ)
case (*FVar* v')
then show *?case*
proof (*cases* $v' \in$ *fmdom′* ϑ)
case *True*
then have ϑ \$\$! $v' \in$ *fmran′* ϑ
by (*simp add*: *fmlookup-dom′-iff fmran′I*)
with *FVar*(*3*) **have** $v \notin$ *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 $v \notin$ *vars* B **and** $v \neq w$
using *surj-pair*[*of* w] **and** *FAbs.prems*(*4*) **by** *fastforce+*
then show *?case*
proof (*cases* $w \notin$ *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** $v \notin$ *vars* (**S** ϑ B)
using *FAbs.prems*(*1−3*) **and** ‹$v \notin$ *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** $\lambda x_\delta.$ *rename-bound-var* (y, γ) z $A \in$ *wffs*$_{\delta\to\beta}$
    by *blast*
  with *False* **show** *?thesis*
    by *auto*
qed
qed *auto*

lemma *old-bound-var-not-free-in-abs-after-renaming*:
  assumes $A \in$ *wffs*$_\alpha$
  and $z_\gamma \neq y_\gamma$
  and $(z, \gamma) \notin$ *vars* A
  shows $(y, \gamma) \notin$ *free-vars* (*rename-bound-var* (y, γ) z $(\lambda y_\gamma.\ A)$)
  using *assms* **and** *free-var-in-renaming-substitution* **by** (*induction* A) *auto*

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

lemma *old-bound-var-not-free-after-renaming*:
  assumes $A \in$ *wffs*$_\alpha$
  and $z_\gamma \neq y_\gamma$
  and $(z, \gamma) \notin$ *vars* A
  and $(y, \gamma) \notin$ *free-vars* A
  shows $(y, \gamma) \notin$ *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* $(x, \alpha) = (y, \gamma)$)
    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 $A \in$ *wffs*$_\alpha$
  and $z_\gamma \neq y_\gamma$
  shows $\neg$ *occurs-at* (y, γ) p (**S** $\{(y, \gamma) \rightarrowtail z_\gamma\}$ (*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 $A \in wffs_\alpha$
and $z_\gamma \neq y_\gamma$
and $(z, \gamma) \notin vars\ A$
and *occurs-at* $(y, \gamma)\ p\ ($*rename-bound-var* $(y, \gamma)\ z\ A)$
shows $\neg$ *in-scope-of-abs* $(z, \gamma)\ p\ ($*rename-bound-var* $(y, \gamma)\ z\ A)$
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** $(z, \gamma) \notin vars\ B$ **and** $(z, \gamma) \notin vars\ C$
by *simp-all*
from *app-is-wff*.*prems*(*2*)
have *occurs-at* $(y, \gamma)\ p\ ($*rename-bound-var* $(y, \gamma)\ z\ B \bullet$ *rename-bound-var* $(y, \gamma)\ z\ C)$
by *simp*
then consider
$(a)\ \exists p'.\ p = « \# p' \wedge$ *occurs-at* $(y, \gamma)\ p'\ ($*rename-bound-var* $(y, \gamma)\ z\ B)$
| $(b)\ \exists p'.\ p = » \# p' \wedge$ *occurs-at* $(y, \gamma)\ p'\ ($*rename-bound-var* $(y, \gamma)\ z\ C)$
using *subforms-from-app* **by** *force*
then show *?case*
proof *cases*
case *a*
then obtain p' **where** $p = « \# p'$ **and** *occurs-at* $(y, \gamma)\ p'\ ($*rename-bound-var* $(y, \gamma)\ z\ B)$
by *blast*
then have $\neg$ *in-scope-of-abs* $(z, \gamma)\ p'\ ($*rename-bound-var* $(y, \gamma)\ z\ B)$
using *app-is-wff*.*IH*(*1*)[*OF* ‹$(z, \gamma) \notin vars\ B$›] **by** *blast*
then have $\neg$ *in-scope-of-abs* $(z, \gamma)\ p\ ($*rename-bound-var* $(y, \gamma)\ z\ (B \bullet C))$ **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* $(y, \gamma)\ p'\ ($*rename-bound-var* $(y, \gamma)\ z\ C)$
by *blast*
then have $\neg$ *in-scope-of-abs* $(z, \gamma)\ p'\ ($*rename-bound-var* $(y, \gamma)\ z\ C)$
using *app-is-wff*.*IH*(*2*)[*OF* ‹$(z, \gamma) \notin vars\ C$›] **by** *blast*

then have ¬ *in-scope-of-abs* (z, γ) p (*rename-bound-var* (y, γ) z $(B \cdot C)$) **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** $(z, \gamma) \notin$ *vars* A **and** $(z, \gamma) \neq (x, \alpha)$
  by *fastforce+*
then show *?case*
proof (*cases* $(y, \gamma) = (x, \alpha)$)
  case *True*
  then have *occurs-at* (y, γ) p (λz_γ. **S** $\{(y, \gamma) \rightarrowtail z_\gamma\}$ (*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** $\{(y, \gamma) \rightarrowtail z_\gamma\}$ (*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* ‹$(z, \gamma) \notin$ *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** ‹$(z, \gamma) \neq (x, \alpha)$› **by** *auto*
  then show *?thesis*
    using ∗ **and** ‹p = « # p'› **by** *simp*
qed
qed

lemma *is-free-for-in-rename-bound-var*:
assumes $A \in$ *wffs*$_\alpha$
and $z_\gamma \neq y_\gamma$
and $(z, \gamma) \notin$ *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* $\Rightarrow$ *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* $\Rightarrow$ *bool* **where**
  [*simp*]: *bool-from-V* = *inv bool-to-V*

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

[*simp*]: $\mathbf{T} = \textit{bool-to-V True}$

definition *bottom* :: V ($\mathbf{F}$) **where**
[*simp*]: $\mathbf{F} = \textit{bool-to-V False}$

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

definition *negation* :: $V \Rightarrow V$ ($\sim$ - [*141*] *141*) **where**
[*simp*]: $\sim p = \textit{bool-to-V}\ (\neg\ \textit{bool-from-V}\ p)$

definition *conjunction* :: $V \Rightarrow V \Rightarrow V$ (**infixr** $\wedge$ *136*) **where**
[*simp*]: $p \wedge q = \textit{bool-to-V}\ (\textit{bool-from-V}\ p \wedge \textit{bool-from-V}\ q)$

definition *disjunction* :: $V \Rightarrow V \Rightarrow V$ (**infixr** $\vee$ *131*) **where**
[*simp*]: $p \vee q = \sim (\sim p \wedge \sim q)$

definition *implication* :: $V \Rightarrow V \Rightarrow V$ (**infixr** $\supset$ *121*) **where**
[*simp*]: $p \supset q = \sim p \vee q$

definition *iff* :: $V \Rightarrow V \Rightarrow V$ (**infixl** $\equiv$ *150*) **where**
[*simp*]: $p \equiv q = (p \supset q) \wedge (q \supset p)$

lemma *boolean-algebra-simps* [*simp*]:
assumes $p \in \textit{elts}\ \mathbb{B}$ **and** $q \in \textit{elts}\ \mathbb{B}$ **and** $r \in \textit{elts}\ \mathbb{B}$
shows $\sim \sim p = p$
and $((\sim p) \equiv (\sim q)) = (p \equiv q)$
and $\sim (p \equiv q) = (p \equiv (\sim q))$
and $(p \vee \sim p) = \mathbf{T}$
and $(\sim p \vee p) = \mathbf{T}$
and $(p \equiv p) = \mathbf{T}$
and $(\sim p) \neq p$
and $p \neq (\sim p)$
and $(\mathbf{T} \equiv p) = p$
and $(p \equiv \mathbf{T}) = p$
and $(\mathbf{F} \equiv p) = (\sim p)$
and $(p \equiv \mathbf{F}) = (\sim p)$
and $(\mathbf{T} \supset p) = p$
and $(\mathbf{F} \supset p) = \mathbf{T}$
and $(p \supset \mathbf{T}) = \mathbf{T}$
and $(p \supset p) = \mathbf{T}$
and $(p \supset \mathbf{F}) = (\sim p)$
and $(p \supset \sim p) = (\sim p)$
and $(p \wedge \mathbf{T}) = p$
and $(\mathbf{T} \wedge p) = p$
and $(p \wedge \mathbf{F}) = \mathbf{F}$
and $(\mathbf{F} \wedge p) = \mathbf{F}$
and $(p \wedge p) = p$
and $(p \wedge (p \wedge q)) = (p \wedge q)$

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*: $\sim^{\mathcal{Q}} A \in pwffs$ **if** $A \in pwffs$
| *conj-pwff*: $A \wedge^{\mathcal{Q}} B \in pwffs$ **if** $A \in pwffs$ **and** $B \in pwffs$
| *disj-pwff*: $A \vee^{\mathcal{Q}} B \in pwffs$ **if** $A \in pwffs$ **and** $B \in pwffs$
| *imp-pwff*: $A \supset^{\mathcal{Q}} B \in pwffs$ **if** $A \in pwffs$ **and** $B \in pwffs$
| *eqv-pwff*: $A \equiv^{\mathcal{Q}} B \in pwffs$ **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 $T_o \neq \sim^{\mathcal{Q}} A$
and $T_o \neq A \wedge^{\mathcal{Q}} B$
and $T_o \neq A \vee^{\mathcal{Q}} B$
and $T_o \neq A \supset^{\mathcal{Q}} B$
and $T_o \neq A \equiv^{\mathcal{Q}} B$
and $F_o \neq p_o$
and $F_o \neq \sim^{\mathcal{Q}} A$
and $F_o \neq A \wedge^{\mathcal{Q}} B$
and $F_o \neq A \vee^{\mathcal{Q}} B$
and $F_o \neq A \supset^{\mathcal{Q}} B$
and $F_o \neq A \equiv^{\mathcal{Q}} B$
and $p_o \neq \sim^{\mathcal{Q}} A$
and $p_o \neq A \wedge^{\mathcal{Q}} B$
and $p_o \neq A \vee^{\mathcal{Q}} B$
and $p_o \neq A \supset^{\mathcal{Q}} B$
and $p_o \neq A \equiv^{\mathcal{Q}} B$
and $\sim^{\mathcal{Q}} A \neq B \wedge^{\mathcal{Q}} C$
and $\sim^{\mathcal{Q}} A \neq B \vee^{\mathcal{Q}} C$
and $\sim^{\mathcal{Q}} A \neq B \supset^{\mathcal{Q}} C$
and $\neg (B = F_o \wedge A = C) \Longrightarrow \sim^{\mathcal{Q}} A \neq B \equiv^{\mathcal{Q}} C$ — $\sim^{\mathcal{Q}} A$ is the same as $F_o \equiv^{\mathcal{Q}} A$
and $A \wedge^{\mathcal{Q}} B \neq C \vee^{\mathcal{Q}} D$
and $A \wedge^{\mathcal{Q}} B \neq C \supset^{\mathcal{Q}} D$
and $A \wedge^{\mathcal{Q}} B \neq C \equiv^{\mathcal{Q}} D$
and $A \vee^{\mathcal{Q}} B \neq C \supset^{\mathcal{Q}} D$
and $A \vee^{\mathcal{Q}} B \neq C \equiv^{\mathcal{Q}} D$
and $A \supset^{\mathcal{Q}} B \neq C \equiv^{\mathcal{Q}} D$
by *simp-all*

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

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

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

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

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

lemma *pwffs-from-eqv-pwff* [*elim*!]:
assumes $A \equiv^{\mathcal{Q}} B \in$ *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* $(\sim^{\mathcal{Q}} A)$ = *free-vars* A
and *conj-fv*: *free-vars* $(A \wedge^{\mathcal{Q}} B)$ = *free-vars* $A \cup$ *free-vars* B
and *disj-fv*: *free-vars* $(A \vee^{\mathcal{Q}} B)$ = *free-vars* $A \cup$ *free-vars* B

and *imp-fv*: *free-vars* $(A \supset^{\mathcal{Q}} B)$ = *free-vars* $A \cup$ *free-vars* B
and *eqv-fv*: *free-vars* $(A \equiv^{\mathcal{Q}} B)$ = *free-vars* $A \cup$ *free-vars* B
by *force+*

lemma *pwffs-free-vars-are-propositional*:
assumes $A \in$ *pwffs*
and $v \in$ *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 $A \in$ *pwffs*
and $v \in$ *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) $v \in$ *free-vars* C **and** $v \in$ *free-vars* D
| (b) $v \in$ *free-vars* C **and** $v \notin$ *free-vars* D
| (c) $v \notin$ *free-vars* C **and** $v \in$ *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) $v \in$ *free-vars* C **and** $v \in$ *free-vars* D
| (b) $v \in$ *free-vars* C **and** $v \notin$ *free-vars* D
| (c) $v \notin$ *free-vars* C **and** $v \in$ *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) $v \in$ *free-vars* C **and** $v \in$ *free-vars* D
| (b) $v \in$ *free-vars* C **and** $v \notin$ *free-vars* D
| (c) $v \notin$ *free-vars* C **and** $v \in$ *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*) $v \in$ *free-vars C* **and** $v \in$ *free-vars D*
| (*b*) $v \in$ *free-vars C* **and** $v \notin$ *free-vars D*
| (*c*) $v \notin$ *free-vars C* **and** $v \in$ *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* :: $(nat \Rightarrow V) \Rightarrow bool$ **where**
[*iff*]: *is-tv-assignment* $\varphi \longleftrightarrow (\forall p.\ \varphi\ p \in elts\ \mathbb{B})$

Denotation of a pwff:

definition *is-pwff-denotation-function* **where**
[*iff*]: *is-pwff-denotation-function* $\mathcal{V} \longleftrightarrow$
(
$\forall \varphi.$ *is-tv-assignment* $\varphi \longrightarrow$

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

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 $\mathcal{V}\ \varphi\ (\sim^{\mathcal{Q}}\ A) = \sim \mathcal{V}\ \varphi\ A$
using *assms(2,3)* **by** *auto*
then show *?case*
using *neg-pwff.IH* **by** *auto*
next
case (*conj-pwff A B*)
then have $\mathcal{V}\ \varphi\ (A \wedge^{\mathcal{Q}} B) = \mathcal{V}\ \varphi\ A \wedge \mathcal{V}\ \varphi\ B$
using *assms(2,3)* **by** *auto*
then show *?case*
using *conj-pwff.IH* **by** *auto*
next
case (*disj-pwff A B*)
then have $\mathcal{V}\ \varphi\ (A \vee^{\mathcal{Q}} B) = \mathcal{V}\ \varphi\ A \vee \mathcal{V}\ \varphi\ B$
using *assms(2,3)* **by** *auto*
then show *?case*
using *disj-pwff.IH* **by** *auto*
next
case (*imp-pwff A B*)
then have $\mathcal{V}\ \varphi\ (A \supset^{\mathcal{Q}} B) = \mathcal{V}\ \varphi\ A \supset \mathcal{V}\ \varphi\ B$
using *assms(2,3)* **by** *blast*
then show *?case*
using *imp-pwff.IH* **by** *auto*
next
case (*eqv-pwff A B*)
then have $\mathcal{V}\ \varphi\ (A \equiv^{\mathcal{Q}} B) = \mathcal{V}\ \varphi\ A \equiv \mathcal{V}\ \varphi\ B$
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 $b \in elts\ \mathbb{B}$
using *assms* **proof** *induction*
case $(\mathcal{V}_B\text{-}graph\text{-}neg\ A\ b_A)$
show *?case*
using $\mathcal{V}_B$*-graph-neg.IH*[*OF assms*(*2*)] **by** *force*
next
case $(\mathcal{V}_B\text{-}graph\text{-}conj\ A\ b_A\ B\ b_B)$
then show *?case*
using $\mathcal{V}_B$*-graph-conj.IH* **and** *assms*(*2*) **by** *force*
next
case $(\mathcal{V}_B\text{-}graph\text{-}disj\ A\ b_A\ B\ b_B)$
then show *?case*
using $\mathcal{V}_B$*-graph-disj.IH* **and** *assms*(*2*) **by** *force*
next
case $(\mathcal{V}_B\text{-}graph\text{-}imp\ A\ b_A\ B\ b_B)$
then show *?case*
using $\mathcal{V}_B$*-graph-imp.IH* **and** *assms*(*2*) **by** *force*
next
case $(\mathcal{V}_B\text{-}graph\text{-}eqv\ A\ b_A\ B\ b_B)$
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 $A \in pwffs$
and *is-tv-assignment* φ
and $\mathcal{V}_B\text{-}graph\ \varphi\ A\ b$ **and** $\mathcal{V}_B\text{-}graph\ \varphi\ A\ b'$
shows $b = b'$
using *assms*(*3*,*1*,*4*) **proof** (*induction arbitrary*: b')
case $\mathcal{V}_B\text{-}graph\text{-}T$
from ‹$\mathcal{V}_B\text{-}graph\ \varphi\ T_o\ b'$› **show** *?case*
by (*cases rule*: $\mathcal{V}_B$*-graph.cases*) *simp-all*
next
case $\mathcal{V}_B\text{-}graph\text{-}F$
from ‹$\mathcal{V}_B\text{-}graph\ \varphi\ F_o\ b'$› **show** *?case*
by (*cases rule*: $\mathcal{V}_B$*-graph.cases*) *simp-all*
next
case $(\mathcal{V}_B\text{-}graph\text{-}var\ p)$
from ‹$\mathcal{V}_B\text{-}graph\ \varphi\ (p_o)\ b'$› **show** *?case*
by (*cases rule*: $\mathcal{V}_B$*-graph.cases*) *simp-all*
next
case $(\mathcal{V}_B\text{-}graph\text{-}neg\ A\ b_A)$
with ‹$\mathcal{V}_B\text{-}graph\ \varphi\ ({\sim}^{\mathcal{Q}}\ A)\ b'$› **have** $\mathcal{V}_B\text{-}graph\ \varphi\ A\ ({\sim}\ b')$
proof (*cases rule*: $\mathcal{V}_B$*-graph.cases*)
case $(\mathcal{V}_B\text{-}graph\text{-}neg\ A'\ b_A)$
from ‹${\sim}^{\mathcal{Q}}\ A = {\sim}^{\mathcal{Q}}\ A'$› **have** $A = A'$
by *simp*
with ‹$\mathcal{V}_B\text{-}graph\ \varphi\ A'\ b_A$› **have** $\mathcal{V}_B\text{-}graph\ \varphi\ A\ b_A$
by *simp*

moreover have $b_A = \sim b'$
proof −
have $b_A \in$ *elts* $\mathbb{B}$
by (*fact* $\mathcal{V}_B$*-graph-denotation-is-truth-value*[*OF* $\mathcal{V}_B$*-graph-neg*(*3*) *assms*(*2*)])
moreover from ‹$b_A \in$ *elts* $\mathbb{B}$› **and** $\mathcal{V}_B$*-graph-neg*(*2*) **have** $\sim b' \in$ *elts* $\mathbb{B}$
by *fastforce*
ultimately show *?thesis*
using $\mathcal{V}_B$*-graph-neg*(*2*) **by** *fastforce*
qed
ultimately show *?thesis*
by *blast*
qed *simp-all*
moreover from $\mathcal{V}_B$*-graph-neg.prems*(*1*) **have** $A \in$ *pwffs*
by (*force elim*: *pwffs.cases*)
moreover have $b_A \in$ *elts* $\mathbb{B}$ **and** $b' \in$ *elts* $\mathbb{B}$ **and** $b_A = \sim b'$
proof −
show $b_A \in$ *elts* $\mathbb{B}$
by (*fact* $\mathcal{V}_B$*-graph-denotation-is-truth-value*[*OF* ‹$\mathcal{V}_B$*-graph* φ A b_A› *assms*(*2*)])
show $b' \in$ *elts* $\mathbb{B}$
by (*fact* $\mathcal{V}_B$*-graph-denotation-is-truth-value*[*OF* ‹$\mathcal{V}_B$*-graph* φ $(\sim^{\mathcal{Q}} A)$ b'› *assms*(*2*)])
show $b_A = \sim b'$
by (*fact* $\mathcal{V}_B$*-graph-neg*(*2*)[*OF* ‹$A \in$ *pwffs*› ‹$\mathcal{V}_B$*-graph* φ A $(\sim b')$›])
qed
ultimately show *?case*
by *force*
next
case ($\mathcal{V}_B$*-graph-conj* A b_A B b_B)
with ‹$\mathcal{V}_B$*-graph* φ $(A \wedge^{\mathcal{Q}} B)$ b'› **obtain** $b_A{}'$ **and** $b_B{}'$
where $b' = b_A{}' \wedge b_B{}'$ **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-conj-pwff*[*OF* $\mathcal{V}_B$*-graph-conj.prems*(*1*)] **by** *blast+*
ultimately show *?case*
using $\mathcal{V}_B$*-graph-conj.IH* **and** $\mathcal{V}_B$*-graph-conj.prems*(*2*) **by** *blast*
next
case ($\mathcal{V}_B$*-graph-disj* A b_A B b_B)
from ‹$\mathcal{V}_B$*-graph* φ $(A \vee^{\mathcal{Q}} B)$ b'› **obtain** $b_A{}'$ **and** $b_B{}'$
where $b' = b_A{}' \vee b_B{}'$ **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-disj-pwff*[*OF* $\mathcal{V}_B$*-graph-disj.prems*(*1*)] **by** *blast+*
ultimately show *?case*
using $\mathcal{V}_B$*-graph-disj.IH* **and** $\mathcal{V}_B$*-graph-disj.prems*(*2*) **by** *blast*
next
case ($\mathcal{V}_B$*-graph-imp* A b_A B b_B)
from ‹$\mathcal{V}_B$*-graph* φ $(A \supset^{\mathcal{Q}} B)$ b'› **obtain** $b_A{}'$ **and** $b_B{}'$
where $b' = b_A{}' \supset b_B{}'$ **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-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 $A \in \mathit{pwffs}$ **and** $B \in \mathit{pwffs}$
and *is-tv-assignment* φ
shows $\mathcal{V}_B\ \varphi\ (A \wedge^{\mathcal{Q}} B) = \mathcal{V}_B\ \varphi\ A \wedge \mathcal{V}_B\ \varphi\ B$
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* $\varphi\ (A \wedge^{\mathcal{Q}} B)\ (\mathcal{V}_B\ \varphi\ A \wedge \mathcal{V}_B\ \varphi\ B)$
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 $A \in \mathit{pwffs}$ **and** $B \in \mathit{pwffs}$
and *is-tv-assignment* φ
shows $\mathcal{V}_B\ \varphi\ (A \supset^{\mathcal{Q}} B) = \mathcal{V}_B\ \varphi\ A \supset \mathcal{V}_B\ \varphi\ B$
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* $\varphi\ (A \supset^{\mathcal{Q}} B)\ (\mathcal{V}_B\ \varphi\ A \supset \mathcal{V}_B\ \varphi\ B)$
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 $A \in \mathit{pwffs}$ **and** $B \in \mathit{pwffs}$
and *is-tv-assignment* φ
shows $\mathcal{V}_B\ \varphi\ (A \equiv^{\mathcal{Q}} B) = \mathcal{V}_B\ \varphi\ A \equiv \mathcal{V}_B\ \varphi\ B$
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* $\varphi\ (A \equiv^{\mathcal{Q}} B)\ (\mathcal{V}_B\ \varphi\ A \equiv \mathcal{V}_B\ \varphi\ B)$
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 $\mathbf{S}\ \{(p,\ o) \rightarrowtail A\}\ T_o = T_o$
  and $\mathbf{S}\ \{(p,\ o) \rightarrowtail A\}\ F_o = F_o$
  and $\mathbf{S}\ \{(p,\ o) \rightarrowtail A\}\ (p'_o) = (\mathit{if}\ p = p'\ \mathit{then}\ A\ \mathit{else}\ (p'_o))$
  and $\mathbf{S}\ \{(p,\ o) \rightarrowtail A\}\ ({\sim^{\mathcal{Q}}}\ B) = {\sim^{\mathcal{Q}}}\ (\mathbf{S}\ \{(p,\ o) \rightarrowtail A\}\ B)$
  and $\mathbf{S}\ \{(p,\ o) \rightarrowtail A\}\ (B \wedge^{\mathcal{Q}} C) = (\mathbf{S}\ \{(p,\ o) \rightarrowtail A\}\ B) \wedge^{\mathcal{Q}} (\mathbf{S}\ \{(p,\ o) \rightarrowtail A\}\ C)$
  and $\mathbf{S}\ \{(p,\ o) \rightarrowtail A\}\ (B \vee^{\mathcal{Q}} C) = (\mathbf{S}\ \{(p,\ o) \rightarrowtail A\}\ B) \vee^{\mathcal{Q}} (\mathbf{S}\ \{(p,\ o) \rightarrowtail A\}\ C)$
  and $\mathbf{S}\ \{(p,\ o) \rightarrowtail A\}\ (B \supset^{\mathcal{Q}} C) = (\mathbf{S}\ \{(p,\ o) \rightarrowtail A\}\ B) \supset^{\mathcal{Q}} (\mathbf{S}\ \{(p,\ o) \rightarrowtail A\}\ C)$
  and $\mathbf{S}\ \{(p,\ o) \rightarrowtail A\}\ (B \equiv^{\mathcal{Q}} C) = (\mathbf{S}\ \{(p,\ o) \rightarrowtail A\}\ B) \equiv^{\mathcal{Q}} (\mathbf{S}\ \{(p,\ o) \rightarrowtail A\}\ C)$
  by *simp-all*

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

lemma closed-pwff-substitution-free-vars:
  assumes A ∈ pwffs and B ∈ pwffs
  and free-vars A = {}
  and (p, o) ∈ free-vars B
  shows free-vars (S {(p, o) ↣ A} B) = free-vars B − {(p, o)} (is ‹free-vars (S ?ϑ B) = -›)
using assms(2,4) proof induction
  case (conj-pwff C D)
  have free-vars (S ?ϑ (C ∧^𝒬 D)) = free-vars ((S ?ϑ C) ∧^𝒬 (S ?ϑ D))
    by simp
  also have ... = free-vars (S ?ϑ C) ∪ free-vars (S ?ϑ D)
    by (fact conj-fv)
  finally have ∗: free-vars (S ?ϑ (C ∧^𝒬 D)) = free-vars (S ?ϑ C) ∪ free-vars (S ?ϑ D) .
  from conj-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 conj-pwff.IH show ?case
    using free-var-singleton-substitution-neutrality by cases auto
next
  case (disj-pwff C D)
  have free-vars (S ?ϑ (C ∨^𝒬 D)) = free-vars ((S ?ϑ C) ∨^𝒬 (S ?ϑ D))
    by simp
  also have ... = free-vars (S ?ϑ C) ∪ free-vars (S ?ϑ D)
    by (fact disj-fv)
  finally have ∗: free-vars (S ?ϑ (C ∨^𝒬 D)) = free-vars (S ?ϑ C) ∪ free-vars (S ?ϑ D) .
  from disj-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 disj-pwff.IH show ?case
    using free-var-singleton-substitution-neutrality by cases auto
next
  case (imp-pwff C D)
  have free-vars (S ?ϑ (C ⊃^𝒬 D)) = free-vars ((S ?ϑ C) ⊃^𝒬 (S ?ϑ D))
    by simp
  also have ... = free-vars (S ?ϑ C) ∪ free-vars (S ?ϑ D)
    by (fact imp-fv)
```

finally have $*$: *free-vars* (**S** *?ϑ* ($C \supset^{\mathcal{Q}} D$)) = *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* ($C \equiv^{\mathcal{Q}} D$)) = *free-vars* ((**S** *?ϑ* C) $\equiv^{\mathcal{Q}}$ (**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* ($C \equiv^{\mathcal{Q}} D$)) = *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. $\alpha = o$)

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 $A \in wffs_o$
using *assms* **and** *substitution-preserves-typing* **and** *tautology-is-wffo* **by** *blast*

lemma *implication-reflexivity-is-tautologous*:
assumes $A \in wffs_o$
shows *is-tautologous* ($A \supset^{\mathcal{Q}} A$)
proof −
let *?ϑ* = {($\mathfrak{x}$, o) ↣ A}
have *is-tautology* ($\mathfrak{x}_o \supset^{\mathcal{Q}} \mathfrak{x}_o$)
by (*fact propositional-implication-reflexivity-is-tautology*)
moreover have *is-pwff-substitution* *?ϑ*
using *assms* **by** *auto*
moreover have $A \supset^{\mathcal{Q}} A$ = **S** *?ϑ* ($\mathfrak{x}_o \supset^{\mathcal{Q}} \mathfrak{x}_o$)
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* $(A \supset^{\mathcal{Q}} (B \supset^{\mathcal{Q}} A))$
proof −
let $?\vartheta = \{(\mathfrak{x}, o) \rightarrowtail A, (\mathfrak{y}, o) \rightarrowtail B\}$
have *is-tautology* $(\mathfrak{x}_o \supset^{\mathcal{Q}} (\mathfrak{y}_o \supset^{\mathcal{Q}} \mathfrak{x}_o))$
by (*fact propositional-principle-of-simplification-is-tautology*)
moreover have *is-pwff-substitution* $?\vartheta$
using *assms* **by** *auto*
moreover have $A \supset^{\mathcal{Q}} (B \supset^{\mathcal{Q}} A) = \mathbf{S}\ ?\vartheta\ (\mathfrak{x}_o \supset^{\mathcal{Q}} (\mathfrak{y}_o \supset^{\mathcal{Q}} \mathfrak{x}_o))$
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* $((A \supset^{\mathcal{Q}} \sim^{\mathcal{Q}} B) \supset^{\mathcal{Q}} (B \supset^{\mathcal{Q}} \sim^{\mathcal{Q}} A))$
proof −
let $?\vartheta = \{(\mathfrak{x}, o) \rightarrowtail A, (\mathfrak{y}, o) \rightarrowtail B\}$
have *is-tautology* $((\mathfrak{x}_o \supset^{\mathcal{Q}} \sim^{\mathcal{Q}} \mathfrak{y}_o) \supset^{\mathcal{Q}} (\mathfrak{y}_o \supset^{\mathcal{Q}} \sim^{\mathcal{Q}} \mathfrak{x}_o))$
using $\mathcal{V}_B$*-simps* **by** (*safe*, *fold is-tv-assignment-def*, *simp only*:) *simp*
moreover have *is-pwff-substitution* $?\vartheta$
using *assms* **by** *auto*
moreover have $(A \supset^{\mathcal{Q}} \sim^{\mathcal{Q}} B) \supset^{\mathcal{Q}} (B \supset^{\mathcal{Q}} \sim^{\mathcal{Q}} A) = \mathbf{S}\ ?\vartheta\ ((\mathfrak{x}_o \supset^{\mathcal{Q}} \sim^{\mathcal{Q}} \mathfrak{y}_o) \supset^{\mathcal{Q}} (\mathfrak{y}_o \supset^{\mathcal{Q}} \sim^{\mathcal{Q}} \mathfrak{x}_o))$
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:
$\mathfrak{g}_{o\to o} \bullet T_o \wedge^{\mathcal{Q}} \mathfrak{g}_{o\to o} \bullet F_o \equiv^{\mathcal{Q}} \forall \mathfrak{x}_o.\ \mathfrak{g}_{o\to o} \bullet \mathfrak{x}_o \in axioms$

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

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* ⟷
  (
   $\exists \alpha\ A\ B.$
    $E = A =_\alpha B \wedge A \in \mathit{wffs}_\alpha \wedge B \in \mathit{wffs}_\alpha \wedge$ — 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* $\Rightarrow$ *nat* $\Rightarrow$ *bool* **where**
[*iff*]: *is-proof-step* $\mathcal{S}$ $i' \longleftrightarrow$
$\mathcal{S}$! $i' \in$ *axioms* $\vee$
($\exists p\ j\ k$. $\{j, k\} \subseteq \{0..<i'\} \wedge$ *is-rule-R-app* p ($\mathcal{S}$! i') ($\mathcal{S}$! j) ($\mathcal{S}$! k))

definition *is-proof* :: *form list* $\Rightarrow$ *bool* **where**
[*iff*]: *is-proof* $\mathcal{S} \longleftrightarrow$ ($\forall i' <$ *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* $\vee$
$\{j, k\} \subseteq \{0..<i'\} \wedge$ *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*) $\{j, k\} \subseteq \{0..<i'\} \wedge$ *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* $\vee$
($\{j, k\} \subseteq \{0..<i'\} \wedge$ *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 $\{j, k\} \subseteq \{0..<i'\} \wedge$ *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* $\mathcal{S}$
and $i' <$ *length* $(\mathcal{S} @ \mathcal{S}') -$ *length* $\mathcal{S}'$
shows *is-proof-step* $(\mathcal{S} @ \mathcal{S}')$ i'
using *assms* **and** *common-prefix-is-subproof*[**where** $\mathcal{S}_1 = []$] **by** *simp*

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

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

then show *?thesis* **..**
qed
with $*$ **show** *?thesis*
by *fastforce*
qed
with *assms*(*1*) **and** ‹*?i* $<$ *length* $\mathcal{S}'$› **show** *?thesis*
by *simp*
qed

lemma *proof-but-last-is-proof*:
assumes *is-proof* ($\mathcal{S}$ @ [A])
shows *is-proof* $\mathcal{S}$
using *assms* **and** *common-prefix-is-subproof*[**where** $\mathcal{S}_1 = [A]$ **and** $\mathcal{S}_2 = []$] **by** *simp*

lemma *proof-prefix-is-proof*:
assumes *is-proof* ($\mathcal{S}_1$ @ $\mathcal{S}_2$)
shows *is-proof* $\mathcal{S}_1$
using *assms* **and** *proof-but-last-is-proof*
by (*induction* $\mathcal{S}_2$ *arbitrary*: $\mathcal{S}_1$ *rule*: *rev-induct*) (*simp*, *metis append.assoc*)

lemma *single-axiom-is-proof*:
assumes $A \in$ *axioms*
shows *is-proof* [A]
using *assms* **by** *fastforce*

lemma *proofs-concatenation-is-proof*:
assumes *is-proof* $\mathcal{S}_1$ **and** *is-proof* $\mathcal{S}_2$
shows *is-proof* ($\mathcal{S}_1$ @ $\mathcal{S}_2$)
proof $-$
from *assms*(*1*) **have** $\forall\, i' <$ *length* $\mathcal{S}_1$. *is-proof-step* ($\mathcal{S}_1$ @ $\mathcal{S}_2$) i'
using *added-suffix-proof-preservation* **by** *auto*
moreover from *assms*(*2*) **have** $\forall\, i' \in$ {*length* $\mathcal{S}_1$..<*length* ($\mathcal{S}_1$ @ $\mathcal{S}_2$)}. *is-proof-step* ($\mathcal{S}_1$ @ $\mathcal{S}_2$) 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* $\mathcal{S}$ **and** $A \in$ *lset* $\mathcal{S}$
shows $A \in wffs_o$
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* $\mathcal{S}$) (*simp*, *metis* (*full-types*) *in-mono in-set-conv-nth*)

lemma *axiom-prepended-to-proof-is-proof*:
assumes *is-proof* $\mathcal{S}$
and $A \in$ *axioms*
shows *is-proof* ([A] @ $\mathcal{S}$)
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 ∈ wffs_o
  using assms and elem-of-proof-is-wffo by blast

lemma proof-form-is-theorem:
  assumes is-proof 𝒮 and 𝒮 ≠ []
  and i' < length 𝒮
  shows is-theorem (𝒮 ! i')
proof -
  let ?𝒮₁ = take (Suc i') 𝒮
  from assms(1) obtain 𝒮₂ where is-proof (?𝒮₁ @ 𝒮₂)
    by (metis append-take-drop-id)
  then have is-proof ?𝒮₁
    by (fact proof-prefix-is-proof)
  moreover from assms(3) have last ?𝒮₁ = 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 𝒮_C and 𝒮_E where
    is-proof 𝒮_C and 𝒮_C ≠ [] and last 𝒮_C = C and
    is-proof 𝒮_E and 𝒮_E ≠ [] and last 𝒮_E = E
    using dv-rule-R.IH by fastforce
  let ?i_C = length 𝒮_C − 1 and ?i_E = length 𝒮_C + length 𝒮_E − 1 and ?i_D = length 𝒮_C + length
𝒮_E
  let ?𝒮 = 𝒮_C @ 𝒮_E @ [D]
  from ‹𝒮_C ≠ []› have ?i_C < length (𝒮_C @ 𝒮_E) and ?i_E < length (𝒮_C @ 𝒮_E)
    using linorder-not-le by fastforce+
  moreover have (𝒮_C @ 𝒮_E) ! ?i_C = C and (𝒮_C @ 𝒮_E) ! ?i_E = E
    using ‹𝒮_C ≠ []› and ‹last 𝒮_C = C›
    by
      (
        simp add: last-conv-nth nth-append,
        metis ‹last 𝒮_E = E› ‹𝒮_E ≠ []› append-is-Nil-conv last-appendR last-conv-nth length-append
      )
  with ‹is-rule-R-app p D C E› have is-rule-R-app p D ((𝒮_C @ 𝒮_E) ! ?i_C) ((𝒮_C @ 𝒮_E) ! ?i_E)
```

using ‹$(\mathcal{S}_C$ @ $\mathcal{S}_E)$! *?*$i_C = C$› **by** *fastforce*
moreover from ‹*is-proof* $\mathcal{S}_C$› **and** ‹*is-proof* $\mathcal{S}_E$› **have** *is-proof* $(\mathcal{S}_C$ @ $\mathcal{S}_E)$
by (*fact proofs-concatenation-is-proof*)
ultimately have *is-proof* $((\mathcal{S}_C$ @ $\mathcal{S}_E)$ @ $[D])$
using *rule-R-app-appended-to-proof-is-proof* **by** *presburger*
with ‹$\mathcal{S}_C \neq []$› **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** $\mathcal{S}$ **where** *is-proof* $\mathcal{S}$ **and** $\mathcal{S} \neq []$ **and** *last* $\mathcal{S} = A$
by *fastforce*
then show *?thesis*
proof (*induction length* $\mathcal{S}$ *arbitrary*: $\mathcal{S}$ A *rule*: *less-induct*)
case *less*
let *?*i' = *length* $\mathcal{S}$ − *1*
from ‹$\mathcal{S} \neq []$› **and** ‹*last* $\mathcal{S} = A$› **have** $\mathcal{S}$! *?*$i' = A$
by (*simp add*: *last-conv-nth*)
from ‹*is-proof* $\mathcal{S}$› **and** ‹$\mathcal{S} \neq []$› **and** ‹*last* $\mathcal{S} = A$› **have** *is-proof-step* $\mathcal{S}$ *?*i'
using *added-suffix-proof-preservation*[**where** $\mathcal{S}' = []$] **by** *simp*
then consider
(*axiom*) $\mathcal{S}$! *?*$i' \in$ *axioms*
| (*rule-R*) $\exists\, p\ j\ k.\ \{j, k\} \subseteq \{0..<?i'\} \wedge$ *is-rule-R-app* p $(\mathcal{S}$! *?*$i')$ $(\mathcal{S}$! $j)$ $(\mathcal{S}$! $k)$
by *fastforce*
then show *?case*
proof *cases*
case *axiom*
with ‹$\mathcal{S}$! *?*$i' = A$› **show** *?thesis*
by (*fastforce intro*: *dv-axiom*)
next
case *rule-R*
then obtain p **and** j **and** k
where $\{j, k\} \subseteq \{0..<?i'\}$ **and** *is-rule-R-app* p $(\mathcal{S}$! *?*$i')$ $(\mathcal{S}$! $j)$ $(\mathcal{S}$! $k)$
by *force*
let *?*$\mathcal{S}_j$ = *take* (*Suc* j) $\mathcal{S}$
let *?*$\mathcal{S}_k$ = *take* (*Suc* k) $\mathcal{S}$
obtain $\mathcal{S}_j{}'$ **and** $\mathcal{S}_k{}'$ **where** $\mathcal{S}$ = *?*$\mathcal{S}_j$ @ $\mathcal{S}_j{}'$ **and** $\mathcal{S}$ = *?*$\mathcal{S}_k$ @ $\mathcal{S}_k{}'$
by (*metis append-take-drop-id*)
with ‹*is-proof* $\mathcal{S}$› **have** *is-proof* (*?*$\mathcal{S}_j$ @ $\mathcal{S}_j{}'$) **and** *is-proof* (*?*$\mathcal{S}_k$ @ $\mathcal{S}_k{}'$)
by (*simp-all only*:)
moreover
from ‹$\mathcal{S}$ = *?*$\mathcal{S}_j$ @ $\mathcal{S}_j{}'$› **and** ‹$\mathcal{S}$ = *?*$\mathcal{S}_k$ @ $\mathcal{S}_k{}'$› **and** ‹*last* $\mathcal{S} = A$› **and** ‹$\{j, k\} \subseteq \{0..<$*length* $\mathcal{S}$ −
1$\}$›
have *last* $\mathcal{S}_j{}' = A$ **and** *last* $\mathcal{S}_k{}' = A$
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 ‹$\mathcal{S} \neq []$› **have** $?\mathcal{S}_j \neq []$ **and** $?\mathcal{S}_k \neq []$
  by *simp-all*
ultimately have *is-proof-of* $?\mathcal{S}_j$ (*last* $?\mathcal{S}_j$) **and** *is-proof-of* $?\mathcal{S}_k$ (*last* $?\mathcal{S}_k$)
  using *proof-prefix-is-proof-of-last* [**where** $\mathcal{S} = ?\mathcal{S}_j$ **and** $\mathcal{S}' = \mathcal{S}_j{}'$]
  and *proof-prefix-is-proof-of-last* [**where** $\mathcal{S} = ?\mathcal{S}_k$ **and** $\mathcal{S}' = \mathcal{S}_k{}'$]
  by *fastforce+*
moreover from ‹*last* $\mathcal{S}_j{}' = A$› **and** ‹*last* $\mathcal{S}_k{}' = A$›
have *length* $?\mathcal{S}_j <$ *length* $\mathcal{S}$ **and** *length* $?\mathcal{S}_k <$ *length* $\mathcal{S}$
  using ‹$\{j, k\} \subseteq \{0..<length\ \mathcal{S} - 1\}$› **by** *force+*
moreover from *calculation*(*3*,*4*) **have** *last* $?\mathcal{S}_j = \mathcal{S}\ !\ j$ **and** *last* $?\mathcal{S}_k = \mathcal{S}\ !\ k$
  by (*metis Suc-lessD last-snoc linorder-not-le nat-neq-iff take-Suc-conv-app-nth take-all-iff*)+
ultimately have *is-derivable* ($\mathcal{S}$! j) **and** *is-derivable* ($\mathcal{S}$! k)
  using ‹$?\mathcal{S}_j \neq []$› **and** ‹$?\mathcal{S}_k \neq []$› **and** *less*(*1*) **by** *blast+*
with ‹*is-rule-R-app* p ($\mathcal{S}$! $?i'$) ($\mathcal{S}$! j) ($\mathcal{S}$! k)› **and** ‹$\mathcal{S}$! $?i' = A$› **show** *?thesis*
  by (*blast intro*: *dv-rule-R*)
qed
qed
qed

theorem *theoremhood-derivability-equivalence*:
  shows *is-theorem* $A \longleftrightarrow$ *is-derivable* A
  using *derivable-form-is-theorem* **and** *theorem-is-derivable-form* **by** *blast*

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

lemma *equality-reflexivity*:
  assumes $A \in wffs_\alpha$
  shows *is-theorem* ($A =_\alpha A$) (**is** *is-theorem* $?A_2$)
proof −
  let $?A_1 = (\lambda\mathfrak{x}_\alpha.\ \mathfrak{x}_\alpha) \cdot A =_\alpha A$
  let $?\mathcal{S} = [?A_1,\ ?A_2]$
  — (.1) Axiom 4.2
  have *is-proof-step* $?\mathcal{S}$ *0*
  proof −
    from *assms* **have** $?A_1 \in axioms$
      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 (λ𝔵α. 𝔵α) • 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 (λ𝔵α. 𝔵α) • 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 = (λ𝔵α. 𝔵α) • 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** $?C = ?A_1$ **and** $?E = ?A_1$ **and** $?D = ?A_2$
have *is-rule-R-app ?p ?D ?C ?E*
proof −
have $(\lambda\mathfrak{x}_\alpha.\ \mathfrak{x}_\alpha) \bullet A \preceq_{?p} ?C$
by *force*
moreover have $?C\langle\!|?p \leftarrow A|\!\rangle \rhd ?D$
by *force*
moreover from ‹$A \in wffs_\alpha$› **have** $?D \in wffs_o$
by (*intro equality-wff*)
moreover from ‹$A \in wffs_\alpha$› **have** $(\lambda\mathfrak{x}_\alpha.\ \mathfrak{x}_\alpha) \bullet A \in wffs_\alpha$
by (*meson wffs-of-type-simps*)
ultimately show *?thesis*
using ‹$A \in wffs_\alpha$› **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*]: $capture\text{-}exposed\text{-}vars\text{-}at\ p\ A\ \mathcal{X} =$
$\{(x, \beta) \mid x\ \beta\ p'\ E.\ strict\text{-}prefix\ p'\ p \land \lambda x_\beta.\ E \preceq_{p'} A \land (x, \beta) \in free\text{-}vars\ \mathcal{X}\}$

lemma *capture-exposed-vars-at-alt-def*:
assumes $p \in positions\ A$
shows $capture\text{-}exposed\text{-}vars\text{-}at\ p\ A\ \mathcal{X} = binders\text{-}at\ A\ p \cap free\text{-}vars\ \mathcal{X}$
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*]: $rule\text{-}R'\text{-}side\text{-}condition\ \mathcal{H}\ p\ D\ C\ E \longleftrightarrow$
$capture\text{-}exposed\text{-}vars\text{-}at\ p\ C\ E \cap capture\text{-}exposed\text{-}vars\text{-}at\ p\ C\ \mathcal{H} = \{\}$

lemma *rule-R′-side-condition-alt-def*:
fixes $\mathcal{H}$ :: *form set*
assumes $C \in wffs_\alpha$
shows
$rule\text{-}R'\text{-}side\text{-}condition\ \mathcal{H}\ p\ D\ C\ (A =_\alpha B)$
$\longleftrightarrow$
(
$\nexists x\ \beta\ E\ p'.$
$strict\text{-}prefix\ p'\ p\ \land$
$\lambda x_\beta.\ E \preceq_{p'} C\ \land$

$(x, \beta) \in$ *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
$\ldots$
$=$
$\{(x, \beta) \mid x\ \beta\ p'\ E.$ *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* $\Rightarrow$ *position* $\Rightarrow$ *form* $\Rightarrow$ *form* $\Rightarrow$ *form* $\Rightarrow$ *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
$A \preceq_p C \wedge D \in wffs_o \wedge$
$C⦇p \leftarrow B⦈ \rhd D \wedge$
(
$\nexists x\ \beta\ E\ p'.$
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 $?\mathcal{S} = \mathcal{S}_2\ @\ \mathcal{S}_2{}'$
from *assms*(2) **have** $?\mathcal{S} ! i' = (\mathcal{S}_2\ @\ \mathcal{S}_2{}'') ! i'$
by (*simp add*: *nth-append*)
moreover from *assms*(2) **have** $i' < length\ ?\mathcal{S}$
by *simp*
ultimately obtain *p* **and** *j* **and** *k* **where**
$?\mathcal{S} ! i' \in \mathcal{H} \vee$
$?\mathcal{S} ! i' \in lset\ \mathcal{S}_1 \vee$
$\{j, k\} \subseteq \{0..<i'\} \wedge$ *is-rule-R′-app* $\mathcal{H}\ p\ (?\mathcal{S} ! i')\ (?\mathcal{S} ! j)\ (?\mathcal{S} ! k)$
using *assms*(1) **unfolding** *is-hyp-proof-def* **and** *is-hyp-proof-step-def* **by** *meson*
then consider

  (*hyp*) $?\mathcal{S}$! $i' \in \mathcal{H}$
  | (*seq*) $?\mathcal{S}$! $i' \in$ *lset* $\mathcal{S}_1$
  | (*rule-R'*) $\{j, k\} \subseteq \{0..<i'\} \wedge$ *is-rule-R'-app* $\mathcal{H}$ *p* ($?\mathcal{S}$! i') ($?\mathcal{S}$! j) ($?\mathcal{S}$! k)
  by *blast*
 then have
  $(\mathcal{S}_2 \ @ \ \mathcal{S}_2'')$! $i' \in \mathcal{H} \vee$
  $(\mathcal{S}_2 \ @ \ \mathcal{S}_2'')$! $i' \in$ *lset* $\mathcal{S}_1 \vee$
  $(\{j, k\} \subseteq \{0..<i'\} \wedge$ *is-rule-R'-app* $\mathcal{H}$ *p* $((\mathcal{S}_2 \ @ \ \mathcal{S}_2'')$! $i')$ $((\mathcal{S}_2 \ @ \ \mathcal{S}_2'')$! $j)$ $((\mathcal{S}_2 \ @ \ \mathcal{S}_2'')$! $k))$
 proof *cases*
  case *hyp*
  with *assms*(*2*) **have** $(\mathcal{S}_2 \ @ \ \mathcal{S}_2'')$! $i' \in \mathcal{H}$
   by (*simp add*: *nth-append*)
  then show *?thesis* **..**
 next
  case *seq*
  with *assms*(*2*) **have** $(\mathcal{S}_2 \ @ \ \mathcal{S}_2'')$! $i' \in$ *lset* $\mathcal{S}_1$
   by (*simp add*: *nth-append*)
  then show *?thesis*
   by (*intro disjI1 disjI2*)
 next
  case *rule-R'*
  with *assms*(*2*) **have** $?\mathcal{S}$! $j = (\mathcal{S}_2 \ @ \ \mathcal{S}_2'')$! j **and** $?\mathcal{S}$! $k = (\mathcal{S}_2 \ @ \ \mathcal{S}_2'')$! k
   by (*simp-all add*: *nth-append*)
  with *assms*(*2*) **and** *rule-R'* **have**
   $\{j, k\} \subseteq \{0..<i'\} \wedge$ *is-rule-R'-app* $\mathcal{H}$ *p* $((\mathcal{S}_2 \ @ \ \mathcal{S}_2'')$! $i')$ $((\mathcal{S}_2 \ @ \ \mathcal{S}_2'')$! $j)$ $((\mathcal{S}_2 \ @ \ \mathcal{S}_2'')$! $k)$
   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* $\mathcal{H}$ $\mathcal{S}_1$ $\mathcal{S}_2$
 shows *is-hyp-proof* $\mathcal{H}$ $(\mathcal{S}_1 \ @ \ \mathcal{S}_1')$ $\mathcal{S}_2$
 using *assms* **by** *auto*

lemma *added-suffix-hyp-proof-preservation*:
 assumes *is-hyp-proof* $\mathcal{H}$ $\mathcal{S}_1$ $\mathcal{S}_2$
 and $i' <$ *length* $(\mathcal{S}_2 \ @ \ \mathcal{S}_2') -$ *length* $\mathcal{S}_2'$
 shows *is-hyp-proof-step* $\mathcal{H}$ $\mathcal{S}_1$ $(\mathcal{S}_2 \ @ \ \mathcal{S}_2')$ i'
 using *assms* **and** *common-prefix-is-hyp-subproof-from*[**where** $\mathcal{S}_2' = []$] **by** *auto*

lemma *appended-hyp-proof-step-is-hyp-proof*:
 assumes *is-hyp-proof* $\mathcal{H}$ $\mathcal{S}_1$ $\mathcal{S}_2$
 and *is-hyp-proof-step* $\mathcal{H}$ $\mathcal{S}_1$ $(\mathcal{S}_2 \ @ \ [A])$ (*length* $(\mathcal{S}_2 \ @ \ [A]) - 1$)
 shows *is-hyp-proof* $\mathcal{H}$ $\mathcal{S}_1$ $(\mathcal{S}_2 \ @ \ [A])$
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 $\mathit{is\text{-}hyp\text{-}proof\text{-}step}\ \mathcal{H}\ \mathcal{S}_1\ (\mathcal{S}_2\ @\ [A])\ i'$
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 $\mathit{is\text{-}hyp\text{-}proof}\ \mathcal{H}\ \mathcal{S}_1\ \mathcal{S}_2{}'$
and $i' \in \{\mathit{length}\ \mathcal{S}_2..<\mathit{length}\ (\mathcal{S}_2\ @\ \mathcal{S}_2{}')\}$
shows $\mathit{is\text{-}hyp\text{-}proof\text{-}step}\ \mathcal{H}\ \mathcal{S}_1\ (\mathcal{S}_2\ @\ \mathcal{S}_2{}')\ i'$
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 $\mathit{is\text{-}hyp\text{-}proof\text{-}step}\ \mathcal{H}\ \mathcal{S}_1\ ?\mathcal{S}\ i' = \mathit{is\text{-}hyp\text{-}proof\text{-}step}\ \mathcal{H}\ \mathcal{S}_1\ \mathcal{S}_2{}'\ ?i$
proof $-$
from *assms*(*1*) **and** $\langle ?i < \mathit{length}\ \mathcal{S}_2{}' \rangle$ **obtain** j **and** k **and** p **where**
$\mathcal{S}_2{}'\ !\ ?i \in \mathcal{H}\ \vee$
$\mathcal{S}_2{}'\ !\ ?i \in \mathit{lset}\ \mathcal{S}_1\ \vee$
$(\{j, k\} \subseteq \{0..<?i\} \wedge \mathit{is\text{-}rule\text{-}R'\text{-}app}\ \mathcal{H}\ p\ (\mathcal{S}_2{}'\ !\ ?i)\ (\mathcal{S}_2{}'\ !\ j)\ (\mathcal{S}_2{}'\ !\ k))$
unfolding *is-hyp-proof-def* **and** *is-hyp-proof-step-def* **by** *meson*
then consider
$(\mathit{hyp})\ \mathcal{S}_2{}'\ !\ ?i \in \mathcal{H}$
$\mid (\mathit{seq})\ \mathcal{S}_2{}'\ !\ ?i \in \mathit{lset}\ \mathcal{S}_1$
$\mid (\mathit{rule\text{-}R'})\ \{j, k\} \subseteq \{0..<?i\} \wedge \mathit{is\text{-}rule\text{-}R'\text{-}app}\ \mathcal{H}\ p\ (\mathcal{S}_2{}'\ !\ ?i)\ (\mathcal{S}_2{}'\ !\ j)\ (\mathcal{S}_2{}'\ !\ k)$
by *blast*
then have
$?\mathcal{S}\ !\ i' \in \mathcal{H}\ \vee$
$?\mathcal{S}\ !\ i' \in \mathit{lset}\ \mathcal{S}_1\ \vee$
$(\{j + \mathit{length}\ \mathcal{S}_2,\ k + \mathit{length}\ \mathcal{S}_2\} \subseteq \{0..<i'\} \wedge$
$\mathit{is\text{-}rule\text{-}R'\text{-}app}\ \mathcal{H}\ p\ (?\mathcal{S}\ !\ i')\ (?\mathcal{S}\ !\ (j + \mathit{length}\ \mathcal{S}_2))\ (?\mathcal{S}\ !\ (k + \mathit{length}\ \mathcal{S}_2)))$
proof *cases*
case *hyp*
with $\langle ?\mathcal{S}\ !\ i' = \mathcal{S}_2{}'\ !\ ?i \rangle$ **have** $?\mathcal{S}\ !\ i' \in \mathcal{H}$
by (*simp only*:)
then show *?thesis* **..**
next
case *seq*

with ‹$?\mathcal{S}\ !\ i' = \mathcal{S}_2{}'\ !\ ?i$› **have** $?\mathcal{S}\ !\ i' \in$ *lset* $\mathcal{S}_1$
by (*simp only*:)
then show *?thesis*
by (*intro disjI1 disjI2*)
next
case *rule-R'*
with *assms*(*2*) **have** $?\mathcal{S}\ !\ (j +$ *length* $\mathcal{S}_2) = \mathcal{S}_2{}'\ !\ j$ **and** $?\mathcal{S}\ !\ (k +$ *length* $\mathcal{S}_2) = \mathcal{S}_2{}'\ !\ k$
by (*simp-all add*: *nth-append*)
with ‹$?\mathcal{S}\ !\ i' = \mathcal{S}_2{}'\ !\ ?i$› **and** *rule-R'* **have**
$\{j +$ *length* $\mathcal{S}_2,\ k +$ *length* $\mathcal{S}_2\} \subseteq \{0..<i'\} \wedge$
is-rule-R'-app $\mathcal{H}$ p $(?\mathcal{S}\ !\ i')$ $(?\mathcal{S}\ !\ (j +$ *length* $\mathcal{S}_2))$ $(?\mathcal{S}\ !\ (k +$ *length* $\mathcal{S}_2))$
by *auto*
then show *?thesis*
by (*intro disjI2*)
qed
with *assms*(*1*) **and** ‹$?i <$ *length* $\mathcal{S}_2{}'$› **show** *?thesis*
unfolding *is-hyp-proof-def* **and** *is-hyp-proof-step-def* **by** *meson*
qed
with *assms*(*1*) **and** ‹$?i <$ *length* $\mathcal{S}_2{}'$› **show** *?thesis*
by *simp*
qed

lemma *hyp-proof-but-last-is-hyp-proof*:
assumes *is-hyp-proof* $\mathcal{H}$ $\mathcal{S}_1$ $(\mathcal{S}_2\ @\ [A])$
shows *is-hyp-proof* $\mathcal{H}$ $\mathcal{S}_1$ $\mathcal{S}_2$
using *assms* **and** *common-prefix-is-hyp-subproof-from*[**where** $\mathcal{S}_2{}' = [A]$ **and** $\mathcal{S}_2{}'' = []$]
by *simp*

lemma *hyp-proof-prefix-is-hyp-proof*:
assumes *is-hyp-proof* $\mathcal{H}$ $\mathcal{S}_1$ $(\mathcal{S}_2\ @\ \mathcal{S}_2{}')$
shows *is-hyp-proof* $\mathcal{H}$ $\mathcal{S}_1$ $\mathcal{S}_2$
using *assms* **and** *hyp-proof-but-last-is-hyp-proof*
by (*induction* $\mathcal{S}_2{}'$ *arbitrary*: $\mathcal{S}_2$ *rule*: *rev-induct*) (*simp*, *metis append.assoc*)

lemma *single-hyp-is-hyp-proof*:
assumes $A \in \mathcal{H}$
shows *is-hyp-proof* $\mathcal{H}$ $\mathcal{S}_1$ $[A]$
using *assms* **by** *fastforce*

lemma *single-thm-is-hyp-proof*:
assumes $A \in$ *lset* $\mathcal{S}_1$
shows *is-hyp-proof* $\mathcal{H}$ $\mathcal{S}_1$ $[A]$
using *assms* **by** *fastforce*

lemma *hyp-proofs-from-concatenation-is-hyp-proof*:
assumes *is-hyp-proof* $\mathcal{H}$ $\mathcal{S}_1$ $\mathcal{S}_1{}'$ **and** *is-hyp-proof* $\mathcal{H}$ $\mathcal{S}_2$ $\mathcal{S}_2{}'$
shows *is-hyp-proof* $\mathcal{H}$ $(\mathcal{S}_1\ @\ \mathcal{S}_2)$ $(\mathcal{S}_1{}'\ @\ \mathcal{S}_2{}')$
proof (*standard*, *intro allI impI*)
let $?\mathcal{S} = \mathcal{S}_1\ @\ \mathcal{S}_2$ **and** $?\mathcal{S}' = \mathcal{S}_1{}'\ @\ \mathcal{S}_2{}'$

fix i'
assume $i' <$ *length* $?\mathcal{S}'$
then consider (a) $i' <$ *length* $\mathcal{S}_1{}'$ | (b) $i' \in \{$*length* $\mathcal{S}_1{}'..<$*length* $?\mathcal{S}'\}$
by *fastforce*
then show *is-hyp-proof-step* $\mathcal{H}$ $?\mathcal{S}$ $?\mathcal{S}'$ i'
proof *cases*
case *a*
from ‹*is-hyp-proof* $\mathcal{H}$ $\mathcal{S}_1$ $\mathcal{S}_1{}'$› **have** *is-hyp-proof* $\mathcal{H}$ $(\mathcal{S}_1$ @ $\mathcal{S}_2)$ $\mathcal{S}_1{}'$
by *auto*
with *assms*(*1*) **and** *a* **show** *?thesis*
using *added-suffix-hyp-proof-preservation*[**where** $\mathcal{S}_1 = \mathcal{S}_1$ @ $\mathcal{S}_2$] **by** *auto*
next
case *b*
from *assms*(*2*) **have** *is-hyp-proof* $\mathcal{H}$ $(\mathcal{S}_1$ @ $\mathcal{S}_2)$ $\mathcal{S}_2{}'$
by *auto*
with *b* **show** *?thesis*
using *added-prefix-hyp-proof-preservation*[**where** $\mathcal{S}_1 = \mathcal{S}_1$ @ $\mathcal{S}_2$] **by** *auto*
qed
qed

lemma *elem-of-hyp-proof-is-wffo*:
assumes *is-hyps* $\mathcal{H}$
and *lset* $\mathcal{S}_1 \subseteq$ *wffs*$_o$
and *is-hyp-proof* $\mathcal{H}$ $\mathcal{S}_1$ $\mathcal{S}_2$
and $A \in$ *lset* $\mathcal{S}_2$
shows $A \in$ *wffs*$_o$
using *assms* **proof** (*induction* $\mathcal{S}_2$ *rule*: *rev-induct*)
case *Nil*
then show *?case*
by *simp*
next
case (*snoc* A' $\mathcal{S}_2$)
from ‹*is-hyp-proof* $\mathcal{H}$ $\mathcal{S}_1$ $(\mathcal{S}_2$ @ $[A'])$› **have** *is-hyp-proof* $\mathcal{H}$ $\mathcal{S}_1$ $\mathcal{S}_2$
using *hyp-proof-prefix-is-hyp-proof*[**where** $\mathcal{S}_2{}' = [A']$] **by** *presburger*
then show *?case*
proof (*cases* $A \in$ *lset* $\mathcal{S}_2$)
case *True*
with *snoc.prems*(*1*,*2*) **and** ‹*is-hyp-proof* $\mathcal{H}$ $\mathcal{S}_1$ $\mathcal{S}_2$› **show** *?thesis*
by (*fact snoc.IH*)
next
case *False*
with *snoc.prems*(*4*) **have** $A' = A$
by *simp*
with *snoc.prems*(*3*) **have**
$(\mathcal{S}_2$ @ $[A])$! $i' \in \mathcal{H}$ $\vee$
$(\mathcal{S}_2$ @ $[A])$! $i' \in$ *lset* $\mathcal{S}_1$ $\vee$
$(\mathcal{S}_2$ @ $[A])$! $i' \in$ *wffs*$_o$ **if** $i' \in \{0..<$*length* $(\mathcal{S}_2$ @ $[A])\}$ **for** i'
using *that* **by** *auto*
then have $A \in$ *wffs*$_o$ $\vee$ $A \in \mathcal{H}$ $\vee$ $A \in$ *lset* $\mathcal{S}_1$ $\vee$ *length* $\mathcal{S}_2 \notin \{0..<$*Suc* (*length* $\mathcal{S}_2)\}$

by (*metis* (*no-types*) *length-append-singleton nth-append-length*)
with *assms*(*1*) **and** ‹*lset* $\mathcal{S}_1 \subseteq$ *wffs*$_o$› **show** *?thesis*
using *atLeast0-lessThan-Suc* **by** *blast*
qed
qed

lemma *hyp-prepended-to-hyp-proof-is-hyp-proof*:
assumes *is-hyp-proof* $\mathcal{H}$ $\mathcal{S}_1$ $\mathcal{S}_2$
and $A \in \mathcal{H}$
shows *is-hyp-proof* $\mathcal{H}$ $\mathcal{S}_1$ ([A] @ $\mathcal{S}_2$)
using
hyp-proofs-from-concatenation-is-hyp-proof
[
OF single-hyp-is-hyp-proof[*OF assms*(*2*)] *assms*(*1*),
where $\mathcal{S}_1$ = []
]
by *simp*

lemma *hyp-appended-to-hyp-proof-is-hyp-proof*:
assumes *is-hyp-proof* $\mathcal{H}$ $\mathcal{S}_1$ $\mathcal{S}_2$
and $A \in \mathcal{H}$
shows *is-hyp-proof* $\mathcal{H}$ $\mathcal{S}_1$ ($\mathcal{S}_2$ @ [A])
using
hyp-proofs-from-concatenation-is-hyp-proof
[
OF assms(*1*) *single-hyp-is-hyp-proof*[*OF assms*(*2*)],
where $\mathcal{S}_2$ = []
]
by *simp*

lemma *dropped-duplicated-thm-in-hyp-proof-is-hyp-proof*:
assumes *is-hyp-proof* $\mathcal{H}$ (A # $\mathcal{S}_1$) $\mathcal{S}_2$
and $A \in$ *lset* $\mathcal{S}_1$
shows *is-hyp-proof* $\mathcal{H}$ $\mathcal{S}_1$ $\mathcal{S}_2$
using *assms* **by** *auto*

lemma *thm-prepended-to-hyp-proof-is-hyp-proof*:
assumes *is-hyp-proof* $\mathcal{H}$ $\mathcal{S}_1$ $\mathcal{S}_2$
and $A \in$ *lset* $\mathcal{S}_1$
shows *is-hyp-proof* $\mathcal{H}$ $\mathcal{S}_1$ ([A] @ $\mathcal{S}_2$)
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* $\mathcal{H}$ $\mathcal{S}_1$ $\mathcal{S}_2$
and $A \in$ *lset* $\mathcal{S}_1$
shows *is-hyp-proof* $\mathcal{H}$ $\mathcal{S}_1$ ($\mathcal{S}_2$ @ [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 < *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* $\mathcal{H}$ p D C E
  shows *is-hyp-proof* $\mathcal{H}$ $\mathcal{S}'$ ($\mathcal{S}$ @ [D])
proof (*standard*, *intro allI impI*)
  let *?*$\mathcal{S}$ = $\mathcal{S}$ @ [D]
  fix i'
  assume i' < *length ?*$\mathcal{S}$
  then consider (a) i' < *length* $\mathcal{S}$ | (b) i' = *length* $\mathcal{S}$
    by *fastforce*
  then show *is-hyp-proof-step* $\mathcal{H}$ $\mathcal{S}'$ ($\mathcal{S}$ @ [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 = *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* $\mathcal{H}$ $\mathcal{S}'$ *?*$\mathcal{S}$ (*length ?*$\mathcal{S}$ − *1*)
      by *simp*
    moreover from b **have** i' = *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 $\longleftrightarrow$
    is-hyps $\mathcal{H}$ $\wedge$
    is-proof $\mathcal{S}_1$ $\wedge$
    $\mathcal{S}_2 \neq []$ $\wedge$
    is-hyp-proof $\mathcal{H}$ $\mathcal{S}_1$ $\mathcal{S}_2$ $\wedge$
    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* $\mathcal{H}$ $\mathcal{S}''$ ($\mathcal{S}$ @ $\mathcal{S}'$) **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* $\mathcal{H}$ $(\mathcal{S}_C{}'$ @ $\mathcal{S}_E{}')$ $((\mathcal{S}_C$ @ $\mathcal{S}_E)$ @ $[D])$
using *rule-R′-app-appended-to-hyp-proof-is-hyp-proof*
by *presburger*
moreover from ‹*is-proof* $\mathcal{S}_C{}'$› **and** ‹*is-proof* $\mathcal{S}_E{}'$› **have** *is-proof* $(\mathcal{S}_C{}'$ @ $\mathcal{S}_E{}')$
by (*fact proofs-concatenation-is-proof*)
ultimately have *is-hyp-proof-of* $\mathcal{H}$ $(\mathcal{S}_C{}'$ @ $\mathcal{S}_E{}')$ $((\mathcal{S}_C$ @ $\mathcal{S}_E)$ @ $[D])$ D
using ‹*is-hyps* $\mathcal{H}$› **by** *fastforce*
then show *?case*
by (*intro exI*)
qed

theorem *hyp-proof-existence-implies-hyp-derivability*:
assumes $\exists \mathcal{S}_1\ \mathcal{S}_2.$ *is-hyp-proof-of* $\mathcal{H}\ \mathcal{S}_1\ \mathcal{S}_2\ A$
shows $\mathcal{H} \vdash A$
proof −
from *assms* **obtain** $\mathcal{S}_1$ **and** $\mathcal{S}_2$
where *is-hyps* $\mathcal{H}$ **and** *is-proof* $\mathcal{S}_1$ **and** $\mathcal{S}_2 \neq []$ **and** *is-hyp-proof* $\mathcal{H}\ \mathcal{S}_1\ \mathcal{S}_2$ **and** *last* $\mathcal{S}_2 = A$
by *fastforce*
then show *?thesis*
proof (*induction length* $\mathcal{S}_2$ *arbitrary*: $\mathcal{S}_2$ A *rule*: *less-induct*)
case *less*
let *?i′* = *length* $\mathcal{S}_2 - 1$
from ‹$\mathcal{S}_2 \neq []$› **and** ‹*last* $\mathcal{S}_2 = A$› **have** $\mathcal{S}_2$! *?i′* = A
by (*simp add*: *last-conv-nth*)
from ‹*is-hyp-proof* $\mathcal{H}\ \mathcal{S}_1\ \mathcal{S}_2$› **and** ‹$\mathcal{S}_2 \neq []$› **have** *is-hyp-proof-step* $\mathcal{H}\ \mathcal{S}_1\ \mathcal{S}_2$ *?i′*
by *simp*
then consider
(*hyp*) $\mathcal{S}_2$! *?i′* $\in \mathcal{H}$
| (*seq*) $\mathcal{S}_2$! *?i′* $\in$ *lset* $\mathcal{S}_1$
| (*rule-R′*) $\exists p\ j\ k.\ \{j, k\} \subseteq \{0..<$*?i′*$\} \wedge$ *is-rule-R′-app* $\mathcal{H}$ p $(\mathcal{S}_2$! *?i′*$)$ $(\mathcal{S}_2$! $j)$ $(\mathcal{S}_2$! $k)$
by *force*
then show *?case*
proof *cases*
case *hyp*
with ‹$\mathcal{S}_2$! *?i′* = A› **and** ‹*is-hyps* $\mathcal{H}$› **show** *?thesis*
by (*fastforce intro*: *dv-hyp*)
next
case *seq*
from ‹$\mathcal{S}_2$! *?i′* $\in$ *lset* $\mathcal{S}_1$› **and** ‹$\mathcal{S}_2$! *?i′* = A›
obtain j **where** $\mathcal{S}_1$! $j = A$ **and** $\mathcal{S}_1 \neq []$ **and** $j <$ *length* $\mathcal{S}_1$
by (*metis empty-iff in-set-conv-nth list.set*(*1*))
with ‹*is-proof* $\mathcal{S}_1$› **have** *is-proof* (*take* (*Suc* j) $\mathcal{S}_1$) **and** *take* (*Suc* j) $\mathcal{S}_1 \neq []$
using *proof-prefix-is-proof*[**where** $\mathcal{S}_1$ = *take* (*Suc* j) $\mathcal{S}_1$ **and** $\mathcal{S}_2$ = *drop* (*Suc* j) $\mathcal{S}_1$]
by *simp-all*
moreover from ‹$\mathcal{S}_1$! $j = A$› **and** ‹$j <$ *length* $\mathcal{S}_1$› **have** *last* (*take* (*Suc* j) $\mathcal{S}_1$) $= A$
by (*simp add*: *take-Suc-conv-app-nth*)
ultimately have *is-proof-of* (*take* (*Suc* j) $\mathcal{S}_1$) A
by *fastforce*
then have *is-theorem* A

using *is-theorem-def* **by** *blast*
with ‹*is-hyps* $\mathcal{H}$› **show** *?thesis*
by (*intro dv-thm*)
next
case *rule-R′*
then obtain p **and** j **and** k
where $\{j, k\} \subseteq \{0..<?i'\}$ **and** *is-rule-R′-app* $\mathcal{H}$ p $(\mathcal{S}_2\ !\ ?i')$ $(\mathcal{S}_2\ !\ j)$ $(\mathcal{S}_2\ !\ k)$
by *force*
let $?\mathcal{S}_j$ = *take* (*Suc* j) $\mathcal{S}_2$ **and** $?\mathcal{S}_k$ = *take* (*Suc* k) $\mathcal{S}_2$
obtain $\mathcal{S}_j{}'$ **and** $\mathcal{S}_k{}'$ **where** $\mathcal{S}_2 = ?\mathcal{S}_j\ @\ \mathcal{S}_j{}'$ **and** $\mathcal{S}_2 = ?\mathcal{S}_k\ @\ \mathcal{S}_k{}'$
by (*metis append-take-drop-id*)
then have *is-hyp-proof* $\mathcal{H}$ $\mathcal{S}_1$ $(?\mathcal{S}_j\ @\ \mathcal{S}_j{}')$ **and** *is-hyp-proof* $\mathcal{H}$ $\mathcal{S}_1$ $(?\mathcal{S}_k\ @\ \mathcal{S}_k{}')$
by (*simp-all only*: ‹*is-hyp-proof* $\mathcal{H}$ $\mathcal{S}_1$ $\mathcal{S}_2$›)
moreover from ‹$\mathcal{S}_2 \neq []$› **and** ‹$\mathcal{S}_2 = ?\mathcal{S}_j\ @\ \mathcal{S}_j{}'$› **and** ‹$\mathcal{S}_2 = ?\mathcal{S}_k\ @\ \mathcal{S}_k{}'$› **and** ‹*last* $\mathcal{S}_2 = A$›
have *last* $\mathcal{S}_j{}' = A$ **and** *last* $\mathcal{S}_k{}' = A$
using ‹$\{j, k\} \subseteq \{0..<$*length* $\mathcal{S}_2 - 1\}$› **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 ‹$\mathcal{S}_2 \neq []$› **have** $?\mathcal{S}_j \neq []$ **and** $?\mathcal{S}_k \neq []$
by *simp-all*
ultimately have *is-hyp-proof-of* $\mathcal{H}$ $\mathcal{S}_1$ $?\mathcal{S}_j$ (*last* $?\mathcal{S}_j$) **and** *is-hyp-proof-of* $\mathcal{H}$ $\mathcal{S}_1$ $?\mathcal{S}_k$ (*last* $?\mathcal{S}_k$)
using *hyp-proof-prefix-is-hyp-proof-of-last*
[*OF* ‹*is-hyps* $\mathcal{H}$› ‹*is-proof* $\mathcal{S}_1$› ‹*is-hyp-proof* $\mathcal{H}$ $\mathcal{S}_1$ $(?\mathcal{S}_j\ @\ \mathcal{S}_j{}')$› ‹$?\mathcal{S}_j \neq []$›]
and *hyp-proof-prefix-is-hyp-proof-of-last*
[*OF* ‹*is-hyps* $\mathcal{H}$› ‹*is-proof* $\mathcal{S}_1$› ‹*is-hyp-proof* $\mathcal{H}$ $\mathcal{S}_1$ $(?\mathcal{S}_k\ @\ \mathcal{S}_k{}')$› ‹$?\mathcal{S}_k \neq []$›]
by *fastforce*+
moreover from ‹*last* $\mathcal{S}_j{}' = A$› **and** ‹*last* $\mathcal{S}_k{}' = A$›
have *length* $?\mathcal{S}_j <$ *length* $\mathcal{S}_2$ **and** *length* $?\mathcal{S}_k <$ *length* $\mathcal{S}_2$
using ‹$\{j, k\} \subseteq \{0..<$*length* $\mathcal{S}_2 - 1\}$› **by** *force*+
moreover from *calculation*(*3*,*4*) **have** *last* $?\mathcal{S}_j = \mathcal{S}_2\ !\ j$ **and** *last* $?\mathcal{S}_k = \mathcal{S}_2\ !\ k$
by (*metis Suc-lessD last-snoc linorder-not-le nat-neq-iff take-Suc-conv-app-nth take-all-iff*)+
ultimately have $\mathcal{H} \vdash \mathcal{S}_2\ !\ j$ **and** $\mathcal{H} \vdash \mathcal{S}_2\ !\ k$
using ‹*is-hyps* $\mathcal{H}$›
and *less*(*1*)[*OF* ‹*length* $?\mathcal{S}_j <$ *length* $\mathcal{S}_2$›] **and** *less*(*1*)[*OF* ‹*length* $?\mathcal{S}_k <$ *length* $\mathcal{S}_2$›]
by *fast*+
with ‹*is-hyps* $\mathcal{H}$› **and** ‹$\mathcal{S}_2\ !\ ?i' = A$› **show** *?thesis*
using ‹*is-rule-R′-app* $\mathcal{H}$ p $(\mathcal{S}_2\ !\ ?i')$ $(\mathcal{S}_2\ !\ j)$ $(\mathcal{S}_2\ !\ k)$› **by** (*blast intro*: *dv-rule-R′*)
qed
qed
qed

theorem *hypothetical-derivability-proof-existence-equivalence*:
shows $\mathcal{H} \vdash A \longleftrightarrow (\exists \mathcal{S}_1\ \mathcal{S}_2.$ *is-hyp-proof-of* $\mathcal{H}$ $\mathcal{S}_1$ $\mathcal{S}_2$ $A)$
using *hyp-derivability-implies-hyp-proof-existence* **and** *hyp-proof-existence-implies-hyp-derivability* **..**

proposition *derivability-from-no-hyps-theoremhood-equivalence*:
shows $\{\} \vdash A \longleftrightarrow$ *is-theorem* A
proof
assume $\{\} \vdash A$
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** $C\langle\!\langle p \leftarrow B\rangle\!\rangle \rhd D$
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** $\mathcal{H} \vdash A \equiv^{\mathcal{Q}} B$
  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 $A \in \mathit{wffs}_{\alpha}$
      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** $A \in \mathit{wffs}_{\alpha\to\beta}$ **and** $C \in \mathit{wffs}_{\alpha}$
      using *hyp-derivable-form-is-wffso* **and** *wffs-from-equality* **by** *blast+*
    then show $A \cdot C \in \mathit{wffs}_{\beta}$
      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** $C \in wffs_\alpha$
  shows $\mathcal{H} \vdash A \cdot C =_\beta B \cdot C$
proof −
  have $\mathcal{H} \vdash A \cdot C =_\beta A \cdot C$
  proof (*rule hyp-prop-5200*)
    from *assms*(*1*) **show** *is-hyps* $\mathcal{H}$
      by (*blast elim*: *is-derivable-from-hyps.cases*)
    have $A \in wffs_{\alpha\to\beta}$
      by (*fact hyp-derivable-form-is-wffso*[*OF assms*(*1*), *THEN wffs-from-equality*(*1*)])
    with *assms*(*2*) **show** $A \cdot C \in wffs_\beta$
      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** $A \in wffs_{\alpha\to\beta}$
  shows $\mathcal{H} \vdash A \cdot C =_\beta A \cdot D$
proof −
  have $\mathcal{H} \vdash A \cdot C =_\beta A \cdot C$
  proof (*rule hyp-prop-5200*)
    from *assms*(*1*) **show** *is-hyps* $\mathcal{H}$
      by (*blast elim*: *is-derivable-from-hyps.cases*)
    have $C \in wffs_\alpha$
      by (*fact hyp-derivable-form-is-wffso*[*OF assms*(*1*), *THEN wffs-from-equality*(*1*)])
    with *assms*(*2*) **show** $A \cdot C \in wffs_\beta$
      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 $\vdash A =_\alpha B \lor \vdash B =_\alpha A$
  and $p \in$ *positions* C **and** $A \preceq_p C$ **and** $C⦃p \leftarrow B⦄ \rhd D$
  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** $p = \text{»} \ \# \ p$]) (*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 ‹$\vdash C =_o D$› **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 $\vdash (\lambda x_\alpha.\ B) \bullet A =_\beta \mathbf{S}\ \{(x, \alpha) \rightarrowtail A\}\ B$
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** $\vdash (\lambda x_\alpha.\ x_\alpha) \bullet A =_\alpha A$
using *axiom-4-2* **by** (*intro axiom-is-derivable-from-no-hyps*)
moreover have $\mathbf{S}\ \{(x, \alpha) \rightarrowtail A\}\ (x_\alpha) = A$
by *force*
ultimately show *?thesis*
using *True* **by** (*simp only*:)
next
case *False*
with *assms*(*1*) **have** $\vdash (\lambda x_\alpha.\ y_\beta) \bullet A =_\beta y_\beta$
using *axiom-4-1-var* **by** (*intro axiom-is-derivable-from-no-hyps*)
moreover from *False* **have** $\mathbf{S}\ \{(x, \alpha) \rightarrowtail A\}\ (y_\beta) = y_\beta$
by *auto*
ultimately show *?thesis*
by (*simp only*:)
qed
next
case (*con-is-wff* β *c*)
from *assms*(*1*) **have** $\vdash (\lambda x_\alpha.\ \{\!|c|\!\}_\beta) \bullet A =_\beta \{\!|c|\!\}_\beta$
using *axiom-4-1-con* **by** (*intro axiom-is-derivable-from-no-hyps*)

moreover have $\mathbf{S}\ \{(x, \alpha) \rightarrowtail A\}\ (\{\!|c|\!\}_\beta) = \{\!|c|\!\}_\beta$
  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*: $\forall v \in$ *vars* A. $\neg$ *is-bound* v D $\wedge$ $\neg$ *is-bound* v C
  using *is-bound-in-app-homomorphism* **by** *fast*
  from ‹$D \in$ *wffs*$_{\gamma\to\beta}$› **have** $\vdash (\lambda x_\alpha.\ D) \cdot A =_{\gamma\to\beta} \mathbf{S}\ \{(x, \alpha) \rightarrowtail A\}\ D$
  using *app-is-wff.IH*(*1*)[*OF assms*(*1*)] **and** *not-bound-subforms* **by** *simp*
  moreover from ‹$C \in$ *wffs*$_\gamma$› **have** $\vdash (\lambda x_\alpha.\ C) \cdot A =_\gamma \mathbf{S}\ \{(x, \alpha) \rightarrowtail A\}\ C$
  using *app-is-wff.IH*(*2*)[*OF assms*(*1*)] **and** *not-bound-subforms* **by** *simp*
  moreover have $\vdash (\lambda x_\alpha.\ D \cdot C) \cdot A =_\beta ((\lambda x_\alpha.\ D) \cdot A) \cdot ((\lambda x_\alpha.\ C) \cdot A)$
  using *axiom-is-derivable-from-no-hyps*[*OF axiom-4-3*[*OF assms*(*1*) ‹$D \in$ *wffs*$_{\gamma\to\beta}$› ‹$C \in$ *wffs*$_\gamma$›]] .
  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* $y_\gamma = x_\alpha$)
  case *True*
  then have $\vdash (\lambda x_\alpha.\ \lambda y_\gamma.\ D) \cdot A =_{\gamma\to\beta} \lambda y_\gamma.\ D$
  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** $\mathbf{S}\ \{(x, \alpha) \rightarrowtail A\}\ (\lambda y_\gamma.\ D) = \lambda y_\gamma.\ D$
  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* $(\lambda y_\gamma.\ D)$ [«] $= \{(y, \gamma)\}$
  by *simp*
  then have *is-bound* (y, γ) $(\lambda y_\gamma.\ D)$
  by *fastforce*
  with *abs-is-wff.prems*(*2*) **have** $(y, \gamma) \notin$ *vars* A
  by *blast*
  with ‹$y_\gamma \neq x_\alpha$› **have** $\vdash (\lambda x_\alpha.\ \lambda y_\gamma.\ D) \cdot A =_{\gamma\to\beta} \lambda y_\gamma.\ (\lambda x_\alpha.\ D) \cdot A$
  using *axiom-4-4*[*OF assms*(*1*) *abs-is-wff.hyps*(*1*)] **and** *axiom-is-derivable-from-no-hyps* **by** *blast*
  moreover have $\vdash (\lambda x_\alpha.\ D) \cdot A =_\beta \mathbf{S}\ \{(x, \alpha) \rightarrowtail A\}\ D$
  proof −
  have $\forall p.\ y_\gamma \preceq_{\text{«} \# p} \lambda y_\gamma.\ D \longrightarrow y_\gamma \preceq_p D$
  using *subforms-from-abs* **by** *fastforce*
  from *abs-is-wff.prems*(*2*) **have** $\forall v \in$ *vars* A. $\neg$ *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 $\vdash (\lambda x_\alpha.\ \lambda y_\gamma.\ D) \cdot A =_{\gamma\to\beta} \lambda y_\gamma.\ \mathbf{S}\ \{(x, \alpha) \rightarrowtail A\}\ D$
  by (*rule rule-R*[**where** $p =$ [»,«]]) *force+*

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

6.5 Proposition 5204

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

6.6 Proposition 5205 (η-conversion)

proposition *prop-5205*:
shows $\vdash \mathfrak{f}_{\alpha\to\beta} =_{\alpha\to\beta} (\lambda y_\alpha.\ \mathfrak{f}_{\alpha\to\beta} \boldsymbol{\cdot} y_\alpha)$
proof −
{
fix y
assume $y_\alpha \neq \mathfrak{x}_\alpha$
let $?A = \lambda y_\alpha.\ \mathfrak{f}_{\alpha\to\beta} \boldsymbol{\cdot} y_\alpha$
have $\vdash (\mathfrak{f}_{\alpha\to\beta} =_{\alpha\to\beta} ?A) =_o \forall \mathfrak{x}_\alpha.\ (\mathfrak{f}_{\alpha\to\beta} \boldsymbol{\cdot} \mathfrak{x}_\alpha =_\beta ?A \boldsymbol{\cdot} \mathfrak{x}_\alpha)$
proof −
have $\vdash (\mathfrak{f}_{\alpha\to\beta} =_{\alpha\to\beta} \mathfrak{g}_{\alpha\to\beta}) =_o \forall \mathfrak{x}_\alpha.\ (\mathfrak{f}_{\alpha\to\beta} \boldsymbol{\cdot} \mathfrak{x}_\alpha =_\beta \mathfrak{g}_{\alpha\to\beta} \boldsymbol{\cdot} \mathfrak{x}_\alpha)$ (**is** $\vdash ?B =_o ?C$)
using *axiom-3*[*unfolded equivalence-def*] **by** (*rule axiom-is-derivable-from-no-hyps*)
have $\vdash \mathbf{S}\ \{(\mathfrak{g},\, \alpha\to\beta) \rightarrowtail ?A\}\ (?B =_o ?C)$
proof −
have $?A \in wffs_{\alpha\to\beta}$ **and** $?B \in wffs_o$ **and** $?C \in wffs_o$
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 $\langle y_\alpha \neq \mathfrak{x}_\alpha \rangle$ **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 $\langle v \in \mathit{vars}\ ?A \rangle$ **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 $\langle \vdash\ ?B =_o\ ?C \rangle$ **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 $\langle y_\alpha \neq \mathfrak{x}_\alpha \rangle$ **by** *auto*
moreover have $\mathbf{S}\ \{(y, \alpha) \rightarrowtail \mathfrak{x}_\alpha\}\ (\mathfrak{f}_{\alpha\to\beta} \cdot y_\alpha) = \mathfrak{f}_{\alpha\to\beta} \cdot \mathfrak{x}_\alpha$
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 = [»,»,«,»]]) *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 $\vdash \mathbf{S}\ \{(\mathfrak{g}, \alpha{\to}\beta) \rightarrowtail\ ?A\}\ (?B =_o\ ?C)$
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* $= \{(\mathfrak{f}, \alpha{\to}\beta), (\mathfrak{g}, \alpha{\to}\beta)\}$ **and** *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** $(y, \alpha) \notin$ *vars ?B* **and** $(y, \alpha) \notin$ *vars ?C*
by *force+*
then have $\neg$ *is-bound* (y, α) *?B* **and** $\neg$ *is-bound* (y, α) *?C*
using *absent-var-is-not-bound* **by** *blast+*
moreover have $\neg$ *is-bound* $(\mathfrak{f}, \alpha{\to}\beta)$ *?B* **and** $\neg$ *is-bound* $(\mathfrak{f}, \alpha{\to}\beta)$ *?C*
by *code-simp+*
moreover from ‹$v \in$ *vars ?A* ›**have** $v \in \{(y, \alpha), (\mathfrak{f}, \alpha{\to}\beta)\}$
by *auto*
ultimately show $\neg$ *is-bound v ?B* $\wedge$ $\neg$ *is-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
ultimately have $\vdash \mathfrak{f}_{\alpha\to\beta} =_{\alpha\to\beta} (\lambda y_\alpha.\ \mathfrak{f}_{\alpha\to\beta} \bullet y_\alpha)$
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*: $\vdash \mathfrak{f}_{\alpha\to\beta} =_{\alpha\to\beta} \lambda\mathfrak{y}_\alpha.\ \mathfrak{f}_{\alpha\to\beta} \bullet \mathfrak{y}_\alpha$ (**is** $\vdash$ *?B* $=_{-}$ *?C*)
by *simp*
then have §*7*: $\vdash (\lambda\mathfrak{x}_\alpha.\ \mathfrak{f}_{\alpha\to\beta} \bullet \mathfrak{x}_\alpha) =_{\alpha\to\beta} (\lambda\mathfrak{y}_\alpha.\ (\lambda\mathfrak{x}_\alpha.\ \mathfrak{f}_{\alpha\to\beta} \bullet \mathfrak{x}_\alpha) \bullet \mathfrak{y}_\alpha)$
proof −
let *?A* $= \lambda\mathfrak{x}_\alpha.\ \mathfrak{f}_{\alpha\to\beta} \bullet \mathfrak{x}_\alpha$
have *?A* $\in$ $wffs_{\alpha\to\beta}$ **and** *?B* $\in$ $wffs_{\alpha\to\beta}$ **and** *?C* $\in$ $wffs_{\alpha\to\beta}$
by *auto*
moreover have $\forall\, v \in$ *vars ?A*. $\neg$ *is-bound v ?B* $\wedge$ $\neg$ *is-bound v ?C*
proof
fix *v*
assume $v \in$ *vars ?A*
have $\neg$ *is-bound* $(\mathfrak{x}, \alpha)$ *?B* **and** $\neg$ *is-bound* $(\mathfrak{x}, \alpha)$ *?C*
by *code-simp+*
moreover have $\neg$ *is-bound* $(\mathfrak{f}, \alpha{\to}\beta)$ *?B* **and** $\neg$ *is-bound* $(\mathfrak{f}, \alpha{\to}\beta)$ *?C*
by *code-simp+*
moreover from ‹$v \in$ *vars ?A* ›**have** $v \in \{(\mathfrak{x}, \alpha), (\mathfrak{f}, \alpha{\to}\beta)\}$
by *auto*
ultimately show $\neg$ *is-bound v ?B* $\wedge$ $\neg$ *is-bound v ?C*
by *fast*
qed
ultimately have $\vdash$ **S** $\{(\mathfrak{f}, \alpha{\to}\beta) \rightarrowtail$ *?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 $\mathfrak{y}_\alpha \in$ *wffs*$_\alpha$ **and** $\mathfrak{f}_{\alpha\to\beta} \cdot \mathfrak{x}_\alpha \in$ *wffs*$_\beta$
by *auto*
moreover have $\forall\, v \in$ *vars* $(\mathfrak{y}_\alpha)$. $\neg$ *is-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 = [»,«]]) *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 $A \in$ *wffs*$_\alpha$
and $(z, \beta) \notin$ *free-vars* A
and *is-free-for* (z_β) (x, β) A
shows $\vdash (\lambda x_\beta.\ A) =_{\beta\to\alpha} (\lambda z_\beta.$ **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** $\{(x, \beta) \rightarrowtail z_\beta\}\ A \in$ *wffs*$_\alpha$
by (*fact substitution-preserves-typing*)
obtain y **where** $(y, \beta) \notin \{(x, \beta), (z, \beta)\} \cup$ *vars* A
proof −
have *finite* $(\{(x, \beta), (z, \beta)\} \cup$ *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** $(y, \beta) \notin$ *vars* A **and** $(y, \beta) \notin$ *free-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* $=_{-}$ *?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* $\in$ *wffs*$_{\beta\to\alpha}$ **and** *?B* $\in$ *wffs*$_{\beta\to\alpha}$ **and** *?C* $\in$ *wffs*$_{\beta\to\alpha}$

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 $v \in \mathit{vars}\ ?A$
then consider $(a)\ v = (x, \beta) \mid (b)\ v \in \mathit{vars}\ A$
by *fastforce*
then show $\neg\ \mathit{is\text{-}bound}\ v\ ?B \wedge \neg\ \mathit{is\text{-}bound}\ v\ ?C$
proof *cases*
case a
then show *?thesis*
using ‹$(y, \beta) \neq (x, \beta)$› **by** *force*
next
case b
then have $\neg\ \mathit{is\text{-}bound}\ v\ ?B$
by *simp*
moreover have $\neg\ \mathit{is\text{-}bound}\ v\ ?C$
using b **and** ‹$(y, \beta) \notin \mathit{vars}\ A$› **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*: $\vdash (\lambda x_\beta.\ A) =_{\beta\to\alpha} (\lambda y_\beta.\ \mathbf{S}\ \{(x, \beta) \rightarrowtail y_\beta\}\ A)$
proof $-$
have $\vdash (\lambda x_\beta.\ A) \bullet y_\beta =_\alpha \mathbf{S}\ \{(x, \beta) \rightarrowtail y_\beta\}\ A$ (**is** $\vdash (\lambda x_\beta.\ ?B) \bullet ?A =_{-}$ -)
proof $-$
have $?A \in \mathit{wffs}_\beta$ **and** $?B \in \mathit{wffs}_\alpha$
by *blast fact*
moreover have $\forall\, v \in \mathit{vars}\ ?A.\ \neg\ \mathit{is\text{-}bound}\ v\ ?B$
using ‹$(y, \beta) \notin \mathit{vars}\ A$› **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*: $\vdash (\lambda z_\beta.\ \mathbf{S}\ \{(x, \beta) \rightarrowtail z_\beta\}\ A) =_{\beta\to\alpha} (\lambda y_\beta.\ (\lambda z_\beta.\ \mathbf{S}\ \{(x, \beta) \rightarrowtail z_\beta\}\ A) \bullet y_\beta)$
proof $-$
let $?A = \lambda z_\beta.\ \mathbf{S}\ \{(x, \beta) \rightarrowtail z_\beta\}\ A$
have $*$: $\vdash \mathfrak{f}_{\beta\to\alpha} =_{\beta\to\alpha} (\lambda y_\beta.\ \mathfrak{f}_{\beta\to\alpha} \bullet y_\beta)$ (**is** $\vdash ?B =_{-} ?C$)
by (*fact prop-5205*)
moreover have $\vdash \mathbf{S}\ \{(\mathfrak{f}, \beta\to\alpha) \rightarrowtail ?A\}\ (?B =_{\beta\to\alpha} ?C)$

proof −
have $?A \in wffs_{\beta\to\alpha}$ **and** $?B \in wffs_{\beta\to\alpha}$ **and** $?C \in wffs_{\beta\to\alpha}$
using ‹**S** $\{(x, \beta) \rightarrowtail z_\beta\}$ $A \in wffs_\alpha$› **by** *auto*
moreover have $\forall v \in vars\ ?A.\ \neg\ is\text{-}bound\ v\ ?B \wedge \neg\ is\text{-}bound\ v\ ?C$
proof
fix v
assume $v \in vars\ ?A$
then consider (a) $v = (z, \beta)$ | (b) $v \in vars$ (**S** $\{(x, \beta) \rightarrowtail z_\beta\}$ A)
by *fastforce*
then show $\neg\ is\text{-}bound\ v\ ?B \wedge \neg\ is\text{-}bound\ v\ ?C$
proof *cases*
case a
then show *?thesis*
using ‹$(y, \beta) \neq (z, \beta)$› **by** *auto*
next
case b
then have $\neg\ is\text{-}bound\ v\ ?B$
by *simp*
moreover from b **and** ‹$(y, \beta) \notin vars\ A$› **and** ‹$(y, \beta) \neq (z, \beta)$› **have** $v \neq (y, \beta)$
using *renaming-substitution-minimal-change* **by** *blast*
then have $\neg\ is\text{-}bound\ v\ ?C$
by *code-simp simp*
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 §*4*: $\vdash (\lambda z_\beta.$ **S** $\{(x, \beta) \rightarrowtail z_\beta\}\ A) =_{\beta\to\alpha} (\lambda y_\beta.$ **S** $\{(x, \beta) \rightarrowtail y_\beta\}\ A)$
proof −
have $\vdash (\lambda z_\beta.$ **S** $\{(x, \beta) \rightarrowtail z_\beta\}\ A) \bullet y_\beta =_\alpha$ **S** $\{(x, \beta) \rightarrowtail y_\beta\}\ A$ (**is** $\vdash (\lambda z_\beta.\ ?B) \bullet ?A =_{-}$ -)
proof −
have $?A \in wffs_\beta$ **and** $?B \in wffs_\alpha$
by *blast fact*
moreover from ‹$(y, \beta) \notin vars\ A$› **and** ‹$(y, \beta) \neq (z, \beta)$› **have** $\forall v \in vars\ ?A.\ \neg\ is\text{-}bound\ v\ ?B$
using *absent-var-is-not-bound* **and** *renaming-substitution-minimal-change* **by** *auto*
ultimately have $\vdash (\lambda z_\beta.$ **S** $\{(x, \beta) \rightarrowtail z_\beta\}\ A) \bullet y_\beta =_\alpha$ **S** $\{(z, \beta) \rightarrowtail y_\beta\}$ **S** $\{(x, \beta) \rightarrowtail z_\beta\}\ A$
using *prop-5203* **by** *fast*
moreover have S $\{(z, \beta) \rightarrowtail y_\beta\}$ **S** $\{(x, \beta) \rightarrowtail z_\beta\}\ A =$ **S** $\{(x, \beta) \rightarrowtail y_\beta\}\ A$
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-bound-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-bound-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-bound-var* $(y, \gamma)\ z\ A$
by (*rule rule-R*[**where** p = [»,«]]) *force*+
moreover
have
$\vdash \lambda y_\gamma.$ *rename-bound-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-bound-var* $(y, \gamma)\ z\ A$›]
by (*blast dest*: *wffs-from-equality*)

moreover from *abs-is-wff.prems(2)* **have** $(z, \gamma) \notin$ *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 $\vdash \lambda y_\gamma.\ A =_{\gamma \to \beta} \lambda z_\gamma.$ **S** $\{(y, \gamma) \rightarrowtail z_\gamma\}$ (*rename-bound-var* (y, γ) z A)
by (*rule Equality-Rules(3)*)
then show *?thesis*
using *True* **by** *auto*
next
case *False*
have $\vdash \lambda x_\alpha.\ A =_{\alpha \to \beta} \lambda x_\alpha.\ A$
by (*fact abs-is-wff.hyps*[*THEN prop-5200*[*OF wffs-of-type-intros(4)*]])
moreover have $\vdash A =_\beta$ *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 $\vdash \lambda x_\alpha.\ A =_{\alpha \to \beta} \lambda x_\alpha.$ *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 $A \in wffs_\alpha$ **and** $B \in wffs_\beta$
and *is-free-for* A (x, α) B
shows $\vdash (\lambda x_\alpha.\ B) \bullet A =_\beta$ **S** $\{(x, \alpha) \rightarrowtail A\}$ 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* $y_\beta = x_\alpha$)
case *True*
then have $\alpha = \beta$
by *simp*
moreover from *assms(1)* **have** $\vdash (\lambda x_\alpha.\ x_\alpha) \bullet A =_\alpha A$
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*
unfolding *True* **and** *var-is-wff* **by** *simp*
next
case *False*
with *assms(1)* **have** $\vdash (\lambda x_\alpha.\ y_\beta) \bullet A =_\beta y_\beta$
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*
unfolding *False* **and** *var-is-wff* **by** *simp*
qed
next
case (*con-is-wff c*)
from *assms*(*1*) **have** $\vdash (\lambda x_\alpha.\ \{\!\!\{c\}\!\!\}_\beta) \bullet A =_\beta \{\!\!\{c\}\!\!\}_\beta$
using *axiom-4-1-con* **by** (*intro axiom-is-derivable-from-no-hyps*)
moreover have $\mathbf{S}\ \{(x, \alpha) \rightarrowtail A\}\ (\{\!\!\{c\}\!\!\}_\beta) = \{\!\!\{c\}\!\!\}_\beta$
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* $A\ (x, \alpha)\ D$ **and** *is-free-for* $A\ (x, \alpha)\ C$
using *is-free-for-from-app* **by** *iprover+*
from ‹*is-free-for* $A\ (x, \alpha)\ D$› **have** $\vdash (\lambda x_\alpha.\ D) \bullet A =_{\gamma\to\beta} \mathbf{S}\ \{(x, \alpha) \rightarrowtail A\}\ D$
by (*fact less*(*1*)[*OF* ‹*form-size* $D <$ *form-size* B› *assms*(*1*) *app-is-wff*(*2*)])
moreover from ‹*is-free-for* $A\ (x, \alpha)\ C$› **have** $\vdash (\lambda x_\alpha.\ C) \bullet A =_\gamma \mathbf{S}\ \{(x, \alpha) \rightarrowtail A\}\ C$
by (*fact less*(*1*)[*OF* ‹*form-size* $C <$ *form-size* B› *assms*(*1*) *app-is-wff*(*3*)])
moreover have $\vdash (\lambda x_\alpha.\ D \bullet C) \bullet A =_\beta ((\lambda x_\alpha.\ D) \bullet A) \bullet ((\lambda x_\alpha.\ C) \bullet A)$
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* $y_\gamma = x_\alpha$)
case *True*
with *abs-is-wff*(*1*) **have** $\vdash (\lambda x_\alpha.\ \lambda y_\gamma.\ D) \bullet A =_\beta \lambda y_\gamma.\ D$
using *axiom-4-5*[*OF assms*(*1*) *abs-is-wff*(*3*)] **by** (*simp add*: *axiom-is-derivable-from-no-hyps*)
moreover have $\mathbf{S}\ \{(x, \alpha) \rightarrowtail A\}\ (\lambda y_\gamma.\ D) = \lambda y_\gamma.\ D$
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* $A\ (x, \alpha)\ D$
using *is-free-for-from-abs*[*OF less*(*4*)[*unfolded abs-is-wff*(*2*)]] **and** ‹$y_\gamma \neq x_\alpha$› **by** *blast*
have $\vdash (\lambda x_\alpha.\ (\lambda y_\gamma.\ D)) \bullet A =_\beta \lambda y_\gamma.\ \mathbf{S}\ \{(x, \alpha) \rightarrowtail A\}\ D$
proof (*cases* $(y, \gamma) \notin$ *vars* A)
case *True*
with ‹$y_\gamma \neq x_\alpha$› **have** $\vdash (\lambda x_\alpha.\ \lambda y_\gamma.\ D) \bullet A =_{\gamma\to\delta} \lambda y_\gamma.\ (\lambda x_\alpha.\ D) \bullet A$
using *axiom-4-4*[*OF assms*(*1*) *abs-is-wff*(*3*)] **and** *axiom-is-derivable-from-no-hyps* **by** *auto*
moreover have $\vdash (\lambda x_\alpha.\ D) \bullet A =_\delta \mathbf{S}\ \{(x, \alpha) \rightarrowtail A\}\ D$

```
  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*: $\vdash (\lambda x_\alpha.\ \lambda y_\gamma.\ D) \bullet A =_{\gamma \to \delta} \lambda y_\gamma.\ D$
proof −
note *rule-RR*$'$ = *rule-RR*[*OF disjI2*]
have §*5*$_1$: $\vdash (\lambda x_\alpha.\ \lambda y_\gamma.\ D) \bullet A =_{\gamma \to \delta} \lambda z_\gamma.\ D'$ (**is** ‹$\vdash$ *?A5*$_1$›)
by (*rule rule-RR*$'$[*OF* §*4*, **where** p = [«,»,«,«] **and** C = *?A3*]) (*use* §*3* **in** ‹*force*+›)
show *?thesis*
by (*rule rule-RR*$'$[*OF* §*4*, **where** p = [»] **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*) ‹$z_\gamma \neq y_\gamma$›] **and** ‹$(z, \gamma) \notin$ *vars* $\{A, D\}$›
by *simp*
moreover have §*7*: $\vdash (\lambda x_\alpha.\ \lambda y_\gamma.\ D) \bullet A' =_{\gamma\to\delta} \lambda y_\gamma.\ (\lambda x_\alpha.\ D) \bullet A'$ (**is** ‹$\vdash$ *?A7*›)
using *axiom-4-4*[*OF* ‹$A' \in$ *wffs*$_\alpha$› ‹$D \in$ *wffs*$_\delta$›]
and ‹$(y, \gamma) \notin$ *vars* A'› **and** ‹$y_\gamma \neq x_\alpha$› **and** *axiom-is-derivable-from-no-hyps*
by *auto*
ultimately have §*8*: $\vdash (\lambda x_\alpha.\ \lambda y_\gamma.\ D) \bullet A =_{\gamma\to\delta} \lambda y_\gamma.\ (\lambda x_\alpha.\ D) \bullet A$
proof −
note *rule-RR*$'$ = *rule-RR*[*OF disjI2*]
have §*8*$_1$: $\vdash (\lambda x_\alpha.\ \lambda y_\gamma.\ D) \bullet A =_{\gamma\to\delta} \lambda y_\gamma.\ (\lambda x_\alpha.\ D) \bullet A'$ (**is** ‹$\vdash$ *?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*: $\vdash (\lambda x_\alpha.\ D) \bullet A =_\delta$ **S** $\{(x, \alpha) \rightarrowtail A\}\ D$
using *less*(*1*) **and** ‹$D \in$ *wffs*$_\delta$› **and** ‹*is-free-for* A (x, α) D› **by** (*simp only*:)
ultimately show *?thesis*
unfolding ‹$\beta = \gamma\to\delta$› **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* $\neq$ [] **and** $B \in$ *wffs*$_\beta$
shows $\vdash \bullet^{\mathcal{Q}}_\star\ (\lambda^{\mathcal{Q}}_\star\ vs\ B)\ (map\ FVar\ vs) =_\beta B$
using *assms*(*1*) **proof** (*induction vs rule*: *list-nonempty-induct*)
case (*single v*)
obtain x **and** α **where** $v = (x, \alpha)$
by *fastforce*
then have $\bullet^{\mathcal{Q}}_\star\ (\lambda^{\mathcal{Q}}_\star\ [v]\ B)\ (map\ FVar\ [v]) = (\lambda x_\alpha.\ B) \bullet x_\alpha$
by *simp*
moreover have $\vdash (\lambda x_\alpha.\ B) \bullet x_\alpha =_\beta B$
proof −
have *is-free-for* (x_α) (x, α) B
by *fastforce*
then have $\vdash (\lambda x_\alpha.\ B) \bullet x_\alpha =_\beta$ **S** $\{(x, \alpha) \rightarrowtail x_\alpha\}\ B$
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** $v = (x, \alpha)$
by *fastforce*
have $\vdash \bullet^{\mathcal{Q}}{}_{\star}\ (\lambda^{\mathcal{Q}}{}_{\star}\ (v \# vs)\ B)\ (map\ FVar\ (v \# vs)) =_{\beta} \bullet^{\mathcal{Q}}{}_{\star}\ (\lambda^{\mathcal{Q}}{}_{\star}\ vs\ B)\ (map\ FVar\ vs)$
proof −
have $\bullet^{\mathcal{Q}}{}_{\star}\ (\lambda^{\mathcal{Q}}{}_{\star}\ (v \# vs)\ B)\ (map\ FVar\ (v \# vs)) \in wffs_{\beta}$
proof −
have $\lambda^{\mathcal{Q}}{}_{\star}\ (v \# vs)\ B \in wffs_{foldr\ (\rightarrow)\ (map\ snd\ (v \# vs))\ \beta}$
using *generalized-abs-wff* [*OF assms*(*2*)] **by** *blast*
moreover
have $\forall k < length\ (map\ FVar\ (v \# vs)).\ map\ FVar\ (v \# vs)\ !\ k \in wffs_{map\ snd\ (v \# vs)\ !\ k}$
proof *safe*
fix k
assume $*$: $k < length\ (map\ FVar\ (v \# vs))$
moreover obtain x **and** α **where** $(v \# vs)\ !\ k = (x, \alpha)$
by *fastforce*
with $*$ **have** $map\ FVar\ (v \# vs)\ !\ k = x_{\alpha}$ **and** $map\ snd\ (v \# vs)\ !\ k = \alpha$
by (*metis length-map nth-map snd-conv*)+
ultimately show $map\ FVar\ (v \# vs)\ !\ k \in wffs_{map\ snd\ (v \# vs)\ !\ k}$
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
$\vdash \bullet^{\mathcal{Q}}{}_{\star}\ (\lambda^{\mathcal{Q}}{}_{\star}\ (v \# vs)\ B)\ (map\ FVar\ (v \# vs)) =_{\beta} \bullet^{\mathcal{Q}}{}_{\star}\ (\lambda^{\mathcal{Q}}{}_{\star}\ (v \# vs)\ B)\ (map\ FVar\ (v \# vs))$
by (*fact prop-5200*)
then have
$\vdash \bullet^{\mathcal{Q}}{}_{\star}\ (\lambda^{\mathcal{Q}}{}_{\star}\ (v \# vs)\ B)\ (map\ FVar\ (v \# vs)) =_{\beta} \bullet^{\mathcal{Q}}{}_{\star}\ ((\lambda^{\mathcal{Q}}{}_{\star}\ (v \# vs)\ B) \bullet FVar\ v)\ (map\ FVar\ vs)$
by *simp*
moreover have $\vdash (\lambda^{\mathcal{Q}}{}_{\star}\ (v \# vs)\ B) \bullet FVar\ v =_{foldr\ (\rightarrow)\ (map\ snd\ vs)\ \beta} (\lambda^{\mathcal{Q}}{}_{\star}\ vs\ B)$
proof −
have $\vdash (\lambda^{\mathcal{Q}}{}_{\star}\ (v \# vs)\ B) \bullet FVar\ v =_{foldr\ (\rightarrow)\ (map\ snd\ vs)\ \beta} \mathbf{S}\ \{v \rightarrowtail FVar\ v\}\ (\lambda^{\mathcal{Q}}{}_{\star}\ vs\ B)$
proof −
from ‹$v = (x, \alpha)$› **have** $\lambda^{\mathcal{Q}}{}_{\star}\ (v \# vs)\ B = \lambda x_{\alpha}.\ \lambda^{\mathcal{Q}}{}_{\star}\ vs\ B$
by *simp*
have $\lambda^{\mathcal{Q}}{}_{\star}\ vs\ B \in wffs_{foldr\ (\rightarrow)\ (map\ snd\ vs)\ \beta}$
using *generalized-abs-wff*[*OF assms*(*2*)] **by** *blast*
moreover have *is-free-for* $(x_{\alpha})\ (x, \alpha)\ (\lambda^{\mathcal{Q}}{}_{\star}\ vs\ B)$
by *fastforce*
ultimately
have $\vdash (\lambda x_{\alpha}.\ \lambda^{\mathcal{Q}}{}_{\star}\ vs\ B) \bullet x_{\alpha} =_{foldr\ (\rightarrow)\ (map\ snd\ vs)\ \beta} \mathbf{S}\ \{(x, \alpha) \rightarrowtail x_{\alpha}\}\ \lambda^{\mathcal{Q}}{}_{\star}\ vs\ B$

```
        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 ⊢ ·𝒬⋆ (λ𝒬⋆ 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 ∈ wffsα and B ∈ wffsβ and C ∈ wffsβ
  and ⊢ B =β C
  and is-free-for A (x, α) (B =β C)
  shows ⊢ S {(x, α) ↣ A} (B =β C)
proof –
  have ⊢ (λxα. B) · A =β (λxα. B) · A
  proof –
    have (λxα. B) · A ∈ wffsβ
      using assms(1,2) by blast
    then show ?thesis
      by (fact prop-5200)
  qed
  from this and assms(4) have ⊢ (λxα. B) · A =β (λxα. C) · A
    by (rule rule-R [where p = [»,«,«]]) force+
  moreover have ⊢ (λxα. B) · A =β S {(x, α) ↣ A} B
  proof –
    from assms(5)[unfolded equality-of-type-def] have is-free-for A (x, α) (Qβ · 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 $\vdash (\lambda x_\alpha.\ C) \cdot A =_\beta$ **S** $\{(x, \alpha) \rightarrowtail A\}\ C$
proof −
from *assms*(*5*)[*unfolded equality-of-type-def*] **have** *is-free-for* $A\ (x, \alpha)\ C$
by (*rule is-free-for-from-app*)
with *assms*(*1*,*3*) **show** *?thesis*
by (*rule prop-5207*)
qed
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

6.11 Proposition 5210

proposition *prop-5210*:
assumes $B \in wffs_\beta$
shows $\vdash T_o =_o (B =_\beta B)$
proof −
have §*1*:
$\vdash$
$((\lambda\mathfrak{y}_\beta.\ \mathfrak{y}_\beta) =_{\beta\to\beta} (\lambda\mathfrak{y}_\beta.\ \mathfrak{y}_\beta))$
$=_o$
$\forall\mathfrak{x}_\beta.\ ((\lambda\mathfrak{y}_\beta.\ \mathfrak{y}_\beta) \cdot \mathfrak{x}_\beta =_\beta (\lambda\mathfrak{y}_\beta.\ \mathfrak{y}_\beta) \cdot \mathfrak{x}_\beta)$
proof −
have $\vdash (\mathfrak{f}_{\beta\to\beta} =_{\beta\to\beta} \mathfrak{g}_{\beta\to\beta}) =_o \forall\mathfrak{x}_\beta.\ (\mathfrak{f}_{\beta\to\beta} \cdot \mathfrak{x}_\beta =_\beta \mathfrak{g}_{\beta\to\beta} \cdot \mathfrak{x}_\beta)$ (**is** $\vdash$ *?B* $=_o$ *?C*)
using *axiom-3*[*unfolded equivalence-def*] **by** (*rule axiom-is-derivable-from-no-hyps*)
moreover have $(\lambda\mathfrak{y}_\beta.\ \mathfrak{y}_\beta) \in wffs_{\beta\to\beta}$ **and** *?B* $\in wffs_o$ **and** *?C* $\in wffs_o$
by *auto*
moreover have *is-free-for* $(\lambda\mathfrak{y}_\beta.\ \mathfrak{y}_\beta)\ (\mathfrak{f}, \beta\to\beta)$ (*?B* $=_o$ *?C*)
by *simp*
ultimately have $\vdash$ **S** $\{(\mathfrak{f}, \beta\to\beta) \rightarrowtail (\lambda\mathfrak{y}_\beta.\ \mathfrak{y}_\beta)\}$ (*?B* $=_o$ *?C*) (**is** $\vdash$ *?S*)
using *prop-5209* **by** *presburger*
moreover have *?S* =
(
$(\lambda\mathfrak{y}_\beta.\ \mathfrak{y}_\beta) =_{\beta\to\beta} \mathfrak{g}_{\beta\to\beta}) =_o \forall\mathfrak{x}_\beta.\ ((\lambda\mathfrak{y}_\beta.\ \mathfrak{y}_\beta) \cdot \mathfrak{x}_\beta =_\beta \mathfrak{g}_{\beta\to\beta} \cdot \mathfrak{x}_\beta$
) (**is** - = *?B′* $=_o$ *?C′*)
by *simp*
ultimately have $\vdash$ *?B′* $=_o$ *?C′*
by (*simp only*:)
moreover from ‹$(\lambda\mathfrak{y}_\beta.\ \mathfrak{y}_\beta) \in wffs_{\beta\to\beta}$› **have** *?B′* $\in wffs_o$ **and** *?C′* $\in wffs_o$
by *auto*
moreover have *is-free-for* $(\lambda\mathfrak{y}_\beta.\ \mathfrak{y}_\beta)\ (\mathfrak{g}, \beta\to\beta)$ (*?B′* $=_o$ *?C′*)
by *simp*
ultimately have $\vdash$ **S** $\{(\mathfrak{g}, \beta\to\beta) \rightarrowtail (\lambda\mathfrak{y}_\beta.\ \mathfrak{y}_\beta)\}$ (*?B′* $=_o$ *?C′*) (**is** $\vdash$ *?S′*)

using *prop-5209*[*OF* ‹$(\lambda\mathfrak{y}_\beta.\ \mathfrak{y}_\beta) \in wffs_{\beta\to\beta}$›] **by** *blast*
then show *?thesis*
by *simp*
qed
then have $\vdash (\lambda\mathfrak{x}_\beta.\ T_o) =_{\beta\to o} (\lambda\mathfrak{x}_\beta.\ (\mathfrak{x}_\beta =_\beta \mathfrak{x}_\beta))$
proof −
have $\lambda\mathfrak{y}_\beta.\ \mathfrak{y}_\beta \in wffs_{\beta\to\beta}$
by *blast*
then have $\vdash \lambda\mathfrak{y}_\beta.\ \mathfrak{y}_\beta =_{\beta\to\beta} \lambda\mathfrak{y}_\beta.\ \mathfrak{y}_\beta$
by (*fact prop-5200*)
with §*1* **have** $\vdash \forall\mathfrak{x}_\beta.\ ((\lambda\mathfrak{y}_\beta.\ \mathfrak{y}_\beta) \cdot \mathfrak{x}_\beta =_\beta (\lambda\mathfrak{y}_\beta.\ \mathfrak{y}_\beta) \cdot \mathfrak{x}_\beta)$
using *rule-R* **and** *is-subform-at.simps*(*1*) **by** *blast*
moreover have $\vdash (\lambda\mathfrak{y}_\beta.\ \mathfrak{y}_\beta) \cdot \mathfrak{x}_\beta =_\beta \mathfrak{x}_\beta$
using *axiom-4-2*[*OF wffs-of-type-intros*(*1*)] **by** (*rule axiom-is-derivable-from-no-hyps*)
ultimately have $\vdash \forall\mathfrak{x}_\beta.\ (\mathfrak{x}_\beta =_\beta (\lambda\mathfrak{y}_\beta.\ \mathfrak{y}_\beta) \cdot \mathfrak{x}_\beta)$
by (*rule rule-R*[**where** *p* = [»,«,«,»]]) *auto*
from *this* **and** ‹$\vdash (\lambda\mathfrak{y}_\beta.\ \mathfrak{y}_\beta) \cdot \mathfrak{x}_\beta =_\beta \mathfrak{x}_\beta$› **have** $\vdash \forall\mathfrak{x}_\beta.\ (\mathfrak{x}_\beta =_\beta \mathfrak{x}_\beta)$
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*: $\vdash (\lambda\mathfrak{x}_\beta.\ T_o) \cdot B =_o (\lambda\mathfrak{x}_\beta.\ (\mathfrak{x}_\beta =_\beta \mathfrak{x}_\beta)) \cdot B$
by (*rule Equality-Rules*(*5*))
then show *?thesis*
proof −
have $\vdash (\lambda\mathfrak{x}_\beta.\ T_o) \cdot B =_o T_o$
using *prop-5207*[*OF assms true-wff*] **by** *fastforce*
from *3* **and** *this* **have** $\vdash T_o =_o (\lambda\mathfrak{x}_\beta.\ (\mathfrak{x}_\beta =_\beta \mathfrak{x}_\beta)) \cdot B$
by (*rule rule-R*[**where** *p* = [«,»]]) *auto*
moreover have $\vdash (\lambda\mathfrak{x}_\beta.\ (\mathfrak{x}_\beta =_\beta \mathfrak{x}_\beta)) \cdot B =_o (B =_\beta B)$
proof −
have $\mathfrak{x}_\beta =_\beta \mathfrak{x}_\beta \in wffs_o$ **and** *is-free-for* B $(\mathfrak{x}, \beta)$ $(\mathfrak{x}_\beta =_\beta \mathfrak{x}_\beta)$
by (*blast*, *intro is-free-for-in-equality is-free-for-in-var*)
moreover have $\mathbf{S}\ \{(\mathfrak{x}, \beta) \rightarrowtail B\}\ (\mathfrak{x}_\beta =_\beta \mathfrak{x}_\beta) = (B =_\beta B)$
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 $\vdash (T_o \wedge^{\mathcal{Q}} T_o) =_o T_o$
proof −
have *const-T-wff*: $(\lambda x_o.\ T_o) \in wffs_{o\to o}$ **for** x

by *blast*
have §*1*: $\vdash (\lambda\mathfrak{y}_o.\ T_o) \bullet T_o \wedge^{\mathcal{Q}} (\lambda\mathfrak{y}_o.\ T_o) \bullet F_o =_o \forall\mathfrak{x}_o.\ (\lambda\mathfrak{y}_o.\ T_o) \bullet \mathfrak{x}_o$
proof −
have $\vdash \mathfrak{g}_{o\to o} \bullet T_o \wedge^{\mathcal{Q}} \mathfrak{g}_{o\to o} \bullet F_o =_o \forall\mathfrak{x}_o.\ \mathfrak{g}_{o\to o} \bullet \mathfrak{x}_o$ (**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*$_o$
by *auto*
moreover have *is-free-for* $(\lambda\mathfrak{y}_o.\ T_o)$ $(\mathfrak{g},\ o\to o)$ (*?B* $=_o$ *?C*)
by *simp*
ultimately have $\vdash$ **S** $\{(\mathfrak{g},\ o\to o) \rightarrowtail (\lambda\mathfrak{y}_o.\ T_o)\}$ (*?B* $=_o$ *?C*)
using *const-T-wff* **and** *prop-5209* **by** *presburger*
then show *?thesis*
by *simp*
qed
then have $\vdash T_o \wedge^{\mathcal{Q}} T_o =_o \forall\mathfrak{x}_o.\ T_o$
proof −
have *T-β-redex*: $\vdash (\lambda\mathfrak{y}_o.\ T_o) \bullet A =_o T_o$ **if** $A \in$ *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 $\vdash T_o \wedge^{\mathcal{Q}} (\lambda\mathfrak{y}_o.\ T_o) \bullet F_o =_o \forall\mathfrak{x}_o.\ (\lambda\mathfrak{y}_o.\ T_o) \bullet \mathfrak{x}_o$
by (*rule rule-R*[**where** p = [«,»,«,»]]) *force+*
from *this* **and** *T-β-redex*[*OF false-wff*] **have** $\vdash T_o \wedge^{\mathcal{Q}} T_o =_o \forall\mathfrak{x}_o.\ (\lambda\mathfrak{y}_o.\ T_o) \bullet \mathfrak{x}_o$
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 $\vdash T_o =_o \forall\mathfrak{x}_o.\ T_o$
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 $\vdash T_o$
unfolding *true-def* **using** *Q-wff* **by** (*rule prop-5200*)

6.13 Proposition 5212

proposition *prop-5212*:
shows $\vdash T_o \wedge^{\mathcal{Q}} T_o$
proof −
have $\vdash T_o$
by (*fact true-is-derivable*)
moreover have $\vdash (T_o \wedge^{\mathcal{Q}} T_o) =_o T_o$
by (*fact prop-5211*)
then have $\vdash T_o \equiv^{\mathcal{Q}} (T_o \wedge^{\mathcal{Q}} T_o)$
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 $\vdash (A =_{\alpha} B) \wedge^{\mathcal{Q}} (C =_{\beta} D)$
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 = [»,»]]) *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 = [»,»]]) *force+*
  then show *?thesis*
  proof –
    have $\vdash T_o \wedge^{\mathcal{Q}} T_o$
      by (*fact prop-5212*)
    from *this* **and** ‹$\vdash T_o =_o (A =_{\alpha} B)$› **have** $\vdash (A =_{\alpha} B) \wedge^{\mathcal{Q}} T_o$
      by (*rule rule-R*[**where** p = [«,»]]) *force+*
    from *this* **and** ‹$\vdash T_o =_o (C =_{\beta} D)$› **show** *?thesis*
      by (*rule rule-R*[**where** p = [»]]) *force+*
  qed
qed

6.15 Proposition 5214

proposition *prop-5214*:
  shows $\vdash T_o \wedge^{\mathcal{Q}} F_o =_o F_o$
proof –
  have *id-on-o-is-wff*: $(\lambda\mathfrak{x}_o.\ \mathfrak{x}_o) \in wffs_{o \to o}$
    by *blast*
  have §*1*: $\vdash (\lambda\mathfrak{x}_o.\ \mathfrak{x}_o) \cdot T_o \wedge^{\mathcal{Q}} (\lambda\mathfrak{x}_o.\ \mathfrak{x}_o) \cdot F_o =_o \forall\mathfrak{x}_o.\ (\lambda\mathfrak{x}_o.\ \mathfrak{x}_o) \cdot \mathfrak{x}_o$
  proof –
    have $\vdash \mathfrak{g}_{o \to o} \cdot T_o \wedge^{\mathcal{Q}} \mathfrak{g}_{o \to o} \cdot F_o =_o \forall\mathfrak{x}_o.\ \mathfrak{g}_{o \to o} \cdot \mathfrak{x}_o$ (**is** $\vdash$ *?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 $\vdash T_o \wedge^{\mathcal{Q}} F_o =_o \forall \mathfrak{x}_o.\ \mathfrak{x}_o$
proof –
have *id-β-redex*: $\vdash (\lambda\mathfrak{x}_o.\ \mathfrak{x}_o) \boldsymbol{\cdot} A =_o A$ **if** $A \in wffs_o$ **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 $\vdash T_o \wedge^{\mathcal{Q}} (\lambda\mathfrak{x}_o.\ \mathfrak{x}_o) \boldsymbol{\cdot} F_o =_o \forall \mathfrak{x}_o.\ (\lambda\mathfrak{x}_o.\ \mathfrak{x}_o) \boldsymbol{\cdot} \mathfrak{x}_o$
by (*rule rule-R*[**where** p = [«,»,«,»]]) *force+*
from *this* **and** *id-β-redex*[*OF false-wff*] **have** $\vdash T_o \wedge^{\mathcal{Q}} F_o =_o \forall \mathfrak{x}_o.\ (\lambda\mathfrak{x}_o.\ \mathfrak{x}_o) \boldsymbol{\cdot} \mathfrak{x}_o$
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 $\mathcal{H} \vdash \forall x_\alpha.\ B$ **and** $A \in wffs_\alpha$
and *is-free-for* A (x, α) B
shows $\mathcal{H} \vdash \mathbf{S}\ \{(x, \alpha) \rightarrowtail A\}\ B$
proof –
from *assms*(*1*) **have** *is-hyps* $\mathcal{H}$
by (*blast elim*: *is-derivable-from-hyps.cases*)
from *assms*(*1*) **have** $\mathcal{H} \vdash (\lambda\mathfrak{x}_\alpha.\ T_o) =_{\alpha\to o} (\lambda x_\alpha.\ B)$
by *simp*
with *assms*(*2*) **have** $\mathcal{H} \vdash (\lambda\mathfrak{x}_\alpha.\ T_o) \boldsymbol{\cdot} A =_o (\lambda x_\alpha.\ B) \boldsymbol{\cdot} A$
by (*intro Equality-Rules*(*5*))
then have $\mathcal{H} \vdash T_o =_o \mathbf{S}\ \{(x, \alpha) \rightarrowtail A\}\ B$
proof –
have $\mathcal{H} \vdash (\lambda\mathfrak{x}_\alpha.\ T_o) \boldsymbol{\cdot} A =_o T_o$
proof –
have $\vdash (\lambda\mathfrak{x}_\alpha.\ T_o) \boldsymbol{\cdot} A =_o T_o$
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* $\mathcal{H}$› **show** *?thesis*
by (*rule derivability-implies-hyp-derivability*)
qed
moreover have $\mathcal{H} \vdash (\lambda x_\alpha.\ B) \boldsymbol{\cdot} A =_o \mathbf{S}\ \{(x, \alpha) \rightarrowtail A\}\ B$
proof –
have $B \in wffs_o$
using *hyp-derivable-form-is-wffso*[*OF assms*(*1*)] **by** (*fastforce elim*: *wffs-from-forall*)
with *assms*(*2*,*3*) **have** $\vdash (\lambda x_\alpha.\ B) \boldsymbol{\cdot} A =_o \mathbf{S}\ \{(x, \alpha) \rightarrowtail A\}\ B$
using *prop-5207* **by** (*simp only*:)
from *this* **and** ‹*is-hyps* $\mathcal{H}$› **show** *?thesis*
by (*rule derivability-implies-hyp-derivability*)

qed
ultimately show *?thesis*
using ‹$\mathcal{H} \vdash (\lambda \mathfrak{x}_\alpha.\ T_o) \boldsymbol{\cdot} A =_o (\lambda x_\alpha.\ B) \boldsymbol{\cdot} A$› **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** ‹$\mathcal{H} \vdash T_o =_o$ **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 $A \in wffs_o$
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*: $?B \in wffs_{o \to o}$
by *auto*
have §*1*: $\vdash ?B \boldsymbol{\cdot} T_o \wedge^{\mathcal{Q}} ?B \boldsymbol{\cdot} F_o =_o \forall \mathfrak{x}_o.\ ?B \boldsymbol{\cdot} \mathfrak{x}_o$
proof −
have $\vdash \mathfrak{g}_{o \to o} \boldsymbol{\cdot} T_o \wedge^{\mathcal{Q}} \mathfrak{g}_{o \to o} \boldsymbol{\cdot} F_o =_o \forall \mathfrak{x}_o.\ \mathfrak{g}_{o \to o} \boldsymbol{\cdot} \mathfrak{x}_o$ (**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 $\vdash$ **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*: $\vdash ?B \boldsymbol{\cdot} A =_o (T_o \wedge^{\mathcal{Q}} A =_o A)$ **if** $A \in wffs_o$ **for** A
proof −
have $T_o \wedge^{\mathcal{Q}} \mathfrak{x}_o =_o \mathfrak{x}_o \in wffs_o$
by *blast*
moreover have 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 $\vdash (T_o \wedge^{\mathcal{Q}} T_o =_o T_o) \wedge^{\mathcal{Q}}$ *?B* $\bullet F_o =_o \forall \mathfrak{x}_o.$ *?B* $\bullet \mathfrak{x}_o$
by (*rule rule-R*[**where** p = [«,»,«,»]]) *force+*
from *this* **and** *B-β-redex*[*OF false-wff*]
have $\vdash (T_o \wedge^{\mathcal{Q}} T_o =_o T_o) \wedge^{\mathcal{Q}} (T_o \wedge^{\mathcal{Q}} F_o =_o F_o) =_o \forall \mathfrak{x}_o.$ *?B* $\bullet \mathfrak{x}_o$
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 $\vdash \forall \mathfrak{x}_o. (T_o \wedge^{\mathcal{Q}} \mathfrak{x}_o =_o \mathfrak{x}_o)$
by (*rule rule-R*[**where** p = []]) *fastforce+*
show *?thesis*
using $\forall I$[*OF* ‹$\vdash \forall \mathfrak{x}_o. (T_o \wedge^{\mathcal{Q}} \mathfrak{x}_o =_o \mathfrak{x}_o)$› *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* $\bullet T_o \wedge^{\mathcal{Q}}$ *?B* $\bullet F_o =_o \forall \mathfrak{x}_o.$ *?B* $\bullet \mathfrak{x}_o$
proof −
have $\vdash \mathfrak{g}_{o \to o} \bullet T_o \wedge^{\mathcal{Q}} \mathfrak{g}_{o \to o} \bullet F_o =_o \forall \mathfrak{x}_o. \mathfrak{g}_{o \to o} \bullet \mathfrak{x}_o$ (**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 $\vdash$ **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 $\vdash (T_o =_o T_o) \wedge^{\mathcal{Q}} (T_o =_o F_o) =_o \forall \mathfrak{x}_o. (T_o =_o \mathfrak{x}_o)$ (**is** $\vdash$ *?A*)
proof −
have *B-β-redex*: $\vdash$ *?B* $\bullet A =_o (T_o =_o A)$ **if** $A \in wffs_o$ **for** A
proof −
have $T_o =_o \mathfrak{x}_o \in wffs_o$
by *auto*
moreover have 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** $\vdash (T_o =_o T_o) \wedge^{\mathcal{Q}} ?B \cdot F_o =_o \forall \mathfrak{x}_o.\ ?B \cdot \mathfrak{x}_o$
by (*rule rule-R*[**where** p = [«,»,«,»]]) *force+*
from *this* **and** *B-β-redex*[*OF false-wff*]
have $\vdash (T_o =_o T_o) \wedge^{\mathcal{Q}} (T_o =_o F_o) =_o \forall \mathfrak{x}_o.\ ?B \cdot \mathfrak{x}_o$
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** $\vdash T_o \wedge^{\mathcal{Q}} (T_o =_o F_o) =_o \forall \mathfrak{x}_o.\ (T_o =_o \mathfrak{x}_o)$
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 $\vdash \mathbf{S}\ \{(\mathfrak{f},\ o\to o) \rightarrowtail (\lambda\mathfrak{x}_o.\ T_o)\}\ (?C =_o ?D)$
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 ‹$(\lambda\mathfrak{x}_o.\ T_o) \in wffs_{o\to o}$› **by** *auto*
ultimately have $\vdash \mathbf{S}\ \{(\mathfrak{g},\ o\to o) \rightarrowtail (\lambda\mathfrak{x}_o.\ \mathfrak{x}_o)\}\ (?C' =_o ?D')$
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 $A \in wffs_o$
shows $\vdash (T_o =_o A) =_o A$
proof −
let $?B = \lambda\mathfrak{x}_o.\ ((T_o =_o \mathfrak{x}_o) =_o \mathfrak{x}_o)$
have *B-is-wff*: $?B \in wffs_{o \to o}$
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 $?C \in wffs_o$ **and** $?D \in wffs_o$ **and** *is-free-for* $?B$ $(\mathfrak{g},\ o\to o)$ $(?C =_o\ ?D)$
by *auto*
ultimately have $\vdash$ **S** $\{(\mathfrak{g},\ o\to o) \rightarrowtail\ ?B\}$ $(?C =_o\ ?D)$
using *prop-5209*[*OF B-is-wff*] **by** *presburger*
then show *?thesis*
by *simp*
qed
have $*$: *is-free-for* A $(\mathfrak{x},\ o)$ $((T_o =_o \mathfrak{x}_o) =_o \mathfrak{x}_o)$ **for** A
by (*intro is-free-for-in-equality is-free-for-in-true is-free-for-in-var*)
have §*2*:
$\vdash$
$((T_o =_o T_o) =_o T_o) \wedge^{\mathcal{Q}} ((T_o =_o F_o) =_o F_o)$
$=_o$
$\forall\mathfrak{x}_o.\ ((T_o =_o \mathfrak{x}_o) =_o \mathfrak{x}_o)$
proof −
have *B-β-redex*: $\vdash\ ?B \cdot A =_o ((T_o =_o A) =_o A)$ **if** $A \in wffs_o$ **for** A
proof −
have $(T_o =_o \mathfrak{x}_o) =_o \mathfrak{x}_o \in wffs_o$
by *auto*
moreover have S $\{(\mathfrak{x},\ o) \rightarrowtail A\}$ $((T_o =_o \mathfrak{x}_o) =_o \mathfrak{x}_o) = ((T_o =_o A) =_o A)$
by *simp*
ultimately show *?thesis*
using $*$ **and** *prop-5207*[*OF that*] **by** *metis*
qed
from §*1* **and** *B-β-redex*[*OF true-wff*]
have $\vdash ((T_o =_o T_o) =_o T_o) \wedge^{\mathcal{Q}}\ ?B \cdot F_o =_o \forall\mathfrak{x}_o.\ ?B \cdot \mathfrak{x}_o$
by (*rule rule-R*[**where** p = [«,»,«,»]]) *force+*
from *this* **and** *B-β-redex*[*OF false-wff*]
have $\vdash ((T_o =_o T_o) =_o T_o) \wedge^{\mathcal{Q}} ((T_o =_o F_o) =_o F_o) =_o \forall\mathfrak{x}_o.\ ?B \cdot \mathfrak{x}_o$
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
  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_α) (x, α) T_o
by *auto*
then have $\vdash (\lambda x_\alpha.\ T_o) =_{\alpha \to o} \lambda z_\alpha.\ \mathbf{S}\ \{(x, \alpha) \rightarrowtail z_\alpha\}\ T_o$
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 = [», «]])
case *no-capture*
have ∗: [»,«] ∈ *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* ∩ *free-vars* $\mathcal{H}$ = {}
shows $\mathcal{H} \vdash \forall^{\mathcal{Q}}{}_{\star}$ *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* $\cap$ *free-vars* $\mathcal{H}$ = {} **and** $(x, \alpha) \notin$ *free-vars* $\mathcal{H}$
by *simp-all*
from ‹*lset vs* $\cap$ *free-vars* $\mathcal{H}$ = {}› **have** $\mathcal{H} \vdash \forall^{\mathcal{Q}}{}_{\star}$ *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 $A \in wffs_\alpha$
shows $\mathcal{H} \vdash$ **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 $\forall\, v \in$ *fmdom′* ϑ. *var-name v* $\notin$ *free-var-names* $\mathcal{H}$ $\wedge$ *is-free-for* (ϑ \$\$! v) v B
and $\vartheta \neq$ {\$\$}
shows $\mathcal{H} \vdash$ **S** ϑ 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* $\cup$ *vars* (*lset As*) $\cup$ *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*›)
— $\mathcal{H} \vdash \mathbf{S}^{x^1_{\alpha_1}\ \dots\ x^k_{\alpha_k}\ x^{k+1}_{\alpha_{k+1}}\ \dots\ x^n_{\alpha_n}}_{A^1_{\alpha_1}\ \dots\ A^k_{\alpha_k}\ y^{k+1}_{\alpha_{k+1}}\ \dots\ y^n_{\alpha_n}}\ B$
have $\mathcal{H}$ ⊢ **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 $\mathcal{H}$ ⊢ **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 $?\vartheta$ = *fmap-of-list* (*zip xs* (*map FVar ys*))
from *Cons.hyps*(*1*) **have** *lset xs* = *fmdom′* $?\vartheta$
by *simp*
from *Cons.hyps*(*1*) **and** *Cons.prems*(*6*) **have** *fmran′* $?\vartheta$ = *FVar* ‘ *lset ys*
by *force*
have *is-substitution* $?\vartheta$
unfolding *is-substitution-def* **proof**
fix *v*
assume $v \in$ *fmdom′* $?\vartheta$
with ‹*lset xs* = *fmdom′* $?\vartheta$› **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** $?\vartheta$ $$! *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** $?\vartheta$ \$\$! $v = y_\beta$
  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* $(x, \alpha) \Rightarrow ?\vartheta$ \$\$! $(x, \alpha) \in wffs_\alpha$
  using ‹$?\vartheta$ \$\$! $v = y_\beta$› **and** ‹*var-type* $v = \alpha$› **by** *fastforce*
qed
have $v_x \notin fmdom'\ ?\vartheta$
  using *Cons.prems*(*6*) **and** ‹*lset xs* = *fmdom'* $?\vartheta$› **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'* $?\vartheta$. *var-name* v ∉ *free-var-names* $\mathcal{H}$ ∧ *is-free-for* ($?\vartheta$ \$\$! v) v B
proof
  fix v

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

by (*fastforce intro*: *fmap.map-ident-strong*)
with ‹$v_x \notin$ *fmdom′ ?ϑ*› **show** *?thesis*
using ‹$\forall\, v \in$ *fmdom′ ?ϑ*. *var-name v* $\notin$ *free-var-names* $\mathcal{H} \wedge$ *is-free-for* (*?ϑ* \$\$! *v*) *v B*›
and *substitution-consolidation* **by** *auto*
qed
— $\mathcal{H} \vdash \text{Ş}^{x^1_{\alpha_1} \dots x^n_{\alpha_n}}_{y^1_{\alpha_1} \dots y^n_{\alpha_n}} B$
ultimately show *?case*
using ‹$v_x = (x, \alpha)$› **and** ‹$v_x \notin$ *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* $\in$ *lset ys*
using *Suc.prems* ‹*length ys* = *length xs*› **by** *simp*
have $\forall\, j <$ *length ys*. *ys* ! *j* $\notin$ *vars* ($\mathcal{H}$::*form set*) $\wedge$ *ys* ! *j* $\notin$ *vars B*
using ‹(*var-name* ‘ *lset ys*) $\cap$ (*var-name* ‘ (*vars B* $\cup$ *vars* (*lset As*) $\cup$ *vars* $\mathcal{H}$ $\cup$ *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* $\in$ 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* $\in$ *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* $(x, \alpha) \Rightarrow$ (*?ϑ k*) \$\$! $(x, \alpha) \in \mathit{wffs}_\alpha$
proof (*cases j* < *k*)

    case *True*
    with ‹$j <$ *length xs*› **and** ‹$v =$ *xs* ! j› **have** (*?ϑ k*) $$! $v =$ *As* ! j
      using ‹*distinct xs*› **and** ‹*length xs* = *length As*› **and** ‹*length ys* = *length xs*› **by** *force*
    with *assms*(*2*) ‹$v =$ *xs* ! j› **and** ‹$v \in$ *fmdom′* (*?ϑ k*)› **and** ‹*var-type* $v = \alpha$› **and** ‹$j <$ *length xs*›
    have (*?ϑ k*) $$! $v \in$ *wffs*$_\alpha$
      using ‹*As* = *map* (($$!) *ϑ*) *xs*› **and** ‹*fmdom′* (*?ϑ k*) = *lset xs*› **and** ‹*lset xs* = *fmdom′ ϑ*›
      by *auto*
    then show *?thesis*
      using ‹*var-type* $v = \alpha$› **by** *force*
  next
    case *False*
    with ‹$j <$ *length xs*› **and** ‹$v =$ *xs* ! j› **have** (*?ϑ k*) $$! $v =$ *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* $v = \alpha$› **and** ‹*length ys* = *length xs*›
    have (*?ϑ k*) $$! $v \in$ *wffs*$_\alpha$
      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* $v = \alpha$› **by** *force*
  qed
qed
have *ϑ′-alt-def*: *?ϑ′ k* = *fmap-of-list*
  (*zip xs*
    (*take k* (*map* ($\lambda A'$. **S** {*?y* ↣ *?A*} A') *As*)
    @
    (*drop k* (*map* ($\lambda A'$. **S** {*?y* ↣ *?A*} A') (*map FVar ys*)))))
proof −
  have
    fmap-of-list (*zip xs* (*map* ($\lambda A'$. **S** {*?y* ↣ *?A*} A') (*take k As* @ *drop k* (*map FVar ys*))))
    =
    fmap-of-list
      (*zip xs*
        (*map* ($\lambda A'$. **S** {*?y* ↣ *?A*} A') (*take k As*)
        @
        (*drop k* (*map* ($\lambda A'$. **S** {*?y* ↣ *?A*} A') (*map FVar ys*)))))
    by (*simp add*: *drop-map*)
  then show *?thesis*
    by (*metis take-map zip-map2*)
qed
— $\mathcal{H} \vdash \mathsf{S}^{x^1_{\alpha_1} \ldots x^k_{\alpha_k} x^{k+1}_{\alpha_{k+1}} \ldots x^n_{\alpha_n}}_{A^1_{\alpha_1} \ldots A^k_{\alpha_k} y^{k+1}_{\alpha_{k+1}} \ldots y^n_{\alpha_n}} B$
have $\mathcal{H} \vdash$ **S** (*?ϑ k*) B
  by (*fact Suc.IH*[*OF Suc-leD*[*OF Suc.prems*]])
— $\mathcal{H} \vdash \mathsf{S}^{y^{k+1}_{\alpha_{k+1}}}_{A^{k+1}_{\alpha_{k+1}}} \mathsf{S}^{x^1_{\alpha_1} \ldots x^k_{\alpha_k} x^{k+1}_{\alpha_{k+1}} \ldots x^n_{\alpha_n}}_{A^1_{\alpha_1} \ldots A^k_{\alpha_k} y^{k+1}_{\alpha_{k+1}} \ldots y^n_{\alpha_n}} B$
then have $\mathcal{H} \vdash$ **S** {*?y* ↣ *?A*} **S** (*?ϑ k*) B
proof −
  from ‹$(n_y, \alpha_y) =$ *?y*› **and** ‹*length ys* = *length xs*› **have** $(n_y, \alpha_y) \notin$ *free-vars* $\mathcal{H}$
    using ‹$\forall j <$ *length ys*. *ys* ! $j \notin$ *vars* ($\mathcal{H}$::*form set*) $\wedge$ *ys* ! $j \notin$ *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* $\mathcal{H}$ ∧ *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* ‹$\mathcal{H} \vdash$ **S** (*?ϑ k*) *B*›] **and** ‹*?A* $\in$ *wffs*$_{\alpha_y}$› **and** ‹(n_y, α_y) = *?y*› **by** *metis*
qed
— $\mathsf{S}^{y^{k+1}_{\alpha_{k+1}}}_{A^{k+1}_{\alpha_{k+1}}} \mathsf{S}^{x^1_{\alpha_1} \dots x^k_{\alpha_k} x^{k+1}_{\alpha_{k+1}} \dots x^n_{\alpha_n}}_{A^1_{\alpha_1} \dots A^k_{\alpha_k} y^{k+1}_{\alpha_{k+1}} \dots y^n_{\alpha_n}} B = \mathsf{S}^{x^1_{\alpha_1} \dots x^k_{\alpha_k} x^{k+1}_{\alpha_{k+1}} x^{k+2}_{\alpha_{k+2}} \dots x^n_{\alpha_n}}_{A^1_{\alpha_1} \dots A^k_{\alpha_k} A^{k+1}_{\alpha_{k+1}} y^{k+2}_{\alpha_{k+2}} \dots y^n_{\alpha_n}} B$
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* (λ*A'*. **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* $\mathcal{H}$ ∧ *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
         ?ϑ' k $$ (xs ! j) = Some (drop k (map (λA'. S {?y ↣ ?A} A') (map FVar ys)) ! (j − k))
          using fmap-of-list-nth-split and ‹length xs = length As› and ‹length ys = length xs›
          by simp
        also from Suc.prems and c have … = Some (S {?y ↣ ?A} (FVar (ys ! j)))
          using ‹length ys = length xs› by simp
       also from Suc-le-lessD[OF Suc.prems] and ‹distinct ys› have … = Some (FVar (ys ! j))
          using ‹j < length xs› and ‹k < j› and ‹length ys = length xs›
          by (metis nless-le nth-eq-iff-index-eq prod.exhaust-sel singleton-substitution-simps(1))
        also from c and ‹distinct xs› have … = ?ϑ (Suc k) $$ v'
          using ‹xs ! j = v'› and ‹length xs = length As› and ‹length ys = length xs› by force
        finally show ?thesis
          using ‹?ϑ' k $$ v' = Some A'› and ‹xs ! j = v'› by force
      qed
    }
    note ϑ-k-in-Sub-k = this
    {
      fix v' and A'
      assume ?ϑ (Suc k) $$ v' = Some A'
      then have v' ∈ fmdom' (?ϑ (Suc k))
        by (intro fmdom'I)
      then obtain j where j < length xs and xs ! j = v'
        using ‹fmdom' (?ϑ (Suc k)) = lset xs› by (metis in-set-conv-nth)
      then consider (a) j < k | (b) j = k | (c) j ∈ {k<..< length xs}
        by fastforce
      with ‹j < length xs› and ‹xs ! j = v'› and ϑ-k-in-Sub-k show ?ϑ' k $$ v' = Some A'
        using ‹⋀k'. fmdom' (?ϑ' k') = lset xs› and ‹?ϑ (Suc k) $$ v' = Some A'›
        by (metis (mono-tags, lifting) fmlookup-dom'-iff nth-mem)+
    }
  qed
  then show ?thesis
    by presburger
qed
also have … = S (?ϑ (Suc k)) B
proof −
  have ?ϑ (Suc k) $$ ?y = None
    using ‹?y ∈ lset ys› ‹⋀k'. fmdom' (?ϑ k') = lset xs› and ys-fresh by blast
  moreover from Suc-le-lessD[OF Suc.prems] have ?y ∉ vars B
    using ‹∀j < length ys. ys ! j ∉ vars ℋ ∧ ys ! j ∉ vars B› and ‹length ys = length xs›
    by auto
  ultimately show ?thesis
    by (rule substitution-absorption)
qed
finally show ?thesis .
qed
```

— $\mathcal{H} \vdash \underset{\cdot}{\mathrm{S}}\,{}^{x^1_{\alpha_1}\,\dots\,x^k_{\alpha_k}\,x^{k+1}_{\alpha_{k+1}}\,x^{k+2}_{\alpha_{k+2}}\,\dots\,x^n_{\alpha_n}}_{A^1_{\alpha_1}\,\dots\,A^k_{\alpha_k}\,A^{k+1}_{\alpha_{k+1}}\,y^{k+2}_{\alpha_{k+2}}\,\dots\,y^n_{\alpha_n}}\; B$

ultimately show *?case*
by (*simp only*:)
qed
— $\mathcal{H} \vdash \mathbf{S}\,^{x^1_{\alpha_1}\ \dots\ x^n_{\alpha_n}}_{A^1_{\alpha_1}\ \dots\ A^n_{\alpha_n}}\ B$
then have $\mathcal{H} \vdash$ **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 $A \in wffs_o$
and $(z, \beta) \notin$ *free-vars A*
and *is-free-for* (z_β) (x, β) *A*
shows $\vdash \forall x_\beta.\ A =_o \forall z_\beta.$ **S** $\{(x, \beta) \rightarrowtail z_\beta\}$ *A*
proof −
from *assms*(*1*) **have** $\forall x_\beta.\ A \in wffs_o$
by (*intro forall-wff*)
then have $\vdash \forall x_\beta.\ A =_o \forall x_\beta.\ A$
by (*fact prop-5200*)
moreover from *assms* **have** $\vdash (\lambda x_\beta.\ A) =_{\beta \to o} (\lambda z_\beta.$ **S** $\{(x, \beta) \rightarrowtail z_\beta\}$ $A)$
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*: $\mathcal{H} \vdash T_o =_o (\lambda x_o.\ A) \bullet T_o$
proof –
have $\vdash (\lambda x_o.\ A) \bullet T_o =_o \mathbf{S}\ \{(x,\ o) \rightarrowtail T_o\}\ A$
using *prop-5207*[*OF true-wff assms*(*3*) *closed-is-free-for*] **by** *simp*
from *this* **and** *assms*(*1*) **have** $\mathcal{H} \vdash (\lambda x_o.\ A) \bullet T_o$
using *rule-RR*[*OF disjI2*, **where** p = []] **by** *fastforce*
moreover have $(\lambda x_o.\ A) \bullet T_o \in \mathit{wffs}_o$
by (*fact hyp-derivable-form-is-wffso*[*OF* ‹$\mathcal{H} \vdash (\lambda x_o.\ A) \bullet T_o$›])
ultimately show *?thesis*
using *rule-T*(*1*) **by** *blast*
qed
moreover have §*2*: $\mathcal{H} \vdash T_o =_o (\lambda x_o.\ A) \bullet F_o$
proof –
have $\vdash (\lambda x_o.\ A) \bullet F_o =_o \mathbf{S}\ \{(x,\ o) \rightarrowtail F_o\}\ A$
using *prop-5207*[*OF false-wff assms*(*3*) *closed-is-free-for*] **by** *simp*
from *this* **and** *assms*(*2*) **have** $\mathcal{H} \vdash (\lambda x_o.\ A) \bullet F_o$
using *rule-RR*[*OF disjI2*, **where** p = []] **by** *fastforce*
moreover have $(\lambda x_o.\ A) \bullet F_o \in \mathit{wffs}_o$
by (*fact hyp-derivable-form-is-wffso*[*OF* ‹$\mathcal{H} \vdash (\lambda x_o.\ A) \bullet F_o$›])
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 $\mathcal{H} \vdash (\lambda x_o.\ A) \bullet T_o \wedge^{\mathcal{Q}} (\lambda x_o.\ A) \bullet F_o$
proof –
from §*3* **and** §*1* **have** $\mathcal{H} \vdash (\lambda x_o.\ A) \bullet T_o \wedge^{\mathcal{Q}} T_o$
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 $\vdash (\lambda x_o.\ A) \bullet T_o \wedge^{\mathcal{Q}} (\lambda x_o.\ A) \bullet F_o =_o \forall x_o.\ A$
proof –
have $\mathfrak{g}_{o \to o} \bullet \mathfrak{x}_o \in \mathit{wffs}_o$
by *blast*
have $\vdash \mathfrak{g}_{o \to o} \bullet T_o \wedge^{\mathcal{Q}} \mathfrak{g}_{o \to o} \bullet F_o =_o \forall \mathfrak{x}_o.\ \mathfrak{g}_{o \to o} \bullet \mathfrak{x}_o$
using *axiom-1*[*unfolded equivalence-def*] **by** (*rule axiom-is-derivable-from-no-hyps*)
— By α-conversion
then have $\vdash \mathfrak{g}_{o \to o} \bullet T_o \wedge^{\mathcal{Q}} \mathfrak{g}_{o \to o} \bullet F_o =_o \forall x_o.\ \mathfrak{g}_{o \to o} \bullet x_o$ (**is** $\vdash$ *?B* $=_o$ *?C*)
proof –
have $\vdash \forall \mathfrak{x}_o.\ \mathfrak{g}_{o \to o} \bullet \mathfrak{x}_o =_o \forall x_o.\ \mathfrak{g}_{o \to o} \bullet x_o$

```
    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 ∧^𝒬 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 ∧^𝒬 (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 $?A = (\lambda\mathfrak{x}_o.\ \lambda\mathfrak{y}_o.\ (\mathfrak{x}_o \equiv^{\mathcal{Q}} \mathfrak{x}_o \wedge^{\mathcal{Q}} \mathfrak{y}_o)) \boldsymbol{\cdot} T_o \boldsymbol{\cdot} \mathfrak{y}_o$
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 $\vdash (\lambda\mathfrak{x}_o.\ \lambda\mathfrak{y}_o.\ (\mathfrak{x}_o \equiv^{\mathcal{Q}} \mathfrak{x}_o \wedge^{\mathcal{Q}} \mathfrak{y}_o)) \boldsymbol{\cdot} T_o =_{o\to o} \lambda\mathfrak{y}_o.\ (T_o \equiv^{\mathcal{Q}} T_o \wedge^{\mathcal{Q}} \mathfrak{y}_o)$
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 $\mathbf{S}\ \{(\mathfrak{x},\ o) \rightarrowtail T_o\}\ (\lambda\mathfrak{y}_o.\ (\mathfrak{x}_o \equiv^{\mathcal{Q}} \mathfrak{x}_o \wedge^{\mathcal{Q}} \mathfrak{y}_o)) = (\lambda\mathfrak{y}_o.\ (T_o \equiv^{\mathcal{Q}} T_o \wedge^{\mathcal{Q}} \mathfrak{y}_o))$
by *simp*
ultimately show *?thesis*
using *prop-5207*[*OF true-wff*] **by** *metis*
qed
ultimately have $*$: $\vdash (T_o \supset^{\mathcal{Q}} \mathfrak{y}_o) =_o (\lambda\mathfrak{y}_o.\ (T_o \equiv^{\mathcal{Q}} T_o \wedge^{\mathcal{Q}} \mathfrak{y}_o)) \boldsymbol{\cdot} \mathfrak{y}_o$
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 $\vdash (\lambda\mathfrak{y}_o.\ (T_o \equiv^{\mathcal{Q}} T_o \wedge^{\mathcal{Q}} \mathfrak{y}_o)) \boldsymbol{\cdot} \mathfrak{y}_o =_o (T_o \equiv^{\mathcal{Q}} T_o \wedge^{\mathcal{Q}} \mathfrak{y}_o)$
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 $A \in wffs_o$
shows $\vdash (T_o \supset^{\mathcal{Q}} A) =_o A$
proof −
have $T_o \supset^{\mathcal{Q}} \mathfrak{y}_o \in wffs_o$ **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** $\vdash$ **S** $\{(\mathfrak{y}, o) \rightarrowtail A\}$ $((T_o \supset^{\mathcal{Q}} \mathfrak{y}_o) =_o \mathfrak{y}_o)$
by (*rule prop-5209*[*OF assms* ‹$T_o \supset^{\mathcal{Q}} \mathfrak{y}_o \in wffs_o$› *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** $A \in wffs_o$
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 $?\vartheta = \{(\mathfrak{y}, o) \rightarrowtail B\}$
have $B \in wffs_o$
by (*fact hyp-derivable-form-is-wffso*[*OF assms*(*2*), *THEN wffs-from-imp-op*(*2*)])
then have *is-substitution* $?\vartheta$
by *simp*
moreover have
$\forall v \in fmdom'\ ?\vartheta.$
var-name $v \notin$ *free-var-names* ({}::*form set*) $\wedge$
is-free-for ($?\vartheta$ \$\$! 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 $?\vartheta \neq$ {\$\$}
by *simp*
ultimately have $\vdash$ **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 $\vdash \prod{}_{\alpha} \bullet \mathfrak{f}_{\alpha \to o} \supset^{\mathcal{Q}} \mathfrak{f}_{\alpha \to o} \bullet \mathfrak{x}_{\alpha}$
proof −
have $\mathfrak{f}_{\alpha \to o} \bullet \mathfrak{x}_{\alpha} \in wffs_o$
by *blast*
have §*1*:
$\vdash$
$\prod{}_{\alpha} \bullet \mathfrak{f}_{\alpha \to o} \supset^{\mathcal{Q}} (((\lambda \mathfrak{f}_{\alpha \to o}.\ \mathfrak{f}_{\alpha \to o} \bullet \mathfrak{x}_{\alpha}) \bullet (\lambda \mathfrak{x}_{\alpha}.\ T_o))$
$=_o$
$((\lambda \mathfrak{f}_{\alpha \to o}.\ \mathfrak{f}_{\alpha \to o} \bullet \mathfrak{x}_{\alpha}) \bullet \mathfrak{f}_{\alpha \to o}))$
proof −
let $?\vartheta = \{(\mathfrak{h}, (\alpha \to o) \to o) \rightarrowtail \lambda \mathfrak{f}_{\alpha \to o}.\ \mathfrak{f}_{\alpha \to o} \bullet \mathfrak{x}_{\alpha}, (\mathfrak{x}, \alpha \to o) \rightarrowtail \lambda \mathfrak{x}_{\alpha}.\ T_o, (\mathfrak{y}, \alpha \to o) \rightarrowtail \mathfrak{f}_{\alpha \to o}\}$
and $?A = (\mathfrak{x}_{\alpha \to o} =_{\alpha \to o} \mathfrak{y}_{\alpha \to o}) \supset^{\mathcal{Q}} (\mathfrak{h}_{(\alpha \to o) \to o} \bullet \mathfrak{x}_{\alpha \to o} \equiv^{\mathcal{Q}} \mathfrak{h}_{(\alpha \to o) \to o} \bullet \mathfrak{y}_{\alpha \to o})$
have $\vdash ?A$
by (*fact axiom-is-derivable-from-no-hyps*[*OF axiom-2*])
moreover have $\lambda \mathfrak{f}_{\alpha \to o}.\ \mathfrak{f}_{\alpha \to o} \bullet \mathfrak{x}_{\alpha} \in wffs_{(\alpha \to o) \to o}$ **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* $?\vartheta$
by *simp*
moreover have
$\forall v \in fmdom'\ ?\vartheta.\ var\text{-}name\ v \notin free\text{-}var\text{-}names\ (\{\}::form\ set) \wedge is\text{-}free\text{-}for\ (?\vartheta\ \$\$!\ v)\ v\ ?A$
by (*code-simp*, (*unfold atomize-conj*[*symmetric*])?, *simp*)+ *blast*
moreover have $?\vartheta \neq \{\$\$\}$
by *simp*
ultimately have $\vdash$ **S** $?\vartheta\ ?A$
by (*rule Sub*)
then show *?thesis*
by *simp*
qed
have $\vdash \prod_\alpha \bullet \mathfrak{f}_{\alpha\to o} \supset^{\mathcal{Q}} (T_o =_o \mathfrak{f}_{\alpha\to o} \bullet \mathfrak{x}_\alpha)$
proof −
have
$\vdash (\lambda\mathfrak{f}_{\alpha\to o}.\ \mathfrak{f}_{\alpha\to o} \bullet \mathfrak{x}_\alpha) \bullet (\lambda\mathfrak{x}_\alpha.\ T_o) =_o (\lambda\mathfrak{x}_\alpha.\ T_o) \bullet \mathfrak{x}_\alpha$
(**is** $\vdash (\lambda ?x_{?\beta}.\ ?B) \bullet ?A =_o ?C$)
proof −
have $\vdash (\lambda ?x_{?\beta}.\ ?B) \bullet ?A =_o$ **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 $\vdash (\lambda\mathfrak{x}_\alpha.\ T_o) \bullet \mathfrak{x}_\alpha =_o T_o$
using *prop-5208*[**where** $vs = [(\mathfrak{x},\ \alpha)]$] **and** *true-wff* **by** *simp*
ultimately have $\vdash (\lambda\mathfrak{f}_{\alpha\to o}.\ \mathfrak{f}_{\alpha\to o} \bullet \mathfrak{x}_\alpha) \bullet (\lambda\mathfrak{x}_\alpha.\ T_o) =_o T_o$
by (*rule Equality-Rules*(*3*))
from §*1* **and** *this* **have** $\vdash \prod_\alpha \bullet \mathfrak{f}_{\alpha\to o} \supset^{\mathcal{Q}} (T_o =_o ((\lambda\mathfrak{f}_{\alpha\to o}.\ \mathfrak{f}_{\alpha\to o} \bullet \mathfrak{x}_\alpha) \bullet \mathfrak{f}_{\alpha\to o}))$
by (*rule rule-R*[**where** p = [»,«,»]]) *force+*
moreover have $\vdash (\lambda\mathfrak{f}_{\alpha\to o}.\ \mathfrak{f}_{\alpha\to o} \bullet \mathfrak{x}_\alpha) \bullet \mathfrak{f}_{\alpha\to o} =_o \mathfrak{f}_{\alpha\to o} \bullet \mathfrak{x}_\alpha$
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* ‹$\mathfrak{f}_{\alpha\to o} \bullet \mathfrak{x}_\alpha \in wffs_o$›] **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* $A\ (x,\ \alpha)\ B$
shows $\vdash \forall x_\alpha.\ B \supset^{\mathcal{Q}}$ **S** $\{(x,\ \alpha) \rightarrowtail A\}\ B$
proof −
have $\vdash \prod_\alpha \bullet (\lambda x_\alpha.\ B) \supset^{\mathcal{Q}} (\lambda x_\alpha.\ B) \bullet A$
proof −
let $?\vartheta = \{(\mathfrak{f},\ \alpha\to o) \rightarrowtail \lambda x_\alpha.\ B,\ (\mathfrak{x},\ \alpha) \rightarrowtail A\}$
have $\vdash \prod_\alpha \bullet \mathfrak{f}_{\alpha\to o} \supset^{\mathcal{Q}} \mathfrak{f}_{\alpha\to o} \bullet \mathfrak{x}_\alpha$ (**is** $\vdash ?C$)

by (*fact prop-5225*)
moreover from *assms* 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* $$! *v*) *v ?C*
by (*code-simp*, (*unfold atomize-conj*[*symmetric*])?, *fastforce*)+ *blast*
moreover have *?ϑ* $\neq$ {$$}
by *simp*
ultimately have $\vdash$ **S** *?ϑ ?C*
by (*rule Sub*)
moreover have **S** *?ϑ ?C* $= \prod_\alpha \bullet (\lambda x_\alpha.\ B) \supset^{\mathcal{Q}} (\lambda x_\alpha.\ B) \bullet A$
by *simp*
ultimately show *?thesis*
by (*simp only*:)
qed
moreover from *assms* have $\vdash (\lambda x_\alpha.\ B) \bullet A =_o$ **S** $\{(x, \alpha) \rightarrowtail A\}\ B$
by (*rule prop-5207*)
ultimately show *?thesis*
by (*rule rule-R* [where p = [»]]) *force+*
qed

6.28 Proposition 5227

corollary *prop-5227*:
shows $\vdash F_o \supset^{\mathcal{Q}} \mathfrak{x}_o$
proof –
have $\vdash \forall \mathfrak{x}_o.\ \mathfrak{x}_o \supset^{\mathcal{Q}}$ **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 $A \in wffs_o$
shows $\vdash F_o \supset^{\mathcal{Q}} A$
proof –
let $?\vartheta = \{(\mathfrak{x}, o) \rightarrowtail A\}$ **and** $?B = F_o \supset^{\mathcal{Q}} \mathfrak{x}_o$
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
$\forall v \in$ *fmdom′ ?ϑ. var-name v* $\notin$ *free-var-names* ({}::*form set*) $\wedge$ *is-free-for* (*?ϑ* $$! *v*) *v ?B*
by *force*
ultimately have $\vdash$ **S** $\{(\mathfrak{x}, o) \rightarrowtail A\}\ (F_o \supset^{\mathcal{Q}} \mathfrak{x}_o)$
using *Sub*[*OF prop-5227*, **where** ϑ = *?ϑ*] **by** *fastforce*
then show *?thesis*
by *simp*
qed

6.29 Proposition 5228

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

6.30 Proposition 5229

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

have $?B \in wffs_{?\beta}$ **and** $?A \in wffs_o$
using *assms* **by** *auto*
moreover have *is-free-for* $?B$ $(?x,\ ?\beta)$ $?A$
by (*intro is-free-for-in-equivalence is-free-for-in-conj is-free-for-in-false*) *fastforce*
ultimately have $\vdash (\lambda ?x_{?\beta}.\ ?A) \bullet ?B =_o \mathbf{S}\ \{(?x,\ ?\beta) \rightarrowtail ?B\}\ ?A$
by (*rule prop-5207*)
moreover
have S $\{(?x,\ ?\beta) \rightarrowtail ?B\}\ ?A = ?C$
by *simp*
ultimately show *?thesis*
by (*simp only*:)
qed
ultimately have $\vdash F_o \equiv^{\mathcal{Q}} F_o \wedge^{\mathcal{Q}} A$
by (*rule rule-R*[**where** $p = []$]) *auto*
then show *?thesis*
unfolding *equivalence-def* **by** (*rule Equality-Rules*(*2*))
qed

proposition *prop-5229*:
shows $\vdash (T_o \wedge^{\mathcal{Q}} T_o) =_o T_o$
and $\vdash (T_o \wedge^{\mathcal{Q}} F_o) =_o F_o$
and $\vdash (F_o \wedge^{\mathcal{Q}} T_o) =_o F_o$
and $\vdash (F_o \wedge^{\mathcal{Q}} F_o) =_o F_o$
proof −
show $\vdash (T_o \wedge^{\mathcal{Q}} T_o) =_o T_o$ **and** $\vdash (T_o \wedge^{\mathcal{Q}} F_o) =_o F_o$
using *prop-5216* **by** *blast+*
next
show $\vdash (F_o \wedge^{\mathcal{Q}} T_o) =_o F_o$ **and** $\vdash (F_o \wedge^{\mathcal{Q}} F_o) =_o F_o$
using *false-in-conj-provability* **and** *true-wff* **and** *false-wff* **by** *simp-all*
qed

6.31 Proposition 5230

proposition *prop-5230*:
shows $\vdash (T_o \equiv^{\mathcal{Q}} T_o) =_o T_o$
and $\vdash (T_o \equiv^{\mathcal{Q}} F_o) =_o F_o$
and $\vdash (F_o \equiv^{\mathcal{Q}} T_o) =_o F_o$
and $\vdash (F_o \equiv^{\mathcal{Q}} F_o) =_o T_o$
proof −
show $\vdash (T_o \equiv^{\mathcal{Q}} T_o) =_o T_o$ **and** $\vdash (T_o \equiv^{\mathcal{Q}} F_o) =_o F_o$
unfolding *equivalence-def* **using** *prop-5218* **by** *blast+*
next
show $\vdash (F_o \equiv^{\mathcal{Q}} F_o) =_o T_o$
unfolding *equivalence-def* **by** (*rule Equality-Rules*(*2*)[*OF prop-5210*[*OF false-wff*]])
next
have §*1*: $\vdash (F_o \equiv^{\mathcal{Q}} T_o) \supset^{\mathcal{Q}} ((\lambda\mathfrak{x}_o.\ (\mathfrak{x}_o \equiv^{\mathcal{Q}} F_o)) \bullet F_o \equiv^{\mathcal{Q}} (\lambda\mathfrak{x}_o.\ (\mathfrak{x}_o \equiv^{\mathcal{Q}} F_o)) \bullet T_o)$
proof −
let $?\vartheta = \{(\mathfrak{h},\ o{\to}o) \rightarrowtail \lambda\mathfrak{x}_o.\ (\mathfrak{x}_o \equiv^{\mathcal{Q}} F_o),\ (\mathfrak{x},\ o) \rightarrowtail F_o,\ (\mathfrak{y},\ o) \rightarrowtail T_o\}$
and $?A = (\mathfrak{x}_o =_o \mathfrak{y}_o) \supset^{\mathcal{Q}} (\mathfrak{h}_{o\to o} \bullet \mathfrak{x}_o \equiv^{\mathcal{Q}} \mathfrak{h}_{o\to o} \bullet \mathfrak{y}_o)$

```
  have ⊢ ?A
    by (fact axiom-is-derivable-from-no-hyps[OF axiom-2])
  moreover have is-substitution ?ϑ
    by auto
  moreover have
    ∀ v ∈ fmdom′ ?ϑ. var-name v ∉ free-var-names ({}::form set) ∧ is-free-for (?ϑ $$! v) v ?A
    by (code-simp, unfold atomize-conj[symmetric], simp, force)+ blast
  moreover have ?ϑ ≠ {$$}
    by simp
  ultimately have ⊢ S ?ϑ ?A
    by (rule Sub)
  then show ?thesis
    by simp
qed
then have §2: ⊢ (F_o ≡^Q T_o) ⊃^Q ((F_o ≡^Q F_o) ≡^Q (T_o ≡^Q F_o)) (is ⊢ ?A2)
proof −
  have is-free-for A (𝔵, o) (𝔵_o ≡^Q F_o) for A
    by code-simp blast
  have β-reduction: ⊢ (λ𝔵_o. (𝔵_o ≡^Q F_o)) • A =_o (A ≡^Q F_o) if A ∈ wffs_o for A
    using
      prop-5207
      [
        OF that equivalence-wff[OF wffs-of-type-intros(1) false-wff]
        ‹is-free-for A (𝔵, o) (𝔵_o ≡^Q F_o)›
      ]
    by simp
  from §1 and β-reduction[OF false-wff] have
    ⊢ (F_o =_o T_o) ⊃^Q ((F_o ≡^Q F_o) ≡^Q (λ𝔵_o. (𝔵_o ≡^Q F_o)) • 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: ⊢ (F_o ≡^Q T_o) ⊃^Q F_o
proof −
  note r1 = rule-RR[OF disjI1] and r2 = rule-RR[OF disjI2]
  have §3_1: ⊢ (F_o ≡^Q T_o) ⊃^Q ((F_o ≡^Q F_o) ≡^Q F_o) (is ‹⊢ ?A3_1›)
    by (rule r1[OF prop-5218[OF false-wff], where p = [»,»] and C = ?A2]) (use §2 in ‹force+›)
  have §3_2: ⊢ (F_o ≡^Q T_o) ⊃^Q (T_o ≡^Q F_o) (is ‹⊢ ?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 ⊢ (F_o ≡^Q T_o) ≡^Q ((F_o ≡^Q T_o) ∧^Q F_o)
proof −
  have
    ⊢
      (λ𝔵_o. λ𝔶_o. (𝔵_o ≡^Q 𝔵_o ∧^Q 𝔶_o)) • (F_o ≡^Q T_o)
      =_{o→o}
      S {(𝔵, o) ↣ F_o ≡^Q T_o} (λ𝔶_o. (𝔵_o ≡^Q 𝔵_o ∧^Q 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 = [«]]) *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 $?\vartheta = \{(\mathfrak{x},\ o) \rightarrowtail F_o \equiv^{\mathcal{Q}} T_o\}$
  have *is-substitution* $?\vartheta$
    by *auto*
  moreover have $\forall\, v \in \mathit{fmdom}'\ ?\vartheta.$
    var-name $v \notin$ *free-var-names* ({}::*form set*) $\wedge$ *is-free-for* ($?\vartheta$ $$! v) v $(\mathfrak{x}_o \wedge^{\mathcal{Q}} F_o \equiv^{\mathcal{Q}} F_o)$
    by *simp*
  moreover have $?\vartheta \neq$ {$$}
    by *simp*
  ultimately have $\vdash$ **S** $?\vartheta\ (\mathfrak{x}_o \wedge^{\mathcal{Q}} F_o \equiv^{\mathcal{Q}} F_o)$
    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 $A \in wffs_o$ **and** $B \in wffs_o$
shows $\vdash A \vee^{\mathcal{Q}} B =_o \sim^{\mathcal{Q}} (\sim^{\mathcal{Q}} A \wedge^{\mathcal{Q}} \sim^{\mathcal{Q}} B)$
proof −
let $?C = (\lambda\mathfrak{x}_o.\ \lambda\mathfrak{y}_o.\ \sim^{\mathcal{Q}} (\sim^{\mathcal{Q}} \mathfrak{x}_o \wedge^{\mathcal{Q}} \sim^{\mathcal{Q}} \mathfrak{y}_o)) \cdot A \cdot B$
from *assms* **have** $?C \in wffs_o$
by *blast*
have $(\sim^{\mathcal{Q}} (\sim^{\mathcal{Q}} \mathfrak{x}_o \wedge^{\mathcal{Q}} \sim^{\mathcal{Q}} \mathfrak{y}_o)) \in wffs_o$
by *auto*
moreover obtain z **where** $(z,\ o) \notin \{(\mathfrak{x},\ o),\ (\mathfrak{y},\ o)\}$ **and** $(z,\ o) \notin$ *free-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 $(z,\ o) \notin$ *free-vars* $(\sim^{\mathcal{Q}} (\sim^{\mathcal{Q}} \mathfrak{x}_o \wedge^{\mathcal{Q}} \sim^{\mathcal{Q}} \mathfrak{y}_o))$
by *simp*
moreover have *is-free-for* $(z_o)\ (\mathfrak{y},\ o)\ (\sim^{\mathcal{Q}} (\sim^{\mathcal{Q}} \mathfrak{x}_o \wedge^{\mathcal{Q}} \sim^{\mathcal{Q}} \mathfrak{y}_o))$
by (*intro is-free-for-in-conj is-free-for-in-neg is-free-for-in-var*)
ultimately have
$\vdash (\lambda\mathfrak{y}_o.\ \sim^{\mathcal{Q}} (\sim^{\mathcal{Q}} \mathfrak{x}_o \wedge^{\mathcal{Q}} \sim^{\mathcal{Q}} \mathfrak{y}_o)) =_{o\to o} (\lambda z_o.\ \mathbf{S}\ \{(\mathfrak{y},\ o) \rightarrowtail z_o\} \sim^{\mathcal{Q}} (\sim^{\mathcal{Q}} \mathfrak{x}_o \wedge^{\mathcal{Q}} \sim^{\mathcal{Q}} \mathfrak{y}_o))$
by (*rule* α)
then have $\vdash (\lambda\mathfrak{y}_o.\ \sim^{\mathcal{Q}} (\sim^{\mathcal{Q}} \mathfrak{x}_o \wedge^{\mathcal{Q}} \sim^{\mathcal{Q}} \mathfrak{y}_o)) =_{o\to o} (\lambda z_o.\ \sim^{\mathcal{Q}} (\sim^{\mathcal{Q}} \mathfrak{x}_o \wedge^{\mathcal{Q}} \sim^{\mathcal{Q}} z_o))$
by *simp*
from *prop-5200*[*OF* ‹$?C \in wffs_o$›] **and** *this* **have**
$\vdash$
$(\lambda\mathfrak{x}_o.\ \lambda z_o.\ \sim^{\mathcal{Q}} (\sim^{\mathcal{Q}} \mathfrak{x}_o \wedge^{\mathcal{Q}} \sim^{\mathcal{Q}} z_o)) \cdot A \cdot B$
$=_o$
$(\lambda\mathfrak{x}_o.\ \lambda\mathfrak{y}_o.\ \sim^{\mathcal{Q}} (\sim^{\mathcal{Q}} \mathfrak{x}_o \wedge^{\mathcal{Q}} \sim^{\mathcal{Q}} \mathfrak{y}_o)) \cdot A \cdot B$
by (*rule rule-R*[**where** p = [«,»,«,«,«]]) *force*+
moreover have $\lambda z_o.\ \sim^{\mathcal{Q}} (\sim^{\mathcal{Q}} \mathfrak{x}_o \wedge^{\mathcal{Q}} \sim^{\mathcal{Q}} z_o) \in wffs_{o\to o}$
by *blast*
have
$\vdash$
$(\lambda\mathfrak{x}_o.\ \lambda z_o.\ \sim^{\mathcal{Q}} (\sim^{\mathcal{Q}} \mathfrak{x}_o \wedge^{\mathcal{Q}} \sim^{\mathcal{Q}} z_o)) \cdot A$
$=_{o\to o}$
$\mathbf{S}\ \{(\mathfrak{x},\ o) \rightarrowtail A\}\ (\lambda z_o.\ \sim^{\mathcal{Q}} (\sim^{\mathcal{Q}} \mathfrak{x}_o \wedge^{\mathcal{Q}} \sim^{\mathcal{Q}} z_o))$
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*: ‹$(z,\ o) \notin$ *free-vars* A›)+
)
then have $\vdash (\lambda\mathfrak{x}_o.\ \lambda z_o.\ \sim^{\mathcal{Q}} (\sim^{\mathcal{Q}} \mathfrak{x}_o \wedge^{\mathcal{Q}} \sim^{\mathcal{Q}} z_o)) \cdot A =_{o\to o} (\lambda z_o.\ \sim^{\mathcal{Q}} (\sim^{\mathcal{Q}} A \wedge^{\mathcal{Q}} \sim^{\mathcal{Q}} z_o))$
using ‹$(z,\ o) \notin$ *free-vars* $(\sim^{\mathcal{Q}} (\sim^{\mathcal{Q}} \mathfrak{x}_o \wedge^{\mathcal{Q}} \sim^{\mathcal{Q}} \mathfrak{y}_o))$› **by** *simp*
ultimately have
$\vdash (\lambda z_o.\ \sim^{\mathcal{Q}} (\sim^{\mathcal{Q}} A \wedge^{\mathcal{Q}} \sim^{\mathcal{Q}} z_o)) \cdot B =_o (\lambda\mathfrak{x}_o.\ \lambda\mathfrak{y}_o.\ \sim^{\mathcal{Q}} (\sim^{\mathcal{Q}} \mathfrak{x}_o \wedge^{\mathcal{Q}} \sim^{\mathcal{Q}} \mathfrak{y}_o)) \cdot A \cdot B$
by (*rule rule-R*[**where** p = [«,»,«]]) *force*+
moreover have $\vdash (\lambda z_o.\ \sim^{\mathcal{Q}} (\sim^{\mathcal{Q}} A \wedge^{\mathcal{Q}} \sim^{\mathcal{Q}} z_o)) \cdot B =_o \mathbf{S}\ \{(z,\ o) \rightarrowtail B\}\ (\sim^{\mathcal{Q}} (\sim^{\mathcal{Q}} A \wedge^{\mathcal{Q}} \sim^{\mathcal{Q}} z_o))$
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 $\{(z, o) \rightarrowtail B\}$ $(\sim^{\mathcal{Q}} (\sim^{\mathcal{Q}} A \wedge^{\mathcal{Q}} \sim^{\mathcal{Q}} z_o)) = \sim^{\mathcal{Q}} (\sim^{\mathcal{Q}} A \wedge^{\mathcal{Q}} \sim^{\mathcal{Q}} B)$
using *free-var-singleton-substitution-neutrality*[*OF* ‹$(z, o) \notin$ *free-vars* A›] **by** *simp*
ultimately have $\vdash (\lambda\mathfrak{x}_o.\ \lambda\mathfrak{y}_o.\ \sim^{\mathcal{Q}} (\sim^{\mathcal{Q}} \mathfrak{x}_o \wedge^{\mathcal{Q}} \sim^{\mathcal{Q}} \mathfrak{y}_o)) \bullet A \bullet B =_o \sim^{\mathcal{Q}} (\sim^{\mathcal{Q}} A \wedge^{\mathcal{Q}} \sim^{\mathcal{Q}} B)$
using *Equality-Rules*(*2*,*3*) **by** *metis*
then show *?thesis*
by *simp*
qed

context begin

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

proposition *prop-5232*:
shows $\vdash (T_o \vee^{\mathcal{Q}} T_o) =_o T_o$
and $\vdash (T_o \vee^{\mathcal{Q}} F_o) =_o T_o$
and $\vdash (F_o \vee^{\mathcal{Q}} T_o) =_o T_o$
and $\vdash (F_o \vee^{\mathcal{Q}} F_o) =_o F_o$
proof −
from *prop-5231*(*2*) **have** $\vdash \sim^{\mathcal{Q}} F_o =_o T_o$ (**is** ‹$\vdash$ *?A*›) **.**
from *prop-5229*(*4*) **have** $\vdash \sim^{\mathcal{Q}} (F_o \wedge^{\mathcal{Q}} F_o) =_o T_o$
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** $\vdash (T_o \vee^{\mathcal{Q}} T_o) =_o T_o$ **.**
from *prop-5229*(*3*) **have** $\vdash \sim^{\mathcal{Q}} (F_o \wedge^{\mathcal{Q}} T_o) =_o T_o$
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** $\vdash (T_o \vee^{\mathcal{Q}} F_o) =_o T_o$ **.**
from *prop-5229*(*2*) **have** $\vdash \sim^{\mathcal{Q}} (T_o \wedge^{\mathcal{Q}} F_o) =_o T_o$
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** $\vdash (F_o \vee^{\mathcal{Q}} T_o) =_o T_o$ **.**
next

from *prop-5231*(*1*) **have** $\vdash \sim^{\mathcal{Q}}\ T_o =_o F_o$ (**is** ‹⊢ *?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* ‹⊢ *?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* $\mathcal{V}_B$*-graph-denotation-is-truth-value*[*OF* $\mathcal{V}_B$*-graph-*$\mathcal{V}_B$]]
by (*auto dest*: *tv-cases*) *metis*
then show *?case*
proof *cases*
case *a*
with ‹*free-vars* $B = \{\}$› **have** $\vdash T_o =_o B$
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 $\vdash \sim^{\mathcal{Q}} B =_o F_o$
unfolding *neg-def*[*folded equality-of-type-def*] **by** (*rule rule-R*[**where** p = [«,»,»]]) *force+*
moreover from *a* **have** $\forall\varphi.$ *is-tv-assignment* $\varphi \longrightarrow \mathcal{V}_B\ \varphi\ (\sim^{\mathcal{Q}} B) = \mathbf{F}$
using $\mathcal{V}_B$*-neg*[*OF neg-pwff.hyps*] **by** *simp*
ultimately show *?thesis*
by (*auto simp*: *inj-eq*)
next
case *b*
with ‹*free-vars* $B = \{\}$› **have** $\vdash F_o =_o B$
using *neg-pwff.IH* **and** *Equality-Rules*(*2*) **by** *blast*
then have $\vdash \sim^{\mathcal{Q}} B =_o T_o$
unfolding *neg-def*[*folded equality-of-type-def*]
using *rule-T*(*2*)[*OF hyp-derivable-form-is-wffso*] **by** *blast*
moreover from *b* **have** $\forall\varphi.$ *is-tv-assignment* $\varphi \longrightarrow \mathcal{V}_B\ \varphi\ (\sim^{\mathcal{Q}} B) = \mathbf{T}$
using $\mathcal{V}_B$*-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*: $\forall\varphi.$ *is-tv-assignment* $\varphi \longrightarrow \mathcal{V}_B\ \varphi\ B = b$
and *C-den*: $\forall\varphi.$ *is-tv-assignment* $\varphi \longrightarrow \mathcal{V}_B\ \varphi\ C = b'$
using *closed-pwff-denotation-uniqueness* **by** *metis*
then have $b \in$ *elts* $\mathbb{B}$ **and** $b' \in$ *elts* $\mathbb{B}$
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* $\mathcal{V}_B$*-graph-denotation-is-truth-value*[*OF* $\mathcal{V}_B$*-graph-*$\mathcal{V}_B$]]
by *force+*
with *conj-pwff.hyps* **consider**
(*a*) $b = \mathbf{T}$ **and** $b' = \mathbf{T}$
| (*b*) $b = \mathbf{T}$ **and** $b' = \mathbf{F}$
| (*c*) $b = \mathbf{F}$ **and** $b' = \mathbf{T}$
| (*d*) $b = \mathbf{F}$ **and** $b' = \mathbf{F}$
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** ‹⊢ *?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** ‹⊢ *?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 *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* ‹⊢ *?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** ‹⊢ *?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** ‹⊢ *?A2*›)
  by (*rule rule-RR*[*OF disjI2*, **where** *p* = [«,»,«,»] **and** *C* = *?A1*]) (*use* ‹⊢ *?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* ‹⊢ *?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** ‹⊢ *?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** ‹⊢ *?A2*›)
  by (*rule rule-RR*[*OF disjI2*, **where** *p* = [«,»,«,»] **and** *C* = *?A1*]) (*use* ‹⊢ *?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* ‹⊢ *?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** $\vdash F_o \wedge^{\mathcal{Q}} F_o =_o F_o$ (**is** ‹⊢ *?A1*›) **.**
from *B-den*[*unfolded d*(*1*)] **and** ‹*free-vars B* = {}› **have** $\vdash B =_o F_o$
using *conj-pwff.IH*(*1*) **by** *simp*
then have $\vdash B \wedge^{\mathcal{Q}} F_o =_o F_o$ (**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** $\vdash C =_o F_o$
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* ‹⊢ *?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 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*: $\forall \varphi$. *is-tv-assignment* $\varphi \longrightarrow \mathcal{V}_B\ \varphi\ B = b$
and *C-den*: $\forall \varphi$. *is-tv-assignment* $\varphi \longrightarrow \mathcal{V}_B\ \varphi\ C = b'$
using *closed-pwff-denotation-uniqueness* **by** *metis*
then have $b \in$ *elts* $\mathbb{B}$ **and** $b' \in$ *elts* $\mathbb{B}$
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* $\mathcal{V}_B$*-graph-denotation-is-truth-value*[*OF* $\mathcal{V}_B$*-graph-*$\mathcal{V}_B$]]
by *force+*
with *disj-pwff.hyps* **consider**
(*a*) $b = \mathbf{T}$ **and** $b' = \mathbf{T}$
| (*b*) $b = \mathbf{T}$ **and** $b' = \mathbf{F}$
| (*c*) $b = \mathbf{F}$ **and** $b' = \mathbf{T}$
| (*d*) $b = \mathbf{F}$ **and** $b' = \mathbf{F}$
by *auto*
then show *?case*
proof *cases*
case *a*
from *prop-5232*(*1*) **have** $\vdash T_o \vee^{\mathcal{Q}} T_o =_o T_o$ (**is** ‹⊢ *?A1*›) **.**
from *B-den*[*unfolded a*(*1*)] **and** ‹*free-vars B* = {}› **have** $\vdash B =_o T_o$
using *disj-pwff.IH*(*1*) **by** *simp*
then have $\vdash B \vee^{\mathcal{Q}} T_o =_o T_o$ (**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 $\vdash B \vee^{\mathcal{Q}} C =_o T_o$
  by (*rule rule-RR*[*OF disjI2*, **where** *p* = [«,»,»] **and** *C* = *?A2*]) (*use* ‹⊢ *?A2*› **in** ‹*force*+›)
then have ($\forall\varphi$. *is-tv-assignment* $\varphi \longrightarrow \mathcal{V}_B\ \varphi\ (B \vee^{\mathcal{Q}} C) = \mathbf{T}) \longrightarrow\ \vdash B \vee^{\mathcal{Q}} C =_o T_o$
  by *blast*
moreover have $\forall\varphi$. *is-tv-assignment* $\varphi \longrightarrow \mathcal{V}_B\ \varphi\ (B \vee^{\mathcal{Q}} C) \neq \mathbf{F}$
  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** $\vdash T_o \vee^{\mathcal{Q}} F_o =_o T_o$ (**is** ‹⊢ *?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 $\vdash B \vee^{\mathcal{Q}} F_o =_o T_o$ (**is** ‹⊢ *?A2*›)
  by (*rule rule-RR*[*OF disjI2*, **where** *p* = [«,»,«,»] **and** *C* = *?A1*]) (*use* ‹⊢ *?A1*› **in** ‹*force*+›)
from *C-den*[*unfolded b*(*2*)] **and** ‹*free-vars* $C = \{\}$› **have** $\vdash C =_o F_o$
  using *disj-pwff.IH*(*2*) **by** *simp*
then have $\vdash B \vee^{\mathcal{Q}} C =_o T_o$
  by (*rule rule-RR*[*OF disjI2*, **where** *p* = [«,»,»] **and** *C* = *?A2*]) (*use* ‹⊢ *?A2*› **in** ‹*force*+›)
then have ($\forall\varphi$. *is-tv-assignment* $\varphi \longrightarrow \mathcal{V}_B\ \varphi\ (B \vee^{\mathcal{Q}} C) = \mathbf{T}) \longrightarrow\ \vdash B \vee^{\mathcal{Q}} C =_o T_o$
  by *blast*
moreover have $\forall\varphi$. *is-tv-assignment* $\varphi \longrightarrow \mathcal{V}_B\ \varphi\ (B \vee^{\mathcal{Q}} C) \neq \mathbf{F}$
  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** $\vdash F_o \vee^{\mathcal{Q}} T_o =_o T_o$ (**is** ‹⊢ *?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 $\vdash B \vee^{\mathcal{Q}} T_o =_o T_o$ (**is** ‹⊢ *?A2*›)
  by (*rule rule-RR*[*OF disjI2*, **where** *p* = [«,»,«,»] **and** *C* = *?A1*]) (*use* ‹⊢ *?A1*› **in** ‹*force*+›)
from *C-den*[*unfolded c*(*2*)] **and** ‹*free-vars* $C = \{\}$› **have** $\vdash C =_o T_o$
  using *disj-pwff.IH*(*2*) **by** *simp*
then have $\vdash B \vee^{\mathcal{Q}} C =_o T_o$
  by (*rule rule-RR*[*OF disjI2*, **where** *p* = [«,»,»] **and** *C* = *?A2*]) (*use* ‹⊢ *?A2*› **in** ‹*force*+›)
then have ($\forall\varphi$. *is-tv-assignment* $\varphi \longrightarrow \mathcal{V}_B\ \varphi\ (B \vee^{\mathcal{Q}} C) = \mathbf{T}) \longrightarrow\ \vdash B \vee^{\mathcal{Q}} C =_o T_o$
  by *blast*
moreover have $\forall\varphi$. *is-tv-assignment* $\varphi \longrightarrow \mathcal{V}_B\ \varphi\ (B \vee^{\mathcal{Q}} C) \neq \mathbf{F}$
  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** $\vdash F_o \vee^{\mathcal{Q}} F_o =_o F_o$ (**is** ‹$\vdash$ *?A1*›) **.**
from *B-den*[*unfolded d*(*1*)] **and** ‹*free-vars* $B = \{\}$› **have** $\vdash B =_o F_o$
  using *disj-pwff.IH*(*1*) **by** *simp*
then have $\vdash B \vee^{\mathcal{Q}} F_o =_o F_o$ (**is** ‹$\vdash$ *?A2*›)
  by (*rule rule-RR*[*OF disjI2*, **where** p = [«,»,«,»] **and** C = *?A1*]) (*use* ‹$\vdash$ *?A1*› **in** ‹*force+*›)
from *C-den*[*unfolded d*(*2*)] **and** ‹*free-vars* $C = \{\}$› **have** $\vdash C =_o F_o$
  using *disj-pwff.IH*(*2*) **by** *simp*
then have $\vdash B \vee^{\mathcal{Q}} C =_o F_o$
  by (*rule rule-RR*[*OF disjI2*, **where** p = [«,»,»] **and** C = *?A2*]) (*use* ‹$\vdash$ *?A2*› **in** ‹*force+*›)
then have ($\forall \varphi$. *is-tv-assignment* $\varphi \longrightarrow \mathcal{V}_B\ \varphi\ (B \vee^{\mathcal{Q}} C) = \mathbf{T}) \longrightarrow \vdash B \vee^{\mathcal{Q}} C =_o F_o$
  by *blast*
moreover have $\forall \varphi$. *is-tv-assignment* $\varphi \longrightarrow \mathcal{V}_B\ \varphi\ (B \vee^{\mathcal{Q}} C) \neq \mathbf{T}$
  using $\mathcal{V}_B$-*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 ‹$\vdash B \vee^{\mathcal{Q}} C =_o F_o$› **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*: $\forall \varphi$. *is-tv-assignment* $\varphi \longrightarrow \mathcal{V}_B\ \varphi\ B = b$
  and *C-den*: $\forall \varphi$. *is-tv-assignment* $\varphi \longrightarrow \mathcal{V}_B\ \varphi\ C = b'$
  using *closed-pwff-denotation-uniqueness* **by** *metis*
then have $b \in$ *elts* $\mathbb{B}$ **and** $b' \in$ *elts* $\mathbb{B}$
  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* $\mathcal{V}_B$-*graph-denotation-is-truth-value*[*OF* $\mathcal{V}_B$-*graph*-$\mathcal{V}_B$]]
  by *force+*
with *imp-pwff.hyps* **consider**
  (a) $b = \mathbf{T}$ **and** $b' = \mathbf{T}$
| (b) $b = \mathbf{T}$ **and** $b' = \mathbf{F}$
| (c) $b = \mathbf{F}$ **and** $b' = \mathbf{T}$
| (d) $b = \mathbf{F}$ **and** $b' = \mathbf{F}$
  by *auto*
then show *?case*
proof *cases*
  case *a*
  from *prop-5228*(*1*) **have** $\vdash T_o \supset^{\mathcal{Q}} T_o =_o T_o$ (**is** ‹$\vdash$ *?A1*›) **.**
  from *B-den*[*unfolded a*(*1*)] **and** ‹*free-vars* $B = \{\}$› **have** $\vdash B =_o T_o$
    using *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 a*(*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 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** ‹⊢ *?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** ‹⊢ *?A2*›)
by (*rule rule-RR*[*OF disjI2*, **where** p = [«,»,«,»] **and** C = *?A1*]) (*use* ‹⊢ *?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* ‹⊢ *?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** ‹⊢ *?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** ‹⊢ *?A2*›)
by (*rule rule-RR*[*OF disjI2*, **where** p = [«,»,«,»] **and** C = *?A1*]) (*use* ‹⊢ *?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* ‹⊢ *?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** ‹⊢ *?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** ‹⊢ *?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** $\vdash C =_o F_o$
  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 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*: $\forall \varphi$. *is-tv-assignment* $\varphi \longrightarrow \mathcal{V}_B\ \varphi\ B = b$
  and *C-den*: $\forall \varphi$. *is-tv-assignment* $\varphi \longrightarrow \mathcal{V}_B\ \varphi\ C = b'$
  using *closed-pwff-denotation-uniqueness* **by** *metis*
then have $b \in$ *elts* $\mathbb{B}$ **and** $b' \in$ *elts* $\mathbb{B}$
  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* $\mathcal{V}_B$*-graph-denotation-is-truth-value*[*OF* $\mathcal{V}_B$*-graph-*$\mathcal{V}_B$]]
  by *force+*
with *eqv-pwff.hyps* **consider**
  (*a*) $b = \mathbf{T}$ **and** $b' = \mathbf{T}$
| (*b*) $b = \mathbf{T}$ **and** $b' = \mathbf{F}$
| (*c*) $b = \mathbf{F}$ **and** $b' = \mathbf{T}$
| (*d*) $b = \mathbf{F}$ **and** $b' = \mathbf{F}$
  by *auto*
then show *?case*
proof *cases*
  case *a*
  from *prop-5230*(*1*) **have** $\vdash (T_o \equiv^{\mathcal{Q}} T_o) =_o T_o$ (**is** ‹$\vdash$ *?A1*›) **.**
  from *B-den*[*unfolded a*(*1*)] **and** ‹*free-vars* $B = \{\}$› **have** $\vdash B =_o T_o$
    using *eqv-pwff.IH*(*1*) **by** *simp*
  then have $\vdash (B \equiv^{\mathcal{Q}} T_o) =_o T_o$ (**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 *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 $\forall \varphi$. *is-tv-assignment* $\varphi \longrightarrow \mathcal{V}_B\ \varphi\ (B \equiv^{\mathcal{Q}} C) \neq \mathbf{F}$
    using $\mathcal{V}_B$*-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** ‹⊢ *?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** ‹⊢ *?A2*›)
by (*rule rule-RR*[*OF disjI2*, **where** *p* = [«,»,«,»] **and** *C* = *?A1*]) (*use* ‹⊢ *?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* ‹⊢ *?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** ‹⊢ *?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** ‹⊢ *?A2*›)
by (*rule rule-RR*[*OF disjI2*, **where** *p* = [«,»,«,»] **and** *C* = *?A1*]) (*use* ‹⊢ *?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* ‹⊢ *?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** ‹⊢ *?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** ‹⊢ *?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** $\vdash C =_o F_o$
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* ‹⊢ *?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 $\forall \varphi.$ *is-tv-assignment* $\varphi \longrightarrow \mathcal{V}_B\ \varphi\ (B \equiv^{\mathcal{Q}} C) \neq \mathbf{F}$
using $\mathcal{V}_B$*-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*$_T \longrightarrow \vdash A =_o T_o$ **and** *?A*$_F \longrightarrow \vdash A =_o F_o$
by *blast+*
qed

proposition *prop-5233*:
assumes *is-tautology A*
shows $\vdash A$
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** $A \in$ *pwffs* **and** $\forall \varphi.$ *is-tv-assignment* $\varphi \longrightarrow \mathcal{V}_B\ \varphi\ A = \mathbf{T}$
unfolding *is-tautology-def* **by** *blast+*
with *empty*(*1*) **have** $\vdash A =_o T_o$
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** $A \in$ *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 ($\mathbf{S}$ $\{(p,\ o) \rightarrowtail T_o\}$ *A*) **and** *is-tautology* ($\mathbf{S}$ $\{(p,\ o) \rightarrowtail F_o\}$ *A*)
using *pwff-substitution-tautology-preservation* [*OF insert.prems*] **by** *blast+*
moreover from *insert.hyps*(*2*,*4*) **and** ‹$v = (p,\ o)$› **and** ‹$A \in$ *pwffs*›
have *free-vars* ($\mathbf{S}$ $\{(p,\ o) \rightarrowtail T_o\}$ *A*) = *F* **and** *free-vars* ($\mathbf{S}$ $\{(p,\ o) \rightarrowtail F_o\}$ *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 $\vdash \mathbf{S}\ \{(p,\ o) \rightarrowtail T_o\}\ A$ **and** $\vdash \mathbf{S}\ \{(p,\ o) \rightarrowtail F_o\}\ A$
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* $(hs \supset^{\mathcal{Q}}{}_{\star}\ B)$
  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 $hs \supset^{\mathcal{Q}}{}_{\star}\ B = \mathbf{S}\ \vartheta\ C$
    by *blast*
  then have $\vdash hs \supset^{\mathcal{Q}}{}_{\star}\ B$
  proof (*cases* $\vartheta = \{\$\$\}$)
    case *True*
    with ‹$hs \supset^{\mathcal{Q}}{}_{\star}\ B = \mathbf{S}\ \vartheta\ C$› **have** $C = hs \supset^{\mathcal{Q}}{}_{\star}\ B$
      using *empty-substitution-neutrality* **by** *simp*
    with ‹$hs \supset^{\mathcal{Q}}{}_{\star}\ B = \mathbf{S}\ \vartheta\ C$› **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
      $\forall v \in fmdom'\ \vartheta.\ var\text{-}name\ v \notin free\text{-}var\text{-}names\ (\{\}::form\ set) \wedge is\text{-}free\text{-}for\ (\vartheta\ \$\$!\ v)\ v\ C$
    proof
      fix v
      assume $v \in fmdom'\ \vartheta$
      then show $var\text{-}name\ v \notin free\text{-}var\text{-}names\ (\{\}::form\ set) \wedge is\text{-}free\text{-}for\ (\vartheta\ \$\$!\ v)\ v\ C$
      proof (*cases* $v \in free\text{-}vars\ C$)
        case *True*
        with ‹$C \in pwffs$› **show** *?thesis*
          using *is-free-for-in-pwff* **by** *simp*
      next
        case *False*
        then have $is\text{-}free\text{-}for\ (\vartheta\ \$\$!\ v)\ v\ C$
          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*: ‹$hs \supset^{\mathcal{Q}}{}_{\star}\ B = \mathbf{S}\ \vartheta\ C$›[*unfolded generalized-imp-op-def*])
qed
from *this* **and** *assms*(*1*) **have** $\mathcal{H} \vdash hs \supset^{\mathcal{Q}}{}_{\star}\ B$
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 $\vdash \forall x_\alpha.\ (A \vee^{\mathcal{Q}} B) \supset^{\mathcal{Q}} (A \vee^{\mathcal{Q}} \forall x_\alpha.\ B)$
proof −
have §*1*: $\vdash \forall x_\alpha.\ (T_o \vee^{\mathcal{Q}} B) \supset^{\mathcal{Q}} (T_o \vee^{\mathcal{Q}} \forall x_\alpha.\ B)$
proof (*intro rule-P*(*2*))
show *is-tautologous* $(\forall x_\alpha.\ (T_o \vee^{\mathcal{Q}} B) \supset^{\mathcal{Q}} T_o \vee^{\mathcal{Q}} \forall x_\alpha.\ B)$
proof −
let $?\vartheta = \{(\mathfrak{x}, o) \rightarrowtail \forall x_\alpha.\ (T_o \vee^{\mathcal{Q}} B), (\mathfrak{y}, o) \rightarrowtail \forall x_\alpha.\ B\}$ **and** $?C = \mathfrak{x}_o \supset^{\mathcal{Q}} (T_o \vee^{\mathcal{Q}} (\mathfrak{y}_o))$
have *is-tautology* $?C$
using $\mathcal{V}_B$*-simps* **by** *simp*
moreover from *assms*(*2*) **have** *is-pwff-substitution* $?\vartheta$
using *pwffs-subset-of-wffso* **by** *fastforce*
moreover have $\forall x_\alpha.\ (T_o \vee^{\mathcal{Q}} B) \supset^{\mathcal{Q}} T_o \vee^{\mathcal{Q}} \forall x_\alpha.\ B = \mathbf{S}\ ?\vartheta\ ?C$
by *simp*
ultimately show *?thesis*
by *blast*
qed
qed *simp*
have §*2*: $\vdash \forall x_\alpha.\ B \supset^{\mathcal{Q}} (F_o \vee^{\mathcal{Q}} \forall x_\alpha.\ B)$
proof (*intro rule-P*(*2*))
show *is-tautologous* $(\forall x_\alpha.\ B \supset^{\mathcal{Q}} (F_o \vee^{\mathcal{Q}} \forall x_\alpha.\ B))$
proof −
let $?\vartheta = \{(\mathfrak{x}, o) \rightarrowtail \forall x_\alpha.\ B\}$ **and** $?C = \mathfrak{x}_o \supset^{\mathcal{Q}} (F_o \vee^{\mathcal{Q}} (\mathfrak{x}_o))$
have *is-tautology* $(\mathfrak{x}_o \supset^{\mathcal{Q}} (F_o \vee^{\mathcal{Q}} (\mathfrak{x}_o)))$ (**is** ‹*is-tautology* $?C$›)
using $\mathcal{V}_B$*-simps* **by** *simp*
moreover from *assms*(*2*) **have** *is-pwff-substitution* $?\vartheta$
using *pwffs-subset-of-wffso* **by** *auto*

moreover have $\forall x_\alpha.\ B \supset^{\mathcal{Q}} (F_o \vee^{\mathcal{Q}} \forall x_\alpha.\ B) = \mathbf{S}\ ?\vartheta\ ?C$
by *simp*
ultimately show *?thesis*
by *blast*
qed
qed *simp*
have §*3*: $\vdash B \equiv^{\mathcal{Q}} (F_o \vee^{\mathcal{Q}} B)$
proof (*intro rule-P*(*2*))
show *is-tautologous* $(B \equiv^{\mathcal{Q}} (F_o \vee^{\mathcal{Q}} B))$
proof −
let $?\vartheta = \{(\mathfrak{x},\ o) \rightarrowtail B\}$ **and** $?C = \mathfrak{x}_o \equiv^{\mathcal{Q}} (F_o \vee^{\mathcal{Q}} (\mathfrak{x}_o))$
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 $B \equiv^{\mathcal{Q}} (F_o \vee^{\mathcal{Q}} B) = \mathbf{S}\ ?\vartheta\ ?C$
by *simp*
ultimately show *?thesis*
by *blast*
qed
qed *simp*
from §*2* **and** §*3*[*unfolded equivalence-def*] **have** §*4*:
$\vdash \forall x_\alpha.\ (F_o \vee^{\mathcal{Q}} B) \supset^{\mathcal{Q}} (F_o \vee^{\mathcal{Q}} \forall x_\alpha.\ B)$
by (*rule rule-R*[**where** p = [«,»,»,«]]) *force+*
obtain p **where** $(p,\ o) \notin$ *vars* $(\forall x_\alpha.\ (A \vee^{\mathcal{Q}} B) \supset^{\mathcal{Q}} (A \vee^{\mathcal{Q}} \forall x_\alpha.\ B))$
by (*meson fresh-var-existence vars-form-finiteness*)
then have $(p,\ o) \neq (x,\ \alpha)$ **and** $(p,\ o) \notin$ *vars* A **and** $(p,\ o) \notin$ *vars* B
by *simp-all*
from ‹$(p,\ o) \notin$ *vars* B› **have** *sub*: $\mathbf{S}\ \{(p,\ o) \rightarrowtail C\}\ B = B$ **for** C
using *free-var-singleton-substitution-neutrality* **and** *free-vars-in-all-vars* **by** *blast*
have §*5*: $\vdash \forall x_\alpha.\ (p_o \vee^{\mathcal{Q}} B) \supset^{\mathcal{Q}} (p_o \vee^{\mathcal{Q}} \forall x_\alpha.\ B)$ (**is** ‹$\vdash\ ?C$›)
proof −
from *sub* **and** §*1* **have** $\vdash \mathbf{S}\ \{(p,\ o) \rightarrowtail T_o\}\ ?C$
using ‹$(p,\ o) \neq (x,\ \alpha)$› **by** *auto*
moreover from *sub* **and** §*4* **have** $\vdash \mathbf{S}\ \{(p,\ o) \rightarrowtail F_o\}\ ?C$
using ‹$(p,\ o) \neq (x,\ \alpha)$› **by** *auto*
moreover from *assms*(*2*) **have** $?C \in wffs_o$
using *pwffs-subset-of-wffso* **by** *auto*
ultimately show *?thesis*
by (*rule Cases*)
qed
then show *?thesis*
proof −
let $?\vartheta = \{(p,\ o) \rightarrowtail A\}$
from *assms*(*1*) **have** *is-substitution ?ϑ*
using *pwffs-subset-of-wffso* **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 *?C*
proof

fix v
assume $v \in$ *fmdom'* $?\vartheta$
then have $v = (p, o)$
by *simp*
with *assms*(*3*) **and** ‹$(p, o) \notin$ *vars* B› **have** *is-free-for* ($?\vartheta$ \$\$! 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* $v \notin$ *free-var-names* ({}::*form set*)
by *simp*
ultimately show *var-name* $v \notin$ *free-var-names* ({}::*form set*) $\wedge$ *is-free-for* ($?\vartheta$ \$\$! v) v $?C$
unfolding ‹$v = (p, o)$› **by** *blast*
qed
moreover have $?\vartheta \neq$ {\$\$}
by *simp*
ultimately have $\vdash$ **S** $?\vartheta$ $?C$
by (*rule Sub*[*OF* §*5*])
moreover have S $?\vartheta$ $?C = \forall x_\alpha.\ (A \vee^{\mathcal{Q}} B) \supset^{\mathcal{Q}} (A \vee^{\mathcal{Q}} \forall x_\alpha.\ B)$
using ‹$(p, o) \neq (x, \alpha)$› **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 $A \in wffs_o$ **and** $B \in wffs_o$
and $(x, \alpha) \notin$ *free-vars* A
shows $\vdash \forall x_\alpha.\ (A \supset^{\mathcal{Q}} B) \equiv^{\mathcal{Q}} (A \supset^{\mathcal{Q}} (\forall x_\alpha.\ B))$
proof −
have *is-tautology* $(\mathfrak{x}_o \equiv^{\mathcal{Q}} (T_o \supset^{\mathcal{Q}} \mathfrak{x}_o))$ (**is** ‹*is-tautology* $?C_1$›)
using $\mathcal{V}_B$*-simps* **by** *simp*
have *is-tautology* $(\mathfrak{x}_o \supset^{\mathcal{Q}} (\mathfrak{x}_o \equiv^{\mathcal{Q}} (F_o \supset^{\mathcal{Q}} \mathfrak{y}_o)))$ (**is** ‹*is-tautology* $?C_2$›)
using $\mathcal{V}_B$*-simps* **by** *simp*
have §*1*: $\vdash \forall x_\alpha.\ B \equiv^{\mathcal{Q}} (T_o \supset^{\mathcal{Q}} \forall x_\alpha.\ B)$
proof (*intro rule-P*(*2*))
show *is-tautologous* $(\forall x_\alpha.\ B \equiv^{\mathcal{Q}} (T_o \supset^{\mathcal{Q}} \forall x_\alpha.\ B))$
proof −
let $?\vartheta = \{(\mathfrak{x}, o) \rightarrowtail \forall x_\alpha.\ B\}$
from *assms*(*2*) **have** *is-pwff-substitution* $?\vartheta$
using *pwffs-subset-of-wffso* **by** *auto*
moreover have $\forall x_\alpha.\ B \equiv^{\mathcal{Q}} (T_o \supset^{\mathcal{Q}} \forall x_\alpha.\ B) =$ **S** $?\vartheta$ $?C_1$
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 $?\vartheta = \{(\mathfrak{x},\ o) \rightarrowtail B\}$
from *assms*(*2*) **have** *is-pwff-substitution* $?\vartheta$
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 $?\vartheta = \{(\mathfrak{x},\ o) \rightarrowtail \forall x_\alpha.\ T_o,\ (\mathfrak{y},\ o) \rightarrowtail \forall x_\alpha.\ B\}$
from *assms*(*2*) **have** *is-pwff-substitution* $?\vartheta$
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 $?\vartheta = \{(\mathfrak{x},\ o) \rightarrowtail B\}$ **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* $?\vartheta$
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*: $\vdash \forall x_\alpha.\ (T_o \supset^{\mathcal{Q}} B) \equiv^{\mathcal{Q}} (T_o \supset^{\mathcal{Q}} \forall x_\alpha.\ B)$
  unfolding *equivalence-def* **by** (*rule rule-R*[**where** *p* = [«,»,»,«]]) *force+*
obtain *p* **where** $(p, o) \notin$ *vars B* **and** $p \neq x$
  using *fresh-var-existence* **and** *vars-form-finiteness* **by** (*metis finite-insert insert-iff*)
from ‹$(p, o) \notin$ *vars B*› **have** *sub*: **S** $\{(p, o) \rightarrowtail C\}$ $B = B$ **for** *C*
  using *free-var-singleton-substitution-neutrality* **and** *free-vars-in-all-vars* **by** *blast*
have §*8*: $\vdash \forall x_\alpha.\ (p_o \supset^{\mathcal{Q}} B) \equiv^{\mathcal{Q}} (p_o \supset^{\mathcal{Q}} \forall x_\alpha.\ B)$ (**is** ‹$\vdash$ *?C*$_3$›)
proof −
  from *sub* **and** §*7* **have** $\vdash$ **S** $\{(p, o) \rightarrowtail T_o\}$ *?C*$_3$
    using ‹$p \neq x$› **by** *auto*
  moreover from *sub* **and** §*6* **have** $\vdash$ **S** $\{(p, o) \rightarrowtail F_o\}$ *?C*$_3$
    using ‹$p \neq x$› **by** *auto*
  moreover from *assms*(*2*) **have** *?C*$_3 \in$ *wffs*$_o$
    using *pwffs-subset-of-wffso* **by** *auto*
  ultimately show *?thesis*
    by (*rule Cases*)
qed
then show *?thesis*
proof −
  let *?ϑ* = $\{(p, o) \rightarrowtail A\}$
  from *assms*(*1*) **have** *is-substitution ?ϑ*
    using *pwffs-subset-of-wffso* **by** *auto*
  moreover have
    $\forall v \in$ *fmdom′ ?ϑ*. *var-name v* $\notin$ *free-var-names* ({}::*form set*) $\wedge$ *is-free-for* (*?ϑ* $$! *v*) *v ?C*$_3$
  proof
    fix *v*
    assume $v \in$ *fmdom′ ?ϑ*
    then have $v = (p, o)$
      by *simp*
    with *assms*(*3*) **and** ‹$(p, o) \notin$ *vars B*› **have** *is-free-for* (*?ϑ* $$! *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* $\notin$ *free-var-names* ({}::*form set*)
      by *simp*
    ultimately show *var-name v* $\notin$ *free-var-names* ({}::*form set*) $\wedge$ *is-free-for* (*?ϑ* $$! *v*) *v ?C*$_3$
      unfolding ‹$v = (p, o)$› **by** *blast*
  qed
  moreover have *?ϑ* $\neq$ {$$}
    by *simp*
  ultimately have $\vdash$ **S** *?ϑ ?C*$_3$
    by (*rule Sub*[*OF* §*8*])
  moreover have S *?ϑ ?C*$_3$ $= \forall x_\alpha.\ (A \supset^{\mathcal{Q}} B) \equiv^{\mathcal{Q}} (A \supset^{\mathcal{Q}} \forall x_\alpha.\ B)$
    using ‹$p \neq x$› **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** $A \in wffs_o$ **and** $B \in wffs_o$
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* $v \in$ *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** $\forall v' \in S - \{v\}.\ v' \notin$ *free-vars* $(\{A\} \cup \mathcal{H})$
by *blast*
ultimately have $\mathcal{H} \vdash A \supset^{\mathcal{Q}} \forall^{\mathcal{Q}}{}_{\star}\ vs\ B$
using *assms*(*1*,*2*) **by** (*intro Cons.IH*)
moreover from *Cons.prems*(*4*) **and** ‹$v = (x, \alpha)$› **and** $*$ **have** $(x, \alpha) \notin$ *free-vars* $(\{A\} \cup \mathcal{H})$
by *auto*
ultimately have $\mathcal{H} \vdash A \supset^{\mathcal{Q}} (\forall x_{\alpha}.\ \forall^{\mathcal{Q}}{}_{\star}\ vs\ B)$
using *assms*(*1*) **by** (*intro prop-5237*)
with ‹$v = (x, \alpha)$› **show** *?thesis*
by *simp*
qed
qed

end

6.38 Proposition 5238

context begin

private lemma *prop-5238-aux*:
assumes $A \in wffs_{\alpha}$ **and** $B \in wffs_{\alpha}$
shows $\vdash ((\lambda x_{\beta}.\ A) =_{\beta\to\alpha} (\lambda x_{\beta}.\ B)) \equiv^{\mathcal{Q}} \forall x_{\beta}.\ (A =_{\alpha} B)$
proof −
have §*1*:
$\vdash (\mathfrak{f}_{\beta\to\alpha} =_{\beta\to\alpha} \mathfrak{g}_{\beta\to\alpha}) \equiv^{\mathcal{Q}} \forall \mathfrak{x}_{\beta}.\ (\mathfrak{f}_{\beta\to\alpha} \bullet \mathfrak{x}_{\beta} =_{\alpha} \mathfrak{g}_{\beta\to\alpha} \bullet \mathfrak{x}_{\beta})$ (**is** ‹$\vdash$ - $\equiv^{\mathcal{Q}} \forall \mathfrak{x}_{\beta}.\ ?C_1$›)
by (*fact axiom-is-derivable-from-no-hyps*[*OF axiom-3*])
then have §*2*:
$\vdash (\mathfrak{f}_{\beta\to\alpha} =_{\beta\to\alpha} \mathfrak{g}_{\beta\to\alpha}) \equiv^{\mathcal{Q}} \forall x_{\beta}.\ (\mathfrak{f}_{\beta\to\alpha} \bullet x_{\beta} =_{\alpha} \mathfrak{g}_{\beta\to\alpha} \bullet x_{\beta})$ (**is** ‹$\vdash\ ?C_2$›)
proof (*cases* $x = \mathfrak{x}$)
case *True*
with §*1* **show** *?thesis*
by (*simp only*:)
next
case *False*
have $?C_1 \in wffs_{o}$
by *blast*
moreover from *False* **have** $(x, \beta) \notin$ *free-vars* $?C_1$
by *simp*
moreover have *is-free-for* $(x_{\beta})\ (\mathfrak{x}, \beta)\ ?C_1$
by (*intro is-free-for-in-equality is-free-for-to-app*) *simp-all*
ultimately have $\vdash \lambda\mathfrak{x}_{\beta}.\ ?C_1 =_{\beta\to o} \lambda x_{\beta}.\ (\mathbf{S}\ \{(\mathfrak{x}, \beta) \rightarrowtail x_{\beta}\}\ ?C_1)$
by (*rule* α)
from §*1* **and** *this* **show** *?thesis*
by (*rule rule-R*[**where** p = [»,»]]) *force+*
qed

then have §*3*:
$\vdash ((\lambda x_\beta.\ A) =_{\beta\to\alpha} (\lambda x_\beta.\ B)) \equiv^{\mathcal{Q}} \forall x_\beta.\ ((\lambda x_\beta.\ A) \bullet x_\beta =_\alpha (\lambda x_\beta.\ B) \bullet x_\beta)$
proof −
let $?\vartheta = \{(\mathfrak{f},\ \beta{\to}\alpha) \rightarrowtail \lambda x_\beta.\ A,\ (\mathfrak{g},\ \beta{\to}\alpha) \rightarrowtail \lambda x_\beta.\ B\}$
have $\lambda x_\beta.\ A \in wffs_{\beta\to\alpha}$ **and** $\lambda x_\beta.\ B \in wffs_{\beta\to\alpha}$
by (*blast intro*: *assms*(*1*,*2*))+
then have *is-substitution* $?\vartheta$
by *simp*
moreover have
$\forall v \in fmdom'\ ?\vartheta.$ *var-name* $v \notin$ *free-var-names* ({}::*form set*) $\wedge$ *is-free-for* ($?\vartheta$ \$\$! v) v $?C_2$
proof
fix v
assume $v \in fmdom'\ ?\vartheta$
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* ($?\vartheta$ \$\$! 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 $?\vartheta \neq$ {\$\$}
by *simp*
ultimately have $\vdash$ **S** $?\vartheta\ ?C_2$
by (*rule Sub*[*OF* §*2*])
then show *?thesis*
by *simp*
qed
then have §*4*: $\vdash ((\lambda x_\beta.\ A) =_{\beta\to\alpha} (\lambda x_\beta.\ B)) \equiv^{\mathcal{Q}} \forall x_\beta.\ (A =_\alpha (\lambda x_\beta.\ B) \bullet x_\beta)$
proof −
have $\vdash (\lambda x_\beta.\ A) \bullet x_\beta =_\alpha A$
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 $\vdash (\lambda x_\beta.\ B) \cdot x_\beta =_\alpha B$
using *prop-5208*[**where** $vs = [(x,\ \beta)]$] **and** *assms*(*2*) **by** *simp*
from §*4* **and** *this* **show** *?thesis*
by (*rule rule-R*[**where** p = [»,»,«,»]]) *force+*
qed
qed

proposition *prop-5238*:
assumes $vs \neq []$ **and** $A \in wffs_\alpha$ **and** $B \in wffs_\alpha$
shows $\vdash \lambda^{\mathcal{Q}}{}_\star\ vs\ A =_{foldr\ (\to)\ (map\ var\text{-}type\ vs)\ \alpha} \lambda^{\mathcal{Q}}{}_\star\ vs\ B \equiv^{\mathcal{Q}} \forall^{\mathcal{Q}}{}_\star\ vs\ (A =_\alpha B)$
using *assms* **proof** (*induction vs arbitrary*: *A B* α *rule*: *rev-nonempty-induct*)
case (*single v*)
obtain x **and** β **where** $v = (x,\ \beta)$
by *fastforce*
from *single.prems* **have**
$\lambda^{\mathcal{Q}}{}_\star\ vs\ A =_{foldr\ (\to)\ (map\ var\text{-}type\ vs)\ \alpha} \lambda^{\mathcal{Q}}{}_\star\ vs\ B \equiv^{\mathcal{Q}} \forall^{\mathcal{Q}}{}_\star\ vs\ (A =_\alpha B) \in wffs_o$
by *blast*
with *single.prems* **and** ‹$v = (x,\ \beta)$› **show** *?case*
using *prop-5238-aux* **by** *simp*
next
case (*snoc v vs*)
obtain x **and** β **where** $v = (x,\ \beta)$
by *fastforce*
from *snoc.prems* **have** $\lambda x_\beta.\ A \in wffs_{\beta\to\alpha}$ **and** $\lambda x_\beta.\ B \in wffs_{\beta\to\alpha}$
by *auto*
then have
$\vdash$
$\lambda^{\mathcal{Q}}{}_\star\ vs\ (\lambda x_\beta.\ A) =_{foldr\ (\to)\ (map\ var\text{-}type\ vs)\ (\beta\to\alpha)} \lambda^{\mathcal{Q}}{}_\star\ vs\ (\lambda x_\beta.\ B)$
$\equiv^{\mathcal{Q}}$
$\forall^{\mathcal{Q}}{}_\star\ vs\ ((\lambda x_\beta.\ A) =_{\beta\to\alpha} (\lambda x_\beta.\ B))$
by (*fact snoc.IH*)
moreover from *snoc.prems* **have** $\vdash \lambda x_\beta.\ A =_{\beta\to\alpha} \lambda x_\beta.\ B \equiv^{\mathcal{Q}} \forall x_\beta.\ (A =_\alpha B)$
by (*fact prop-5238-aux*)
ultimately have
$\vdash$
$\lambda^{\mathcal{Q}}{}_\star\ vs\ (\lambda x_\beta.\ A) =_{foldr\ (\to)\ (map\ var\text{-}type\ vs)\ (\beta\to\alpha)} \lambda^{\mathcal{Q}}{}_\star\ vs\ (\lambda x_\beta.\ B)$
$\equiv^{\mathcal{Q}}$
$\forall^{\mathcal{Q}}{}_\star\ vs\ \forall x_\beta.\ (A =_\alpha B)$
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 $C⦃p \leftarrow B⦄ \rhd D$
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** $A = C_1 \cdot C_2$
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 ‹$A = C_1 \cdot C_2$› **and** ‹$D = B$› **and** *app-is-wff.hyps(1,2)* **and** *assms(3)* **have** $\alpha = \delta$
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** $A \preceq_{p'} C_1$
by *blast*
moreover obtain D_1 **where** $D = D_1 \cdot C_2$ **and** $C_1 \langle\!| p' \leftarrow B |\!\rangle \rhd D_1$
using *app-is-wff.prems(3)* **and** ‹$p = « \# p'$› **by** (*force dest*: *is-replacement-at.cases*)
ultimately have $\vdash C_1 =_{\gamma \to \delta} D_1$
using *app-is-wff.IH(1)* **and** *assms(3)* **by** *blast*
moreover have $\vdash C_2 =_\gamma C_2$
by (*fact prop-5200[OF app-is-wff.hyps(2)]*)
ultimately have $\vdash C_1 \cdot C_2 =_\delta D_1 \cdot C_2$
using *Equality-Rules(4)* **by** (*simp only*:)
with ‹$D = D_1 \cdot C_2$› **show** *?thesis*
by (*simp only*:)
next
case *c*
then obtain p' **where** $p = » \# p'$ **and** $A \preceq_{p'} C_2$
by *blast*
moreover obtain D_2 **where** $D = C_1 \cdot D_2$ **and** $C_2 \langle\!| p' \leftarrow B |\!\rangle \rhd D_2$
using *app-is-wff.prems(3)* **and** ‹$p = » \# p'$› **by** (*force dest*: *is-replacement-at.cases*)
ultimately have $\vdash C_2 =_\gamma D_2$
using *app-is-wff.IH(2)* **and** *assms(3)* **by** *blast*
moreover have $\vdash C_1 =_{\gamma \to \delta} C_1$
by (*fact prop-5200[OF app-is-wff.hyps(1)]*)
ultimately have $\vdash C_1 \cdot C_2 =_\delta C_1 \cdot D_2$
using *Equality-Rules(4)* **by** (*simp only*:)
with ‹$D = C_1 \cdot D_2$› **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*) $\exists p'.\ p = « \# p' \wedge A \preceq_{p'} C'$
using *subforms-from-abs* **by** *blast*
then show *?case*
proof *cases*
case *a*
with *abs-is-wff.prems(1)* **have** $A = \lambda x_\gamma.\ C'$
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 ‹$A = \lambda x_\gamma.\ C'$› **and** ‹$D = B$› **and** *abs-is-wff.hyps*(*1*) **and** *assms*(*3*) **have** $\alpha = \gamma{\to}\delta$
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** $A \preceq_{p'} C'$
by *blast*
moreover obtain D' **where** $D = \lambda x_\gamma.\ D'$ **and** $C'\llparenthesis p' \leftarrow B\rrparenthesis \rhd D'$
using *abs-is-wff.prems*(*3*) **and** ‹$p = « \# p'$› **by** (*force dest*: *is-replacement-at.cases*)
ultimately have $\vdash C' =_\delta D'$
using *abs-is-wff.IH* **and** *assms*(*3*) **by** *blast*
then have $\vdash \lambda x_\gamma.\ C' =_{\gamma\to\delta} \lambda x_\gamma.\ D'$
proof −
from ‹$\vdash C' =_\delta D'$› **have** $\vdash \forall x_\gamma.\ (C' =_\delta D')$
using *Gen* **by** *simp*
moreover from ‹$\vdash C' =_\delta D'$› **and** *abs-is-wff.hyps* **have** $D' \in wffs_\delta$
using *hyp-derivable-form-is-wffso* **by** (*blast dest*: *wffs-from-equality*)
with *abs-is-wff.hyps* **have** $\vdash (\lambda x_\gamma.\ C' =_{\gamma\to\delta} \lambda x_\gamma.\ D') \equiv^{\mathcal{Q}} \forall x_\gamma.\ (C' =_\delta D')$
using *prop-5238*[**where** $vs = [(x, \gamma)]$] **by** *simp*
ultimately show *?thesis*
using *Equality-Rules*(*1*,*2*) **unfolding** *equivalence-def* **by** *blast*
qed
with ‹$D = \lambda x_\gamma.\ D'$› **show** *?thesis*
by (*simp only*:)
qed
qed

context
begin

private lemma *prop-5239-aux-1*:
assumes $p \in positions\ (\bullet^{\mathcal{Q}}_\star\ (FVar\ v)\ (map\ FVar\ vs))$
and $p \neq replicate\ (length\ vs)$ «
shows
$(\exists A\ B.\ A \bullet B \preceq_p (\bullet^{\mathcal{Q}}_\star\ (FVar\ v)\ (map\ FVar\ vs)))$
$\vee$
$(\exists v \in lset\ vs.\ occurs\text{-}at\ v\ p\ (\bullet^{\mathcal{Q}}_\star\ (FVar\ v)\ (map\ FVar\ vs)))$
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) $\exists p' \in$ *positions* $({\cdot}^{\mathcal{Q}}{}_{\star}$ $(FVar\ v)$ $(map\ FVar\ vs))$. $p = « \# p'$
using *surj-pair*[*of v'*] by *fastforce*
then show *?case*
proof *cases*
case *c*
then obtain p' where $p' \in$ *positions* $({\cdot}^{\mathcal{Q}}{}_{\star}$ $(FVar\ v)$ $(map\ FVar\ vs))$ and $p = « \# p'$
by *blast*
from ‹$p = « \# p'$› and *snoc.prems*(*2*) have $p' \neq$ *replicate* (*length vs*) «
by *force*
then have
$(\exists A\ B.\ A \cdot B \preceq_{p'} {\cdot}^{\mathcal{Q}}{}_{\star}$ $(FVar\ v)$ $(map\ FVar\ vs))$
$\vee$
$(\exists v \in lset\ vs.\ occurs\text{-}at\ v\ p'$ $({\cdot}^{\mathcal{Q}}{}_{\star}$ $(FVar\ v)$ $(map\ FVar\ vs)))$
using ‹$p' \in$ *positions* $({\cdot}^{\mathcal{Q}}{}_{\star}$ $(FVar\ v)$ $(map\ FVar\ vs))$› and *snoc.IH* by *simp*
with ‹$p = « \# p'$› show *?thesis*
by *auto*
qed *simp-all*
qed

private lemma *prop-5239-aux-2*:
assumes $t \notin lset\ vs \cup vars\ C$
and $C\langle\!|p \leftarrow ({\cdot}^{\mathcal{Q}}{}_{\star}$ $(FVar\ t)$ $(map\ FVar\ vs))|\!\rangle \rhd G$
and $C\langle\!|p \leftarrow ({\cdot}^{\mathcal{Q}}{}_{\star}$ $(\lambda^{\mathcal{Q}}{}_{\star}\ vs\ A)$ $(map\ FVar\ vs))|\!\rangle \rhd G'$
shows **S** $\{t \rightarrowtail \lambda^{\mathcal{Q}}{}_{\star}\ vs\ A\}$ $G = G'$ (is ‹**S** $?\vartheta$ $G = G'$›)
proof −
have **S** $?\vartheta$ $({\cdot}^{\mathcal{Q}}{}_{\star}$ $(FVar\ t)$ $(map\ FVar\ vs))$ $=$ ${\cdot}^{\mathcal{Q}}{}_{\star}$ (**S** $?\vartheta$ $(FVar\ t))$ $(map$ $(\lambda v'.$ **S** $?\vartheta$ $v')$ $(map\ FVar$
$vs))$
using *generalized-app-substitution* by *blast*
moreover have **S** $?\vartheta$ $(FVar\ t) = \lambda^{\mathcal{Q}}{}_{\star}\ vs\ A$
using *surj-pair*[*of t*] by *fastforce*
moreover from *assms*(*1*) have *map* $(\lambda v'.$ **S** $?\vartheta$ $v')$ $(map\ FVar\ vs) = map\ FVar\ vs$
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 $G = {\cdot}^{\mathcal{Q}}{}_{\star}$ $(FVar\ t)$ $(map\ FVar\ vs)$
by (*blast dest*: *is-replacement-at.cases*)+
moreover from *FVar.prems*(*6*) and ‹$p = []$› have $G' = {\cdot}^{\mathcal{Q}}{}_{\star}$ $(\lambda^{\mathcal{Q}}{}_{\star}\ vs\ A)$ $(map\ FVar\ vs)$
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 $G = {\cdot}^{\mathcal{Q}}{}_{\star}$ $(FVar\ t)$ $(map\ FVar\ vs)$
by (*blast dest*: *is-replacement-at.cases*)+
moreover from *FCon.prems*(*6*) and ‹$p = []$› have $G' = {\cdot}^{\mathcal{Q}}{}_{\star}$ $(\lambda^{\mathcal{Q}}{}_{\star}\ vs\ A)$ $(map\ FVar\ vs)$
by (*blast dest*: *is-replacement-at.cases*)
ultimately show *?case*

using *FCon.prems(1−3)* **by** (*simp only*:)
next
case (*FApp* C_1 C_2)
from *FApp.prems(4)* **have** $t \notin$ *lset vs* $\cup$ *vars* C_1 **and** $t \notin$ *lset vs* $\cup$ *vars* C_2
by *auto*
consider (a) $p = []$ | (b) $\exists\, p'.\ p = «\ \#\ p'$ | (c) $\exists\, p'.\ p = »\ \#\ p'$
by (*metis direction.exhaust list.exhaust*)
then show *?case*
proof *cases*
case a
with *FApp.prems(5)* **have** $G = \bullet^{\mathcal{Q}}{}_{\star}\ (FVar\ t)\ (map\ FVar\ vs)$
by (*blast dest*: *is-replacement-at.cases*)
moreover from *FApp.prems(6)* **and** ‹$p = []$› **have** $G' = \bullet^{\mathcal{Q}}{}_{\star}\ (\lambda^{\mathcal{Q}}{}_{\star}\ vs\ A)\ (map\ FVar\ vs)$
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** G_1 **where** $G = G_1 \bullet C_2$ **and** $C_1 \langle\!\langle p' \leftarrow (\bullet^{\mathcal{Q}}{}_{\star}\ (FVar\ t)\ (map\ FVar\ vs)) \rangle\!\rangle \rhd G_1$
by (*blast elim*: *is-replacement-at.cases*)
moreover from ‹$p = «\ \#\ p'$› **and** *FApp.prems(6)*
obtain G'_1 **where** $G' = G'_1 \bullet C_2$ **and** $C_1 \langle\!\langle p' \leftarrow (\bullet^{\mathcal{Q}}{}_{\star}\ (\lambda^{\mathcal{Q}}{}_{\star}\ vs\ A)\ (map\ FVar\ vs)) \rangle\!\rangle \rhd G'_1$
by (*blast elim*: *is-replacement-at.cases*)
moreover from ‹$t \notin$ *lset vs* $\cup$ *vars* C_2› **have** **S** $\{t \rightarrowtail \lambda^{\mathcal{Q}}{}_{\star}\ vs\ A\}\ C_2 = C_2$
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)* ‹$t \notin$ *lset vs* $\cup$ *vars* C_1›] **by** *simp*
next
case c
then obtain p' **where** $p = »\ \#\ p'$
by *blast*
with *FApp.prems(5)* **obtain** G_2 **where** $G = C_1 \bullet G_2$ **and** $C_2 \langle\!\langle p' \leftarrow (\bullet^{\mathcal{Q}}{}_{\star}\ (FVar\ t)\ (map\ FVar\ vs)) \rangle\!\rangle \rhd G_2$
by (*blast elim*: *is-replacement-at.cases*)
moreover from ‹$p = »\ \#\ p'$› **and** *FApp.prems(6)*
obtain G'_2 **where** $G' = C_1 \bullet G'_2$ **and** $C_2 \langle\!\langle p' \leftarrow (\bullet^{\mathcal{Q}}{}_{\star}\ (\lambda^{\mathcal{Q}}{}_{\star}\ vs\ A)\ (map\ FVar\ vs)) \rangle\!\rangle \rhd G'_2$
by (*blast elim*: *is-replacement-at.cases*)
moreover from ‹$t \notin$ *lset vs* $\cup$ *vars* C_1› **have** **S** $\{t \rightarrowtail \lambda^{\mathcal{Q}}{}_{\star}\ vs\ A\}\ C_1 = C_1$
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)* ‹$t \notin$ *lset vs* $\cup$ *vars* C_2›] **by** *simp*
qed
next
case (*FAbs v* C')

from *FAbs.prems*(*4*) **have** $t \notin$ *lset vs* $\cup$ *vars* C' **and** $t \neq v$
  using *vars-form.elims* **by** *blast+*
from *FAbs.prems*(*5*) **consider** (*a*) $p = []$ | (*b*) $\exists p'.\ p = « \# p'$
  using *is-replacement-at.simps* **by** *blast*
then show *?case*
proof *cases*
  case *a*
  with *FAbs.prems*(*5*) **have** $G = \bullet^{\mathcal{Q}}_{\star}$ (*FVar t*) (*map FVar vs*)
    by (*blast dest*: *is-replacement-at.cases*)
  moreover from *FAbs.prems*(*6*) **and** ‹$p = []$› **have** $G' = \bullet^{\mathcal{Q}}_{\star}$ ($\lambda^{\mathcal{Q}}_{\star}$ *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 G_1 **where** $G =$ *FAbs v* G_1 **and** $C'\langle\!|p' \leftarrow (\bullet^{\mathcal{Q}}_{\star}$ (*FVar t*) (*map FVar vs*))$|\!\rangle \rhd G_1$
    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** $C'\langle\!|p' \leftarrow (\bullet^{\mathcal{Q}}_{\star}$ ($\lambda^{\mathcal{Q}}_{\star}$ *vs A*) (*map FVar vs*))$|\!\rangle \rhd G'_1$
    by (*blast elim*: *is-replacement-at.cases*)
  ultimately have S $\{t \rightarrowtail \lambda^{\mathcal{Q}}_{\star}$ *vs A*$\}$ $G_1 = G'_1$
    using *FAbs.IH*[*OF FAbs.prems*(*1*−*3*) ‹$t \notin$ *lset vs* $\cup$ *vars* C'›] **by** *simp*
  with ‹$G =$ *FAbs v* G_1› **and** ‹$G' =$ *FAbs v* G'_1› **and** ‹$t \neq v$› **show** *?thesis*
    using *surj-pair*[*of v*] **by** *fastforce*
qed
qed
qed

private lemma *prop-5239-aux-3*:
  assumes $t \notin$ *lset vs* $\cup$ *vars* $\{A, C\}$
  and $C\langle\!|p \leftarrow (\bullet^{\mathcal{Q}}_{\star}$ (*FVar t*) (*map FVar vs*))$|\!\rangle \rhd G$
  and *occurs-at t* p' *G*
  shows $p' = p$ @ *replicate* (*length vs*) « (**is** ‹$p' = ?p_t$›)
proof (*cases vs* $= []$)
  case *True*
  then have $t \notin$ *vars C*
    using *assms*(*1*) **by** *auto*
  moreover from *True* **and** *assms*(*2*) **have** $C\langle\!|p \leftarrow$ *FVar t*$|\!\rangle \rhd G$
    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 $p' \neq ?p_t$
    have $\neg$ *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 b_2
with ‹*occurs-at* t ?p_t G› **and** *assms*(3) **show** *?thesis*
using *loop-subform-impossibility* **by** *blast*
next
case b_3
from b **obtain** p'' **where** $p' = p$ @ p'' **and** $p'' \in$ *positions* $(\bullet^{\mathcal{Q}}{}_{\star}$ $(FVar\ t)$ $(map\ FVar\ vs))$
using *is-replacement-at-new-positions* **and** ‹$p' \in$ *positions* G› **and** *assms*(2) **by** *blast*
moreover have $p'' \neq$ *replicate* (*length vs*) «
using ‹$p' = p$ @ p''› **and** ‹$p' \neq$?p_t› **by** *blast*
ultimately consider
$(b_{3\text{-}1})$ $\exists F_1\ F_2.\ F_1 \bullet F_2 \preceq_{p''} (\bullet^{\mathcal{Q}}{}_{\star}\ (FVar\ t)\ (map\ FVar\ vs))$
| $(b_{3\text{-}2})$ $\exists v \in$ *lset vs. occurs-at* v p'' $(\bullet^{\mathcal{Q}}{}_{\star}\ (FVar\ t)\ (map\ FVar\ vs))$
using *prop-5239-aux-1* **and** $b_3(1,2)$ **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 $b_{3\text{-}1}$
with *assms*(2) **and** ‹$p' = p$ @ p''› **have** $\exists F_1\ F_2.\ F_1 \bullet F_2 \preceq_{p'} G$
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 $b_{3\text{-}2}$
with *assms*(2) **and** ‹$p' = p$ @ p''› **have** $\exists v \in$ *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 ?p_t G› **and** *assms*(3) **show** *?thesis*
using *loop-subform-impossibility* **by** *blast*
qed
qed
qed
qed

private lemma *prop-5239-aux-4*:
assumes $t \notin$ *lset vs* $\cup$ *vars* $\{A,\ C\}$
and $A \preceq_p C$
and *lset vs* $\supseteq$ *capture-exposed-vars-at* p C A
and $C\langle\!|p \leftarrow (\bullet^{\mathcal{Q}}{}_{\star}\ (FVar\ t)\ (map\ FVar\ vs))|\!\rangle \rhd G$
shows *is-free-for* $(\lambda^{\mathcal{Q}}{}_{\star}\ vs\ A)$ t G
unfolding *is-free-for-def* **proof** (*intro ballI impI*)
let ?$p_t = p$ @ *replicate* (*length vs*) «
from *assms*(4) **have** *FVar* $t \preceq_{?p_t} G$
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 $v' \in$ *free-vars* ($\lambda^{\mathcal{Q}}{}_\star$ *vs A*) **and** $p' \in$ *positions G* **and** *is-free-at t* p' *G*
have $v' \notin$ *binders-at G* $?p_t$
proof −
have *free-vars* ($\lambda^{\mathcal{Q}}{}_\star$ *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* ($\lambda^{\mathcal{Q}}{}_\star$ *vs A*) ⊆ *free-vars A* − (*binders-at G p* ∩ *free-vars A*) **.**
moreover have *binders-at* ($\bullet^{\mathcal{Q}}{}_\star$ (*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 ‹$v' \in$ *free-vars* ($\lambda^{\mathcal{Q}}{}_\star$ *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* ‹$p' \in$ *positions G*›] **and** *in-scope-of-abs-alt-def* **by** *auto*
qed

proposition *prop-5239*:
assumes *is-rule-R-app p D C* ($A =_\alpha B$)
and *lset vs* =
{(x, β) | x β p' *E*. *strict-prefix* p' *p* ∧ λx_β. *E* $\preceq_{p'}$ *C* ∧ (x, β) ∈ *free-vars* ($A =_\alpha B$)}
shows ⊢ $\forall^{\mathcal{Q}}{}_\star$ *vs* ($A =_\alpha B$) $\supset^{\mathcal{Q}}$ ($C \equiv^{\mathcal{Q}} D$)
proof −
let $?\gamma$ = *foldr* (→) (*map var-type vs*) α
obtain *t* **where** (t, $?\gamma$) ∉ *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** $A \in wffs_\alpha$ **and** $B \in wffs_\alpha$ **and** $A \preceq_p C$
using *wffs-from-equality*[*OF equality-wff*] **by** *simp-all*
from *assms*(*1*) **have** $C \in wffs_o$ **and** $D \in wffs_o$
using *replacement-preserves-typing* **by** *fastforce+*
have $\bullet^{\mathcal{Q}}{}_\star$ $t_{?\gamma}$ (*map FVar vs*) $\in wffs_\alpha$
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* ← ($\bullet^{\mathcal{Q}}{}_\star$ $t_{?\gamma}$ (*map FVar vs*))⦈ ⊳ *G*

using *is-replacement-at-existence* **by** *blast*
with ‹$A \preceq_p C$› **and** ‹$\bullet^{\mathcal{Q}}{}_\star\ t_{?\gamma}$ (*map FVar vs*) $\in$ *wffs*$_\alpha$› **have** $G \in$ *wffs*$_o$
using ‹$A \in$ *wffs*$_\alpha$› **and** ‹$C \in$ *wffs*$_o$› **and** *replacement-preserves-typing* **by** *blast*
let $?\vartheta = \{(\mathfrak{h},\ ?\gamma{\to}o) \rightarrowtail \lambda t_{?\gamma}.\ G,\ (\mathfrak{x},\ ?\gamma) \rightarrowtail \lambda^{\mathcal{Q}}{}_\star\ vs\ A,\ (\mathfrak{y},\ ?\gamma) \rightarrowtail \lambda^{\mathcal{Q}}{}_\star\ vs\ B\}$
and $?A = (\mathfrak{x}_{?\gamma} =_{?\gamma} \mathfrak{y}_{?\gamma}) \supset^{\mathcal{Q}} (\mathfrak{h}_{?\gamma\to o} \bullet \mathfrak{x}_{?\gamma} \equiv^{\mathcal{Q}} \mathfrak{h}_{?\gamma\to o} \bullet \mathfrak{y}_{?\gamma})$
have $\vdash\ ?A$
by (*fact axiom-is-derivable-from-no-hyps*[*OF axiom-2*])
moreover have $\lambda t_{?\gamma}.\ G \in$ *wffs*$_{?\gamma\to o}$ **and** $\lambda^{\mathcal{Q}}{}_\star\ vs\ A \in$ *wffs*$_{?\gamma}$ **and** $\lambda^{\mathcal{Q}}{}_\star\ vs\ B \in$ *wffs*$_{?\gamma}$
by (*blast intro*: ‹$G \in$ *wffs*$_o$› ‹$A \in$ *wffs*$_\alpha$› ‹$B \in$ *wffs*$_\alpha$›)+
then have *is-substitution* $?\vartheta$
by *simp*
moreover have
$\forall v \in$ *fmdom′* $?\vartheta$. *var-name* $v \notin$ *free-var-names* ({}::*form set*) $\wedge$ *is-free-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 $?\vartheta\ ?A = (\lambda^{\mathcal{Q}}{}_\star\ vs\ A =_{?\gamma} \lambda^{\mathcal{Q}}{}_\star\ vs\ B) \supset^{\mathcal{Q}} ((\lambda t_{?\gamma}.\ G) \bullet (\lambda^{\mathcal{Q}}{}_\star\ vs\ A) \equiv^{\mathcal{Q}} (\lambda t_{?\gamma}.\ G) \bullet (\lambda^{\mathcal{Q}}{}_\star\ vs\ B))$
by *simp*
ultimately have §*1*:
$\vdash (\lambda^{\mathcal{Q}}{}_\star\ vs\ A =_{?\gamma} \lambda^{\mathcal{Q}}{}_\star\ vs\ B) \supset^{\mathcal{Q}} ((\lambda t_{?\gamma}.\ G) \bullet (\lambda^{\mathcal{Q}}{}_\star\ vs\ A) \equiv^{\mathcal{Q}} (\lambda t_{?\gamma}.\ G) \bullet (\lambda^{\mathcal{Q}}{}_\star\ vs\ B))$
by (*simp only*:)
then have §*2*: $\vdash (\forall^{\mathcal{Q}}{}_\star\ vs\ (A =_\alpha B)) \supset^{\mathcal{Q}} ((\lambda t_{?\gamma}.\ G) \bullet (\lambda^{\mathcal{Q}}{}_\star\ vs\ A) \equiv^{\mathcal{Q}} (\lambda t_{?\gamma}.\ G) \bullet (\lambda^{\mathcal{Q}}{}_\star\ vs\ B))$
proof (*cases vs* = [])
case *True*
with §*1* **show** *?thesis*
by *simp*
next
case *False*
from §*1* **and** *prop-5238*[*OF False* ‹$A \in$ *wffs*$_\alpha$› ‹$B \in$ *wffs*$_\alpha$›] **show** *?thesis*
unfolding *equivalence-def* **by** (*rule rule-R*[**where** p = [«,»]]) *force*+
qed
moreover have $\vdash (\lambda t_{?\gamma}.\ G) \bullet (\lambda^{\mathcal{Q}}{}_\star\ vs\ A) =_o C$ **and** $\vdash (\lambda t_{?\gamma}.\ G) \bullet (\lambda^{\mathcal{Q}}{}_\star\ vs\ B) =_o D$
proof −
from *assms*(*1*) **have** $B \preceq_p D$
using *is-replacement-at-minimal-change*(*1*) **by** *force*
from *assms*(*1*) **have** $D\langle\!\langle p \leftarrow (\bullet^{\mathcal{Q}}{}_\star\ t_{?\gamma}$ (*map FVar vs*))$\rangle\!\rangle \rhd G$
using ‹$C\langle\!\langle p \leftarrow (\bullet^{\mathcal{Q}}{}_\star\ t_{?\gamma}$ (*map FVar vs*))$\rangle\!\rangle \rhd G$› **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** ‹$C\langle\!|p \leftarrow (\bullet^{\mathcal{Q}}{}_\star\ t_{?\gamma}$ (*map FVar vs*))$|\!\rangle$
$\rhd$ G›
    and ‹$D\langle\!|p \leftarrow (\bullet^{\mathcal{Q}}{}_\star\ t_{?\gamma}$ (*map FVar vs*))$|\!\rangle \rhd G$› **by** *meson+*
qed
then have $\vdash (\lambda t_{?\gamma}.\ G) \bullet (\lambda^{\mathcal{Q}}{}_\star$ *vs* $A) =_o$ **S** $\{(t,\ ?\gamma) \rightarrowtail \lambda^{\mathcal{Q}}{}_\star$ *vs* $A\}$ G
  and $\vdash (\lambda t_{?\gamma}.\ G) \bullet (\lambda^{\mathcal{Q}}{}_\star$ *vs* $B) =_o$ **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 $C\langle\!|p \leftarrow (\bullet^{\mathcal{Q}}{}_\star\ (\lambda^{\mathcal{Q}}{}_\star$ *vs* $A)$ (*map FVar vs*))$|\!\rangle \rhd G'_1$
  and $D\langle\!|p \leftarrow (\bullet^{\mathcal{Q}}{}_\star\ (\lambda^{\mathcal{Q}}{}_\star$ *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 ‹$C\langle\!|p \leftarrow (\bullet^{\mathcal{Q}}{}_\star\ t_{?\gamma}$ (*map FVar vs*))$|\!\rangle \rhd G$› **and** ‹$D\langle\!|p \leftarrow (\bullet^{\mathcal{Q}}{}_\star\ t_{?\gamma}$ *map FVar vs*)$|\!\rangle \rhd G$›
    and ‹$C\langle\!|p \leftarrow (\bullet^{\mathcal{Q}}{}_\star\ (\lambda^{\mathcal{Q}}{}_\star$ *vs* $A)$ (*map FVar vs*))$|\!\rangle \rhd G'_1$›
    and ‹$D\langle\!|p \leftarrow (\bullet^{\mathcal{Q}}{}_\star\ (\lambda^{\mathcal{Q}}{}_\star$ *vs* $B)$ (*map FVar vs*))$|\!\rangle \rhd G'_2$› **and** *prop-5239-aux-2* **by** *blast+*
qed
ultimately have $\vdash (\lambda t_{?\gamma}.\ G) \bullet (\lambda^{\mathcal{Q}}{}_\star$ *vs* $A) =_o G'_1$ **and** $\vdash (\lambda t_{?\gamma}.\ G) \bullet (\lambda^{\mathcal{Q}}{}_\star$ *vs* $B) =_o G'_2$
  by (*simp-all only*:)
moreover
have $\vdash A =_\alpha (\bullet^{\mathcal{Q}}{}_\star\ (\lambda^{\mathcal{Q}}{}_\star$ *vs* $A)$ (*map FVar vs*)) **and** $\vdash B =_\alpha (\bullet^{\mathcal{Q}}{}_\star\ (\lambda^{\mathcal{Q}}{}_\star$ *vs* $B)$ (*map FVar vs*))
unfolding *atomize-conj* **proof** (*cases vs* = [])
  assume *vs* = []

show $\vdash A =_\alpha \cdot^{\mathcal{Q}}{}_\star\ (\lambda^{\mathcal{Q}}{}_\star\ vs\ A)\ (map\ FVar\ vs) \wedge \vdash B =_\alpha \cdot^{\mathcal{Q}}{}_\star\ (\lambda^{\mathcal{Q}}{}_\star\ vs\ B)\ (map\ FVar\ vs)$
unfolding ‹$vs = []$› **using** *prop-5200* **and** ‹$A \in wffs_\alpha$› **and** ‹$B \in wffs_\alpha$› **by** *simp*
next
assume $vs \neq []$
show $\vdash A =_\alpha \cdot^{\mathcal{Q}}{}_\star\ (\lambda^{\mathcal{Q}}{}_\star\ vs\ A)\ (map\ FVar\ vs) \wedge \vdash B =_\alpha \cdot^{\mathcal{Q}}{}_\star\ (\lambda^{\mathcal{Q}}{}_\star\ vs\ B)\ (map\ FVar\ vs)$
using *Equality-Rules*(*2*)[*OF prop-5208*[*OF* ‹$vs \neq []$›]] **and** ‹$A \in wffs_\alpha$› **and** ‹$B \in wffs_\alpha$›
by *blast+*
qed
with
‹$C\langle\!|p \leftarrow (\cdot^{\mathcal{Q}}{}_\star\ (\lambda^{\mathcal{Q}}{}_\star\ vs\ A)\ (map\ FVar\ vs))|\!\rangle \rhd G'_1$›
and
‹$D\langle\!|p \leftarrow (\cdot^{\mathcal{Q}}{}_\star\ (\lambda^{\mathcal{Q}}{}_\star\ vs\ B)\ (map\ FVar\ vs))|\!\rangle \rhd G'_2$›
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 $\vdash (\lambda t_{?\gamma}.\ G) \bullet (\lambda^{\mathcal{Q}}{}_\star\ vs\ A) =_o C$ **and** $\vdash (\lambda t_{?\gamma}.\ G) \bullet (\lambda^{\mathcal{Q}}{}_\star\ vs\ B) =_o D$
using *Equality-Rules*(*3*) **by** *blast+*
qed
ultimately show *?thesis*
proof −
from §*2* **and** ‹$\vdash (\lambda t_{?\gamma}.\ G) \bullet (\lambda^{\mathcal{Q}}{}_\star\ vs\ A) =_o C$› **have**
$\vdash (\forall^{\mathcal{Q}}{}_\star\ vs\ (A =_\alpha B)) \supset^{\mathcal{Q}} (C \equiv^{\mathcal{Q}} (\lambda t_{?\gamma}.\ G) \bullet (\lambda^{\mathcal{Q}}{}_\star\ vs\ B))$
by (*rule rule-R*[**where** p = [»,«,»]]) *force+*
from *this* **and** ‹$\vdash (\lambda t_{?\gamma}.\ G) \bullet (\lambda^{\mathcal{Q}}{}_\star\ vs\ B) =_o D$› **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* $(((H \supset^{\mathcal{Q}} C) \supset^{\mathcal{Q}} ((H \supset^{\mathcal{Q}} E) \supset^{\mathcal{Q}} ((E \supset^{\mathcal{Q}} (C \equiv^{\mathcal{Q}} D)) \supset^{\mathcal{Q}} (H \supset^{\mathcal{Q}} D)))))$
proof −
let $?\vartheta = \{(\mathfrak{x},\ o) \rightarrowtail C,\ (\mathfrak{y},\ o) \rightarrowtail D,\ (\mathfrak{z},\ o) \rightarrowtail E,\ (\mathfrak{h},\ o) \rightarrowtail H\}$
have
is-tautology
$(((\mathfrak{h}_o \supset^{\mathcal{Q}} \mathfrak{x}_o) \supset^{\mathcal{Q}} ((\mathfrak{h}_o \supset^{\mathcal{Q}} \mathfrak{z}_o) \supset^{\mathcal{Q}} ((\mathfrak{z}_o \supset^{\mathcal{Q}} (\mathfrak{x}_o \equiv^{\mathcal{Q}} \mathfrak{y}_o)) \supset^{\mathcal{Q}} (\mathfrak{h}_o \supset^{\mathcal{Q}} \mathfrak{y}_o)))))$
using $\mathcal{V}_B$*-simps* **by** *simp*
moreover have *is-substitution* $?\vartheta$
using *assms* **by** *auto*
moreover have $\forall (x,\ \alpha) \in fmdom'\ ?\vartheta.\ \alpha = o$
by *simp*
moreover have
$((H \supset^{\mathcal{Q}} C) \supset^{\mathcal{Q}} ((H \supset^{\mathcal{Q}} E) \supset^{\mathcal{Q}} ((E \supset^{\mathcal{Q}} (C \equiv^{\mathcal{Q}} D)) \supset^{\mathcal{Q}} (H \supset^{\mathcal{Q}} D))))$
=
S $?\vartheta$ $(((\mathfrak{h}_o \supset^{\mathcal{Q}} \mathfrak{x}_o) \supset^{\mathcal{Q}} ((\mathfrak{h}_o \supset^{\mathcal{Q}} \mathfrak{z}_o) \supset^{\mathcal{Q}} ((\mathfrak{z}_o \supset^{\mathcal{Q}} (\mathfrak{x}_o \equiv^{\mathcal{Q}} \mathfrak{y}_o)) \supset^{\mathcal{Q}} (\mathfrak{h}_o \supset^{\mathcal{Q}} \mathfrak{y}_o)))))$

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

syntax
  -HypDer :: *form* ⇒ *form set* ⇒ *form* ⇒ *bool* (-,- ⊢ - [*50*, *50*, *50*] *50*)
translations
  $\mathcal{H}$, $H \vdash P \rightharpoonup \mathcal{H} \cup \{H\} \vdash P$

theorem *thm-5240*:
  assumes *finite* $\mathcal{H}$
  and $\mathcal{H}$, $H \vdash P$
  shows $\mathcal{H} \vdash H \supset^{\mathcal{Q}} P$
proof −
  from ‹$\mathcal{H}$, $H \vdash P$› **obtain** $\mathcal{S}_1$ **and** $\mathcal{S}_2$ **where** ∗: *is-hyp-proof-of* $(\mathcal{H} \cup \{H\})$ $\mathcal{S}_1$ $\mathcal{S}_2$ P
    using *hyp-derivability-implies-hyp-proof-existence* **by** *blast*
  have $\mathcal{H} \vdash H \supset^{\mathcal{Q}} (\mathcal{S}_2 \; ! \; i')$ **if** i' <*length* $\mathcal{S}_2$ **for** i'
  using *that* **proof** (*induction* i' *rule*: *less-induct*)
    case (*less* i')
    let $?R = \mathcal{S}_2 \; ! \; i'$
    from *less.prems*(*1*) **and** ∗ **have** *is-hyps* $\mathcal{H}$
      by *fastforce*
    from *less.prems* **and** ∗ **have** $?R \in wffs_o$
      using *elem-of-proof-is-wffo*[*simplified*] **by** *auto*
    from *less.prems* **and** ∗ **have** *is-hyp-proof-step* $(\mathcal{H} \cup \{H\})$ $\mathcal{S}_1$ $\mathcal{S}_2$ i'
      by *simp*
    then consider
      (*hyp*) $?R \in \mathcal{H} \cup \{H\}$
    | (*seq*) $?R \in$ *lset* $\mathcal{S}_1$
    | (*rule-R'*) $\exists j\ k\ p.\ \{j, k\} \subseteq \{0..<i'\} \wedge$ *is-rule-R'-app* $(\mathcal{H} \cup \{H\})$ p $?R$ $(\mathcal{S}_2 \; ! \; j)$ $(\mathcal{S}_2 \; ! \; k)$
      by *force*
    then show *?case*
    proof *cases*
      case *hyp*
      then show *?thesis*
      proof (*cases* $?R = H$)
        case *True*
        with ‹$?R \in wffs_o$› **have** *is-tautologous* $(H \supset^{\mathcal{Q}} ?R)$
          using *implication-reflexivity-is-tautologous* **by** (*simp only*:)
        with ‹*is-hyps* $\mathcal{H}$› **show** *?thesis*
          by (*rule rule-P*(*2*))
      next
        case *False*
        with *hyp* **have** $?R \in \mathcal{H}$
          by *blast*
        with ‹*is-hyps* $\mathcal{H}$› **have** $\mathcal{H} \vdash ?R$
          by (*intro dv-hyp*)
        moreover from *less.prems*(*1*) **and** ∗ **have** *is-tautologous* $(?R \supset^{\mathcal{Q}} (H \supset^{\mathcal{Q}} ?R))$

using *principle-of-simplification-is-tautologous*[*OF* ‹$?R \in wffs_o$›] **by** *force*
moreover from ‹$?R \in wffs_o$› **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 ‹$?R \in wffs_o$› **and** *less.prems*(*1*) **and** $*$ **have** *is-tautologous* $(?R \supset^{\mathcal{Q}} (H \supset^{\mathcal{Q}} ?R))$
using *principle-of-simplification-is-tautologous* **by** *force*
moreover from ‹$?R \in wffs_o$› **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** $\mathcal{H} \vdash H \supset^{\mathcal{Q}} C$ **and** $\mathcal{H} \vdash H \supset^{\mathcal{Q}} (A =_\alpha B)$
using *less.IH* **and** *less.prems*(*1*) **by** (*simp*, *force*)
define S **where** $S \equiv$
$\{(x, \beta) \mid x \; \beta \; p' \; E.$ *strict-prefix* $p' \; p \wedge \lambda x_\beta. \; E \preceq_{p'} C \wedge (x, \beta) \in$ *free-vars* $(A =_\alpha B)\}$
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 $\mathcal{H} \vdash H \supset^{\mathcal{Q}} \forall^{\mathcal{Q}}_\star \; vs \; (A =_\alpha B)$
using *generalized-prop-5237*[*OF* ‹*is-hyps* $\mathcal{H}$› ‹$\mathcal{H} \vdash H \supset^{\mathcal{Q}} (A =_\alpha B)$›] **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** $\vdash \forall^{\mathcal{Q}}_\star \; vs \; (A =_\alpha B) \supset^{\mathcal{Q}} (C \equiv^{\mathcal{Q}} ?R)$

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** $\mathcal{H} \cup \mathcal{H}'' \vdash H \supset^{\mathcal{Q}} P$
using *Deduction-Theorem* **by** *simp*
with ‹$\mathcal{H}''$ = *lset hs*› **and** ‹*finite* $\mathcal{H}$› **have** $\mathcal{H} \vdash$ *foldr* ($\supset^{\mathcal{Q}}$) *hs* ($H \supset^{\mathcal{Q}} P$)
using *snoc.IH* **by** *fastforce*
moreover have (*hs* @ [H]) $\supset^{\mathcal{Q}}{}_{\star}$ P = *hs* $\supset^{\mathcal{Q}}{}_{\star}$ ($H \supset^{\mathcal{Q}} P$)
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** ⊢ *hs* $\supset^{\mathcal{Q}}{}_{\star}$ 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 $\mathcal{H} \vdash$ **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 $\mathcal{H} \vdash \forall x_{\alpha}.\ \sim^{\mathcal{Q}} B \supset^{\mathcal{Q}} \sim^{\mathcal{Q}}$ **S** $\{(x, \alpha) \rightarrowtail A\}$ B (**is** ‹$\mathcal{H} \vdash$ *?C* $\supset^{\mathcal{Q}} \sim^{\mathcal{Q}}$ *?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* $((?C \supset^{\mathcal{Q}} {\sim}^{\mathcal{Q}}\ ?D) \supset^{\mathcal{Q}} (?D \supset^{\mathcal{Q}} {\sim}^{\mathcal{Q}}\ ?C))$
proof $-$
have $?C \in \mathit{wffs}_o$ **and** $?D \in \mathit{wffs}_o$
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** ‹$\mathcal{H} \vdash ?C \supset^{\mathcal{Q}} {\sim}^{\mathcal{Q}}\ ?D$› **have** *is-hyps* $\{?C \supset^{\mathcal{Q}} {\sim}^{\mathcal{Q}}\ ?D,\ ?D\}$
using *hyp-derivable-form-is-wffso* **by** *force*
ultimately show *?thesis*
unfolding *exists-def* **using** *assms*(*3*)
and *rule-P*(*1*)
[
where $\mathcal{G} = \{?C \supset^{\mathcal{Q}} {\sim}^{\mathcal{Q}}\ ?D,\ ?D\}$ **and** $hs = [?C \supset^{\mathcal{Q}} {\sim}^{\mathcal{Q}}\ ?D,\ ?D]$ **and** $B = {\sim}^{\mathcal{Q}}\ ?C$,
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 $\bullet^{\mathcal{Q}}{}_{\star}\ B\ (map\ FVar\ vs) \in \mathit{wffs}_{\gamma}$
and $B \in \mathit{wffs}_{\beta}$
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 $*$: $\forall\, k < length\ (map\ FVar\ vs).\ (map\ FVar\ vs)\ !\ k \in \mathit{wffs}_{ts\ !\ k}$
and $B \in \mathit{wffs}_{foldr\ (\rightarrow)\ ts\ \gamma}$
using *wffs-from-generalized-app* **by** *force*
have $\beta = foldr\ (\rightarrow)\ ts\ \gamma$
by (*fact wff-has-unique-type*[*OF assms*(*2*) ‹$B \in \mathit{wffs}_{foldr\ (\rightarrow)\ ts\ \gamma}$›])
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** $(map\ FVar\ vs)\ !\ k \in \mathit{wffs}_{ts\ !\ k}$
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 $(u, \gamma) \notin$ *free-vars B*
shows $\vdash \exists\, u_\gamma.\ \forall^{\mathcal{Q}}_\star$ *vs* (($\bullet^{\mathcal{Q}}_\star$ 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** $\{(u, \beta) \rightarrowtail B\}$ $(u_\beta =_\beta B)$
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 = $\{(u, \gamma) \rightarrowtail \lambda^{\mathcal{Q}}_\star$ *vs B*$\}$
from *assms*(*2*) **have** $*$: $(u, \gamma) \neq v$ **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** $\vdash \bullet^{\mathcal{Q}}_\star$ ($\lambda^{\mathcal{Q}}_\star$ *vs B*) (*map FVar vs*) $=_\beta B$
by (*fact prop-5208*)
then have $\vdash \forall^{\mathcal{Q}}_\star$ *vs* ($\bullet^{\mathcal{Q}}_\star$ ($\lambda^{\mathcal{Q}}_\star$ *vs B*) (*map FVar vs*) $=_\beta$ *B*)
using *generalized-Gen* **by** *simp*
moreover
have S *?*ϑ ($\forall^{\mathcal{Q}}_\star$ *vs* (($\bullet^{\mathcal{Q}}_\star$ u_γ (*map FVar vs*)) $=_\beta$ *B*)) = $\forall^{\mathcal{Q}}_\star$ *vs* ($\bullet^{\mathcal{Q}}_\star$ ($\lambda^{\mathcal{Q}}_\star$ *vs B*) (*map FVar vs*) $=_\beta$ *B*)
proof −
from $*$ **have** $**$: *map* (λA. **S** $\{(u, \gamma) \rightarrowtail B\}$ *A*) (*map FVar vs*) = *map FVar vs* **for** *B*
by (*induction vs*) *fastforce+*
from $*$ **have**
S *?*ϑ ($\forall^{\mathcal{Q}}_\star$ *vs* (($\bullet^{\mathcal{Q}}_\star$ u_γ (*map FVar vs*)) $=_\beta$ *B*)) = $\forall^{\mathcal{Q}}_\star$ *vs* (**S** *?*ϑ (($\bullet^{\mathcal{Q}}_\star$ u_γ (*map FVar vs*)) $=_\beta$

B))
    using *generalized-forall-substitution* **by** *force*
  also have ... $= \forall^{\mathcal{Q}}{}_{\star}\ vs\ ((\mathbf{S}\ ?\vartheta\ (\bullet^{\mathcal{Q}}{}_{\star}\ u_\gamma\ (map\ FVar\ vs))) =_\beta \mathbf{S}\ \{(u, \gamma) \rightarrowtail \lambda^{\mathcal{Q}}{}_{\star}\ vs\ B\}\ B)$
    by *simp*
  also from *assms(3)* **have** ... $= \forall^{\mathcal{Q}}{}_{\star}\ vs\ ((\mathbf{S}\ ?\vartheta\ (\bullet^{\mathcal{Q}}{}_{\star}\ u_\gamma\ (map\ FVar\ vs))) =_\beta B)$
    using *free-var-singleton-substitution-neutrality* **by** *simp*
  also have ... $= \forall^{\mathcal{Q}}{}_{\star}\ vs\ (\bullet^{\mathcal{Q}}{}_{\star}\ \mathbf{S}\ ?\vartheta\ (u_\gamma)\ (map\ (\lambda A.\ \mathbf{S}\ ?\vartheta\ A)\ (map\ FVar\ vs)) =_\beta B)$
    using *generalized-app-substitution* **by** *simp*
  also have ... $= \forall^{\mathcal{Q}}{}_{\star}\ vs\ (\bullet^{\mathcal{Q}}{}_{\star}\ (\lambda^{\mathcal{Q}}{}_{\star}\ vs\ B)\ (map\ (\lambda A.\ \mathbf{S}\ ?\vartheta\ A)\ (map\ FVar\ vs)) =_\beta B)$
    by *simp*
  also from $**$ **have** ... $= \forall^{\mathcal{Q}}{}_{\star}\ vs\ (\bullet^{\mathcal{Q}}{}_{\star}\ (\lambda^{\mathcal{Q}}{}_{\star}\ vs\ B)\ (map\ FVar\ vs) =_\beta B)$
    by *presburger*
  finally show *?thesis* **.**
 qed
 ultimately have $\vdash \mathbf{S}\ ?\vartheta\ (\forall^{\mathcal{Q}}{}_{\star}\ vs\ (\bullet^{\mathcal{Q}}{}_{\star}\ u_\gamma\ (map\ FVar\ vs) =_\beta B))$
  by *simp*
 moreover from *assms(3)* **have** *is-free-for* $(\lambda^{\mathcal{Q}}{}_{\star}\ vs\ B)\ (u, \gamma)\ (\forall^{\mathcal{Q}}{}_{\star}\ vs\ (\bullet^{\mathcal{Q}}{}_{\star}\ u_\gamma\ (map\ FVar\ vs) =_\beta$
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 $\lambda^{\mathcal{Q}}{}_{\star}\ vs\ B \in wffs_\gamma$ **and** $\forall^{\mathcal{Q}}{}_{\star}\ vs\ (\bullet^{\mathcal{Q}}{}_{\star}\ u_\gamma\ (map\ FVar\ vs) =_\beta B) \in wffs_o$
 proof –
  have $FVar\ (vs\ !\ k) \in wffs_{var\text{-}type\ (vs\ !\ k)}$ **if** $k < length\ vs$ **for** k
   using *that* **and** *surj-pair*[*of vs ! k*] **by** *fastforce*
  with *assms(2)* **have** $\bullet^{\mathcal{Q}}{}_{\star}\ u_\gamma\ (map\ FVar\ vs) \in wffs_\beta$
   using *generalized-app-wff*[**where** *ts = map var-type vs*] **by** *force*
  with *assms(1)* **show** $\forall^{\mathcal{Q}}{}_{\star}\ vs\ (\bullet^{\mathcal{Q}}{}_{\star}\ u_\gamma\ (map\ FVar\ vs) =_\beta B) \in wffs_o$
   by (*auto simp only*:)
 qed (*use assms(1,2)* **in** *blast*)
 ultimately show *?thesis*
  using $\exists$ *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 $A \in wffs_o$ **and** $B \in wffs_o$
 and $(x, \alpha) \notin free\text{-}vars\ A$
 shows $\vdash \forall x_\alpha.\ (B \supset^{\mathcal{Q}} A) \supset^{\mathcal{Q}} (\exists x_\alpha.\ B \supset^{\mathcal{Q}} A)$
proof –
 have $B \supset^{\mathcal{Q}} A \in wffs_o$
  using *assms* **by** *blast*
 moreover have *is-free-for* $(x_\alpha)\ (x, \alpha)\ (B \supset^{\mathcal{Q}} A)$

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

qed (*use* §*4* ‹*is-hyps* $\{\forall x_\alpha.\ (B \supset^{\mathcal{Q}} A)\}$› *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 $(x, \alpha) \notin$ *free-vars* $(\mathcal{H} \cup \{A\})$
shows $\mathcal{H}, \exists x_\alpha.\ B \vdash A$
proof −
from *assms*(*1*) **have** *is-hyps* $\mathcal{H}$
using *hyp-derivability-implies-hyp-proof-existence* **by** *force*
then have $\mathcal{H} \vdash B \supset^{\mathcal{Q}} A$
using *assms*(*1*) **and** *Deduction-Theorem* **by** *simp*
then have $\mathcal{H} \vdash \forall x_\alpha.\ (B \supset^{\mathcal{Q}} A)$
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* ‹$\mathcal{H} \vdash B \supset^{\mathcal{Q}} A$›, *THEN wffs-from-imp-op*(*1*)]
)
with *assms*(*2*) **and** ‹*is-hyps* $\mathcal{H}$› **have** $\mathcal{H} \vdash \forall x_\alpha.\ (B \supset^{\mathcal{Q}} A) \supset^{\mathcal{Q}} (\exists x_\alpha.\ B \supset^{\mathcal{Q}} A)$
using *prop-5244-aux*[*THEN derivability-implies-hyp-derivability*] **by** *simp*
ultimately have $\mathcal{H} \vdash \exists x_\alpha.\ B \supset^{\mathcal{Q}} A$
by (*rule MP*)
then have $\mathcal{H}, \exists x_\alpha.\ B \vdash \exists x_\alpha.\ B \supset^{\mathcal{Q}} A$
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 $\mathcal{H}, \exists x_\alpha.\ B \vdash \exists x_\alpha.\ B$
using *dv-hyp* **by** *simp*
ultimately show *?thesis*
using *MP* **by** *blast*
qed

lemmas $\exists$*-Rule* = *prop-5244*

6.45 Proposition 5245 (Rule C)

lemma *prop-5245-aux*:
assumes $x \neq y$
and $(y, \alpha) \notin$ *free-vars* $(\exists x_\alpha.\ B)$
and *is-free-for* (y_α) (x, α) B
shows *is-free-for* (x_α) (y, α) $\mathbf{S}\ \{(x, \alpha) \rightarrowtail y_\alpha\}\ B$
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* B_1 B_2)
from *FApp.prems*(*1*) **have** $(y, \alpha) \notin$ *free-vars* $(\exists x_\alpha.\ B_1)$ **and** $(y, \alpha) \notin$ *free-vars* $(\exists x_\alpha.\ B_2)$
by *force+*
moreover from *FApp.prems*(*2*) **have** *is-free-for* (y_α) (x, α) B_1 **and** *is-free-for* (y_α) (x, α) B_2
using *is-free-for-from-app* **by** *iprover+*
ultimately have *is-free-for* (x_α) (y, α) $\mathbf{S}$ $\{(x, \alpha) \rightarrowtail y_\alpha\}$ B_1
and *is-free-for* (x_α) (y, α) $\mathbf{S}$ $\{(x, \alpha) \rightarrowtail y_\alpha\}$ B_2
using *FApp.IH* **by** *simp-all*
then have *is-free-for* (x_α) (y, α) $((\mathbf{S}\ \{(x, \alpha) \rightarrowtail y_\alpha\}\ B_1) \cdot (\mathbf{S}\ \{(x, \alpha) \rightarrowtail y_\alpha\}\ B_2))$
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** $v = (z, \beta)$
by *fastforce*
then show *?case*
proof (*cases* $v = (x, \alpha)$)
case *True*
with *FAbs.prems*(*1*) **have** $(y, \alpha) \notin$ *free-vars* $(\exists x_\alpha.\ B')$
by *simp*
moreover from *assms*(*1*) **have** $(y, \alpha) \neq (x, \alpha)$
by *blast*
ultimately have $(y, \alpha) \notin$ *free-vars* B'
using *FAbs.prems*(*1*) **by** *simp*
with ‹$(y, \alpha) \neq (x, \alpha)$› **have** $(y, \alpha) \notin$ *free-vars* $(\lambda x_\alpha.\ B')$
by *simp*
then have *is-free-for* (x_α) (y, α) $(\lambda x_\alpha.\ B')$
unfolding *is-free-for-def* **using** *is-free-at-in-free-vars* **by** *blast*
then have *is-free-for* (x_α) (y, α) $\mathbf{S}$ $\{(x, \alpha) \rightarrowtail y_\alpha\}$ $(\lambda x_\alpha.\ B')$
using *singleton-substitution-simps*(*4*) **by** *presburger*
then show *?thesis*
unfolding *True* **.**
next
case *False*
from *assms*(*1*) **have** $(y, \alpha) \neq (x, \alpha)$
by *blast*
with *FAbs.prems*(*1*) **have** $*$: $(y, \alpha) \notin$ *free-vars* $(\exists x_\alpha.\ (\lambda z_\beta.\ B'))$
using ‹$v = (z, \beta)$› **by** *fastforce*
then show *?thesis*
proof (*cases* $(y, \alpha) \neq v$)
case *True*
from *True*[*unfolded* ‹$v = (z, \beta)$›] **and** $*$ **have** $(y, \alpha) \notin$ *free-vars* $(\exists x_\alpha.\ B')$

by *simp*
moreover from *False*[*unfolded* ‹$v = (z, \beta)$›] **have** *is-free-for* (y_α) (x, α) B'
using *is-free-for-from-abs*[*OF FAbs.prems*(*2*)[*unfolded* ‹$v = (z, \beta)$›]] **by** *blast*
ultimately have *is-free-for* (x_α) (y, α) (**S** $\{(x, \alpha) \rightarrowtail y_\alpha\}$ B')
by (*fact FAbs.IH*)
then have *is-free-for* (x_α) (y, α) $(\lambda z_\beta.$ (**S** $\{(x, \alpha) \rightarrowtail y_\alpha\}$ B'))
using *False*[*unfolded* ‹$v = (z, \beta)$›] **by** (*intro is-free-for-to-abs*, *fastforce*+)
then show *?thesis*
unfolding *singleton-substitution-simps*(*4*) **and** ‹$v = (z, \beta)$› **using** ‹$(z, \beta) \neq (x, \alpha)$› **by** *auto*
next
case *False*
then have $v = (y, \alpha)$
by *simp*
have *is-free-for* (x_α) (y, α) $(\lambda y_\alpha.$ **S** $\{(x, \alpha) \rightarrowtail y_\alpha\}$ B')
proof−
have $(y, \alpha) \notin$ *free-vars* $(\lambda y_\alpha.$ **S** $\{(x, \alpha) \rightarrowtail y_\alpha\}$ B')
by *simp*
then show *?thesis*
using *is-free-at-in-free-vars* **by** *blast*
qed
with‹$v = (y, \alpha)$› **and** ‹$(y, \alpha) \neq (x, \alpha)$› **show** *?thesis*
using *singleton-substitution-simps*(*4*) **by** *presburger*
qed
qed
qed

proposition *prop-5245*:
assumes $\mathcal{H} \vdash \exists x_\alpha.\ B$
and $\mathcal{H}$, **S** $\{(x, \alpha) \rightarrowtail y_\alpha\}$ $B \vdash A$
and *is-free-for* (y_α) (x, α) B
and $(y, \alpha) \notin$ *free-vars* $(\mathcal{H} \cup \{\exists x_\alpha.\ B,\ A\})$
shows $\mathcal{H} \vdash A$
proof −
from *assms*(*1*) **have** *is-hyps* $\mathcal{H}$
by (*blast elim*: *is-derivable-from-hyps.cases*)
from *assms*(*2*,*4*) **have** $\mathcal{H}$, $\exists y_\alpha.$ **S** $\{(x, \alpha) \rightarrowtail y_\alpha\}$ $B \vdash A$
using $\exists$*-Rule* **by** *simp*
then have $*$: $\mathcal{H} \vdash (\exists y_\alpha.$ **S** $\{(x, \alpha) \rightarrowtail y_\alpha\}$ $B) \supset^{\mathcal{Q}} A$ (**is** ‹- $\vdash$ *?F*›)
using *Deduction-Theorem* **and** ‹*is-hyps* $\mathcal{H}$› **by** *blast*
then have $\mathcal{H} \vdash \exists x_\alpha.\ B \supset^{\mathcal{Q}} A$
proof (*cases* $x = y$)
case *True*
with $*$ **show** *?thesis*
using *identity-singleton-substitution-neutrality* **by** *force*
next
case *False*
from *assms*(*4*) **have** $(y, \alpha) \notin$ *free-vars* $(\exists x_\alpha.\ B)$
using *free-vars-in-all-vars* **by** *auto*
have $\sim^{\mathcal{Q}}$ **S** $\{(x, \alpha) \rightarrowtail y_\alpha\}$ $B \in wffs_o$

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* $(\sim^{\mathcal{Q}}$ **S** $\{(x, \alpha) \rightarrowtail y_\alpha\}$ $B)$
using *free-var-in-renaming-substitution* **by** *simp*
moreover have *is-free-for* (x_α) (y, α) $(\sim^{\mathcal{Q}}$ **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 $\vdash (\lambda y_\alpha.\ \sim^{\mathcal{Q}}$ **S** $\{(x, \alpha) \rightarrowtail y_\alpha\}$ $B) =_{\alpha \to o} (\lambda x_\alpha.\ \sim^{\mathcal{Q}} B)$
using α[**where** $A = \sim^{\mathcal{Q}}$ **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** → *60*)
notation *funcset* (**infixr** ⇸ *60*)

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

notation *app* (**infixl** · *300*)

syntax
-vlambda :: *pttrn* ⇒ *V* ⇒ (*V* ⇒ *V*) ⇒ *V* ((*3***λ-:- .**/ -) [*0*, *0*, *3*] *3*)
translations
λ*x* **:** *A***.** *f* ⇌ *CONST VLambda A* (λ*x*. *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* ⇒ *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* ⇒ *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* ⇒ *V*) ⇒ *bool* **where**
[*iff*]: *is-assignment* $\varphi \longleftrightarrow (\forall x\ \alpha.\ \varphi\ (x, \alpha) \in$ *elts* $(\mathcal{D}\ \alpha))$

end

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

abbreviation *identity-relation-in* ($q_{\_}{}^{\text{-}}$ [*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* ⇒ *V*) ⇒ *var* ⇒ (*var* ⇒ *V*) ⇒ *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* ⇒ *V*
assumes *Q-denotation*: $\forall \alpha.\ \mathcal{J}$ (*Q-constant-of-type* α) $= q_\alpha$
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
$(\forall x\ \alpha.\ \mathcal{V}\ \varphi\ (x_\alpha) = \varphi\ (x, \alpha)) \wedge$
$(\forall c\ \alpha.\ \mathcal{V}\ \varphi\ (\{\!|c|\!\}_\alpha) = \mathcal{J}\ (c, \alpha)) \wedge$
$(\forall A\ B\ \alpha\ \beta.\ A \in wffs_{\beta \to \alpha} \wedge B \in wffs_\beta \longrightarrow \mathcal{V}\ \varphi\ (A \bullet B) = (\mathcal{V}\ \varphi\ A) \cdot (\mathcal{V}\ \varphi\ B)) \wedge$
$(\forall x\ B\ \alpha\ \beta.\ B \in wffs_\beta \longrightarrow \mathcal{V}\ \varphi\ (\lambda x_\alpha.\ B) = (\boldsymbol{\lambda} z : \mathcal{D}\ \alpha.\ \mathcal{V}\ (\varphi((x, \alpha) := z))\ B))$
)

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, α)
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 $\mathcal{V}\ \varphi\ (A \bullet B) = \mathcal{V}\ \varphi\ A \cdot \mathcal{V}\ \varphi\ B$

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 $\mathcal{V}\ \varphi\ (\lambda x_{\alpha}.\ B) = (\boldsymbol{\lambda} z\ \mathbf{:}\ \mathcal{D}\ \alpha\mathbf{.}\ \mathcal{V}\ (\varphi((x, \alpha) := z))\ B)$
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 $\mathcal{V}\ \varphi\ (\lambda x_{\alpha}.\ A) = (\boldsymbol{\lambda} z\ \mathbf{:}\ \mathcal{D}\ \alpha\mathbf{.}\ \mathcal{V}\ (\varphi((x, \alpha) := z))\ A)$
by (*fact wff-abs-denotation*[*OF assms*(*1*) *abs-is-wff.prems abs-is-wff.hyps*])
also have $\ldots = (\boldsymbol{\lambda} z\ \mathbf{:}\ \mathcal{D}\ \alpha\mathbf{.}\ \mathcal{V}'\ (\varphi((x, \alpha) := z))\ A)$
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 $\mathcal{V}\ \varphi\ (\lambda x_{\alpha}.\ B) \cdot y = \mathcal{V}\ (\varphi((x, \alpha) := y))\ B$
  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 $\mathcal{V}\ \varphi\ (\wedge_{o \to o \to o}) \in$ *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 $\{x, y\} \subseteq$ *elts* ($\mathcal{D}$ o)
  shows $\mathcal{V}\ \varphi\ (\wedge_{o \to o \to o}) \cdot x \cdot y \in$ *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 $\mathcal{V}\ \varphi\ (\supset_{o \to o \to o}) \in$ *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 $\{x, y\} \subseteq$ *elts* ($\mathcal{D}$ o)
  shows $\mathcal{V}\ \varphi\ (\supset_{o \to o \to o}) \cdot x \cdot y \in$ *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*: $\forall\, \alpha\ \beta.\ \mathcal{D}\ (\alpha{\to}\beta) = \mathcal{D}\ \alpha \longmapsto \mathcal{D}\ \beta$

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

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

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

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* $A \land \mathcal{M} \models_{undefined} A$ — assignments are not meaningful

abbreviation *is-false-sentence-in* **where**
 is-false-sentence-in $A\ \mathcal{M} \equiv$ *is-sentence* $A \land \neg\ \mathcal{M} \models_{undefined} A$ — 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 $A \in wffs_\alpha$
  and $\varphi \leadsto \mathcal{D}$ **and** $\psi \leadsto \mathcal{D}$
  and $\forall\, v \in$ *free-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* α *x*)
    have $(x, \alpha) \in$ *free-vars* (x_α)
      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 ‹$(x, \alpha) \in$ *free-vars* (x_α)› **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* α *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* $(A \boldsymbol{\cdot} B) =$ *free-vars* $A\ \cup$ *free-vars* B
      by *simp*
    with *app-is-wff*.*prems*(*3*)
    have $\forall\, v \in$ *free-vars* $A.\ \varphi\ v = \psi\ v$ **and** $\forall\, v \in$ *free-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** $\mathcal{V}\ \varphi\ (A \bullet B) = \mathcal{V}\ \varphi\ A \cdot \mathcal{V}\ \varphi\ B$
using *$\mathcal{V}$-is-wff-denotation-function* **by** *fastforce*
also from ‹$\mathcal{V}\ \varphi\ A = \mathcal{V}\ \psi\ A$› **and** ‹$\mathcal{V}\ \varphi\ B = \mathcal{V}\ \psi\ B$› **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** $\ldots = \mathcal{V}\ \psi\ (A \bullet B)$
using *$\mathcal{V}$-is-wff-denotation-function* **by** *fastforce*
finally show *?case* **.**
next
case (*abs-is-wff* β A α 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 α
and $\varphi \leadsto \mathcal{D}$ **and** $\psi \leadsto \mathcal{D}$
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 $\varphi \leadsto \mathcal{D}$
and $A \in wffs_\alpha$
and $B \in wffs_\beta$
shows $\mathcal{V}\ \varphi\ ((\lambda x_\alpha.\ B) \bullet A) = \mathcal{V}\ (\varphi((x, \alpha) := \mathcal{V}\ \varphi\ A))\ B$
proof −
from *assms(2,3)* **have** $\lambda x_\alpha.\ B \in wffs_{\alpha \to \beta}$
by *blast*

with *assms*(*1*,*2*) **have** $\mathcal{V}\ \varphi\ ((\lambda x_\alpha.\ B) \bullet A) = \mathcal{V}\ \varphi\ (\lambda x_\alpha.\ B) \cdot \mathcal{V}\ \varphi\ A$
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 $\mathcal{V}\ \varphi\ (A =_\alpha B) = \mathcal{V}\ \varphi\ (Q_\alpha \bullet A \bullet B)$
by *simp*
also from *assms* **have** $\ldots = \mathcal{V}\ \varphi\ (Q_\alpha \bullet A) \cdot \mathcal{V}\ \varphi\ B$
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 ‹$\{\mathcal{V}\ \varphi\ A, \mathcal{V}\ \varphi\ B\} \subseteq elts\ (\mathcal{D}\ \alpha)$› **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 $\mathcal{V}\ \varphi\ (\lambda\mathfrak{x}_o.\ T_o) = (\boldsymbol{\lambda} z \mathbin{\mathbf{:}} \mathcal{D}\ o\mathbf{.}\ \mathbf{T})$
    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** $\mathcal{V}\ \varphi\ (\lambda\mathfrak{x}_o.\ T_o) = (\boldsymbol{\lambda} z \mathbin{\mathbf{:}} \mathcal{D}\ o\mathbf{.}\ \mathcal{V}\ (\varphi((\mathfrak{x},\ o) := z))\ T_o)$
        using *wff-abs-denotation*[*OF* $\mathcal{V}$*-is-wff-denotation-function*] **by** *blast*
      also from *assms* **and** *T-denotation* **have** $\ldots = (\boldsymbol{\lambda} z \mathbin{\mathbf{:}} \mathcal{D}\ o\mathbf{.}\ \mathbf{T})$
        using *vlambda-extensionality* **by** *fastforce*
      finally show *?thesis* **.**
    qed
    moreover have $\mathcal{V}\ \varphi\ (\lambda\mathfrak{x}_o.\ \mathfrak{x}_o) = (\boldsymbol{\lambda} z \mathbin{\mathbf{:}} \mathcal{D}\ o\mathbf{.}\ z)$
    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** $\mathcal{V}\ \varphi\ (\lambda\mathfrak{x}_o.\ \mathfrak{x}_o) = (\boldsymbol{\lambda} z \mathbin{\mathbf{:}} \mathcal{D}\ o\mathbf{.}\ \mathcal{V}\ (\varphi((\mathfrak{x},\ o) := z))\ (\mathfrak{x}_o))$
        using *wff-abs-denotation*[*OF* $\mathcal{V}$*-is-wff-denotation-function*] **by** *blast*
      also from $\mathfrak{x}$*-denotation* **have** $\ldots = (\boldsymbol{\lambda} z \mathbin{\mathbf{:}} (\mathcal{D}\ o)\mathbf{.}\ z)$
        using *vlambda-extensionality* **by** *fastforce*
      finally show *?thesis* **.**
    qed
    moreover have $(\boldsymbol{\lambda} z \mathbin{\mathbf{:}} \mathcal{D}\ o\mathbf{.}\ \mathbf{T}) \neq (\boldsymbol{\lambda} z \mathbin{\mathbf{:}} \mathcal{D}\ o\mathbf{.}\ z)$
    proof −
      from *assms*(*1*) **have** $(\boldsymbol{\lambda} z \mathbin{\mathbf{:}} \mathcal{D}\ o\mathbf{.}\ \mathbf{T}) \cdot \mathbf{F} = \mathbf{T}$
        by (*simp add*: *truth-values-domain-def*)
      moreover from *assms*(*1*) **have** $(\boldsymbol{\lambda} z \mathbin{\mathbf{:}} \mathcal{D}\ o\mathbf{.}\ z) \cdot \mathbf{F} = \mathbf{F}$
        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 $\{x,\ y\} \subseteq$ *elts* $(\mathcal{D}\ o)$
shows $\mathcal{V}\ \varphi\ (\wedge_{o\to o\to o}) \cdot x \cdot y = (\textit{if } x = \mathbf{T} \wedge y = \mathbf{T} \textit{ then } \mathbf{T} \textit{ else } \mathbf{F})$
proof −
let $?B_{leq} = \lambda\mathfrak{g}_{o\to o\to o}.\ \mathfrak{g}_{o\to o\to o} \bullet T_o \bullet T_o$
let $?B_{req} = \lambda\mathfrak{g}_{o\to o\to o}.\ \mathfrak{g}_{o\to o\to o} \bullet \mathfrak{x}_o \bullet \mathfrak{y}_o$
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 $\mathfrak{g}_{o\to o\to o} \bullet T_o \in wffs_{o\to o}$
by *blast*
have $\mathfrak{g}_{o\to o\to o} \bullet T_o \bullet T_o \in wffs_o$ **and** $\mathfrak{g}_{o\to o\to o} \bullet \mathfrak{x}_o \bullet \mathfrak{y}_o \in wffs_o$
by *blast+*
have $?B_{leq} \in wffs_{(o\to o\to o)\to o}$ **and** $?B_{req} \in wffs_{(o\to o\to o)\to o}$
by *blast+*
then have $?B_{eq} \in wffs_o$ **and** $?B_{\mathfrak{y}} \in wffs_{o\to o}$ **and** $?B_{\mathfrak{x}} \in wffs_{o\to o\to o}$
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** ‹$?B_{\mathfrak{y}} \in wffs_{o\to o}$› **have** $\ldots = \mathcal{V}\ (\varphi((\mathfrak{x},\ o) := x))\ ?B_{\mathfrak{y}} \cdot y$
using *mixed-beta-conversion* **by** *simp*
also from *assms* **and** ‹$?B_{eq} \in wffs_o$› **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** ‹$?B_{leq} \in wffs_{(o\to o\to o)\to o}$› **and** ‹$?B_{req} \in wffs_{(o\to o\to o)\to o}$› **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** $g \in$ *elts* $(\mathcal{D}\ (o\to o\to o))$ **for** g
using *that* **by** *auto*
have *side-eq-denotation*:
$\mathcal{V}\ ?\varphi'\ (\lambda\mathfrak{g}_{o\to o\to o}.\ \mathfrak{g}_{o\to o\to o} \bullet A \bullet B) = (\boldsymbol{\lambda} g : \mathcal{D}\ (o\to o\to o).\ g \cdot \mathcal{V}\ (?\varphi''\ g)\ A \cdot \mathcal{V}\ (?\varphi''\ g)\ B)$
if $A \in wffs_o$ **and** $B \in wffs_o$ **for** A **and** B
proof −
from *that* **have** $\mathfrak{g}_{o\to o\to o} \bullet A \bullet B \in wffs_o$
by *blast*
have $\mathcal{V}\ (?\varphi''\ g)\ (\mathfrak{g}_{o\to o\to o} \bullet A \bullet B) = g \cdot \mathcal{V}\ (?\varphi''\ g)\ A \cdot \mathcal{V}\ (?\varphi''\ g)\ B$
if $g \in$ *elts* $(\mathcal{D}\ (o\to o\to o))$ **for** g
proof −
from ‹$A \in wffs_o$› **have** $\mathfrak{g}_{o\to o\to o} \bullet A \in wffs_{o\to o}$

by *blast*
with *that* **and** *is-assg-*φ'' **and** ‹$B \in wffs_o$› **have**
$\mathcal{V}\ (?\varphi''\ g)\ (\mathfrak{g}_{o \to o \to o} \bullet A \bullet B) = \mathcal{V}\ (?\varphi''\ g)\ (\mathfrak{g}_{o \to o \to o} \bullet A) \cdot \mathcal{V}\ (?\varphi''\ g)\ B$
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* $?\varphi'$
by *auto*
with ‹$\mathfrak{g}_{o \to o \to o} \bullet A \bullet B \in wffs_o$› **have**
$\mathcal{V}\ ?\varphi'\ (\lambda \mathfrak{g}_{o \to o \to o}.\ \mathfrak{g}_{o \to o \to o} \bullet A \bullet B) = (\boldsymbol{\lambda} g : \mathcal{D}\ (o \to o \to o).\ \mathcal{V}\ (?\varphi''\ g)\ (\mathfrak{g}_{o \to o \to o} \bullet A \bullet B))$
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*: $\varphi((\mathfrak{x}, o) := \mathbf{T}) \leadsto \mathcal{D}$
by (*simp add*: *truth-values-domain-def*)
then have *is-assg-2*: $\varphi((\mathfrak{x}, o) := \mathbf{T}, (\mathfrak{y}, o) := \mathbf{T}) \leadsto \mathcal{D}$

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* ($\mathcal{D}$ ($o \rightarrow o \rightarrow o$)) **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*$_{o \rightarrow o}$
by *blast*
then have *is-closed-wff-of-type ?g-witness* ($o \rightarrow o \rightarrow o$)
by *force*
moreover from *assms* **have** *is-assg-*φ': *?*$\varphi' \leadsto \mathcal{D}$
by *simp*
ultimately have $\mathcal{V}$ φ *?g-witness* $\cdot\ \mathbf{T} \cdot \mathbf{T} = \mathcal{V}$ *?*φ' *?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*$_{o \rightarrow o}$› **have**
$\mathcal{V}$ *?*φ' *?g-witness* $\cdot\ \mathbf{T} \cdot \mathbf{T} = \mathcal{V}$ (*?*φ'(($\mathfrak{x}$, 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*$_{o \rightarrow o}$› **and** *is-assg-1* **and** *calculation* **have**
$\ldots = \mathcal{V}$ (*?*φ'(($\mathfrak{x}$, 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 $\mathcal{V}$ φ *?g-witness* $\cdot\ \mathbf{T} \cdot \mathbf{T} = \mathbf{T}$ **.**
with *b* **have** $\mathcal{V}$ φ *?g-witness* $\cdot\ \mathbf{T} \cdot \mathbf{T} \neq x$
by *blast*
moreover have $x = \mathcal{V}$ φ *?g-witness* $\cdot\ x \cdot y$
proof −
from *is-assg-*φ' **have** $x = \mathcal{V}$ *?*φ' ($\mathfrak{x}_o$)
using $\mathcal{V}$*-is-wff-denotation-function* **by** *auto*
also from *assms*(*2*) **and** *is-assg-*φ' **have** $\ldots = \mathcal{V}$ *?*φ' ($\lambda\mathfrak{y}_o.\ \mathfrak{x}_o$) $\cdot\ y$
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** $\ldots = \mathcal{V}$ *?*φ' *?g-witness* $\cdot\ x \cdot y$
using *is-assg-*φ' **and** ‹$\lambda\mathfrak{y}_o.\ \mathfrak{x}_o \in$ *wffs*$_{o \rightarrow o}$›
by (*simp add*: *mixed-beta-conversion fun-upd-twist*)
also from *assms*(*1*,*2*) **have** $\ldots = \mathcal{V}$ φ *?g-witness* $\cdot\ x \cdot y$
using *is-assg-*φ' **and** ‹*is-closed-wff-of-type ?g-witness* ($o \rightarrow o \rightarrow o$)›
and *closed-wff-is-meaningful-regardless-of-assignment* **by** *metis*
finally show *?thesis* **.**
qed
moreover from *assms*(*1*,*2*) **have** $\mathcal{V}$ φ *?g-witness* $\in$ *elts* ($\mathcal{D}$ ($o \rightarrow o \rightarrow o$))

using ‹*is-closed-wff-of-type ?g-witness* $(o{\to}o{\to}o)$› **and** $\mathcal{V}$*-is-wff-denotation-function* **by** *simp*
ultimately have $\exists\, g \in$ *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** $\mathcal{V}\ \varphi\ (\wedge_{o\to o\to o}) \cdot x \cdot y \in$ *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 $\lambda\mathfrak{y}_o.\ \mathfrak{y}_o \in$ *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': *?*$\varphi' \leadsto \mathcal{D}$
by *simp*
ultimately have $\mathcal{V}\ \varphi$ *?g-witness* $\cdot\ \mathbf{T} \cdot \mathbf{T} = \mathcal{V}$ *?*φ' *?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 ‹$\lambda\mathfrak{y}_o.\ \mathfrak{y}_o \in$ *wffs*$_{o\to o}$› **and** *mixed-beta-conversion* **and** *truth-values-domain-def* **by** *auto*
also from *assms*(*1*) **and** ‹$\lambda\mathfrak{y}_o.\ \mathfrak{y}_o \in$ *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** $y = \mathcal{V}$ *?*φ' $(\lambda\mathfrak{y}_o.\ \mathfrak{y}_o) \cdot y$
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** ‹$\lambda\mathfrak{y}_o.\ \mathfrak{y}_o \in$ *wffs*$_{o\to o}$› **have** $\ldots = \mathcal{V}$ *?*φ' *?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* $\in$ *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 $\exists\, g \in$ *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** $\mathcal{V}\ \varphi\ (\wedge_{o\to o\to o}) \cdot x \cdot y \in$ *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 $\varphi \leadsto \mathcal{D}$
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** $\wedge_{o\to o\to o} \bullet A \in \mathit{wffs}_{o\to o}$
by *blast*
have $\mathcal{V}\ \varphi\ (A \wedge^{\mathcal{Q}} B) = \mathcal{V}\ \varphi\ (\wedge_{o\to o\to o} \bullet A \bullet B)$
by *simp*
also from *assms* **have** $\ldots = \mathcal{V}\ \varphi\ (\wedge_{o\to o\to o} \bullet A) \cdot \mathcal{V}\ \varphi\ B$
using *$\mathcal{V}$-is-wff-denotation-function* **and** $\langle \wedge_{o\to o\to o} \bullet A \in \mathit{wffs}_{o\to o} \rangle$ **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 $\varphi \leadsto \mathcal{D}$
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'*: $?\varphi' \leadsto \mathcal{D}$
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 $\langle\lambda\mathfrak{y}_o.\ (\mathfrak{x}_o \equiv^{\mathcal{Q}} \mathfrak{x}_o \wedge^{\mathcal{Q}} \mathfrak{y}_o) \in$ *wffs*$_{o\to o}\rangle$ **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** $\langle(\mathfrak{x}_o \equiv^{\mathcal{Q}} \mathfrak{x}_o \wedge^{\mathcal{Q}} \mathfrak{y}_o) \in$ *wffs*$_o\rangle$ **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 $\langle\mathcal{V}\ ?\varphi'\ (\mathfrak{x}_o) = x\rangle$ **and** $\langle\mathcal{V}\ ?\varphi'\ (\mathfrak{y}_o) = y\rangle$ **..**
also have $\ldots \longleftrightarrow x = ($*if* $x = \mathbf{T} \wedge y = \mathbf{T}$ *then* $\mathbf{T}$ *else* $\mathbf{F})$
using $\langle\{x,\ y\} \subseteq$ *elts* $\mathbb{B}\rangle$ **by** *auto*
also have $\ldots \longleftrightarrow \mathbf{T} = ($*if* $x = \mathbf{T} \wedge y = \mathbf{F}$ *then* $\mathbf{F}$ *else* $\mathbf{T})$
using $\langle\{x,\ y\} \subseteq$ *elts* $\mathbb{B}\rangle$ **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 $\varphi \leadsto \mathcal{D}$
and $A \in$ *wffs*$_o$ **and** $B \in$ *wffs*$_o$
shows $\mathcal{V}\ \varphi\ (A \supset^{\mathcal{Q}} B) = \mathcal{V}\ \varphi\ A \supset \mathcal{V}\ \varphi\ B$
proof $-$
from *assms* **have** $\{\mathcal{V}\ \varphi\ A,\ \mathcal{V}\ \varphi\ B\} \subseteq$ *elts* $(\mathcal{D}\ o)$
using $\mathcal{V}$*-is-wff-denotation-function* **by** *simp*
from *assms*(*2*) **have** $\supset_{o\to o\to o} \bullet A \in$ *wffs*$_{o\to o}$
by *blast*
have $\mathcal{V}\ \varphi\ (A \supset^{\mathcal{Q}} B) = \mathcal{V}\ \varphi\ (\supset_{o\to o\to o} \bullet A \bullet B)$
by *simp*
also from *assms*(*1*,*3*) **have** $\ldots = \mathcal{V}\ \varphi\ (\supset_{o\to o\to o} \bullet A) \cdot \mathcal{V}\ \varphi\ B$
using $\mathcal{V}$*-is-wff-denotation-function* **and** $\langle\supset_{o\to o\to o} \bullet A \in$ *wffs*$_{o\to o}\rangle$ **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** $\ldots = ($*if* $\mathcal{V}\ \varphi\ A = \mathbf{T} \wedge \mathcal{V}\ \varphi\ B = \mathbf{F}$ *then* $\mathbf{F}$ *else* $\mathbf{T})$
using $\langle\{\mathcal{V}\ \varphi\ A,\ \mathcal{V}\ \varphi\ B\} \subseteq$ *elts* $(\mathcal{D}\ o)\rangle$ **and** *prop-5401-f* **by** *simp*
also have $\ldots = \mathcal{V}\ \varphi\ A \supset \mathcal{V}\ \varphi\ B$
using *truth-values-domain-def* **and** $\langle\{\mathcal{V}\ \varphi\ A,\ \mathcal{V}\ \varphi\ B\} \subseteq$ *elts* $(\mathcal{D}\ o)\rangle$ **by** *auto*
finally show *?thesis* **.**
qed

lemma (**in** *general-model*) *forall-denotation*:
assumes $\varphi \leadsto \mathcal{D}$
and $A \in$ *wffs*$_o$
shows $\mathcal{V}\ \varphi\ (\forall\, x_\alpha.\ A) = \mathbf{T} \longleftrightarrow (\forall\, z \in$ *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** $z \in$ *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** $z \in$ *elts* $(\mathcal{D}\ \alpha)$ **for** z
using *that* **by** (*simp add*: *mixed-beta-conversion*)
from *assms*(*2*) **have** $\lambda\mathfrak{x}_\alpha.\ T_o \in$ *wffs*$_{\alpha\to o}$ **and** $\lambda x_\alpha.\ A \in$ *wffs*$_{\alpha\to o}$

by *auto*
have $\mathcal{V}\ \varphi\ (\forall x_\alpha.\ A) = \mathcal{V}\ \varphi\ (\prod_\alpha \bullet (\lambda x_\alpha.\ A))$
unfolding *forall-def* **..**
also have $\ldots = \mathcal{V}\ \varphi\ (Q_{\alpha \to o} \bullet (\lambda \mathfrak{x}_\alpha.\ T_o) \bullet (\lambda x_\alpha.\ A))$
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 $\varphi \leadsto_M \mathcal{M}$
and $A \in \mathit{wffs}_o$
shows $\mathcal{M} \models_\varphi \forall x_\alpha.\ A \longleftrightarrow (\forall \psi.\ \psi \leadsto_M \mathcal{M} \land \psi \sim_{(x,\ \alpha)} \varphi \longrightarrow \mathcal{M} \models_\psi A)$
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**
$\mathcal{M} \models_\varphi \forall x_\alpha.\ A$
$\longleftrightarrow$
$\forall x_\alpha.\ A \in \mathit{wffs}_o \land$ *is-general-model* $(\mathcal{D}, \mathcal{J}, \mathcal{V}) \land \varphi \leadsto \mathcal{D} \land \mathcal{V}\ \varphi\ (\forall x_\alpha.\ A) = \mathbf{T}$
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 $\ldots \longleftrightarrow (\forall \psi.\ \psi \leadsto \mathcal{D} \land \psi \sim_{(x,\ \alpha)} \varphi \longrightarrow \mathcal{M} \models_\psi A)$
proof
assume $*$: $\forall z \in \mathit{elts}\ (\mathcal{D}\ \alpha).\ \mathcal{V}\ (\varphi((x, \alpha) := z))\ A = \mathbf{T}$

{
fix ψ
assume $\psi \rightsquigarrow \mathcal{D}$ **and** $\psi \sim_{(x,\ \alpha)} \varphi$
have $\mathcal{V}\ \psi\ A = \mathbf{T}$
proof −
have $\exists z \in \mathit{elts}\ (\mathcal{D}\ \alpha).\ \psi = \varphi((x,\ \alpha) := z)$
proof (*rule ccontr*)
assume $\neg\ (\exists z \in \mathit{elts}\ (\mathcal{D}\ \alpha).\ \psi = \varphi((x,\ \alpha) := z))$
with ‹$\psi \sim_{(x,\ \alpha)} \varphi$› **have** $\forall z \in \mathit{elts}\ (\mathcal{D}\ \alpha).\ \psi\ (x,\ \alpha) \neq z$
by *fastforce*
then have $\psi\ (x,\ \alpha) \notin \mathit{elts}\ (\mathcal{D}\ \alpha)$
by *blast*
moreover from *assms*(*1*) **and** ‹$\mathcal{M} = (\mathcal{D},\ \mathcal{J},\ \mathcal{V})$› **and** ‹$\psi \rightsquigarrow \mathcal{D}$› **have** $\psi\ (x,\ \alpha) \in \mathit{elts}\ (\mathcal{D}\ \alpha)$
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** ‹$\mathcal{M} = (\mathcal{D},\ \mathcal{J},\ \mathcal{V})$› **have** $\mathcal{M} \models_{\psi} A$
by *simp*
}
then show $\forall \psi.\ \psi \rightsquigarrow \mathcal{D} \land \psi \sim_{(x,\ \alpha)} \varphi \longrightarrow \mathcal{M} \models_{\psi} A$
by *blast*
next
assume ∗: $\forall \psi.\ \psi \rightsquigarrow \mathcal{D} \land \psi \sim_{(x,\ \alpha)} \varphi \longrightarrow \mathcal{M} \models_{\psi} A$
show $\forall z \in \mathit{elts}\ (\mathcal{D}\ \alpha).\ \mathcal{V}\ (\varphi((x,\ \alpha) := z))\ A = \mathbf{T}$
proof
fix z
assume $z \in \mathit{elts}\ (\mathcal{D}\ \alpha)$
with *assms*(*1*,*2*) **and** ‹$\mathcal{M} = (\mathcal{D},\ \mathcal{J},\ \mathcal{V})$› **have** $\varphi((x,\ \alpha) := z) \rightsquigarrow \mathcal{D}$
using *general-model-def* **and** *premodel-def* **and** *frame.is-assignment-def* **by** *auto*
moreover have $\varphi((x,\ \alpha) := z) \sim_{(x,\ \alpha)} \varphi$
by *simp*
ultimately have $\mathcal{M} \models_{\varphi((x,\ \alpha) := z)} A$
using ∗ **by** *blast*
with *assms*(*1*) **and** ‹$\mathcal{M} = (\mathcal{D},\ \mathcal{J},\ \mathcal{V})$› **and** ‹$\varphi((x,\ \alpha) := z) \rightsquigarrow \mathcal{D}$› **show** $\mathcal{V}\ (\varphi((x,\ \alpha) := z))\ A =$
$\mathbf{T}$
by *simp*
qed
qed
finally show *?thesis*
using ‹$\mathcal{M} = (\mathcal{D},\ \mathcal{J},\ \mathcal{V})$›
by *simp*
qed

lemma (**in** *general-model*) *axiom-1-validity-aux*:

assumes $\varphi \rightsquigarrow \mathcal{D}$
shows $\mathcal{V}\ \varphi\ (\mathfrak{g}_{o\to o} \bullet T_o \wedge^{\mathcal{Q}} \mathfrak{g}_{o\to o} \bullet F_o \equiv^{\mathcal{Q}} \forall \mathfrak{x}_o.\ \mathfrak{g}_{o\to o} \bullet \mathfrak{x}_o) = \mathbf{T}$ (**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 $\mathfrak{g}_{o\to o} \bullet T_o \in \mathit{wffs}_o$ **and** $\mathfrak{g}_{o\to o} \bullet F_o \in \mathit{wffs}_o$
by *blast*+
with *assms* **have** $\mathcal{V}\ \varphi\ ?A = \mathcal{V}\ \varphi\ (\mathfrak{g}_{o\to o} \bullet T_o) \wedge \mathcal{V}\ \varphi\ (\mathfrak{g}_{o\to o} \bullet F_o)$
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* $\forall z \in$ *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 $\mathfrak{g}_{o\to o} \bullet \mathfrak{x}_o \in \mathit{wffs}_o$
by *blast*
moreover
{
fix ψ
assume $\psi \rightsquigarrow \mathcal{D}$ **and** $\psi \sim_{(\mathfrak{x},\ o)} \varphi$
with *assms* **have** $\mathcal{V}\ \psi\ (\mathfrak{g}_{o\to o} \bullet \mathfrak{x}_o) = \mathcal{V}\ \psi\ (\mathfrak{g}_{o\to o}) \cdot \mathcal{V}\ \psi\ (\mathfrak{x}_o)$
using $\mathcal{V}$*-is-wff-denotation-function* **by** *blast*
also from ‹$\psi \rightsquigarrow \mathcal{D}$› **have** $\ldots = \psi\ (\mathfrak{g},\ o\to o) \cdot \psi\ (\mathfrak{x},\ o)$
using $\mathcal{V}$*-is-wff-denotation-function* **by** *auto*
also from ‹$\psi \sim_{(\mathfrak{x},\ o)} \varphi$› **have** $\ldots = \varphi\ (\mathfrak{g},\ o\to o) \cdot \psi\ (\mathfrak{x},\ o)$
by *simp*
also from *True* **and** ‹$\psi \rightsquigarrow \mathcal{D}$› **have** $\ldots = \mathbf{T}$
by *blast*
finally have $\mathcal{V}\ \psi\ (\mathfrak{g}_{o\to o} \bullet \mathfrak{x}_o) = \mathbf{T}$ **.**
with *assms* **and** ‹$\mathfrak{g}_{o\to o} \bullet \mathfrak{x}_o \in \mathit{wffs}_o$› **have** $?\mathcal{M} \models_\psi \mathfrak{g}_{o\to o} \bullet \mathfrak{x}_o$

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* ‹$?A \equiv^{\mathcal{Q}} ?B \in wffs_{o}$›] **by** *simp*
next
case *False*
then have $\exists z \in elts\ (\mathcal{D}\ o).\ \varphi\ (\mathfrak{g},\ o{\rightarrow}o) \cdot z \neq \mathbf{T}$
by *blast*
moreover from $*$ **have** $\forall z \in elts\ (\mathcal{D}\ o).\ \varphi\ (\mathfrak{g},\ o{\rightarrow}o) \cdot z \in elts\ (\mathcal{D}\ o)$
using *app-is-domain-respecting* **by** *blast*
ultimately obtain z **where** $z \in elts\ (\mathcal{D}\ o)$ **and** $\varphi\ (\mathfrak{g},\ o{\rightarrow}o) \cdot z = \mathbf{F}$
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** $\psi \leadsto \mathcal{D}$
by *simp*
then have $\mathcal{V}\ \psi\ (\mathfrak{g}_{o\rightarrow o} \bullet \mathfrak{x}_{o}) = \mathcal{V}\ \psi\ (\mathfrak{g}_{o\rightarrow o}) \cdot \mathcal{V}\ \psi\ (\mathfrak{x}_{o})$
using $\mathcal{V}$-*is-wff-denotation-function* **by** *blast*
also from ‹$\psi \leadsto \mathcal{D}$› **have** $\ldots = \psi\ (\mathfrak{g},\ o{\rightarrow}o) \cdot \psi\ (\mathfrak{x},\ o)$
using $\mathcal{V}$-*is-wff-denotation-function* **by** *auto*
also from ψ-*def* **have** $\ldots = \varphi\ (\mathfrak{g},\ o{\rightarrow}o) \cdot z$
by *simp*
also have $\ldots = \mathbf{F}$
unfolding ‹$\varphi\ (\mathfrak{g},\ o{\rightarrow}o) \cdot z = \mathbf{F}$› **..**
finally have $\mathcal{V}\ \psi\ (\mathfrak{g}_{o\rightarrow o} \bullet \mathfrak{x}_{o}) = \mathbf{F}$ **.**
with ‹$\psi \leadsto \mathcal{D}$› **have** $\neg\ ?\mathcal{M} \models_{\psi} \mathfrak{g}_{o\rightarrow o} \bullet \mathfrak{x}_{o}$
by (*auto simp add*: *inj-eq*)
with ‹$\psi \leadsto \mathcal{D}$› **and** ψ-*def* **have** $\neg\ (\forall \psi.\ \psi \leadsto \mathcal{D} \wedge \psi \sim_{(\mathfrak{x},\ o)} \varphi \longrightarrow ?\mathcal{M} \models_{\psi} \mathfrak{g}_{o\rightarrow o} \bullet \mathfrak{x}_{o})$
using *fun-upd-other* **by** *fastforce*
with ‹$\neg\ ?\mathcal{M} \models_{\psi} \mathfrak{g}_{o\rightarrow o} \bullet \mathfrak{x}_{o}$› **have** $\neg\ ?\mathcal{M} \models_{\varphi} ?B$
using *prop-5401-g*[*OF* $*$ *wffs-from-forall*[*OF wffs-from-equivalence*(*2*)[*OF* ‹$?A \equiv^{\mathcal{Q}} ?B \in wffs_{o}$›]]]
by *blast*
then have $\mathcal{V}\ \varphi\ (\forall \mathfrak{x}_{o}.\ \mathfrak{g}_{o\rightarrow o} \bullet \mathfrak{x}_{o}) \neq \mathbf{T}$
by *simp*
moreover from *assms* **have** $\mathcal{V}\ \varphi\ ?B \in elts\ (\mathcal{D}\ o)$
using *wffs-from-equivalence*[*OF* ‹$?A \equiv^{\mathcal{Q}} ?B \in wffs_{o}$›] **and** $\mathcal{V}$-*is-wff-denotation-function* **by** *auto*
ultimately have $\mathcal{V}\ \varphi\ ?B = \mathbf{F}$
by (*simp add*: *truth-values-domain-def*)
moreover have $\mathcal{V}\ \varphi\ (\mathfrak{g}_{o\rightarrow o} \bullet T_{o} \wedge^{\mathcal{Q}} \mathfrak{g}_{o\rightarrow o} \bullet F_{o}) = \mathbf{F}$
proof −
from ‹$z \in elts\ (\mathcal{D}\ o)$› **and** ‹$\varphi\ (\mathfrak{g},\ o{\rightarrow}o) \cdot z = \mathbf{F}$›
have $((\varphi\ (\mathfrak{g},\ o{\rightarrow}o)) \cdot \mathbf{T}) = \mathbf{F} \vee ((\varphi\ (\mathfrak{g},\ o{\rightarrow}o)) \cdot \mathbf{F}) = \mathbf{F}$
using *truth-values-domain-def* **by** *fastforce*
moreover from ‹$z \in elts\ (\mathcal{D}\ o)$› **and** ‹$\varphi\ (\mathfrak{g},\ o{\rightarrow}o) \cdot z = \mathbf{F}$›
and ‹$\forall z \in elts\ (\mathcal{D}\ o).\ \varphi\ (\mathfrak{g},\ o{\rightarrow}o) \cdot z \in elts\ (\mathcal{D}\ o)$›

have $\{(\varphi\ (\mathfrak{g},\ o{\to}o)) \boldsymbol{\cdot} \mathbf{T},\ (\varphi\ (\mathfrak{g},\ o{\to}o)) \boldsymbol{\cdot} \mathbf{F}\} \subseteq$ *elts* $(\mathcal{D}\ o)$
by (*simp add*: *truth-values-domain-def*)
ultimately have $((\varphi\ (\mathfrak{g},\ o{\to}o)) \boldsymbol{\cdot} \mathbf{T}) \wedge ((\varphi\ (\mathfrak{g},\ o{\to}o)) \boldsymbol{\cdot} \mathbf{F}) = \mathbf{F}$
by *auto*
with *lhs* **show** *?thesis*
by (*simp only*:)
qed
ultimately show *?thesis*
using *assms* **and** *prop-5401-b*$'$ **and** *wffs-from-equivalence*[*OF* ‹$?A \equiv^{\mathcal{Q}} ?B \in wffs_o$›] **by** *simp*
qed
then show *?thesis* **.**
qed

lemma *axiom-1-validity*:
shows $\models \mathfrak{g}_{o\to o} \boldsymbol{\cdot} T_o \wedge^{\mathcal{Q}} \mathfrak{g}_{o\to o} \boldsymbol{\cdot} F_o \equiv^{\mathcal{Q}} \forall \mathfrak{x}_o.\ \mathfrak{g}_{o\to o} \boldsymbol{\cdot} \mathfrak{x}_o$ (**is** $\models ?A \equiv^{\mathcal{Q}} ?B$)
proof (*intro allI impI*)
fix $\mathcal{M}$ **and** φ
assume $*$: *is-general-model* $\mathcal{M}\ \varphi \rightsquigarrow_M \mathcal{M}$
show $\mathcal{M} \models_\varphi ?A \equiv^{\mathcal{Q}} ?B$
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** $\mathcal{V}\ \varphi\ (?A \equiv^{\mathcal{Q}} ?B) = \mathbf{T}$
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 $\varphi \rightsquigarrow \mathcal{D}$
shows $\mathcal{V}\ \varphi\ ((\mathfrak{x}_\alpha =_\alpha \mathfrak{y}_\alpha) \supset^{\mathcal{Q}} (\mathfrak{h}_{\alpha\to o} \boldsymbol{\cdot} \mathfrak{x}_\alpha \equiv^{\mathcal{Q}} \mathfrak{h}_{\alpha\to o} \boldsymbol{\cdot} \mathfrak{y}_\alpha)) = \mathbf{T}$ (**is** $\mathcal{V}\ \varphi\ (?A \supset^{\mathcal{Q}} ?B) = \mathbf{T}$)
proof −
have $?A \supset^{\mathcal{Q}} ?B \in wffs_o$
using *axioms.axiom-2* **and** *axioms-are-wffs-of-type-o* **by** *blast*
from ‹$?A \supset^{\mathcal{Q}} ?B \in wffs_o$› **have** $?A \in wffs_o$ **and** $?B \in wffs_o$
using *wffs-from-imp-op* **by** *blast+*
with *assms* **have** $\mathcal{V}\ \varphi\ (?A \supset^{\mathcal{Q}} ?B) = \mathcal{V}\ \varphi\ ?A \supset \mathcal{V}\ \varphi\ ?B$
using *prop-5401-f*$'$ **by** *simp*
moreover from *assms* **and** ‹$?A \in wffs_o$› **and** ‹$?B \in wffs_o$› **have** $\{\mathcal{V}\ \varphi\ ?A,\ \mathcal{V}\ \varphi\ ?B\} \subseteq$ *elts* $(\mathcal{D}\ o)$
using $\mathcal{V}$*-is-wff-denotation-function* **by** *simp*
then have $\{\mathcal{V}\ \varphi\ ?A,\ \mathcal{V}\ \varphi\ ?B\} \subseteq$ *elts* $\mathbb{B}$
by (*simp add*: *truth-values-domain-def*)
ultimately have $\mathcal{V}$*-imp-T*: $\mathcal{V}\ \varphi\ (?A \supset^{\mathcal{Q}} ?B) = \mathbf{T} \longleftrightarrow \mathcal{V}\ \varphi\ ?A = \mathbf{F} \vee \mathcal{V}\ \varphi\ ?B = \mathbf{T}$
by *fastforce*
then show *?thesis*
proof (*cases* $\varphi\ (\mathfrak{x},\ \alpha) = \varphi\ (\mathfrak{y},\ \alpha)$)
case *True*
from *assms* **and** ‹$?B \in wffs_o$› **have** $\mathcal{V}\ \varphi\ ?B = \mathbf{T} \longleftrightarrow \mathcal{V}\ \varphi\ (\mathfrak{h}_{\alpha\to o} \boldsymbol{\cdot} \mathfrak{x}_\alpha) = \mathcal{V}\ \varphi\ (\mathfrak{h}_{\alpha\to o} \boldsymbol{\cdot} \mathfrak{y}_\alpha)$

    using *wffs-from-equivalence* **and** *prop-5401-b'* **by** *metis*
  moreover have $\mathcal{V}\ \varphi\ (\mathfrak{h}_{\alpha\to o} \cdot \mathfrak{x}_\alpha) = \mathcal{V}\ \varphi\ (\mathfrak{h}_{\alpha\to o} \cdot \mathfrak{y}_\alpha)$
  proof −
    from *assms* **and** ‹$?B \in wffs_o$› **have** $\mathcal{V}\ \varphi\ (\mathfrak{h}_{\alpha\to o} \cdot \mathfrak{x}_\alpha) = \mathcal{V}\ \varphi\ (\mathfrak{h}_{\alpha\to o}) \cdot \mathcal{V}\ \varphi\ (\mathfrak{x}_\alpha)$
      using *$\mathcal{V}$-is-wff-denotation-function* **by** *blast*
    also from *assms* **have** $\ldots = \varphi\ (\mathfrak{h},\ \alpha\to o) \cdot \varphi\ (\mathfrak{x},\ \alpha)$
      using *$\mathcal{V}$-is-wff-denotation-function* **by** *auto*
    also from *True* **have** $\ldots = \varphi\ (\mathfrak{h},\ \alpha\to o) \cdot \varphi\ (\mathfrak{y},\ \alpha)$
      by (*simp only*:)
    also from *assms* **have** $\ldots = \mathcal{V}\ \varphi\ (\mathfrak{h}_{\alpha\to o}) \cdot \mathcal{V}\ \varphi\ (\mathfrak{y}_\alpha)$
      using *$\mathcal{V}$-is-wff-denotation-function* **by** *auto*
    also from *assms* **and** ‹$?B \in wffs_o$› **have** $\ldots = \mathcal{V}\ \varphi\ (\mathfrak{h}_{\alpha\to o} \cdot \mathfrak{y}_\alpha)$
      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 $\models (\mathfrak{x}_\alpha =_\alpha \mathfrak{y}_\alpha) \supset^{\mathcal{Q}} (\mathfrak{h}_{\alpha\to o} \cdot \mathfrak{x}_\alpha \equiv^{\mathcal{Q}} \mathfrak{h}_{\alpha\to o} \cdot \mathfrak{y}_\alpha)$ (**is** $\models ?A \supset^{\mathcal{Q}} ?B$)
proof (*intro allI impI*)
fix $\mathcal{M}$ **and** φ
assume $*$: *is-general-model* $\mathcal{M}\ \varphi \rightsquigarrow_M \mathcal{M}$
show $\mathcal{M} \models_\varphi ?A \supset^{\mathcal{Q}} ?B$
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** $\mathcal{V}\ \varphi\ (?A \supset^{\mathcal{Q}} ?B) = \mathbf{T}$
    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 $\mathcal{V}\ \varphi\ ((\mathfrak{f}_{\alpha\to\beta} =_{\alpha\to\beta} \mathfrak{g}_{\alpha\to\beta}) \equiv^{\mathcal{Q}} \forall \mathfrak{x}_\alpha.\ (\mathfrak{f}_{\alpha\to\beta} \cdot \mathfrak{x}_\alpha =_\beta \mathfrak{g}_{\alpha\to\beta} \cdot \mathfrak{x}_\alpha)) = \mathbf{T}$

(**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* $?\mathcal{M}\ \varphi \leadsto_M\ ?\mathcal{M}$
using *general-model-axioms* **by** *blast*+
have *B′-wffo*: $\mathfrak{f}_{\alpha\to\beta} \bullet \mathfrak{x}_\alpha =_\beta \mathfrak{g}_{\alpha\to\beta} \bullet \mathfrak{x}_\alpha \in wffs_o$
by *blast*
have $?A \equiv^{\mathcal{Q}}\ ?B \in wffs_o$ **and** $?A \in wffs_o$ **and** $?B \in wffs_o$
proof −
show $?A \equiv^{\mathcal{Q}}\ ?B \in wffs_o$
using *axioms.axiom-3* **and** *axioms-are-wffs-of-type-o*
by *blast*
then show $?A \in wffs_o$ **and** $?B \in wffs_o$
by (*blast dest*: *wffs-from-equivalence*)+
qed
have $\mathcal{V}\ \varphi\ ?A = \mathcal{V}\ \varphi\ ?B$
proof (*cases* $\varphi\ (\mathfrak{f}, \alpha\to\beta) = \varphi\ (\mathfrak{g}, \alpha\to\beta)$)
case *True*
have $\mathcal{V}\ \varphi\ ?A = \mathbf{T}$
proof −
from *assms* **have** $\mathcal{V}\ \varphi\ (\mathfrak{f}_{\alpha\to\beta}) = \varphi\ (\mathfrak{f}, \alpha\to\beta)$
using *$\mathcal{V}$-is-wff-denotation-function* **by** *auto*
also from *True* **have** $\ldots = \varphi\ (\mathfrak{g}, \alpha\to\beta)$
by (*simp only*:)
also from *assms* **have** $\ldots = \mathcal{V}\ \varphi\ (\mathfrak{g}_{\alpha\to\beta})$
using *$\mathcal{V}$-is-wff-denotation-function* **by** *auto*
finally have $\mathcal{V}\ \varphi\ (\mathfrak{f}_{\alpha\to\beta}) = \mathcal{V}\ \varphi\ (\mathfrak{g}_{\alpha\to\beta})$.
with *assms* **show** *?thesis*
using *prop-5401-b* **by** *blast*
qed
moreover have $\mathcal{V}\ \varphi\ ?B = \mathbf{T}$
proof −
{
fix ψ
assume $\psi \leadsto \mathcal{D}$ **and** $\psi \sim_{(\mathfrak{x},\ \alpha)} \varphi$
from *assms* **and** ‹$\psi \leadsto \mathcal{D}$› **have** $\mathcal{V}\ \psi\ (\mathfrak{f}_{\alpha\to\beta} \bullet \mathfrak{x}_\alpha) = \mathcal{V}\ \psi\ (\mathfrak{f}_{\alpha\to\beta}) \cdot \mathcal{V}\ \psi\ (\mathfrak{x}_\alpha)$
using *$\mathcal{V}$-is-wff-denotation-function* **by** *blast*
also from *assms* **and** ‹$\psi \leadsto \mathcal{D}$› **have** $\ldots = \psi\ (\mathfrak{f}, \alpha\to\beta) \cdot \psi\ (\mathfrak{x}, \alpha)$
using *$\mathcal{V}$-is-wff-denotation-function* **by** *auto*
also from ‹$\psi \sim_{(\mathfrak{x},\ \alpha)} \varphi$› **have** $\ldots = \varphi\ (\mathfrak{f}, \alpha\to\beta) \cdot \psi\ (\mathfrak{x}, \alpha)$
by *simp*
also from *True* **have** $\ldots = \varphi\ (\mathfrak{g}, \alpha\to\beta) \cdot \psi\ (\mathfrak{x}, \alpha)$
by (*simp only*:)
also from ‹$\psi \sim_{(\mathfrak{x},\ \alpha)} \varphi$› **have** $\ldots = \psi\ (\mathfrak{g}, \alpha\to\beta) \cdot \psi\ (\mathfrak{x}, \alpha)$
by *simp*
also from *assms* **and** ‹$\psi \leadsto \mathcal{D}$› **have** $\ldots = \mathcal{V}\ \psi\ (\mathfrak{g}_{\alpha\to\beta}) \cdot \mathcal{V}\ \psi\ (\mathfrak{x}_\alpha)$
using *$\mathcal{V}$-is-wff-denotation-function* **by** *auto*
also from *assms* **and** ‹$\psi \leadsto \mathcal{D}$› **have** $\ldots = \mathcal{V}\ \psi\ (\mathfrak{g}_{\alpha\to\beta} \bullet \mathfrak{x}_\alpha)$
using *wff-app-denotation*[*OF $\mathcal{V}$-is-wff-denotation-function*] **by** (*metis wffs-of-type-intros*(*1*))

finally have $\mathcal{V}\ \psi\ (\mathfrak{f}_{\alpha\to\beta} \bullet \mathfrak{x}_\alpha) = \mathcal{V}\ \psi\ (\mathfrak{g}_{\alpha\to\beta} \bullet \mathfrak{x}_\alpha)$ **.**
with *B′-wffo* **and** *assms* **and** ‹$\psi \rightsquigarrow \mathcal{D}$› **have** $\mathcal{V}\ \psi\ (\mathfrak{f}_{\alpha\to\beta} \bullet \mathfrak{x}_\alpha =_\beta \mathfrak{g}_{\alpha\to\beta} \bullet \mathfrak{x}_\alpha) = \mathbf{T}$
using *prop-5401-b* **and** *wffs-from-equality* **by** *blast*
with $*(2)$ **have** $?\mathcal{M} \models_\psi \mathfrak{f}_{\alpha\to\beta} \bullet \mathfrak{x}_\alpha =_\beta \mathfrak{g}_{\alpha\to\beta} \bullet \mathfrak{x}_\alpha$
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** $\varphi\ (\mathfrak{f}, \alpha\to\beta) \in$ *elts* $(\mathcal{D}\ \alpha \longmapsto \mathcal{D}\ \beta)$ **and** $\varphi\ (\mathfrak{g}, \alpha\to\beta) \in$ *elts* $(\mathcal{D}\ \alpha \longmapsto \mathcal{D}\ \beta)$
by (*simp-all add*: *function-domainD*)
with *False* **obtain** z **where** $z \in$ *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** ‹$z \in$ *elts* $(\mathcal{D}\ \alpha)$› **have** $\psi \rightsquigarrow \mathcal{D}$ **and** $\psi \sim_{(\mathfrak{x},\ \alpha)} \varphi$
unfolding *ψ-def* **by** *fastforce+*
have $\mathcal{V}\ \psi\ (f_{\alpha\to\beta} \bullet \mathfrak{x}_\alpha) = \varphi\ (f, \alpha\to\beta) \cdot z$ **for** f
proof $-$
from ‹$\psi \rightsquigarrow \mathcal{D}$› **have** $\mathcal{V}\ \psi\ (f_{\alpha\to\beta} \bullet \mathfrak{x}_\alpha) = \mathcal{V}\ \psi\ (f_{\alpha\to\beta}) \cdot \mathcal{V}\ \psi\ (\mathfrak{x}_\alpha)$
using *𝒱-is-wff-denotation-function* **by** *blast*
also from ‹$\psi \rightsquigarrow \mathcal{D}$› **have** $\ldots = \psi\ (f, \alpha\to\beta) \cdot \psi\ (\mathfrak{x}, \alpha)$
using *𝒱-is-wff-denotation-function* **by** *auto*
finally show *?thesis*
unfolding *ψ-def* **by** *simp*
qed
then have $\mathcal{V}\ \psi\ (\mathfrak{f}_{\alpha\to\beta} \bullet \mathfrak{x}_\alpha) = \varphi\ (\mathfrak{f}, \alpha\to\beta) \cdot z$ **and** $\mathcal{V}\ \psi\ (\mathfrak{g}_{\alpha\to\beta} \bullet \mathfrak{x}_\alpha) = \varphi\ (\mathfrak{g}, \alpha\to\beta) \cdot z$
by (*simp-all only*:)
with ‹$\varphi\ (\mathfrak{f}, \alpha\to\beta) \cdot z \neq \varphi\ (\mathfrak{g}, \alpha\to\beta) \cdot z$› **have** $\mathcal{V}\ \psi\ (\mathfrak{f}_{\alpha\to\beta} \bullet \mathfrak{x}_\alpha) \neq \mathcal{V}\ \psi\ (\mathfrak{g}_{\alpha\to\beta} \bullet \mathfrak{x}_\alpha)$
by *simp*
then have $\mathcal{V}\ \psi\ (\mathfrak{f}_{\alpha\to\beta} \bullet \mathfrak{x}_\alpha =_\beta \mathfrak{g}_{\alpha\to\beta} \bullet \mathfrak{x}_\alpha) = \mathbf{F}$
proof $-$
from *B′-wffo* **and** ‹$\psi \rightsquigarrow \mathcal{D}$› **and** $*$ **have** $\mathcal{V}\ \psi\ (\mathfrak{f}_{\alpha\to\beta} \bullet \mathfrak{x}_\alpha =_\beta \mathfrak{g}_{\alpha\to\beta} \bullet \mathfrak{x}_\alpha) \in$ *elts* $(\mathcal{D}\ o)$
using *𝒱-is-wff-denotation-function* **by** *auto*
moreover from *B′-wffo* **have** $\{\mathfrak{f}_{\alpha\to\beta} \bullet \mathfrak{x}_\alpha, \mathfrak{g}_{\alpha\to\beta} \bullet \mathfrak{x}_\alpha\} \subseteq wffs_\beta$
by *blast*
with ‹$\psi \rightsquigarrow \mathcal{D}$› **and** ‹$\mathcal{V}\ \psi\ (\mathfrak{f}_{\alpha\to\beta} \bullet \mathfrak{x}_\alpha) \neq \mathcal{V}\ \psi\ (\mathfrak{g}_{\alpha\to\beta} \bullet \mathfrak{x}_\alpha)$› **and** *B′-wffo*
have $\mathcal{V}\ \psi\ (\mathfrak{f}_{\alpha\to\beta} \bullet \mathfrak{x}_\alpha =_\beta \mathfrak{g}_{\alpha\to\beta} \bullet \mathfrak{x}_\alpha) \neq \mathbf{T}$
using *prop-5401-b* **by** *simp*
ultimately show *?thesis*
by (*simp add*: *truth-values-domain-def*)
qed
with ‹$\psi \rightsquigarrow \mathcal{D}$› **have** $\neg\ ?\mathcal{M} \models_\psi \mathfrak{f}_{\alpha\to\beta} \bullet \mathfrak{x}_\alpha =_\beta \mathfrak{g}_{\alpha\to\beta} \bullet \mathfrak{x}_\alpha$
by (*auto simp add*: *inj-eq*)
with ‹$\psi \rightsquigarrow \mathcal{D}$› **and** ‹$\psi \sim_{(\mathfrak{x},\ \alpha)} \varphi$›

have $\exists\,\psi.\ \psi \leadsto \mathcal{D} \wedge \psi \sim_{(\mathfrak{x},\ \alpha)} \varphi \wedge \neg\ ?\mathcal{M} \models_{\psi} \mathfrak{f}_{\alpha\to\beta} \cdot \mathfrak{x}_{\alpha} =_{\beta} \mathfrak{g}_{\alpha\to\beta} \cdot \mathfrak{x}_{\alpha}$
by *blast*
with $*$ **and** *B′-wffo* **have** $\neg\ ?\mathcal{M} \models_{\varphi} ?B$
using *prop-5401-g* **by** *blast*
then have $\mathcal{V}\ \varphi\ ?B = \mathbf{F}$
proof $-$
from ‹$?B \in wffs_{o}$› **and** $*$ **have** $\mathcal{V}\ \varphi\ ?B \in elts\ (\mathcal{D}\ o)$
using *$\mathcal{V}$-is-wff-denotation-function* **by** *auto*
with ‹$\neg\ ?\mathcal{M} \models_{\varphi} ?B$› **and** ‹$?B \in wffs_{o}$› **show** *?thesis*
using *truth-values-domain-def* **by** *fastforce*
qed
moreover have $\mathcal{V}\ \varphi\ (\mathfrak{f}_{\alpha\to\beta} =_{\alpha\to\beta} \mathfrak{g}_{\alpha\to\beta}) = \mathbf{F}$
proof $-$
from $*$ **have** $\mathcal{V}\ \varphi\ (\mathfrak{f}_{\alpha\to\beta}) = \varphi\ (\mathfrak{f},\ \alpha\to\beta)$ **and** $\mathcal{V}\ \varphi\ (\mathfrak{g}_{\alpha\to\beta}) = \varphi\ (\mathfrak{g},\ \alpha\to\beta)$
using *$\mathcal{V}$-is-wff-denotation-function* **by** *auto*
with *False* **have** $\mathcal{V}\ \varphi\ (\mathfrak{f}_{\alpha\to\beta}) \neq \mathcal{V}\ \varphi\ (\mathfrak{g}_{\alpha\to\beta})$
by *simp*
with $*$ **have** $\mathcal{V}\ \varphi\ (\mathfrak{f}_{\alpha\to\beta} =_{\alpha\to\beta} \mathfrak{g}_{\alpha\to\beta}) \neq \mathbf{T}$
using *prop-5401-b* **by** *blast*
moreover from $*$ **and** ‹$?A \in wffs_{o}$› **have** $\mathcal{V}\ \varphi\ (\mathfrak{f}_{\alpha\to\beta} =_{\alpha\to\beta} \mathfrak{g}_{\alpha\to\beta}) \in elts\ (\mathcal{D}\ o)$
using *$\mathcal{V}$-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** ‹$?A \in wffs_{o}$› **and** ‹$?B \in wffs_{o}$› **show** *?thesis*
using *prop-5401-b′* **by** *simp*
qed

lemma *axiom-3-validity*:
shows $\models (\mathfrak{f}_{\alpha\to\beta} =_{\alpha\to\beta} \mathfrak{g}_{\alpha\to\beta}) \equiv^{\mathcal{Q}} \forall \mathfrak{x}_{\alpha}.\ (\mathfrak{f}_{\alpha\to\beta} \cdot \mathfrak{x}_{\alpha} =_{\beta} \mathfrak{g}_{\alpha\to\beta} \cdot \mathfrak{x}_{\alpha})$ (**is** $\models ?A \equiv^{\mathcal{Q}} ?B$)
proof (*intro allI impI*)
fix $\mathcal{M}$ **and** φ
assume $*$: *is-general-model* $\mathcal{M}\ \varphi \leadsto_{M} \mathcal{M}$
show $\mathcal{M} \models_{\varphi} ?A \equiv^{\mathcal{Q}} ?B$
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** $\mathcal{V}\ \varphi\ (?A \equiv^{\mathcal{Q}} ?B) = \mathbf{T}$
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 $\varphi \leadsto \mathcal{D}$

and $A \in wffs_\alpha$
shows $\mathcal{V}\ \varphi\ ((\lambda x_\alpha.\ \{\!|c|\!\}_\beta) \bullet A =_\beta \{\!|c|\!\}_\beta) = \mathbf{T}$
proof –
from *assms(2)* **have** $(\lambda x_\alpha.\ \{\!|c|\!\}_\beta) \bullet A =_\beta \{\!|c|\!\}_\beta \in wffs_o$
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** $\mathcal{V}\ \varphi\ ((\lambda x_\alpha.\ \{\!|c|\!\}_\beta) \bullet A) = \mathcal{V}\ (\varphi((x, \alpha) := \mathcal{V}\ \varphi\ A))\ (\{\!|c|\!\}_\beta)$
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 *𝒱-is-wff-denotation-function* **by** *auto*
finally have $\mathcal{V}\ \varphi\ ((\lambda x_\alpha.\ \{\!|c|\!\}_\beta) \bullet A) = \mathcal{V}\ \varphi\ (\{\!|c|\!\}_\beta)$ **.**
with *assms(1)* **and** ‹$(\lambda x_\alpha.\ \{\!|c|\!\}_\beta) \bullet A =_\beta \{\!|c|\!\}_\beta \in wffs_o$› **show** *?thesis*
using *wffs-from-equality(1)* **and** *prop-5401-b* **by** *blast*
qed

lemma *axiom-4-1-con-validity*:
assumes $A \in wffs_\alpha$
shows $\models (\lambda x_\alpha.\ \{\!|c|\!\}_\beta) \bullet A =_\beta \{\!|c|\!\}_\beta$
proof (*intro allI impI*)
fix $\mathcal{M}$ **and** φ
assume $*$: *is-general-model* $\mathcal{M}\ \varphi \rightsquigarrow_M \mathcal{M}$
show $\mathcal{M} \models_\varphi (\lambda x_\alpha.\ \{\!|c|\!\}_\beta) \bullet A =_\beta \{\!|c|\!\}_\beta$
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** ‹$\mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V})$› **have** $\mathcal{V}\ \varphi\ ((\lambda x_\alpha.\ \{\!|c|\!\}_\beta) \bullet A =_\beta \{\!|c|\!\}_\beta) = \mathbf{T}$
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 $A \in wffs_\alpha$
and $(y, \beta) \neq (x, \alpha)$
shows $\mathcal{V}\ \varphi\ ((\lambda x_\alpha.\ y_\beta) \bullet A =_\beta y_\beta) = \mathbf{T}$
proof –
from *assms(2)* **have** $(\lambda x_\alpha.\ y_\beta) \bullet A =_\beta y_\beta \in wffs_o$
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** $\mathcal{V}\ \varphi\ ((\lambda x_\alpha.\ y_\beta) \bullet A) = \mathcal{V}\ (\varphi((x, \alpha) := \mathcal{V}\ \varphi\ A))\ (y_\beta)$
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** $\mathcal{V}\ \varphi\ A \in$ *elts* $(\mathcal{D}\ \alpha)$
  using $\mathcal{V}$*-is-wff-denotation-function* **by** *auto*
with ψ*-def* **and** *assms*(*1*) **have** $\psi \rightsquigarrow \mathcal{D}$
  by *simp*
moreover have *free-vars* $(y_\beta) = \{(y, \beta)\}$
  by *simp*
with ψ*-def* **and** *assms*(*3*) **have** $\forall\, v \in$ *free-vars* $(y_\beta).\ \varphi\ v = \psi\ v$
  by *auto*
ultimately show *?thesis*
  using *prop-5400*[*OF wffs-of-type-intros*(*1*) *assms*(*1*)] **by** *simp*
qed
finally have $\mathcal{V}\ \varphi\ ((\lambda x_\alpha.\ y_\beta) \bullet A) = \mathcal{V}\ \varphi\ (y_\beta)$ **.**
with ‹$(\lambda x_\alpha.\ y_\beta) \bullet A =_\beta y_\beta \in wffs_o$› **show** *?thesis*
  using *wffs-from-equality*(*1*) **and** *prop-5401-b*[*OF assms*(*1*)] **by** *blast*
qed

lemma *axiom-4-1-var-validity*:
  assumes $A \in wffs_\alpha$
  and $(y, \beta) \neq (x, \alpha)$
  shows $\models (\lambda x_\alpha.\ y_\beta) \bullet A =_\beta y_\beta$
proof (*intro allI impI*)
  fix $\mathcal{M}$ **and** φ
  assume $*$: *is-general-model* $\mathcal{M}\ \varphi \rightsquigarrow_M \mathcal{M}$
  show $\mathcal{M} \models_\varphi (\lambda x_\alpha.\ y_\beta) \bullet A =_\beta y_\beta$
  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** ‹$\mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V})$› **have** $\mathcal{V}\ \varphi\ ((\lambda x_\alpha.\ y_\beta) \bullet A =_\beta y_\beta) = \mathbf{T}$
      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 $\varphi \rightsquigarrow \mathcal{D}$
  and $A \in wffs_\alpha$
  shows $\mathcal{V}\ \varphi\ ((\lambda x_\alpha.\ x_\alpha) \bullet A =_\alpha A) = \mathbf{T}$
proof −
  from *assms*(*2*) **have** $(\lambda x_\alpha.\ x_\alpha) \bullet A =_\alpha A \in wffs_o$
    using *axioms.axiom-4-2* **and** *axioms-are-wffs-of-type-o* **by** *blast*
  define ψ **where** $\psi = \varphi((x, \alpha) := \mathcal{V}\ \varphi\ A)$
  with *assms* **have** $\mathcal{V}\ \varphi\ ((\lambda x_\alpha.\ x_\alpha) \bullet A) = \mathcal{V}\ \psi\ (x_\alpha)$
    using *prop-5401-a* **by** *blast*
  also from *assms* **and** ψ*-def* **have** $\ldots = \psi\ (x, \alpha)$
    using $\mathcal{V}$*-is-wff-denotation-function* **by** *force*
  also from ψ*-def* **have** $\ldots = \mathcal{V}\ \varphi\ A$
    by *simp*
  finally have $\mathcal{V}\ \varphi\ ((\lambda x_\alpha.\ x_\alpha) \bullet A) = \mathcal{V}\ \varphi\ A$ **.**

with *assms*(*1*) **and** ‹$(\lambda x_\alpha.\ x_\alpha) \bullet A =_\alpha A \in$ *wffs*$_o$› **show** *?thesis*
  using *wffs-from-equality* **and** *prop-5401-b* **by** *meson*
qed

lemma *axiom-4-2-validity*:
  assumes $A \in$ *wffs*$_\alpha$
  shows $\models (\lambda x_\alpha.\ x_\alpha) \bullet A =_\alpha A$
proof (*intro allI impI*)
  fix $\mathcal{M}$ **and** φ
  assume $*$: *is-general-model* $\mathcal{M}$ $\varphi \leadsto_M \mathcal{M}$
  show $\mathcal{M} \models_\varphi (\lambda x_\alpha.\ x_\alpha) \bullet A =_\alpha A$
  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** $\mathcal{V}\ \varphi\ ((\lambda x_\alpha.\ x_\alpha) \bullet A =_\alpha A) = \mathbf{T}$
      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 $\varphi \leadsto \mathcal{D}$
  and $A \in$ *wffs*$_\alpha$ **and** $B \in$ *wffs*$_{\gamma \to \beta}$ **and** $C \in$ *wffs*$_\gamma$
  shows $\mathcal{V}\ \varphi\ ((\lambda x_\alpha.\ B \bullet C) \bullet A =_\beta ((\lambda x_\alpha.\ B) \bullet A) \bullet ((\lambda x_\alpha.\ C) \bullet A)) = \mathbf{T}$
  (**is** $\mathcal{V}\ \varphi\ (?A =_\beta\ ?B) = \mathbf{T}$)
proof −
  from *assms*(*2*−*4*) **have** $?A =_\beta\ ?B \in$ *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** $\psi \leadsto \mathcal{D}$
    using $\mathcal{V}$*-is-wff-denotation-function* **by** *auto*
  from *assms* **and** ψ*-def* **have** $\mathcal{V}\ \varphi\ ?A = \mathcal{V}\ \psi\ (B \bullet C)$
    using *prop-5401-a* **by** *blast*
  also from *assms*(*3*,*4*) **and** ψ*-def* **and** ‹$\psi \leadsto \mathcal{D}$› **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** $\ldots = \mathcal{V}\ \varphi\ ((\lambda x_\alpha.\ B) \bullet A) \cdot \mathcal{V}\ \psi\ C$
    using *prop-5401-a* **by** *simp*
  also from *assms*(*1*,*2*,*4*) **and** ψ*-def* **have** $\ldots = \mathcal{V}\ \varphi\ ((\lambda x_\alpha.\ B) \bullet A) \cdot \mathcal{V}\ \varphi\ ((\lambda x_\alpha.\ C) \bullet A)$
    using *prop-5401-a* **by** *simp*
  also have $\ldots = \mathcal{V}\ \varphi\ ?B$
  proof −
    have $(\lambda x_\alpha.\ B) \bullet A \in$ *wffs*$_{\gamma \to \beta}$ **and** $(\lambda x_\alpha.\ C) \bullet A \in$ *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** ‹$?A =_\beta\ ?B \in$ *wffs*$_o$› **show** *?thesis*

using *prop-5401-b* **and** *wffs-from-equality* **by** *meson*
qed

lemma *axiom-4-3-validity*:
assumes $A \in wffs_\alpha$ **and** $B \in wffs_{\gamma\to\beta}$ **and** $C \in wffs_\gamma$
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* $\mathcal{M}\ \varphi \rightsquigarrow_M \mathcal{M}$
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** $\mathcal{V}\ \varphi\ (?A =_\beta\ ?B) = \mathbf{T}$
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 $A \in wffs_\alpha$ **and** $B \in wffs_\delta$
and $(y, \gamma) \notin \{(x, \alpha)\} \cup vars\ A$
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** $\mathcal{V}\ \varphi\ (?A =_{\gamma\to\delta}\ ?B) = \mathbf{T}$)
proof −
from *assms*(*2*,*3*) **have** $?A =_{\gamma\to\delta}\ ?B \in wffs_o$
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 $z \in elts\ (\mathcal{D}\ \gamma)$
define φ' **where** $\varphi' = \varphi((y, \gamma) := z)$
from *assms*(*1*) **and** ‹$z \in elts\ (\mathcal{D}\ \gamma)$› **and** φ'*-def* **have** $\varphi' \rightsquigarrow \mathcal{D}$
by *simp*
moreover from φ'*-def* **and** *assms*(*4*) **have** $\forall\, v \in free\text{-}vars\ A.\ \varphi\ v = \varphi'\ v$
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** ‹$z \in elts\ (\mathcal{D}\ \gamma)$› **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** ‹$z \in$ *elts* ($\mathcal{D}$ γ)› **and** ‹*?A* $=_{\gamma\to\delta}$ *?B* $\in$ *wffs*$_o$›
have ... $= \mathcal{V}$ φ *?B* $\boldsymbol{\cdot}$ z
by (*metis mixed-beta-conversion wffs-from-abs wffs-from-equality*(*2*))
finally have $\mathcal{V}$ ψ *?D* $\boldsymbol{\cdot}$ $z = \mathcal{V}$ φ *?B* $\boldsymbol{\cdot}$ z **.**
}
note $* =$ *this*
then have $\mathcal{V}$ ψ *?D* $= \mathcal{V}$ φ *?B*
proof $-$
from ‹$\psi \leadsto \mathcal{D}$› **and** *assms*(*3*) **have** $\mathcal{V}$ ψ *?D* $=$ ($\boldsymbol{\lambda}z$ **:** $\mathcal{D}$ γ**.** $\mathcal{V}$ (ψ((y, γ) := z)) B)
using *wff-abs-denotation*[*OF* $\mathcal{V}$*-is-wff-denotation-function*] **by** *simp*
moreover from *assms*(*1*) **have** $\mathcal{V}$ φ *?B* $=$ ($\boldsymbol{\lambda}z$ **:** $\mathcal{D}$ γ**.** $\mathcal{V}$ (φ((y, γ) := z)) ((λx_α. B) $\boldsymbol{\cdot}$ A))
using *wffs-from-abs*[*OF wffs-from-equality*(*2*)[*OF* ‹*?A* $=_{\gamma\to\delta}$ *?B* $\in$ *wffs*$_o$›]]
and *wff-abs-denotation*[*OF* $\mathcal{V}$*-is-wff-denotation-function*] **by** *meson*
ultimately show *?thesis*
using *vlambda-extensionality* **and** $*$ **by** *fastforce*
qed
with *assms*(*1*$-$*3*) **and** ψ*-def* **have** $\mathcal{V}$ φ *?A* $= \mathcal{V}$ 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* $=_{\gamma\to\delta}$ *?B* $\in$ *wffs*$_o$›] **by** *blast*
qed

lemma *axiom-4-4-validity*:
assumes $A \in$ *wffs*$_\alpha$ **and** $B \in$ *wffs*$_\delta$
and (y, γ) $\notin$ {(x, α)} $\cup$ *vars* A
shows $\models$ (λx_α. λy_γ. B) $\boldsymbol{\cdot}$ A $=_{\gamma\to\delta}$ (λy_γ. (λx_α. B) $\boldsymbol{\cdot}$ A) (**is** $\models$ *?A* $=_{\gamma\to\delta}$ *?B*)
proof (*intro allI impI*)
fix $\mathcal{M}$ **and** φ
assume $*$: *is-general-model* $\mathcal{M}$ $\varphi \leadsto_M \mathcal{M}$
show $\mathcal{M} \models_\varphi$ *?A* $=_{\gamma\to\delta}$ *?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** $\mathcal{V}$ φ (*?A* $=_{\gamma\to\delta}$ *?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 $\varphi \leadsto \mathcal{D}$
and $A \in$ *wffs*$_\alpha$ **and** $B \in$ *wffs*$_\delta$
shows $\mathcal{V}$ φ ((λx_α. λx_α. B) $\boldsymbol{\cdot}$ A $=_{\alpha\to\delta}$ (λx_α. B)) $=$ **T**
proof $-$
define ψ **where** $\psi = \varphi((x, \alpha) := \mathcal{V}\ \varphi\ A)$
from *assms* **have** *wff*: (λx_α. λx_α. B) $\boldsymbol{\cdot}$ A $=_{\alpha\to\delta}$ (λx_α. B) $\in$ *wffs*$_o$
using *axioms.axiom-4-5* **and** *axioms-are-wffs-of-type-o* **by** *blast*
with *assms*(*1*,*2*) **and** ψ*-def* **have** $\mathcal{V}$ φ ((λx_α. λx_α. B) $\boldsymbol{\cdot}$ A) $= \mathcal{V}$ ψ (λx_α. B)

using *prop-5401-a* **and** *wffs-from-equality*(*2*) **by** *blast*
also have $\ldots = \mathcal{V}\ \varphi\ (\lambda x_\alpha.\ B)$
proof $-$
have $(x, \alpha) \notin$ *free-vars* $(\lambda x_\alpha.\ B)$
by *simp*
with ψ*-def* **have** $\forall\, v \in$ *free-vars* $(\lambda x_\alpha.\ B).\ \varphi\ v = \psi\ v$
by *simp*
moreover from ψ*-def* **and** *assms*(*1*,*2*) **have** $\psi \rightsquigarrow \mathcal{D}$
using $\mathcal{V}$*-is-wff-denotation-function* **by** *simp*
moreover from *assms*(*2*,*3*) **have** $(\lambda x_\alpha.\ B) \in wffs_{\alpha\to\delta}$
by *fastforce*
ultimately show *?thesis*
using *assms*(*1*) **and** *prop-5400* **by** *metis*
qed
finally have $\mathcal{V}\ \varphi\ ((\lambda x_\alpha.\ \lambda x_\alpha.\ B) \cdot A) = \mathcal{V}\ \varphi\ (\lambda x_\alpha.\ B)$ **.**
with *wff* **and** *assms*(*1*) **show** *?thesis*
using *prop-5401-b* **and** *wffs-from-equality* **by** *meson*
qed

lemma *axiom-4-5-validity*:
assumes $A \in wffs_\alpha$ **and** $B \in wffs_\delta$
shows $\models (\lambda x_\alpha.\ \lambda x_\alpha.\ B) \cdot A =_{\alpha\to\delta} (\lambda x_\alpha.\ B)$
proof (*intro allI impI*)
fix $\mathcal{M}$ **and** φ
assume $*$: *is-general-model* $\mathcal{M}\ \varphi \rightsquigarrow_M \mathcal{M}$
show $\mathcal{M} \models_\varphi (\lambda x_\alpha.\ \lambda x_\alpha.\ B) \cdot A =_{\alpha\to\delta} (\lambda x_\alpha.\ B)$
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** $\langle\mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V})\rangle$ **have** $\mathcal{V}\ \varphi\ ((\lambda x_\alpha.\ \lambda x_\alpha.\ B) \cdot A =_{\alpha\to\delta} (\lambda x_\alpha.\ B)) = \mathbf{T}$
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 $\varphi \rightsquigarrow \mathcal{D}$
shows $\mathcal{V}\ \varphi\ (\iota \cdot (Q_i \cdot \mathfrak{y}_i) =_i \mathfrak{y}_i) = \mathbf{T}$
proof $-$
have $\iota \cdot (Q_i \cdot \mathfrak{y}_i) =_i \mathfrak{y}_i \in wffs_o$
using *axioms.axiom-5* **and** *axioms-are-wffs-of-type-o* **by** *blast*
have $Q_i \cdot \mathfrak{y}_i \in wffs_{i\to o}$
by *blast*
with *assms* **have** $\mathcal{V}\ \varphi\ (\iota \cdot (Q_i \cdot \mathfrak{y}_i)) = \mathcal{V}\ \varphi\ \iota \cdot \mathcal{V}\ \varphi\ (Q_i \cdot \mathfrak{y}_i)$
using $\mathcal{V}$*-is-wff-denotation-function* **by** *blast*
also from *assms* **have** $\ldots = \mathcal{V}\ \varphi\ \iota \cdot (\mathcal{V}\ \varphi\ (Q_i) \cdot \mathcal{V}\ \varphi\ (\mathfrak{y}_i))$
using *wff-app-denotation*[*OF* $\mathcal{V}$*-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 $\mathcal{V}\ \varphi\ (\iota \bullet (Q_i \bullet \mathfrak{y}_i)) = \mathcal{J}\ (\mathfrak{c}_\iota,\ (i{\to}o){\to}i) \cdot \{\mathcal{V}\ \varphi\ (\mathfrak{y}_i)\}_i{}^{\mathcal{D}}$ **.**
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 $\mathcal{V}\ \varphi\ (\iota \bullet (Q_i \bullet \mathfrak{y}_i)) = \mathcal{V}\ \varphi\ (\mathfrak{y}_i)$
by (*simp only*:)
with *assms* **and** ‹$Q_i \bullet \mathfrak{y}_i \in wffs_{i\to o}$› **show** *?thesis*
using *prop-5401-b* **by** *blast*
qed

lemma *axiom-5-validity*:
shows $\models \iota \bullet (Q_i \bullet \mathfrak{y}_i) =_i \mathfrak{y}_i$
proof (*intro allI impI*)
fix $\mathcal{M}$ **and** φ
assume $*$: *is-general-model* $\mathcal{M}$ $\varphi \rightsquigarrow_M \mathcal{M}$
show $\mathcal{M} \models_\varphi \iota \bullet (Q_i \bullet \mathfrak{y}_i) =_i \mathfrak{y}_i$
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** ‹$\mathcal{M} = (\mathcal{D},\ \mathcal{J},\ \mathcal{V})$› **have** $\mathcal{V}\ \varphi\ (\iota \bullet (Q_i \bullet \mathfrak{y}_i) =_i \mathfrak{y}_i) = \mathbf{T}$
using *general-model.axiom-5-validity-aux* **by** *simp*
ultimately show *?thesis*
by *simp*
qed
qed

lemma *axioms-validity*:
assumes $A \in$ *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 $A \in wffs_\alpha$ **and** $B \in wffs_\alpha$

and $\forall\varphi.\ \varphi \leadsto \mathcal{D} \longrightarrow \mathcal{V}\ \varphi\ A = \mathcal{V}\ \varphi\ B$
and $C \in \mathit{wffs}_\beta$ **and** $C' \in \mathit{wffs}_\beta$
and $p \in \mathit{positions}\ C$ **and** $A \preceq_p C$ **and** $C\langle\!|p \leftarrow B|\!\rangle \rhd C'$
shows $\forall\varphi.\ \varphi \leadsto \mathcal{D} \longrightarrow \mathcal{V}\ \varphi\ C = \mathcal{V}\ \varphi\ C'$
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 $\varphi \leadsto \mathcal{D}$
from ‹$G \cdot H \in \mathit{wffs}_\beta$› **obtain** γ **where** $G \in \mathit{wffs}_{\gamma\to\beta}$ **and** $H \in \mathit{wffs}_\gamma$
by (*rule wffs-from-app*)
with ‹$G' \cdot H \in \mathit{wffs}_\beta$› **have** $G' \in \mathit{wffs}_{\gamma\to\beta}$
by (*metis wff-has-unique-type wffs-from-app*)
from *assms(1)* **and** ‹$\varphi \leadsto \mathcal{D}$› **and** ‹$G \in \mathit{wffs}_{\gamma\to\beta}$› **and** ‹$H \in \mathit{wffs}_\gamma$›
have $\mathcal{V}\ \varphi\ (G \cdot H) = \mathcal{V}\ \varphi\ G \cdot \mathcal{V}\ \varphi\ H$
using $\mathcal{V}$*-is-wff-denotation-function* **by** *blast*
also from ‹$\varphi \leadsto \mathcal{D}$› **and** ‹$G \in \mathit{wffs}_{\gamma\to\beta}$› **and** ‹$G' \in \mathit{wffs}_{\gamma\to\beta}$› **have** $\ldots = \mathcal{V}\ \varphi\ G' \cdot \mathcal{V}\ \varphi\ H$
using *replace-left-app.IH* **and** *replace-left-app.prems(1,4)* **by** *simp*
also from *assms(1)* **and** ‹$\varphi \leadsto \mathcal{D}$› **and** ‹$G' \in \mathit{wffs}_{\gamma\to\beta}$› **and** ‹$H \in \mathit{wffs}_\gamma$›
have $\ldots = \mathcal{V}\ \varphi\ (G' \cdot H)$
using $\mathcal{V}$*-is-wff-denotation-function* **by** *fastforce*
finally show $\mathcal{V}\ \varphi\ (G \cdot H) = \mathcal{V}\ \varphi\ (G' \cdot H)$ **.**
qed
next
case (*replace-right-app* $p\ H\ B'\ H'\ G$)
show *?case*
proof (*intro allI impI*)
fix φ
assume $\varphi \leadsto \mathcal{D}$
from ‹$G \cdot H \in \mathit{wffs}_\beta$› **obtain** γ **where** $G \in \mathit{wffs}_{\gamma\to\beta}$ **and** $H \in \mathit{wffs}_\gamma$
by (*rule wffs-from-app*)
with ‹$G \cdot H' \in \mathit{wffs}_\beta$› **have** $H' \in \mathit{wffs}_\gamma$
using *wff-has-unique-type* **and** *wffs-from-app* **by** (*metis type.inject*)
from *assms(1)* **and** ‹$\varphi \leadsto \mathcal{D}$› **and** ‹$G \in \mathit{wffs}_{\gamma\to\beta}$› **and** ‹$H \in \mathit{wffs}_\gamma$›
have $\mathcal{V}\ \varphi\ (G \cdot H) = \mathcal{V}\ \varphi\ G \cdot \mathcal{V}\ \varphi\ H$
using $\mathcal{V}$*-is-wff-denotation-function* **by** *blast*
also from ‹$\varphi \leadsto \mathcal{D}$› **and** ‹$H \in \mathit{wffs}_\gamma$› **and** ‹$H' \in \mathit{wffs}_\gamma$› **have** $\ldots = \mathcal{V}\ \varphi\ G \cdot \mathcal{V}\ \varphi\ H'$
using *replace-right-app.IH* **and** *replace-right-app.prems(1,4)* **by** *force*
also from *assms(1)* **and** ‹$\varphi \leadsto \mathcal{D}$› **and** ‹$G \in \mathit{wffs}_{\gamma\to\beta}$› **and** ‹$H' \in \mathit{wffs}_\gamma$›
have $\ldots = \mathcal{V}\ \varphi\ (G \cdot H')$
using $\mathcal{V}$*-is-wff-denotation-function* **by** *fastforce*

finally show $\mathcal{V}\ \varphi\ (G \bullet H) = \mathcal{V}\ \varphi\ (G \bullet H')$ **.**
qed
next
case (*replace-abs p E B′ E′ x γ*)
show *?case*
proof (*intro allI impI*)
fix φ
assume $\varphi \rightsquigarrow \mathcal{D}$
define ψ **where** $\psi\ z = \varphi((x, \gamma) := z)$ **for** z
with ‹$\varphi \rightsquigarrow \mathcal{D}$› **have** ψ-*assg*: $\psi\ z \rightsquigarrow \mathcal{D}$ **if** $z \in$ *elts* $(\mathcal{D}\ \gamma)$ **for** z
by (*simp add*: *that*)
from ‹$\lambda x_\gamma.\ E \in wffs_\beta$› **obtain** δ **where** $\beta = \gamma \rightarrow \delta$ **and** $E \in wffs_\delta$
by (*rule wffs-from-abs*)
with ‹$\lambda x_\gamma.\ E' \in wffs_\beta$› **have** $E' \in wffs_\delta$
using *wffs-from-abs* **by** *blast*
from *assms*(*1*) **and** ‹$\varphi \rightsquigarrow \mathcal{D}$› **and** ‹$E \in wffs_\delta$› **and** ψ-*def*
have $\mathcal{V}\ \varphi\ (\lambda x_\gamma.\ E) = (\boldsymbol{\lambda} z : \mathcal{D}\ \gamma\textbf{.}\ \mathcal{V}\ (\psi\ z)\ E)$
using *wff-abs-denotation*[*OF* $\mathcal{V}$-*is-wff-denotation-function*] **by** *simp*
also have $\ldots = (\boldsymbol{\lambda} z : \mathcal{D}\ \gamma\textbf{.}\ \mathcal{V}\ (\psi\ z)\ E')$
proof (*intro vlambda-extensionality*)
fix z
assume $z \in$ *elts* $(\mathcal{D}\ \gamma)$
from ‹$E \in wffs_\delta$› **and** ‹$E' \in wffs_\delta$› **have** $\forall \varphi.\ \varphi \rightsquigarrow \mathcal{D} \longrightarrow \mathcal{V}\ \varphi\ E = \mathcal{V}\ \varphi\ E'$
using *replace-abs.prems*(*1*,*4*) **and** *replace-abs.IH* **by** *simp*
with ψ-*assg* **and** ‹$z \in$ *elts* $(\mathcal{D}\ \gamma)$› **show** $\mathcal{V}\ (\psi\ z)\ E = \mathcal{V}\ (\psi\ z)\ E'$
by *simp*
qed
also from *assms*(*1*) **and** ‹$\varphi \rightsquigarrow \mathcal{D}$› **and** ‹$E' \in wffs_\delta$› **and** ψ-*def*
have $\ldots = \mathcal{V}\ \varphi\ (\lambda x_\gamma.\ E')$
using *wff-abs-denotation*[*OF* $\mathcal{V}$-*is-wff-denotation-function*] **by** *simp*
finally show $\mathcal{V}\ \varphi\ (\lambda x_\gamma.\ E) = \mathcal{V}\ \varphi\ (\lambda x_\gamma.\ E')$ **.**
qed
qed
qed

lemma *rule-R-validity*:
assumes $C \in wffs_o$ **and** $C' \in wffs_o$ **and** $E \in wffs_o$
and $\models C$ **and** $\models E$
and *is-rule-R-app p C′ C E*
shows $\models C'$
proof (*intro allI impI*)
fix $\mathcal{M}$ **and** φ
assume *is-general-model* $\mathcal{M}$ **and** $\varphi \rightsquigarrow_M \mathcal{M}$
show $\mathcal{M} \models_\varphi C'$
proof −
have $\mathcal{M} \models C'$
proof −
obtain $\mathcal{D}$ **and** $\mathcal{J}$ **and** $\mathcal{V}$ **where** $\mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V})$
using *prod-cases3* **by** *blast*

    from *assms*(*6*) **obtain** A **and** B **and** α **where** $A \in \mathit{wffs}_\alpha$ **and** $B \in \mathit{wffs}_\alpha$ **and** $E = A =_\alpha B$
      using *wffs-from-equality* **by** (*meson is-rule-R-app-def*)
    note $* =$ ‹*is-general-model* $\mathcal{M}$› ‹$\mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V})$› ‹$\varphi \leadsto_M \mathcal{M}$›
    have $\mathcal{V}\ \varphi'\ C = \mathcal{V}\ \varphi'\ C'$ **if** $\varphi' \leadsto \mathcal{D}$ **for** φ'
    proof −
      from *assms*(*5*) **and** $*$(*1*,*2*) **and** ‹$A \in \mathit{wffs}_\alpha$› **and** ‹$B \in \mathit{wffs}_\alpha$› **and** ‹$E = A =_\alpha B$› **and** *that*
      have $\forall\,\varphi'.\ \varphi' \leadsto \mathcal{D} \longrightarrow \mathcal{V}\ \varphi'\ A = \mathcal{V}\ \varphi'\ B$
        using *general-model.prop-5401-b* **by** *blast*
      moreover
      from ‹$E = A =_\alpha B$› **and** *assms*(*6*) **have** $p \in \mathit{positions}\ C$ **and** $A \preceq_p C$ **and** $C\langle\!|p \leftarrow B|\!\rangle \rhd C'$
        using *is-subform-implies-in-positions* **by** *auto*
      ultimately show *?thesis*
        using ‹$A \in \mathit{wffs}_\alpha$› **and** ‹$B \in \mathit{wffs}_\alpha$› **and** ‹$C \in \mathit{wffs}_o$› **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 ‹$\varphi \leadsto_M \mathcal{M}$› **show** *?thesis*
    by *blast*
 qed
qed

lemma *individual-proof-step-validity*:
  assumes *is-proof* $\mathcal{S}$ **and** $A \in \mathit{lset}\ \mathcal{S}$
  shows $\models A$
using *assms* **proof** (*induction length* $\mathcal{S}$ *arbitrary*: $\mathcal{S}$ A *rule*: *less-induct*)
  case *less*
  from ‹$A \in \mathit{lset}\ \mathcal{S}$› **obtain** i' **where** $\mathcal{S}\ !\ i' = A$ **and** $\mathcal{S} \neq []$ **and** $i' < \mathit{length}\ \mathcal{S}$
    by (*metis empty-iff empty-set in-set-conv-nth*)
  with ‹*is-proof* $\mathcal{S}$› **have** *is-proof* (*take* (*Suc* i') $\mathcal{S}$) **and** *take* (*Suc* i') $\mathcal{S} \neq []$
    using *proof-prefix-is-proof*[**where** $\mathcal{S}_1 = \mathit{take}\ (\mathit{Suc}\ i')\ \mathcal{S}$ **and** $\mathcal{S}_2 = \mathit{drop}\ (\mathit{Suc}\ i')\ \mathcal{S}$]
    and *append-take-drop-id* **by** *simp-all*
  from ‹$i' < \mathit{length}\ \mathcal{S}$› **consider** (*a*) $i' < \mathit{length}\ \mathcal{S} - 1$ | (*b*) $i' = \mathit{length}\ \mathcal{S} - 1$
    by *fastforce*
  then show *?case*
  proof *cases*
    case *a*
    then have *length* (*take* (*Suc* i') $\mathcal{S}$) $<$ *length* $\mathcal{S}$
      by *simp*
    with ‹$\mathcal{S}\ !\ i' = A$› **and** ‹*take* (*Suc* i') $\mathcal{S} \neq []$› **have** $A \in \mathit{lset}$ (*take* (*Suc* i') $\mathcal{S}$)
      by (*simp add*: *take-Suc-conv-app-nth*)
    with ‹*length* (*take* (*Suc* i') $\mathcal{S}$) $<$ *length* $\mathcal{S}$› **and** ‹*is-proof* (*take* (*Suc* i') $\mathcal{S}$)› **show** *?thesis*
      using *less*(*1*) **by** *blast*
  next
    case *b*
    with ‹$\mathcal{S}\ !\ i' = A$› **and** ‹$\mathcal{S} \neq []$› **have** *last* $\mathcal{S} = A$
      using *last-conv-nth* **by** *blast*
    with ‹*is-proof* $\mathcal{S}$› **and** ‹$\mathcal{S} \neq []$› **and** *b* **have** *is-proof-step* $\mathcal{S}$ i'

using *added-suffix-proof-preservation*[**where** $\mathcal{S}' = []$] **by** *simp*
then consider
($axiom$) $\mathcal{S} \mathbin{!} i' \in axioms$
| ($rule\text{-}R$) $\exists\, p\ j\ k.\ \{j, k\} \subseteq \{0..<i'\} \wedge is\text{-}rule\text{-}R\text{-}app\ p\ (\mathcal{S} \mathbin{!} i')\ (\mathcal{S} \mathbin{!} j)\ (\mathcal{S} \mathbin{!} k)$
by *fastforce*
then show *?thesis*
proof *cases*
case *axiom*
with ‹$\mathcal{S} \mathbin{!} i' = A$› **show** *?thesis*
by (*blast dest*: *axioms-validity*)
next
case *rule-R*
then obtain p **and** j **and** k
where $\{j, k\} \subseteq \{0..<i'\}$ **and** $is\text{-}rule\text{-}R\text{-}app\ p\ (\mathcal{S} \mathbin{!} i')\ (\mathcal{S} \mathbin{!} j)\ (\mathcal{S} \mathbin{!} k)$
by *blast*
let $?\mathcal{S}_j = take\ (Suc\ j)\ \mathcal{S}$ **and** $?\mathcal{S}_k = take\ (Suc\ k)\ \mathcal{S}$
obtain $\mathcal{S}_j{}'$ **and** $\mathcal{S}_k{}'$ **where** $\mathcal{S} = ?\mathcal{S}_j\ @\ \mathcal{S}_j{}'$ **and** $\mathcal{S} = ?\mathcal{S}_k\ @\ \mathcal{S}_k{}'$
by (*metis append-take-drop-id*)
with ‹*is-proof* $\mathcal{S}$› **have** *is-proof* $(?\mathcal{S}_j\ @\ \mathcal{S}_j{}')$ **and** *is-proof* $(?\mathcal{S}_k\ @\ \mathcal{S}_k{}')$
by (*simp-all only*:)
moreover from ‹$\mathcal{S} \neq []$› **have** $?\mathcal{S}_j \neq []$ **and** $?\mathcal{S}_k \neq []$
by *simp-all*
ultimately have *is-proof-of* $?\mathcal{S}_j$ (*last* $?\mathcal{S}_j$) **and** *is-proof-of* $?\mathcal{S}_k$ (*last* $?\mathcal{S}_k$)
using *proof-prefix-is-proof-of-last*[**where** $\mathcal{S} = ?\mathcal{S}_j$ **and** $\mathcal{S}' = \mathcal{S}_j{}'$]
and *proof-prefix-is-proof-of-last*[**where** $\mathcal{S} = ?\mathcal{S}_k$ **and** $\mathcal{S}' = \mathcal{S}_k{}'$]
by *fastforce+*
moreover
from ‹$\{j, k\} \subseteq \{0..<i'\}$› **and** b **have** *length* $?\mathcal{S}_j <$ *length* $\mathcal{S}$ **and** *length* $?\mathcal{S}_k <$ *length* $\mathcal{S}$
by *force+*
moreover from *calculation*(*3*,*4*) **have** $\mathcal{S} \mathbin{!} j \in lset\ ?\mathcal{S}_j$ **and** $\mathcal{S} \mathbin{!} k \in lset\ ?\mathcal{S}_k$
by (*simp-all add*: *take-Suc-conv-app-nth*)
ultimately have $\models \mathcal{S} \mathbin{!} j$ **and** $\models \mathcal{S} \mathbin{!} k$
using ‹$?\mathcal{S}_j \neq []$› **and** ‹$?\mathcal{S}_k \neq []$› **and** *less*(*1*) **unfolding** *is-proof-of-def* **by** *presburger+*
moreover have $\mathcal{S} \mathbin{!} i' \in wffs_o$ **and** $\mathcal{S} \mathbin{!} j \in wffs_o$ **and** $\mathcal{S} \mathbin{!} k \in wffs_o$
using ‹$is\text{-}rule\text{-}R\text{-}app\ p\ (\mathcal{S} \mathbin{!} i')\ (\mathcal{S} \mathbin{!} j)\ (\mathcal{S} \mathbin{!} k)$› **and** *replacement-preserves-typing*
by *force+*
ultimately show *?thesis*
using ‹$is\text{-}rule\text{-}R\text{-}app\ p\ (\mathcal{S} \mathbin{!} i')\ (\mathcal{S} \mathbin{!} j)\ (\mathcal{S} \mathbin{!} k)$› **and** ‹$\mathcal{S} \mathbin{!} i' = A$›
and *rule-R-validity*[**where** $C' = A$] **by** *blast*
qed
qed
qed

lemma *semantic-modus-ponens*:
assumes *is-general-model* $\mathcal{M}$
and $A \in wffs_o$ **and** $B \in wffs_o$
and $\mathcal{M} \models A \supset^{\mathcal{Q}} B$
and $\mathcal{M} \models A$
shows $\mathcal{M} \models B$

proof (*intro allI impI*)
  fix φ
  assume $\varphi \leadsto_M \mathcal{M}$
  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 $\varphi \leadsto \mathcal{D}$
    by *simp*
  show $\mathcal{M} \models_\varphi B$
  proof −
    from *assms*(*4*) **have** $\mathcal{V}\ \varphi\ (A \supset^{\mathcal{Q}} B) = \mathbf{T}$
      using ‹$\mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V})$› **and** ‹$\varphi \leadsto_M \mathcal{M}$› **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** ‹$\varphi \leadsto_M \mathcal{M}$› **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** ‹$\varphi \leadsto \mathcal{D}$› **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** ‹$\varphi \leadsto \mathcal{D}$› **and** ‹$\mathcal{V}\ \varphi\ A = \mathbf{T}$› **have** $\{\mathcal{V}\ \varphi\ A, \mathcal{V}\ \varphi\ B\} \subseteq$ *elts*
$\mathbb{B}$
      using *general-model.$\mathcal{V}$-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** ‹$\varphi \leadsto \mathcal{D}$› **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* $\supset^{\mathcal{Q}}{}_\star\ P$
  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* @ $[H']$) $\supset^{\mathcal{Q}}{}_\star\ P$› **have** $\mathcal{M} \models$ *hs* $\supset^{\mathcal{Q}}{}_\star\ (H' \supset^{\mathcal{Q}} P)$
    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** $H' \supset^{\mathcal{Q}} P \in$ *wffs*$_o$
    by *auto*
  ultimately have $\mathcal{M} \models H' \supset^{\mathcal{Q}} P$
    by (*elim snoc.IH*)

moreover from ‹$\forall H \in$ *lset* (*hs* @ [H']). $\mathcal{M} \models H$› **have** $\mathcal{M} \models H'$
  by *simp*
moreover from ‹$H' \supset^{\mathcal{Q}} P \in$ *wffs*$_o$› **have** $H' \in$ *wffs*$_o$
  using *wffs-from-imp-op*(*1*) **by** *blast*
ultimately show *?case*
  using *assms*(*1*) **and** ‹$P \in$ *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** $A \in$ *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 $\vdash$ *hs* $\supset^{\mathcal{Q}}{}_\star$ A
    using *generalized-deduction-theorem* **by** *simp*
  with *assms*(*4*) **have** $\mathcal{M} \models$ *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* $\subseteq$ *wffs*$_o$
    by *blast*
  ultimately show *?thesis*
    using *assms*(*1*,*4*) **and** ‹$A \in$ *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* $\mathcal{G} \longleftrightarrow \mathcal{G} \vdash F_o$

definition $\mathcal{Q}_0$*-is-inconsistent* :: *bool* **where**
  [*iff*]: $\mathcal{Q}_0$*-is-inconsistent* $\longleftrightarrow\ \vdash F_o$

definition *is-wffo-consistent-with* :: *form* ⇒ *form set* ⇒ *bool* **where**
  [*iff*]: *is-wffo-consistent-with* $B\ \mathcal{G} \longleftrightarrow \neg$ *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* $\{0\}$
| $\mathcal{D}^S\ o = \mathbb{B}$
| $\mathcal{D}^S\ (\alpha \rightarrow \beta) = \mathcal{D}^S\ \alpha \longmapsto \mathcal{D}^S\ \beta$

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* $\beta\ \gamma$)
      from ‹$\mathcal{D}^S\ \gamma \neq 0$› **obtain** y **where** $y \in$ *elts* $(\mathcal{D}^S\ \gamma)$
        by *fastforce*
      then have $(\boldsymbol{\lambda} z : \mathcal{D}^S\ \beta.\ y) \in$ *elts* $(\mathcal{D}^S\ \beta \longmapsto \mathcal{D}^S\ \gamma)$
        by (*intro VPi-I*)
      then show *?case*
        by *force*
    qed *simp-all*
  }
  then show $\forall \alpha.\ \mathcal{D}^S\ \alpha \neq 0$
    by (*intro allI*)
qed *simp-all*

definition *singleton-standard-constant-denotation-function* ($\mathcal{J}^S$) **where**
  [*simp*]: $\mathcal{J}^S\ k =$
    (
      if
        $\exists \beta.$ *is-Q-constant-of-type* $k\ \beta$
      then

let β = *type-of-Q-constant k in* $q_\beta{}^{\mathcal{D}^S}$
else
if
is-iota-constant k
then
$\boldsymbol{\lambda} z : \mathcal{D}^S\ (i{\to}o)\mathbf{.}\ 0$
else
case k of $(c, \alpha) \Rightarrow$ *SOME z. z* $\in$ *elts* $(\mathcal{D}^S\ \alpha)$
)

interpretation *singleton-standard-premodel*: *premodel* $\mathcal{D}^S\ \mathcal{J}^S$
proof (*unfold-locales*)
show $\forall\,\alpha.\ \mathcal{J}^S$ (*Q-constant-of-type* α) = $q_\alpha{}^{\mathcal{D}^S}$
by *simp*
next
show *singleton-standard-frame.is-unique-member-selector* ($\mathcal{J}^S$ *iota-constant*)
unfolding *singleton-standard-frame.is-unique-member-selector-def* **proof**
fix x
assume $x \in$ *elts* $(\mathcal{D}^S\ i)$
then have $x = 0$
by *simp*
moreover have $(\boldsymbol{\lambda} z : \mathcal{D}^S\ (i{\to}o)\mathbf{.}\ 0) \cdot \{0\}_i{}^{\mathcal{D}^S} = 0$
using *beta*[*OF singleton-standard-frame.one-element-function-is-domain-respecting*]
unfolding *singleton-standard-domain-family.simps*(*3*) **by** *blast*
ultimately show ($\mathcal{J}^S$ *iota-constant*) $\cdot \{x\}_i{}^{\mathcal{D}^S} = x$
by *fastforce*
qed
next
show $\forall\, c\ \alpha.\ \neg$ *is-logical-constant* $(c, \alpha) \longrightarrow \mathcal{J}^S\ (c, \alpha) \in$ *elts* $(\mathcal{D}^S\ \alpha)$
proof (*intro allI impI*)
fix c **and** α
assume $\neg$ *is-logical-constant* (c, α)
then have $\mathcal{J}^S\ (c, \alpha) =$ (*SOME z. z* $\in$ *elts* $(\mathcal{D}^S\ \alpha)$)
by *auto*
moreover have $\exists\, z.\ z \in$ *elts* $(\mathcal{D}^S\ \alpha)$
using *eq0-iff* **and** *singleton-standard-frame.domain-nonemptiness* **by** *presburger*
then have (*SOME z. z* $\in$ *elts* $(\mathcal{D}^S\ \alpha)$) $\in$ *elts* $(\mathcal{D}^S\ \alpha)$
using *some-in-eq* **by** *auto*
ultimately show $\mathcal{J}^S\ (c, \alpha) \in$ *elts* $(\mathcal{D}^S\ \alpha)$
by *auto*
qed
qed

fun *singleton-standard-wff-denotation-function* ($\mathcal{V}^S$) **where**
$\mathcal{V}^S\ \varphi\ (x_\alpha) = \varphi\ (x, \alpha)$
$|\ \mathcal{V}^S\ \varphi\ (\{\!|c|\!\}_\alpha) = \mathcal{J}^S\ (c, \alpha)$
$|\ \mathcal{V}^S\ \varphi\ (A \bullet B) = (\mathcal{V}^S\ \varphi\ A) \cdot (\mathcal{V}^S\ \varphi\ B)$
$|\ \mathcal{V}^S\ \varphi\ (\lambda x_\alpha.\ A) = (\boldsymbol{\lambda} z : \mathcal{D}^S\ \alpha\mathbf{.}\ \mathcal{V}^S\ (\varphi((x, \alpha) := z))\ A)$

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* $(\mathcal{D}^S\ \alpha)$
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 $\mathcal{V}^S\ \varphi\ (\{\!|c|\!\}_\alpha) = {q_\beta}^{\mathcal{D}^S}$
      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 $\mathcal{V}^S\ \varphi\ (\{\!|c|\!\}_\alpha) = (\boldsymbol{\lambda} z : \mathcal{D}^S\ (i{\to}o)\boldsymbol{.}\ 0)$
      by *simp*
    moreover have $(\boldsymbol{\lambda} z : \mathcal{D}^S\ (i{\to}o)\boldsymbol{.}\ 0) \in$ *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 $\mathcal{V}^S\ \varphi\ (A \bullet B) = (\mathcal{V}^S\ \varphi\ A) \cdot (\mathcal{V}^S\ \varphi\ B)$
    using *singleton-standard-wff-denotation-function.simps(3)* **.**
  moreover have $\mathcal{V}^S\ \varphi\ A \in$ *elts* $(\mathcal{D}^S\ (\alpha{\to}\beta))$ **and** $\mathcal{V}^S\ \varphi\ B \in$ *elts* $(\mathcal{D}^S\ \alpha)$
    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 $\mathcal{V}^S\ \varphi\ (\lambda x_\alpha.\ A) = (\boldsymbol{\lambda} z : \mathcal{D}^S\ \alpha\boldsymbol{.}\ \mathcal{V}^S\ (\varphi((x, \alpha) := z))\ A)$
    using *singleton-standard-wff-denotation-function.simps(4)* **.**
  moreover have $\mathcal{V}^S\ (\varphi((x, \alpha) := z))\ A \in$ *elts* $(\mathcal{D}^S\ \beta)$ **if** $z \in$ *elts* $(\mathcal{D}^S\ \alpha)$ **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 $\forall\,\alpha\ \beta.\ \mathcal{D}^S\ (\alpha{\rightarrow}\beta) = \mathcal{D}^S\ \alpha \longmapsto \mathcal{D}^S\ \beta$
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 $\mathcal{V}\ \varphi\ F_o$ = **T if** $\varphi \rightsquigarrow \mathcal{D}$ **for** φ
using *that* **and** ‹$\mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V})$› **by** *force*
moreover have $\mathcal{V}\ \varphi\ F_o$ = **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) $\Rightarrow$ *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 $\mathcal{Q}_0$-*is-consistent*:
shows $\neg$ $\mathcal{Q}_0$-*is-inconsistent*
proof $-$
have $\exists \mathcal{M}$. *is-general-model* $\mathcal{M}$ $\wedge$ *is-model-for* $\mathcal{M}$ $\{\}$
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* = $\mathcal{Q}_0$-*is-consistent model-existence-implies-set-consistency*

proposition *principle-of-explosion*:
assumes *is-hyps* $\mathcal{G}$
shows *is-inconsistent-set* $\mathcal{G} \longleftrightarrow (\forall A \in (wffs_o).\ \mathcal{G} \vdash A)$
proof
assume *is-inconsistent-set* $\mathcal{G}$
show $\forall A \in (wffs_o).\ \mathcal{G} \vdash A$
proof
fix A
assume $A \in wffs_o$
from ‹*is-inconsistent-set* $\mathcal{G}$› **have** $\mathcal{G} \vdash F_o$
unfolding *is-inconsistent-set-def* **.**
then have $\mathcal{G} \vdash \forall \mathfrak{x}_o.\ \mathfrak{x}_o$
unfolding *false-is-forall* **.**
with ‹$A \in wffs_o$› **have** $\mathcal{G} \vdash \mathbf{S}\ \{(\mathfrak{x},\ o) \rightarrowtail A\}\ (\mathfrak{x}_o)$
using $\forall I$ **by** *fastforce*
then show $\mathcal{G} \vdash A$
by *simp*
qed
next
assume $\forall A \in (wffs_o).\ \mathcal{G} \vdash A$
then have $\mathcal{G} \vdash F_o$
using *false-wff* **by** (*elim bspec*)
then show *is-inconsistent-set* $\mathcal{G}$
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.