

Metatheory of \mathcal{Q}_0

Javier Díaz

<javier.diaz.manzi@gmail.com>

November 15, 2023

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 proofs are, to the best of our knowledge, the first to be formalized in a proof assistant.

Contents

1	Utilities	5
1.1	Utilities for lists	5
1.2	Utilities for finite maps	5
2	Syntax	10
2.1	Type symbols	10
2.2	Variables	11
2.3	Constants	12
2.4	Formulas	13
2.5	Generalized operators	13
2.6	Subformulas	13
2.7	Free and bound variables	19
2.8	Free and bound occurrences	21
2.9	Free variables for a formula in another formula	38
2.10	Replacement of subformulas	42
2.11	Logical constants	46
2.12	Definitions and abbreviations	47
2.13	Well-formed formulas	49
2.14	Substitutions	59
2.15	Renaming of bound variables	83
3	Boolean Algebra	88
4	Propositional Well-Formed Formulas	90
4.1	Syntax	91
4.2	Semantics	95
5	Proof System	108
5.1	Axioms	108
5.2	Inference rule R	109
5.3	Proof and derivability	109
5.4	Hypothetical proof and derivability	118
6	Elementary Logic	131
6.1	Proposition 5200	131
6.2	Proposition 5201 (Equality Rules)	132
6.3	Proposition 5202 (Rule RR)	133
6.4	Proposition 5203	134
6.5	Proposition 5204	136
6.6	Proposition 5205 (η -conversion)	136
6.7	Proposition 5206 (α -conversion)	139
6.8	Proposition 5207 (β -conversion)	142
6.9	Proposition 5208	147

6.10	Proposition 5209	149
6.11	Proposition 5210	150
6.12	Proposition 5211	151
6.13	Proposition 5212	152
6.14	Proposition 5213	153
6.15	Proposition 5214	153
6.16	Proposition 5215 (Universal Instantiation)	154
6.17	Proposition 5216	155
6.18	Proposition 5217	156
6.19	Proposition 5218	158
6.20	Proposition 5219 (Rule T)	159
6.21	Proposition 5220 (Universal Generalization)	159
6.22	Proposition 5221 (Substitution)	161
6.23	Proposition 5222 (Rule of Cases)	172
6.24	Proposition 5223	175
6.25	Proposition 5224 (Modus Ponens)	176
6.26	Proposition 5225	177
6.27	Proposition 5226	178
6.28	Proposition 5227	179
6.29	Proposition 5228	180
6.30	Proposition 5229	180
6.31	Proposition 5230	181
6.32	Proposition 5231	183
6.33	Proposition 5232	184
6.34	Proposition 5233	186
6.35	Proposition 5234 (Rule P)	196
6.36	Proposition 5235	197
6.37	Proposition 5237 ($\supset \forall$ Rule)	199
6.38	Proposition 5238	203
6.39	Proposition 5239	206
6.40	Theorem 5240 (Deduction Theorem)	217
6.41	Proposition 5241	221
6.42	Proposition 5242 (Rule of Existential Generalization)	221
6.43	Proposition 5243 (Comprehension Theorem)	222
6.44	Proposition 5244 (Existential Rule)	224
6.45	Proposition 5245 (Rule C)	226
7	Semantics	229
7.1	Frames	230
7.2	Pre-models (interpretations)	231
7.3	General models	234
7.4	Standard models	235
7.5	Validity	235

8	Soundness	236
8.1	Proposition 5400	236
8.2	Proposition 5401	237
8.3	Proposition 5402(a)	267
8.4	Proposition 5402(b)	267
8.5	Theorem 5402 (Soundness Theorem)	267
9	Consistency	268
9.1	Existence of a standard model	268
9.2	Theorem 5403 (Consistency Theorem)	271

1 Utilities

```
theory Utilities
  imports
    Finite-Map-Extras.Finite-Map-Extras
begin
```

1.1 Utilities for lists

```
fun foldr1 :: ('a ⇒ 'a ⇒ 'a) ⇒ 'a list ⇒ 'a where
  foldr1 f [x] = x
| foldr1 f (x # xs) = f x (foldr1 f xs)
| foldr1 f [] = undefined f
```

```
abbreviation lset where lset ≡ List.set
```

```
lemma rev-induct2 [consumes 1, case-names Nil snoc]:
```

```
  assumes length xs = length ys
```

```
  and P [] []
```

```
  and  $\bigwedge x xs y ys. \text{length } xs = \text{length } ys \implies P \ xs \ ys \implies P \ (xs @ [x]) \ (ys @ [y])$ 
```

```
  shows P xs ys
```

```
using assms proof (induction xs arbitrary: ys rule: rev-induct)
```

```
  case (snoc x xs)
```

```
  then show ?case by (cases ys rule: rev-cases) simp-all
```

```
qed simp
```

1.2 Utilities for finite maps

```
no-syntax
```

```
-fmaplet :: ['a, 'a] ⇒ fmaplet (- /$$:=/ -)
```

```
-fmaplets :: ['a, 'a] ⇒ fmaplet (- /[$$:=]/ -)
```

```
syntax
```

```
-fmaplet :: ['a, 'a] ⇒ fmaplet (- /↦/ -)
```

```
-fmaplets :: ['a, 'a] ⇒ fmaplet (- /[↦]/ -)
```

```
lemma fmdom'-fmap-of-list [simp]:
```

```
  shows fmdom' (fmap-of-list ps) = lset (map fst ps)
```

```
  by (induction ps) force+
```

```
lemma fmran'-singleton [simp]:
```

```
  shows fmran' {k ↦ v} = {v}
```

```
proof -
```

```
  have  $v' \in \text{fmran}' \{k \mapsto v\} \implies v' = v$  for  $v'$ 
```

```
  proof -
```

```
    assume  $v' \in \text{fmran}' \{k \mapsto v\}$ 
```

```
    fix  $k'$ 
```

```
    have  $\text{fmdom}' \{k \mapsto v\} = \{k\}$ 
```

```
      by simp
```

```
    then show  $v' = v$ 
```

```

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

```

```

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

```

```

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

```

```

lemma finite-fmran':
  shows finite ( $\text{fmran}' m$ )
  by (simp add: fmran'-alt-def)

```

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

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

lemma *fmap-of-zipped-list-fmran'* [*simp*]:
assumes *distinct* ($\text{map fst } ps$)
shows $\text{fmran}' (\text{fmap-of-list } ps) = \text{lset } (\text{map snd } ps)$
using *assms* **proof** (*induction ps*)
case *Nil*
then show *?case*
by *auto*
next
case (*Cons p ps*)
then show *?case*
proof ($\text{cases } p \in \text{lset } ps$)
case *True*
then show *?thesis*
using *Cons.prem*s **by** *auto*
next
case *False*
obtain k **and** v **where** $p = (k, v)$
by *fastforce*
with *Cons.prem*s **have** $k \notin \text{fmdom}' (\text{fmap-of-list } ps)$
by *auto*
then have $\text{fmap-of-list } (p \# ps) = \{k \mapsto v\} ++_f \text{fmap-of-list } ps$
using $\langle p = (k, v) \rangle$ **and** *fmap-singleton-comm* **by** *fastforce*
with *Cons.prem*s **have** $\text{fmran}' (\text{fmap-of-list } (p \# ps)) = \{v\} \cup \text{fmran}' (\text{fmap-of-list } ps)$
by (*simp add:* $\langle p = (k, v) \rangle$)
then have $\text{fmran}' (\text{fmap-of-list } (p \# ps)) = \{v\} \cup \text{lset } (\text{map snd } ps)$
using *Cons.IH* **and** *Cons.prem*s **by** *force*
then show *?thesis*
by (*simp add:* $\langle p = (k, v) \rangle$)
qed
qed

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

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

lemma $fmap\text{-of-list-nth-split}$ [$simp$]:

assumes $distinct\ xs$

and $j < length\ xs$

and $length\ ys = length\ xs$ **and** $length\ zs = length\ xs$

shows $fmap\text{-of-list } (zip\ xs\ (take\ k\ ys\ @\ drop\ k\ zs)) \ \$\$ (xs ! j) =$

$(if\ j < k\ then\ Some\ (take\ k\ ys ! j)\ else\ Some\ (drop\ k\ zs ! (j - k)))$

using $assms$ **proof** $(induction\ k\ arbitrary: xs\ ys\ zs\ j)$

case 0

then show $?case$

by $(simp\ add: fmap\text{-of-list.rep-eq}\ map\text{-of-zip-nth})$

next

case $(Suc\ k)$

then show $?case$

proof $(cases\ xs)$

case Nil

with $Suc.prem\ 2$ **show** $?thesis$

by $auto$

next

case $(Cons\ x\ xs')$

let $?ps = zip\ xs\ (take\ (Suc\ k)\ ys\ @\ drop\ (Suc\ k)\ zs)$

from $Cons$ **and** $Suc.prem\ 3,4$ **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** $*$: $fmap\text{-of-list } ?ps = fmap\text{-of-list } ((x, y) \# ?ps')$

using $\langle ys = y \# ys' \rangle$ **and** $\langle zs = z \# zs' \rangle$ **by** $fastforce$

also have $\dots = \{x \mapsto y\} ++_f fmap\text{-of-list } ?ps'$

proof $-$

from $\langle ys = y \# ys' \rangle$ **and** $\langle zs = z \# zs' \rangle$ **have** $fmap\text{-of-list } ?ps' \ \$\$ x = None$

using $Cons$ **and** $Suc.prem\ 1,3,4$ **by** $(simp\ add: fmdom'\text{-notD})$

then show $?thesis$

using $fmap\text{-singleton-comm}$ **by** $fastforce$

qed

finally have $fmap\text{-of-list } ?ps = \{x \mapsto y\} ++_f fmap\text{-of-list } ?ps'$.

then show $?thesis$

proof $(cases\ j = 0)$

case $True$

with $\langle ys = y \# ys' \rangle$ **and** $Cons$ **show** $?thesis$

by $simp$

next

case $False$

then have $xs ! j = xs' ! (j - 1)$

by $(simp\ add: Cons)$

moreover from $\langle ys = y \# ys' \rangle$ **and** $\langle zs = z \# zs' \rangle$ **have** $fmdom' (fmap\text{-of-list } ?ps') = lset\ xs'$

using $Cons$ **and** $Suc.prem\ 3,4$ **by** $force$

moreover from $False$ **and** $Suc.prem\ 2$ **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.prem(1) by auto
  with Suc.IH and Suc.prem(1,3,4) and Cons have **: fmap-of-list ?ps $$ (xs ! j) =
    (if j - 1 < k then Some (take k ys' ! (j - 1)) else Some (drop k zs' ! ((j - 1) - k)))
    using ⟨j - 1 < length xs'⟩ and ⟨ys = y # ys'⟩ and ⟨zs = z # zs'⟩ by simp
  then show ?thesis
  proof (cases j - 1 < k)
    case True
      with False and ** show ?thesis
        using ⟨ys = y # ys'⟩ by auto
    next
      case False
        from Suc.prem(1) and Cons and ⟨j - 1 < length xs'⟩ and ⟨xs ! j = xs' ! (j - 1)⟩ have j >
0
          using nth-non-equal-first-eq by fastforce
        with False have j ≥ Suc k
          by simp
        moreover have fmap-of-list ?ps $$ (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.prem 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  $m' \$\$ k = \text{Some } v$ 
  by force
from False have  $\{k \mapsto v\} ++_f \text{fmdrop } k (m'(k' \mapsto v')) = \{k \mapsto v\} ++_f (\text{fmdrop } k m')(k' \mapsto v')$ 
  by (simp add: fmdrop-fmupd)
also have  $\dots = (\{k \mapsto v\} ++_f \text{fmdrop } k m')(k' \mapsto v')$ 
  by fastforce
also from fmupd.prems and fmupd.IH[OF  $\langle m' \$\$ k = \text{Some } v \rangle$ ] have  $\dots = m'(k' \mapsto v')$ 
  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  $g = \text{apsnd } f$ , 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  $\rightarrow$  101)

```

```

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

```

```

primrec subtypes :: type  $\Rightarrow$  type set where
  subtypes i = {}
| subtypes o = {}
| subtypes ( $\alpha \rightarrow \beta$ ) =  $\{\alpha, \beta\} \cup \text{subtypes } \alpha \cup \text{subtypes } \beta$ 

```

```

lemma subtype-size-decrease:
assumes  $\alpha \in \text{subtypes } \beta$ 

```

shows *type-size* $\alpha < \text{type-size } \beta$
using *assms* **by** (*induction rule: type.induct*) *force+*

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

lemma *fun-type-atoms-in-subtypes*:
assumes $k < \text{length } ts$
shows $ts ! k \in \text{subtypes } (\text{foldr } (\rightarrow) ts \gamma)$
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 $k < \text{length } ts$
shows $ts ! k \neq \text{foldr } (\rightarrow) ts \gamma$
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* \times *type*

abbreviation *var-name* :: *var* \Rightarrow *nat* **where**
var-name \equiv *fst*

abbreviation *var-type* :: *var* \Rightarrow *type* **where**
var-type \equiv *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** $x \notin \text{var-name } vs$
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** $\text{length } ns' = n$ **and** *distinct ns'* **and** $\text{lset } ns' \cap ns = \{\}$
using *assms* **proof** (*induction n arbitrary: thesis*)
case *0*
then show *?case*
by *simp*
next

```

case (Suc n)
from assms obtain ns' where  $\text{length } ns' = n$  and  $\text{distinct } ns'$  and  $\text{lset } ns' \cap ns = \{\}$ 
  using Suc.IH by blast
moreover from assms obtain n' where  $n' \notin \text{lset } ns' \cup ns$ 
  using ex-new-if-finite[OF infinite-UNIV-nat] by blast
ultimately
  have  $\text{length } (n' \# ns') = \text{Suc } n$  and  $\text{distinct } (n' \# ns')$  and  $\text{lset } (n' \# ns') \cap ns = \{\}$ 
  by simp-all
with Suc.prem1 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  $\text{length } vs' = \text{length } xs$ 
and  $\text{distinct } vs'$ 
and  $\text{var-name } \text{'lset } vs' \cap (ns \cup \text{var-name } \text{'lset } xs) = \{\}$ 
and  $\text{map var-type } vs' = \text{map var-type } xs$ 
proof –
from assms1 have finite  $(ns \cup \text{var-name } \text{'lset } xs)$ 
  by blast
then obtain ns'
  where  $\text{length } ns' = \text{length } xs$ 
  and  $\text{distinct } ns'$ 
  and  $\text{lset } ns' \cap (ns \cup \text{var-name } \text{'lset } xs) = \{\}$ 
  by (rule fresh-var-name-list-existence)
define vs'' where  $vs'' = \text{zip } ns' (\text{map var-type } xs)$ 
from vs''-def and  $\langle \text{length } ns' = \text{length } xs \rangle$  have  $\text{length } vs'' = \text{length } xs$ 
  by simp
moreover from vs''-def and  $\langle \text{distinct } ns' \rangle$  have  $\text{distinct } vs''$ 
  by (simp add: distinct-zip1)
moreover have  $\text{var-name } \text{'lset } vs'' \cap (ns \cup \text{var-name } \text{'lset } xs) = \{\}$ 
  unfolding vs''-def
  using  $\langle \text{length } ns' = \text{length } xs \rangle$  and  $\langle \text{lset } ns' \cap (ns \cup \text{var-name } \text{'lset } xs) = \{\} \rangle$ 
  by (metis length-map set-map map-fst-zip)
moreover from vs''-def have  $\text{map var-type } vs'' = \text{map var-type } xs$ 
  by (simp add: length ns' = length xs)
ultimately show ?thesis
  by (fact that)
qed

```

2.3 Constants

type-synonym *con* = $\text{nat} \times \text{type}$

2.4 Formulas

```

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

```

syntax

```

-FVar :: nat ⇒ type ⇒ form (- [899, 0] 900)
-FCon :: nat ⇒ type ⇒ form (|-| [899, 0] 900)
-FAbs :: nat ⇒ type ⇒ form ⇒ form ((λλ-./ -) [0, 0, 104] 104)

```

translations

```

xα ⇒ CONST FVar (x, α)
{|c|}α ⇒ CONST FCon (c, α)
λxα. A ⇒ CONST FAbs (x, α) A

```

2.5 Generalized operators

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

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

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

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

fun *form-size* :: form ⇒ nat **where**

```

form-size (xα) = 1
| form-size ({|c|}α) = 1
| form-size (A • B) = Suc (form-size A + form-size B)
| form-size (λxα. A) = Suc (form-size A)

```

fun *form-depth* :: form ⇒ nat **where**

```

form-depth (xα) = 0
| form-depth ({|c|}α) = 0
| form-depth (A • B) = Suc (max (form-depth A) (form-depth B))
| form-depth (λxα. A) = Suc (form-depth A)

```

2.6 Subformulas

fun *subforms* :: form ⇒ form set **where**

```

subforms (xα) = {}
| subforms ({|c|}α) = {}
| subforms (A • B) = {A, B}
| subforms (λxα. A) = {A}

```

datatype *direction* = Left («) | Right (»)

type-synonym *position* = direction list

fun *positions* :: *form* \Rightarrow *position set* **where**
 | *positions* (x_α) = $\{\square\}$
 | *positions* ($\{\!\{c\}\!\}_\alpha$) = $\{\square\}$
 | *positions* ($A \cdot B$) = $\{\square\} \cup \{\ll \# p \mid p. p \in \text{positions } A\} \cup \{\gg \# p \mid p. p \in \text{positions } B\}$
 | *positions* ($\lambda x_\alpha. A$) = $\{\square\} \cup \{\ll \# p \mid p. p \in \text{positions } A\}$

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

fun *subform-at* :: *form* \Rightarrow *position* \rightarrow *form* **where**
 | *subform-at* A \square = *Some* A
 | *subform-at* ($A \cdot B$) ($\ll \# p$) = *subform-at* A p
 | *subform-at* ($A \cdot B$) ($\gg \# p$) = *subform-at* B p
 | *subform-at* ($\lambda x_\alpha. A$) ($\ll \# p$) = *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 \square A' = ($A = A'$)
 | *is-subform-at* C ($\ll \# p$) ($A \cdot B$) = *is-subform-at* C p A
 | *is-subform-at* C ($\gg \# p$) ($A \cdot B$) = *is-subform-at* C p B
 | *is-subform-at* C ($\ll \# p$) ($\lambda x_\alpha. A$) = *is-subform-at* C p A
 | *is-subform-at* - - - = *False*

lemma *is-subform-at-alt-def*:
shows $A' \preceq_p A = (\text{case } \text{subform-at } A \text{ } p \text{ of } \text{Some } B \Rightarrow B = A' \mid \text{None} \Rightarrow \text{False})$
by (*induction* $A' \text{ } p \text{ } A$ *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* $B \text{ } p \text{ } C$ *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* $\{\!\{c\}\!\}_\alpha \text{ } p \text{ } C$ *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* $x_\alpha \text{ } p \text{ } C$ *rule*: *is-subform-at.induct*) *auto*

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

lemma *subform-at-subforms-abs*:

assumes $\lambda x_\alpha. A \preceq_p C$

shows $A \preceq_p @ [\llbracket \cdot \rrbracket] C$

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 $p \in \text{positions } A$

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* $A < \text{form-size } B$

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 $A \preceq_p B \cdot C$

shows

$(A = B \cdot C \wedge p = []) \vee$

$(A \neq B \cdot C \wedge$

$(\exists p' \in \text{positions } B. p = \llbracket \# p' \wedge A \preceq_{p'} B) \vee (\exists p' \in \text{positions } C. p = \rrbracket \# p' \wedge A \preceq_{p'} C))$

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 $(A = \lambda x_\alpha. B \wedge p = []) \vee (A \neq \lambda x_\alpha. B \wedge (\exists p' \in \text{positions } B. p = \llbracket \# p' \wedge A \preceq_{p'} B))$

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 $B \preceq_{\text{replicate}} (\text{length } As) \ll \cdot^{\mathcal{Q}} \star B As$
by (*induction* As *arbitrary*: B) (*simp-all*, *metis replicate-append-same subform-at-subforms-app(1)*)

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

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

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

lemma *is-subform-at-existence*:
assumes $p \in \text{positions } A$
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 $C \preceq_{\text{drop } (\text{length } p')} p B$

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  $\langle A' \preceq_{p'} A \rangle$  and assms(2) have  $A' = B$ 
    by (rule is-subform-at-uniqueness)
  with  $\langle C \preceq_{[d]} A' \rangle$  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  $\langle \textit{strict-prefix } p' p'' \rangle$  by blast
with  $\langle C \preceq_{[d]} A' \rangle$  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  $C < \textit{form-size } B$ 
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* \Rightarrow *form* \Rightarrow *bool* (*infix* \preceq 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  $p \in \text{positions } A \longleftrightarrow (\exists A'. A' \preceq_p A)$ 
  by (meson is-subform-at-existence is-subform-implies-in-positions)

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

```

2.7 Free and bound variables

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

overloading

$vars\text{-}form \equiv vars :: form \Rightarrow var\ set$

$vars\text{-}form\text{-}set \equiv vars :: form\ set \Rightarrow var\ set$

begin

fun $vars\text{-}form :: form \Rightarrow var\ set$ **where**

$vars\text{-}form\ (x_\alpha) = \{(x, \alpha)\}$

| $vars\text{-}form\ (\{c\}_\alpha) = \{\}$

| $vars\text{-}form\ (A \cdot B) = vars\text{-}form\ A \cup vars\text{-}form\ B$

| $vars\text{-}form\ (\lambda x_\alpha. A) = vars\text{-}form\ A \cup \{(x, \alpha)\}$

fun $vars\text{-}form\text{-}set :: form\ set \Rightarrow var\ set$ **where**

$vars\text{-}form\text{-}set\ S = (\bigcup A \in S. vars\ A)$

end

abbreviation $var\text{-}names :: 'a \Rightarrow nat\ set$ **where**

$var\text{-}names\ \mathcal{X} \equiv var\text{-}name\ ' (\vars\ \mathcal{X})$

lemma $vars\text{-}form\text{-}finiteness$:

fixes $A :: form$

shows $finite\ (vars\ A)$

by (*induction rule: vars-form.induct*) *simp-all*

lemma $vars\text{-}form\text{-}set\text{-}finiteness$:

fixes $S :: form\ set$

assumes $finite\ S$

shows $finite\ (vars\ S)$

using *assms* **unfolding** $vars\text{-}form\text{-}set.\text{simps}$ **using** $vars\text{-}form\text{-}finiteness$ **by** *blast*

lemma $form\text{-}var\text{-}names\text{-}finiteness$:

fixes $A :: form$

shows $finite\ (var\text{-}names\ A)$

using $vars\text{-}form\text{-}finiteness$ **by** *blast*

lemma $form\text{-}set\text{-}var\text{-}names\text{-}finiteness$:

fixes $S :: form\ set$

assumes $finite\ S$

shows $finite\ (var\text{-}names\ S)$

using *assms* **and** $vars\text{-}form\text{-}set\text{-}finiteness$ **by** *blast*

consts $free\text{-}vars :: 'a \Rightarrow var\ set$

overloading

$free\text{-}vars\text{-}form \equiv free\text{-}vars :: form \Rightarrow var\ set$

$free\text{-}vars\text{-}form\text{-}set \equiv free\text{-}vars :: form\ set \Rightarrow var\ set$

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 ( $A \cdot B$ ) = 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 =  $(\bigcup A \in S. \text{free-vars } A)$ 
```

end

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

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

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

```
lemma free-vars-of-generalized-abs:  
  shows free-vars ( $\lambda^{\mathcal{Q}}_* vs A$ ) = 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  $\cdot$  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  $\cdot$  B)  
    using vars-form.simps(3)[symmetric] .  
  finally show ?case
```

```

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

```

```

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

```

```

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

```

```

fun bound-vars where
  bound-vars (x $\alpha$ ) = {}
| bound-vars ( $\{c\}$  $\alpha$ ) = {}
| bound-vars (B  $\cdot$  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  $\cdot$  B) ( $\ll$  # p) = binders-at A p
| binders-at (A  $\cdot$  B) ( $\gg$  # p) = binders-at B p
| binders-at ( $\lambda$ x $\alpha$ . A) ( $\ll$  # 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 ( $\ll$  # p) (A  $\cdot$  B)  $\longleftrightarrow$  occurs-at v p A
  and occurs-at v ( $\gg$  # p) (A  $\cdot$  B)  $\longleftrightarrow$  occurs-at v p B
  and occurs-at v ( $\ll$  # p) ( $\lambda$ x $\alpha$ . A)  $\longleftrightarrow$  occurs-at v p A

```

and *occurs-at* v ($d \# p$) ($FVar\ v'$) $\longleftrightarrow False$
and *occurs-at* v ($\gg \# p$) ($\lambda x_\alpha. A$) $\longleftrightarrow False$
and *occurs-at* v \square ($A \bullet B$) $\longleftrightarrow False$
and *occurs-at* v \square ($\lambda x_\alpha. A$) $\longleftrightarrow False$
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 (
 $p \neq \square \wedge$
 (
 $\exists p' \in lset\ (strict-prefixes\ p).$
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
 $p \neq \square \wedge (\exists p' \in positions\ B. \exists C. strict-prefix\ p'\ p \wedge FAbs\ v\ C \preceq_{p'} B)$

proof
assume *in-scope-of-abs* $v\ p\ B$
then show $p \neq \square \wedge (\exists p' \in positions\ B. \exists C. strict-prefix\ p'\ p \wedge FAbs\ v\ C \preceq_{p'} B)$
by (*induction rule: subform-at.induct*) *force* $+$
next
assume $p \neq \square \wedge (\exists p' \in positions\ B. \exists C. strict-prefix\ p'\ p \wedge FAbs\ v\ C \preceq_{p'} B)$
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 ($\ll \# p$) ($A \bullet B$) $\longleftrightarrow in-scope-of-abs\ v\ p\ A$
by *force*

lemma *in-scope-of-abs-in-right-app*:
shows *in-scope-of-abs* v ($\gg \# p$) ($A \bullet B$) $\longleftrightarrow in-scope-of-abs\ v\ p\ B$

by force

lemma *in-scope-of-abs-in-app*:

assumes *in-scope-of-abs v p (A • B)*

obtains p' **where** $(p = \ll \# p' \wedge \text{in-scope-of-abs } v \ p' \ A) \vee (p = \gg \# p' \wedge \text{in-scope-of-abs } v \ p' \ B)$

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** $\langle p = d \# p' \rangle$ **show** *?thesis*

using *that* **and** *in-scope-of-abs-in-left-app* **by** *simp*

next

case *Right*

with *assms* **and** $\langle p = d \# p' \rangle$ **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

$\forall p'$.

$(p = \ll \# p' \longrightarrow \neg \text{in-scope-of-abs } v' \ p' \ A)$

\wedge

$(p = \gg \# p' \longrightarrow \neg \text{in-scope-of-abs } v' \ p' \ B)$

shows $\neg \text{in-scope-of-abs } v' \ p \ (A \bullet B)$

using *assms* **and** *in-scope-of-abs-in-app* **by** *metis*

lemma *in-scope-of-abs-in-abs*:

shows $\text{in-scope-of-abs } v \ (\ll \# p) \ (F\text{Abs } v' \ B) \longleftrightarrow v = v' \vee \text{in-scope-of-abs } v \ p \ B$

proof

assume $\text{in-scope-of-abs } v \ (\ll \# p) \ (F\text{Abs } v' \ B)$

then obtain p' **and** C

where $p' \in \text{positions } (F\text{Abs } v' \ B)$

and *strict-prefix* $p' \ (\ll \# p)$

and $F\text{Abs } v \ C \preceq_{p'} F\text{Abs } v' \ B$

unfolding *in-scope-of-abs-alt-def* **by** *blast*

then show $v = v' \vee \text{in-scope-of-abs } v \ p \ B$

proof (*cases p'*)

case *Nil*

with $\langle F\text{Abs } v \ C \preceq_{p'} F\text{Abs } v' \ B \rangle$ **have** $v = v'$

by *auto*

then show *?thesis*

by *simp*

next

case (*Cons d p''*)

with $\langle \text{strict-prefix } p' \ (\ll \# p) \rangle$ **have** $d = \ll$

by *simp*

from $\langle FAbs\ v\ C\ \preceq_p',\ FAbs\ v'\ B \rangle$ **and** *Cons* **have** $p'' \in \text{positions } B$
by
 (cases (FAbs v C, p', FAbs v' B) rule: is-subform-at.cases)
 (simp-all add: is-subform-implies-in-positions)
moreover from $\langle FAbs\ v\ C\ \preceq_p',\ FAbs\ v'\ B \rangle$ **and** *Cons* **and** $\langle d = \langle \rangle \rangle$ **have** $FAbs\ v\ C\ \preceq_{p''}\ B$
by *(metis is-subform-at.simps(4) old.prod.exhaust)*
moreover from $\langle \text{strict-prefix } p'\ (\langle \# \ p) \rangle$ **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 $v = v' \vee \text{in-scope-of-abs } v\ p\ B$
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 $\neg \text{in-scope-of-abs } v\ p\ (FVar\ v')$
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 $v \in \text{vars } A$
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.prem*s **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.prem*s **and** $\langle p = d \# p' \rangle$ **have** *in-scope-of-abs v p' B*
using *in-scope-of-abs-in-left-app* **by** *blast*
then have $v \in \text{vars } B$
by *(fact FApp.IH(1))*
then show *?thesis*
by *simp*


```

next
  case Right
  with FApp.prems and  $\langle p = d \# p' \rangle$  have in-scope-of-abs  $v \ p' \ C$ 
  using in-scope-of-abs-in-right-app by blast
  then have  $v \in \text{vars } C$ 
  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  $\langle p = d \# p' \rangle$  have in-scope-of-abs  $v \ p' \ B$ 
    using in-scope-of-abs-in-abs by blast
    then have  $v \in \text{vars } B$ 
    by (fact FAbs.IH)
    then show ?thesis
    using surj-pair[of  $v'$ ] by force
  next
  case Right
  with FAbs.prems and  $\langle p = d \# p' \rangle$  and False show ?thesis
  by (cases rule: is-subform-at.cases) auto
qed
qed
qed

```

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

definition *is-bound-at* :: $\text{var} \Rightarrow \text{position} \Rightarrow \text{form} \Rightarrow \text{bool}$ **where**
 [*iff*]: *is-bound-at* $v \ p \ B \iff \text{occurs-at } v \ p \ B \wedge \text{in-scope-of-abs } v \ p \ B$

lemma *not-is-bound-at-in-var*:
 shows $\neg \text{is-bound-at } v \ p \ (\text{FVar } v')$
 by (*fastforce elim: is-subform-at.elims*(2))

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

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

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

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

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

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

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

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

lemma *is-bound-at-in-bound-vars*:

assumes $p \in \text{positions } A$
and *is-bound-at* $v p A \vee v \in \text{binders-at } A p$
shows $v \in \text{bound-vars } A$
using *assms* **proof** (*induction* A *arbitrary*: p)
case (*FApp* $B C$)
from *FApp.prem*s(2) **consider** (a) *is-bound-at* $v p (B \cdot C) \mid (b)$ $v \in \text{binders-at } (B \cdot C) p$
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 $\langle p \in \text{positions } (B \cdot C) \rangle$
consider (a_1) $p = \ll \# p'$ **and** $p' \in \text{positions } B \mid (a_2)$ $p = \gg \# p'$ **and** $p' \in \text{positions } C$
by *force*
then show *?thesis*
proof *cases*
case a_1
from $a_1(1)$ **and** $\langle \text{is-bound-at } v p (B \cdot C) \rangle$ **have** *is-bound-at* $v p' B$
using *is-bound-at-in-left-app* **by** *blast*
with $a_1(2)$ **have** $v \in \text{bound-vars } B$
using *FApp.IH*(1) **by** *blast*
then show *?thesis*
by *simp*
next
case a_2
from $a_2(1)$ **and** $\langle \text{is-bound-at } v p (B \cdot C) \rangle$ **have** *is-bound-at* $v p' C$
using *is-bound-at-in-right-app* **by** *blast*
with $a_2(2)$ **have** $v \in \text{bound-vars } C$
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 ∈ positions (B • C)⟩
consider (b1) p = « # p' and p' ∈ positions B | (b2) p = » # p' and p' ∈ positions C
  by force
then show ?thesis
proof cases
  case b1
  from b1(1) and ⟨v ∈ binders-at (B • C) p⟩ have v ∈ binders-at B p'
  by force
  with b1(2) have v ∈ bound-vars B
  using FApp.IH(1) by blast
  then show ?thesis
  by simp
next
  case b2
  from b2(1) and ⟨v ∈ binders-at (B • C) p⟩ have v ∈ binders-at C p'
  by force
  with b2(2) have v ∈ bound-vars C
  using FApp.IH(2) by blast
  then show ?thesis
  by simp
qed
qed
next
case (FAbs v' B)
from FAbs.prem(2) consider (a) is-bound-at v p (FAbs v' B) | (b) v ∈ binders-at (FAbs v' B) p
  by blast
then show ?case
proof cases
  case a
  then have p ≠ []
  using occurs-at-alt-def(9) by force
  with ⟨p ∈ positions (FAbs v' B)⟩ obtain p' where p = « # p' and p' ∈ positions B
  by (cases FAbs v' B rule: positions.cases) fastforce+
  from ⟨p = « # p'⟩ and ⟨is-bound-at v p (FAbs v' B)⟩ have v = v' ∨ is-bound-at v p' B
  using is-bound-at-from-abs by blast
  then consider (a1) v = v' | (a2) is-bound-at v p' B
  by blast
  then show ?thesis
proof cases
  case a1
  then show ?thesis
  using surj-pair[of v'] by fastforce
next
  case a2
  then have v ∈ bound-vars B
  using ⟨p' ∈ positions B⟩ and FAbs.IH by blast
  then show ?thesis
  using surj-pair[of v'] by fastforce

```

```

qed
next
case b
then have  $p \neq []$ 
  by force
with  $F\text{Abs.prem}(1)$  obtain  $p'$  where  $p = \ll \# p' \text{ and } p' \in \text{positions } B$ 
  by (cases  $F\text{Abs } v' B$  rule:  $\text{positions.cases}$ ) fastforce+
with  $b$  consider  $(b_1) v = v' \mid (b_2) v \in \text{binders-at } B p'$ 
  by (cases  $F\text{Abs } v' B$  rule:  $\text{positions.cases}$ ) fastforce+
then show ?thesis
proof cases
case  $b_1$ 
then show ?thesis
  using  $\text{surj-pair}[of v']$  by fastforce
next
case  $b_2$ 
then have  $v \in \text{bound-vars } B$ 
  using  $\langle p' \in \text{positions } B \rangle$  and  $F\text{Abs.IH}$  by blast
then show ?thesis
  using  $\text{surj-pair}[of v']$  by fastforce
qed
qed
qed fastforce+

```

lemma *bound-vars-in-is-bound-at*:

```

assumes  $v \in \text{bound-vars } A$ 
obtains  $p$  where  $p \in \text{positions } A$  and  $\text{is-bound-at } v p A \vee v \in \text{binders-at } A p$ 
using  $\text{assms}$  proof (induction  $A$  arbitrary:  $\text{thesis}$  rule:  $\text{bound-vars.induct}$ )
case  $(\exists B C)$ 
from  $\langle v \in \text{bound-vars } (B \cdot C) \rangle$  consider  $(a) v \in \text{bound-vars } B \mid (b) v \in \text{bound-vars } C$ 
  by fastforce
then show ?case
proof cases
case  $a$ 
with  $\exists.\text{IH}(1)$  obtain  $p$  where  $p \in \text{positions } B$  and  $\text{is-bound-at } v p B \vee v \in \text{binders-at } B p$ 
  by blast
from  $\langle p \in \text{positions } B \rangle$  have  $\ll \# p \in \text{positions } (B \cdot C)$ 
  by simp
from  $\langle \text{is-bound-at } v p B \vee v \in \text{binders-at } B p \rangle$ 
consider  $(a_1) \text{is-bound-at } v p B \mid (a_2) v \in \text{binders-at } B p$ 
  by blast
then show ?thesis
proof cases
case  $a_1$ 
then have  $\text{is-bound-at } v (\ll \# p) (B \cdot C)$ 
  using  $\text{is-bound-at-in-left-app}$  by blast
then show ?thesis
  using  $\exists.\text{prems}(1)$  and  $\text{is-subform-implies-in-positions}$  by blast
next

```

```

    case a2
    then have  $v \in \text{binders-at } (B \cdot C) (\ll \# p)$ 
      by simp
    then show ?thesis
      using 3.prems(1) and  $\langle \ll \# p \in \text{positions } (B \cdot C) \rangle$  by blast
  qed
next
case b
with 3.IH(2) obtain  $p$  where  $p \in \text{positions } C$  and  $\text{is-bound-at } v \ p \ C \vee v \in \text{binders-at } C \ p$ 
  by blast
from  $\langle p \in \text{positions } C \rangle$  have  $\gg \# p \in \text{positions } (B \cdot C)$ 
  by simp
from  $\langle \text{is-bound-at } v \ p \ C \vee v \in \text{binders-at } C \ p \rangle$ 
consider  $(b_1) \text{ is-bound-at } v \ p \ C \mid (b_2) v \in \text{binders-at } C \ p$ 
  by blast
then show ?thesis
proof cases
  case b1
  then have  $\text{is-bound-at } v \ (\gg \# p) (B \cdot C)$ 
    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 b2
  then have  $v \in \text{binders-at } (B \cdot C) (\gg \# p)$ 
    by simp
  then show ?thesis
    using 3.prems(1) and  $\langle \gg \# p \in \text{positions } (B \cdot C) \rangle$  by blast
  qed
qed
next
case (4  $x \ \alpha \ B$ )
from  $\langle v \in \text{bound-vars } (\lambda x_\alpha. B) \rangle$  consider  $(a) v = (x, \alpha) \mid (b) v \in \text{bound-vars } B$ 
  by force
then show ?case
proof cases
  case a
  then have  $v \in \text{binders-at } (\lambda x_\alpha. B) [\ll]$ 
    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  $p \in \text{positions } B$  and  $\text{is-bound-at } v \ p \ B \vee v \in \text{binders-at } B \ p$ 
    by blast
  from  $\langle p \in \text{positions } B \rangle$  have  $\ll \# p \in \text{positions } (\lambda x_\alpha. B)$ 
    by simp
  from  $\langle \text{is-bound-at } v \ p \ B \vee v \in \text{binders-at } B \ p \rangle$ 
  consider  $(b_1) \text{ is-bound-at } v \ p \ B \mid (b_2) v \in \text{binders-at } B \ p$ 

```

```

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

```

lemma *bound-vars-alt-def*:

```

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

```

definition *is-free-at* :: var ⇒ position ⇒ form ⇒ bool **where**

[*iff*]: *is-free-at* v p B ⟷ occurs-at v p B ∧ ¬ in-scope-of-abs v p B

lemma *is-free-at-in-var*:

```

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

```

lemma *not-is-free-at-in-con*:

```

shows ¬ is-free-at v [] (λc}α)
by simp

```

lemma *is-free-at-in-left-app*:

```

shows is-free-at v (« # p) (B • C) ⟷ is-free-at v p B
by auto

```

lemma *is-free-at-in-right-app*:

```

shows is-free-at v (» # p) (B • C) ⟷ is-free-at v p C
by auto

```

lemma *is-free-at-from-app*:

assumes *is-free-at* v p (B • C)

obtains p' **where** (p = « # p' ∧ *is-free-at* v p' B) ∨ (p = » # p' ∧ *is-free-at* v p' C)

proof –

from *assms* obtain d and p' **where** p = d # p'

using *subforms-from-app* by blast

then show ?thesis

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

lemma *is-free-at-from-abs*:
 assumes *is-free-at* v ($\ll \# p$) (*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 $p = \ll \# p'$ and *is-free-at* v $p' B$
proof –
 obtain x and α where $v' = (x, \alpha)$
 by *fastforce*
 with *assms* obtain p' where $p = \ll \# p'$
 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 ($\ll \# p$) (*FAbs* $v' B$)
unfolding *is-free-at-def* **proof**
 from *assms*(1) **show** *occurs-at* v ($\ll \# p$) (*FAbs* $v' B$)
 using *surj-pair*[*of v'*] by *fastforce*
 from *assms* **show** \neg *in-scope-of-abs* v ($\ll \# p$) (*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 \langle *is-free-at* v p ($B \cdot C$) \rangle **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 $\langle p \in$ *positions* ($B \cdot C$) \rangle
 consider (a) $p = \ll \# p'$ and $p' \in$ *positions* B | (b) $p = \gg \# p'$ 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 $\langle is\text{-free-at } v \ p \ A \rangle$ **have** $is\text{-free-at } v \ (\ll \# \ p) \ (A \cdot B)$
using $is\text{-free-at-in-left-app}$ **by** $blast$
ultimately show $?thesis$
using $\mathcal{P}.prems(1)$ **by** $presburger$
next
case b
with $\mathcal{P}.IH(2)$ **obtain** p **where** $p \in positions \ B$ **and** $is\text{-free-at } v \ p \ B$
by $blast$
from $\langle p \in positions \ B \rangle$ **have** $\gg \# \ p \in positions \ (A \cdot B)$
by $simp$
moreover from $\langle is\text{-free-at } v \ p \ B \rangle$ **have** $is\text{-free-at } v \ (\gg \# \ p) \ (A \cdot B)$
using $is\text{-free-at-in-right-app}$ **by** $blast$
ultimately show $?thesis$
using $\mathcal{P}.prems(1)$ **by** $presburger$
qed
next
case $(\lambda \ x \ \alpha \ A)$
from $\langle v \in free\text{-vars} \ (\lambda x_{\alpha}. A) \rangle$ **have** $v \in free\text{-vars} \ A - \{(x, \alpha)\}$ **and** $v \neq (x, \alpha)$
by $simp\text{-all}$
then have $v \in free\text{-vars} \ A$
by $blast$
with $\mathcal{P}.IH$ **obtain** p **where** $p \in positions \ A$ **and** $is\text{-free-at } v \ p \ A$
by $blast$
from $\langle p \in positions \ A \rangle$ **have** $\ll \# \ p \in positions \ (\lambda x_{\alpha}. A)$
by $simp$
moreover from $\langle is\text{-free-at } v \ p \ A \rangle$ **and** $\langle v \neq (x, \alpha) \rangle$ **have** $is\text{-free-at } v \ (\ll \# \ p) \ (\lambda x_{\alpha}. A)$
using $is\text{-free-at-to-abs}$ **by** $blast$
ultimately show $?case$
using $\mathcal{P}.prems(1)$ **by** $presburger$
qed $simp\text{-all}$

lemma $free\text{-vars-alt-def}$:

shows $free\text{-vars} \ A = \{v \mid v \ p. \ p \in positions \ A \wedge is\text{-free-at } v \ p \ A\}$
using $free\text{-vars-in-is-free-at}$ **and** $is\text{-free-at-in-free-vars}$
by $(intro \ subset\text{-antisym} \ subsetI \ CollectI, \ metis) \ blast$

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

definition $is\text{-bound} :: var \Rightarrow form \Rightarrow bool$ **where**

$[iff]: is\text{-bound } v \ B \longleftrightarrow (\exists p \in positions \ B. is\text{-bound-at } v \ p \ B \vee v \in binders\text{-at } B \ p)$

lemma $is\text{-bound-in-app-homomorphism}$:

shows $is\text{-bound } v \ (A \cdot B) \longleftrightarrow is\text{-bound } v \ A \vee is\text{-bound } v \ B$

proof

assume $is\text{-bound } v \ (A \cdot B)$

then obtain p **where** $p \in positions \ (A \cdot B)$ **and** $is\text{-bound-at } v \ p \ (A \cdot B) \vee v \in binders\text{-at} \ (A \cdot B) \ p$
by $auto$

then have $p \neq []$

by $fastforce$

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

```

qed
qed
next
assume is-bound  $v$   $A \vee$  is-bound  $v$   $B$ 
then show is-bound  $v$   $(A \cdot B)$ 
proof (rule disjE)
  assume is-bound  $v$   $A$ 
  then obtain  $p$  where  $p \in$  positions  $A$  and is-bound-at  $v$   $p$   $A \vee v \in$  binders-at  $A$   $p$ 
    by auto
  from  $\langle p \in$  positions  $A \rangle$  have  $\langle \# p \in$  positions  $(A \cdot B) \rangle$ 
    by auto
  from  $\langle$  is-bound-at  $v$   $p$   $A \vee v \in$  binders-at  $A$   $p \rangle$ 
  consider (a) is-bound-at  $v$   $p$   $A$  | (b)  $v \in$  binders-at  $A$   $p$ 
    by blast
  then show is-bound  $v$   $(A \cdot B)$ 
  proof cases
    case a
    then have occurs-at  $v$   $(\langle \# p \rangle)$   $(A \cdot B)$ 
      by auto
    moreover from a have is-bound-at  $v$   $(\langle \# p \rangle)$   $(A \cdot B)$ 
      by force
    ultimately show is-bound  $v$   $(A \cdot B)$ 
      using  $\langle \langle \# p \in$  positions  $(A \cdot B) \rangle \rangle$  by blast
  next
  case b
  then have  $v \in$  binders-at  $(A \cdot B)$   $(\langle \# p \rangle)$ 
    by auto
  then show is-bound  $v$   $(A \cdot B)$ 
    using  $\langle \langle \# p \in$  positions  $(A \cdot B) \rangle \rangle$  by blast
qed
next
assume is-bound  $v$   $B$ 
then obtain  $p$  where  $p \in$  positions  $B$  and is-bound-at  $v$   $p$   $B \vee v \in$  binders-at  $B$   $p$ 
  by auto
from  $\langle p \in$  positions  $B \rangle$  have  $\langle \# p \in$  positions  $(A \cdot B) \rangle$ 
  by auto
from  $\langle$  is-bound-at  $v$   $p$   $B \vee v \in$  binders-at  $B$   $p \rangle$ 
consider (a) is-bound-at  $v$   $p$   $B$  | (b)  $v \in$  binders-at  $B$   $p$ 
  by blast
then show is-bound  $v$   $(A \cdot B)$ 
proof cases
  case a
  then have occurs-at  $v$   $(\langle \# p \rangle)$   $(A \cdot B)$ 
    by auto
  moreover from a have is-bound-at  $v$   $(\langle \# p \rangle)$   $(A \cdot B)$ 
    by force
  ultimately show is-bound  $v$   $(A \cdot B)$ 
    using  $\langle \langle \# p \in$  positions  $(A \cdot B) \rangle \rangle$  by blast
  next

```

```

    case b
  then have  $v \in \text{binders-at } (A \cdot B) (\gg \# p)$ 
    by auto
  then show  $\text{is-bound } v (A \cdot B)$ 
    using  $\langle \gg \# p \in \text{positions } (A \cdot B) \rangle$  by blast
qed
qed
qed

```

lemma *is-bound-in-abs-body*:

```

  assumes  $\text{is-bound } v A$ 
  shows  $\text{is-bound } v (\lambda x_\alpha. A)$ 
using assms unfolding is-bound-def proof
  fix  $p$ 
  assume  $p \in \text{positions } A$  and  $\text{is-bound-at } v p A \vee v \in \text{binders-at } A p$ 
  moreover from  $\langle p \in \text{positions } A \rangle$  have  $\langle \# p \in \text{positions } (\lambda x_\alpha. A) \rangle$ 
    by simp
  ultimately consider (a)  $\text{is-bound-at } v p A$  | (b)  $v \in \text{binders-at } A p$ 
    by blast
  then show  $\exists p \in \text{positions } (\lambda x_\alpha. A). \text{is-bound-at } v p (\lambda x_\alpha. A) \vee v \in \text{binders-at } (\lambda x_\alpha. A) p$ 
proof cases
  case a
  then have  $\text{is-bound-at } v (\langle \# p \rangle (\lambda x_\alpha. A))$ 
    by force
  with  $\langle \langle \# p \in \text{positions } (\lambda x_\alpha. A) \rangle$  show ?thesis
    by blast
  next
  case b
  then have  $v \in \text{binders-at } (\lambda x_\alpha. A) (\langle \# p \rangle)$ 
    by simp
  with  $\langle \langle \# p \in \text{positions } (\lambda x_\alpha. A) \rangle$  show ?thesis
    by blast
qed
qed

```

lemma *absent-var-is-not-bound*:

```

  assumes  $v \notin \text{vars } A$ 
  shows  $\neg \text{is-bound } v A$ 
using assms and binders-at-alt-def and in-scope-of-abs-in-vars by blast

```

lemma *bound-vars-alt-def2*:

```

  shows  $\text{bound-vars } A = \{v \in \text{vars } A. \text{is-bound } v A\}$ 
unfolding bound-vars-alt-def using absent-var-is-not-bound by fastforce

```

definition *is-free* :: $\text{var} \Rightarrow \text{form} \Rightarrow \text{bool}$ **where**

```

  [iff]:  $\text{is-free } v B \iff (\exists p \in \text{positions } B. \text{is-free-at } v p B)$ 

```

2.9 Free variables for a formula in another formula

definition *is-free-for* :: *form* \Rightarrow *var* \Rightarrow *form* \Rightarrow *bool* **where**

[*iff*]: *is-free-for* *A v B* \longleftrightarrow
 (
 $\forall v' \in \text{free-vars } A.$
 $\forall p \in \text{positions } B.$
 is-free-at *v p B* $\longrightarrow \neg \text{in-scope-of-abs } v' p B$
)

lemma *is-free-for-absent-var* [*intro*]:

assumes $v \notin \text{vars } B$

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 α)*

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 $v' \in \text{free-vars } A$

then have $\forall p \in \text{positions } B. \text{is-free-at } v p B \longrightarrow \neg \text{in-scope-of-abs } v' p B$

proof (*intro ballI impI*)

fix *p*

assume $v' \in \text{free-vars } A$ **and** $p \in \text{positions } B$ **and** *is-free-at* *v p B*

from $\langle p \in \text{positions } B \rangle$ **have** $\langle \# p \in \text{positions } (B \bullet C) \rangle$

by *simp*

moreover from $\langle \text{is-free-at } v p B \rangle$ **have** *is-free-at* *v (c # p) (B • C)*

using *is-free-at-in-left-app* **by** *blast*

ultimately have $\neg \text{in-scope-of-abs } v' (\langle \# p \rangle (B \bullet C))$

using *assms* **and** $\langle v' \in \text{free-vars } A \rangle$ **by** *blast*

then show $\neg \text{in-scope-of-abs } v' p B$

by *simp*

qed

}

then show *is-free-for* *A v B*

by *force*

next

{

fix *v'*

assume $v' \in \text{free-vars } A$

```

then have  $\forall p \in \text{positions } C. \text{is-free-at } v \ p \ C \longrightarrow \neg \text{in-scope-of-abs } v' \ p \ C$ 
proof (intro ballI impI)
  fix  $p$ 
  assume  $v' \in \text{free-vars } A$  and  $p \in \text{positions } C$  and  $\text{is-free-at } v \ p \ C$ 
  from  $\langle p \in \text{positions } C \rangle$  have  $\gg \# p \in \text{positions } (B \cdot C)$ 
  by simp
  moreover from  $\langle \text{is-free-at } v \ p \ C \rangle$  have  $\text{is-free-at } v \ (\gg \# p) (B \cdot C)$ 
  using is-free-at-in-right-app by blast
  ultimately have  $\neg \text{in-scope-of-abs } v' \ (\gg \# p) (B \cdot C)$ 
  using assms and  $\langle v' \in \text{free-vars } A \rangle$  by blast
  then show  $\neg \text{in-scope-of-abs } v' \ p \ C$ 
  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  $A \ v \ (B \cdot C)$ 
unfolding is-free-for-def proof (intro ballI impI)
  fix  $v'$  and  $p$ 
  assume  $v' \in \text{free-vars } A$  and  $p \in \text{positions } (B \cdot C)$  and  $\text{is-free-at } v \ p \ (B \cdot C)$ 
  from  $\langle \text{is-free-at } v \ p \ (B \cdot C) \rangle$  have  $p \neq []$ 
  using occurs-at-alt-def(8) by force
  then obtain  $d$  and  $p'$  where  $p = d \# p'$ 
  by (meson list.exhaust)
  with  $\langle p \in \text{positions } (B \cdot C) \rangle$ 
  consider (b)  $p = \ll \# p'$  and  $p' \in \text{positions } B \mid$  (c)  $p = \gg \# p'$  and  $p' \in \text{positions } C$ 
  by force
  then show  $\neg \text{in-scope-of-abs } v' \ p \ (B \cdot C)$ 
proof cases
  case b
  from b(1) and  $\langle \text{is-free-at } v \ p \ (B \cdot C) \rangle$  have  $\text{is-free-at } v \ p' \ B$ 
  using is-free-at-in-left-app by blast
  with assms(1) and  $\langle v' \in \text{free-vars } A \rangle$  and  $\langle p' \in \text{positions } B \rangle$  have  $\neg \text{in-scope-of-abs } v' \ p' \ B$ 
  by simp
  with b(1) show ?thesis
  using in-scope-of-abs-in-left-app by simp
next
  case c
  from c(1) and  $\langle \text{is-free-at } v \ p \ (B \cdot C) \rangle$  have  $\text{is-free-at } v \ p' \ C$ 
  using is-free-at-in-right-app by blast
  with assms(2) and  $\langle v' \in \text{free-vars } A \rangle$  and  $\langle p' \in \text{positions } C \rangle$  have  $\neg \text{in-scope-of-abs } v' \ p' \ C$ 
  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 $is\text{-free-for } A \ v \ (B \cdot C) \longleftrightarrow is\text{-free-for } A \ v \ B \wedge is\text{-free-for } A \ v \ C$
using *is-free-for-from-app* **and** *is-free-for-to-app* **by** *iprover*

lemma *is-free-for-to-abs* [*intro*]:

assumes $is\text{-free-for } A \ v \ B$ **and** $(x, \alpha) \notin free\text{-vars } A$
shows $is\text{-free-for } A \ v \ (\lambda x_\alpha. B)$

unfolding *is-free-for-def* **proof** (*intro ballI impI*)

fix v' **and** p

assume $v' \in free\text{-vars } A$ **and** $p \in positions \ (\lambda x_\alpha. B)$ **and** $is\text{-free-at } v \ p \ (\lambda x_\alpha. B)$

from $\langle is\text{-free-at } v \ p \ (\lambda x_\alpha. B) \rangle$ **have** $p \neq []$

using *occurs-at-alt-def(9)* **by** *force*

with $\langle p \in positions \ (\lambda x_\alpha. B) \rangle$ **obtain** p' **where** $p = \llcorner \# \ p'$ **and** $p' \in positions \ B$
by *force*

then show $\neg in\text{-scope-of-abs } v' \ p \ (\lambda x_\alpha. B)$

proof –

from $\langle p = \llcorner \# \ p' \rangle$ **and** $\langle is\text{-free-at } v \ p \ (\lambda x_\alpha. B) \rangle$ **have** $is\text{-free-at } v \ p' \ B$

using *is-free-at-from-abs* **by** *blast*

with *assms(1)* **and** $\langle v' \in free\text{-vars } A \rangle$ **and** $\langle p' \in positions \ B \rangle$ **have** $\neg in\text{-scope-of-abs } v' \ p' \ B$
by *force*

moreover from $\langle v' \in free\text{-vars } A \rangle$ **and** *assms(2)* **have** $v' \neq (x, \alpha)$

by *blast*

ultimately show *?thesis*

using $\langle p = \llcorner \# \ p' \rangle$ **and** *in-scope-of-abs-in-abs* **by** *auto*

qed

qed

lemma *is-free-for-from-abs*:

assumes $is\text{-free-for } A \ v \ (\lambda x_\alpha. B)$ **and** $v \neq (x, \alpha)$

shows $is\text{-free-for } A \ v \ B$

unfolding *is-free-for-def* **proof** (*intro ballI impI*)

fix v' **and** p

assume $v' \in free\text{-vars } A$ **and** $p \in positions \ B$ **and** $is\text{-free-at } v \ p \ B$

then show $\neg in\text{-scope-of-abs } v' \ p \ B$

proof –

from $\langle is\text{-free-at } v \ p \ B \rangle$ **and** *assms(2)* **have** $is\text{-free-at } v \ (\llcorner \# \ p) \ (\lambda x_\alpha. B)$

by (*rule is-free-at-to-abs*)

moreover from $\langle p \in positions \ B \rangle$ **have** $\llcorner \# \ p \in positions \ (\lambda x_\alpha. B)$

by *simp*

ultimately have $\neg in\text{-scope-of-abs } v' \ (\llcorner \# \ p) \ (\lambda x_\alpha. B)$

using *assms* **and** $\langle v' \in free\text{-vars } A \rangle$ **by** *blast*

then show *?thesis*

using *in-scope-of-abs-in-abs* **by** *blast*

qed

qed

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

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

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

lemma $is\text{-free-for-alt-def}$:
shows
 $is\text{-free-for } A \ v \ B$
 \longleftrightarrow
 $($
 $\# p.$
 $($
 $p \in positions \ B \wedge is\text{-free-at } v \ p \ B \wedge p \neq [] \wedge$
 $(\exists v' \in free\text{-vars } A. \exists p' \ C. strict\text{-prefix } p' \ p \wedge FAbs \ v' \ C \preceq_{p'} B)$
 $)$
 $)$
unfolding $is\text{-free-for-def}$
using $in\text{-scope-of-abs-alt-def}$ **and** $is\text{-subform-implies-in-positions}$
by $meson$

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

by contradiction
qed

lemma *absent-var-is-free-for* [intro]:
assumes $x \notin \text{vars } A$
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 $x \notin \text{vars } A$
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** $v' \in \text{free-vars } A$ **and** $v \in \text{free-vars } B$
shows $\neg \text{is-free-for } A$ v (FAbs v' B)

proof –
from *assms*(3) **obtain** p **where** $p \in \text{positions } B$ **and** *is-free-at* v p B
using *free-vars-in-is-free-at* **by** *blast*
then have $\langle \# p \in \text{positions } (FAbs v' B) \text{ and } \text{is-free-at } v (\langle \# p) (FAbs v' B) \rangle$
using *surj-pair*[of v'] **and** *is-free-at-to-abs*[OF $\langle \text{is-free-at } v p B \rangle$ *assms*(1)] **by** *force+*
moreover have *in-scope-of-abs* v' ($\langle \# p \rangle$) (FAbs v' B)
using *in-scope-of-abs-in-abs* **by** *blast*
ultimately show *?thesis*
using *assms*(2) **by** *blast*

qed

2.10 Replacement of subformulas

inductive

is-replacement-at :: *form* \Rightarrow *position* \Rightarrow *form* \Rightarrow *form* \Rightarrow *bool*
((λ - \langle - \leftarrow - \rangle \triangleright -) [1000, 0, 0, 0] 900)

where

pos-found: $A \langle p \leftarrow C \rangle \triangleright C'$ **if** $p = []$ **and** $C = C'$
| *replace-left-app*: $(G \cdot H) \langle \langle \# p \leftarrow C \rangle \triangleright (G' \cdot H) \text{ if } p \in \text{positions } G \text{ and } G \langle p \leftarrow C \rangle \triangleright G'$
| *replace-right-app*: $(G \cdot H) \langle \rangle \langle \# p \leftarrow C \rangle \triangleright (G \cdot H')$ **if** $p \in \text{positions } H$ **and** $H \langle p \leftarrow C \rangle \triangleright H'$
| *replace-abs*: $(\lambda x_\gamma. E) \langle \langle \# p \leftarrow C \rangle \triangleright (\lambda x_\gamma. E')$ **if** $p \in \text{positions } E$ **and** $E \langle p \leftarrow C \rangle \triangleright E'$

lemma *is-replacement-at-implies-in-positions*:

assumes $C \langle p \leftarrow A \rangle \triangleright D$
shows $p \in \text{positions } C$
using *assms* **by** (*induction rule: is-replacement-at.induct*) *auto*

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

lemma *is-replacement-at-existence*:

assumes $p \in \text{positions } C$
obtains D **where** $C \langle p \leftarrow A \rangle \triangleright D$
using *assms* **proof** (*induction* C *arbitrary: p thesis*)

```

case (FApp C1 C2)
from FApp.prem(2) consider
  (a) p = []
  | (b) ∃ p'. p = « # p' ∧ p' ∈ positions C1
  | (c) ∃ p'. p = » # p' ∧ p' ∈ positions C2
  by fastforce
then show ?case
proof cases
  case a
  with FApp.prem(1) show ?thesis
  by blast
next
  case b
  with FApp.prem(1) show ?thesis
  using FApp.IH(1) and replace-left-app by meson
next
  case c
  with FApp.prem(1) show ?thesis
  using FApp.IH(2) and replace-right-app by meson
qed
next
case (FAbs v C')
from FAbs.prem(2) consider (a) p = [] | (b) ∃ p'. p = « # p' ∧ p' ∈ positions C'
  using surj-pair[of v] by fastforce
then show ?case
proof cases
  case a
  with FAbs.prem(1) show ?thesis
  by blast
next
  case b
  with FAbs.prem(1,2) show ?thesis
  using FAbs.IH and surj-pair[of v] by blast
qed
qed force+

```

lemma *is-replacement-at-minimal-change*:

assumes $C \langle p \leftarrow A \rangle \triangleright D$

shows $A \preceq_p D$

and $\forall p' \in \text{positions } D. \neg \text{prefix } p' p \wedge \neg \text{prefix } p p' \longrightarrow \text{subform-at } D p' = \text{subform-at } C p'$

using *assms* **by** (induction rule: *is-replacement-at.induct*) *auto*

lemma *is-replacement-at-binders*:

assumes $C \langle p \leftarrow A \rangle \triangleright D$

shows *binders-at* $D p = \text{binders-at } C p$

using *assms* **by** (induction rule: *is-replacement-at.induct*) *simp-all*

lemma *is-replacement-at-occurs*:

assumes $C \langle p \leftarrow A \rangle \triangleright D$

```

and  $\neg$  prefix  $p'$   $p$  and  $\neg$  prefix  $p$   $p'$ 
shows occurs-at  $v$   $p'$   $C \iff$  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  $C \langle p \leftarrow FVar\ v \rangle \triangleright G$ 
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  $v$   $p'$   $C \iff$  occurs-at  $v$   $p'$   $G$  if  $\neg$  prefix  $p'$   $p$  and  $\neg$  prefix  $p$   $p'$ 
    using assms(2) and that and is-replacement-at-occurs by blast
  ultimately show False
  proof (cases  $\neg$  prefix  $p'$   $p \wedge \neg$  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  $\langle p' \neq p \rangle$  and False and loop-subform-impossibility
  by (blast dest: prefix-order.antisym-conv2)
qed
qed

```

lemma *is-replacement-at-new-positions*:

```

  assumes  $C \langle p \leftarrow A \rangle \triangleright D$  and prefix p p' and  $p' \in \text{positions } D$ 
  obtains  $p''$  where  $p' = p @ p''$  and  $p'' \in \text{positions } A$ 
  using assms by (induction arbitrary: thesis p' rule: is-replacement-at.induct, auto) blast+

```

lemma *replacement-override*:

```

  assumes  $C \langle p \leftarrow B \rangle \triangleright D$  and  $C \langle p \leftarrow A \rangle \triangleright F$ 
  shows  $D \langle p \leftarrow A \rangle \triangleright F$ 
  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  $p \in \text{positions } G'$ 
  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  $G \langle p \leftarrow A \rangle \triangleright F'$ 
  by (fastforce elim: is-replacement-at.cases)
  from  $\langle G \langle p \leftarrow A \rangle \triangleright F' \rangle$  have  $G' \langle p \leftarrow A \rangle \triangleright F'$ 
  by (fact replace-left-app.IH)
  with  $\langle p \in \text{positions } G' \rangle$  show ?case
  unfolding  $\langle F = F' \cdot H \rangle$  by blast

```

next

```

  case (replace-right-app p H C H' G)
  have  $p \in \text{positions } H'$ 
  by
  (
    fact is-subform-implies-in-positions
    [OF is-replacement-at-minimal-change(1)[OF replace-right-app.hyps(2)]]
  )

```

)
from *replace-right-app.prem*s **obtain** F' **where** $F = G \cdot F'$ **and** $H\langle p \leftarrow A \rangle \triangleright F'$
by (*fastforce elim: is-replacement-at.cases*)
from $\langle H\langle p \leftarrow A \rangle \triangleright F' \rangle$ **have** $H'\langle p \leftarrow A \rangle \triangleright F'$
by (*fact replace-right-app.IH*)
with $\langle p \in \text{positions } H' \rangle$ **show** *?case*
unfolding $\langle F = G \cdot F' \rangle$ **by** *blast*
next
case (*replace-abs p E C E' x γ*)
have $p \in \text{positions } E'$
by
(*fact is-subform-implies-in-positions*
[*OF is-replacement-at-minimal-change(1)*][*OF replace-abs.hyps(2)*]
)
from *replace-abs.prem*s **obtain** F' **where** $F = \lambda x \gamma. F'$ **and** $E\langle p \leftarrow A \rangle \triangleright F'$
by (*fastforce elim: is-replacement-at.cases*)
from $\langle E\langle p \leftarrow A \rangle \triangleright F' \rangle$ **have** $E'\langle p \leftarrow A \rangle \triangleright F'$
by (*fact replace-abs.IH*)
with $\langle p \in \text{positions } E' \rangle$ **show** *?case*
unfolding $\langle F = \lambda x \gamma. F' \rangle$ **by** *blast*
qed

lemma *leftmost-subform-in-generalized-app-replacement:*
shows $(\cdot^{\mathcal{Q}}_{\star} C As)\langle \text{replicate } (\text{length } As) \ll \leftarrow D \rangle \triangleright (\cdot^{\mathcal{Q}}_{\star} D As)$
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{r} **where** $\mathfrak{r} \equiv 0$
abbreviation (*input*) \mathfrak{h} **where** $\mathfrak{h} \equiv \text{Suc } \mathfrak{r}$
abbreviation (*input*) \mathfrak{z} **where** $\mathfrak{z} \equiv \text{Suc } \mathfrak{h}$
abbreviation (*input*) \mathfrak{f} **where** $\mathfrak{f} \equiv \text{Suc } \mathfrak{z}$
abbreviation (*input*) \mathfrak{g} **where** $\mathfrak{g} \equiv \text{Suc } \mathfrak{f}$
abbreviation (*input*) \mathfrak{h} **where** $\mathfrak{h} \equiv \text{Suc } \mathfrak{g}$
abbreviation (*input*) \mathfrak{c} **where** $\mathfrak{c} \equiv \text{Suc } \mathfrak{h}$
abbreviation (*input*) \mathfrak{c}_Q **where** $\mathfrak{c}_Q \equiv \text{Suc } \mathfrak{c}$
abbreviation (*input*) \mathfrak{c}_i **where** $\mathfrak{c}_i \equiv \text{Suc } \mathfrak{c}_Q$

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

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

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

definition *iota* :: form (ι) **where**

[simp]: $\iota = FCon\ iota\text{-constant}$

definition *is-Q-constant-of-type* :: con \Rightarrow type \Rightarrow bool **where**

[iff]: *is-Q-constant-of-type* $p\ \alpha \longleftrightarrow p = Q\text{-constant-of-type}\ \alpha$

definition *is-iota-constant* :: con \Rightarrow bool **where**

[iff]: *is-iota-constant* $p \longleftrightarrow p = iota\text{-constant}$

definition *is-logical-constant* :: con \Rightarrow bool **where**

[iff]: *is-logical-constant* $p \longleftrightarrow (\exists \beta. is\text{-}Q\text{-constant-of-type}\ p\ \beta) \vee is\text{-}iota\text{-constant}\ p$

definition *type-of-Q-constant* :: con \Rightarrow type **where**

[simp]: *type-of-Q-constant* $p = (THE\ \alpha. is\text{-}Q\text{-constant-of-type}\ p\ \alpha)$

lemma *constant-cases*[case-names non-logical Q-constant ι -constant, cases type: con]:

assumes $\neg is\text{-}logical\text{-constant}\ p \Longrightarrow P$

and $\bigwedge \beta. is\text{-}Q\text{-constant-of-type}\ p\ \beta \Longrightarrow P$

and *is-iota-constant* $p \Longrightarrow P$

shows P

using *assms* by *blast*

2.12 Definitions and abbreviations

definition *equality-of-type* :: form \Rightarrow type \Rightarrow form \Rightarrow form ((- =./ -) [103, 0, 103] 102) **where**

[simp]: $A =_{\alpha} B = Q_{\alpha} \cdot A \cdot B$

definition *equivalence* :: form \Rightarrow form \Rightarrow form (infixl $\equiv^{\mathcal{Q}}$ 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]: $T_o = Q_o =_{o \rightarrow o \rightarrow o} Q_o$

definition *false* :: form (F_o) **where**

[simp]: $F_o = \lambda \mathfrak{r}_o. T_o =_{o \rightarrow o} \lambda \mathfrak{r}_o. \mathfrak{r}_o$

definition *PI* :: type \Rightarrow form (\prod -) **where**

[simp]: $\prod \alpha = Q_{\alpha \rightarrow o} \cdot (\lambda \mathfrak{r}_\alpha. T_o)$

definition *forall* :: nat \Rightarrow type \Rightarrow form \Rightarrow form ((\forall -./ -) [0, 0, 141] 141) **where**

[simp]: $\forall x_{\alpha}. A = \prod \alpha \cdot (\lambda x_{\alpha}. A)$

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

definition *generalized-forall* :: var list \Rightarrow form \Rightarrow form ($\forall^{\mathcal{Q}}_{\star}$ - - [141, 141] 141) **where**

[simp]: $\forall^{\mathcal{Q}}_{\star} vs\ A = foldr\ (\lambda(x, \alpha) B. \forall x_{\alpha}. B)\ vs\ A$

lemma *innermost-subform-in-generalized-forall*:

assumes $vs \neq []$

shows $A \preceq_{foldr} (\lambda p. [»,«] @ p)\ vs\ []\ \forall^{\mathcal{Q}}_{\star} vs\ A$

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

lemma *innermost-replacement-in-generalized-forall*:

assumes $vs \neq []$

shows $(\forall^{\mathcal{Q}_*} vs C) \langle \text{foldr } (\lambda-. (@) [»,«]) vs [] \leftarrow B \rangle \triangleright (\forall^{\mathcal{Q}_*} vs B)$

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 $\langle v = (x, \alpha) \rangle$ **show** *?thesis*

unfolding *True* **by** *force*

next

case *False*

then have $\text{foldr } (\lambda-. (@) [»,«]) vs [] \in \text{positions } (\forall^{\mathcal{Q}_*} vs C)$

using *innermost-subform-in-generalized-forall* **and** *is-subform-implies-in-positions* **by** *blast*

moreover from *False* **have** $(\forall^{\mathcal{Q}_*} vs C) \langle \text{foldr } (\lambda-. (@) [»,«]) vs [] \leftarrow B \rangle \triangleright (\forall^{\mathcal{Q}_*} vs B)$

by (*fact Cons.IH*)

ultimately have $(\lambda x_\alpha. \forall^{\mathcal{Q}_*} vs C) \langle \ll \# \text{foldr } (\lambda-. (@) [»,«]) vs [] \leftarrow B \rangle \triangleright (\lambda x_\alpha. \forall^{\mathcal{Q}_*} vs B)$

by (*rule replace-abs*)

moreover have $\ll \# \text{foldr } (\lambda-. (@) [»,«]) vs [] \in \text{positions } (\lambda x_\alpha. \forall^{\mathcal{Q}_*} vs C)$

using $\langle \text{foldr } (\lambda-. (@) [»,«]) vs [] \in \text{positions } (\forall^{\mathcal{Q}_*} vs C) \rangle$ **by** *simp*

ultimately have

$(\prod \alpha \cdot (\lambda x_\alpha. \forall^{\mathcal{Q}_*} vs C)) \langle \ll \# \ll \# \text{foldr } (\lambda-. (@) [»,«]) vs [] \leftarrow B \rangle \triangleright (\prod \alpha \cdot (\lambda x_\alpha. \forall^{\mathcal{Q}_*} vs B))$

by *blast*

then have $(\forall x_\alpha. \forall^{\mathcal{Q}_*} vs C) \langle [»,«] @ \text{foldr } (\lambda-. (@) [»,«]) vs [] \leftarrow B \rangle \triangleright (\forall x_\alpha. \forall^{\mathcal{Q}_*} vs B)$

by *simp*

then show *?thesis*

unfolding $\langle v = (x, \alpha) \rangle$ **and** *generalized-forall-def* **and** *foldr.simps(2)* **and** *o-apply*

and *case-prod-conv* .

qed

qed

lemma *false-is-forall*:

shows $F_o = \forall \mathfrak{r}_o. \mathfrak{r}_o$

unfolding *false-def* **and** *forall-def* **and** *PI-def* **and** *equality-of-type-def* ..

definition *conj-fun* :: *form* $(\wedge_{o \rightarrow o \rightarrow o})$ **where**

[*simp*]: $\wedge_{o \rightarrow o \rightarrow o} =$

$\lambda \mathfrak{r}_o. \lambda \mathfrak{h}_o.$

(

$(\lambda \mathfrak{g}_{o \rightarrow o \rightarrow o}. \mathfrak{g}_{o \rightarrow o \rightarrow o} \cdot T_o \cdot T_o) =_{(o \rightarrow o \rightarrow o) \rightarrow o} (\lambda \mathfrak{g}_{o \rightarrow o \rightarrow o}. \mathfrak{g}_{o \rightarrow o \rightarrow o} \cdot \mathfrak{r}_o \cdot \mathfrak{h}_o)$

)

definition *conj-op* :: form \Rightarrow form \Rightarrow form (**infixl** $\wedge^{\mathcal{Q}}$ 131) **where**

[simp]: $A \wedge^{\mathcal{Q}} B = \wedge_{o \rightarrow o \rightarrow o} \cdot A \cdot B$

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

definition *generalized-conj-op* :: form list \Rightarrow form ($\wedge^{\mathcal{Q}}_*$ - [0] 131) **where**

[simp]: $\wedge^{\mathcal{Q}}_* As = \text{foldr1 } (\wedge^{\mathcal{Q}}) As$

definition *imp-fun* :: form ($\supset_{o \rightarrow o \rightarrow o}$) **where** $- \equiv$ used instead of $=$, see [2]

[simp]: $\supset_{o \rightarrow o \rightarrow o} = \lambda \mathfrak{r}_o. \lambda \eta_o. (\mathfrak{r}_o \equiv^{\mathcal{Q}} \mathfrak{r}_o \wedge^{\mathcal{Q}} \eta_o)$

definition *imp-op* :: form \Rightarrow form \Rightarrow form (**infixl** $\supset^{\mathcal{Q}}$ 111) **where**

[simp]: $A \supset^{\mathcal{Q}} B = \supset_{o \rightarrow o \rightarrow o} \cdot A \cdot B$

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

definition *generalized-imp-op* :: form list \Rightarrow form \Rightarrow form (**infixl** $\supset^{\mathcal{Q}}_*$ 111) **where**

[simp]: $As \supset^{\mathcal{Q}}_* B = \text{foldr } (\supset^{\mathcal{Q}}) As B$

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 \Rightarrow form ($\sim^{\mathcal{Q}}$ - [141] 141) **where**

[simp]: $\sim^{\mathcal{Q}} A = Q_o \cdot F_o \cdot A$

definition *disj-fun* :: form ($\vee_{o \rightarrow o \rightarrow o}$) **where**

[simp]: $\vee_{o \rightarrow o \rightarrow o} = \lambda \mathfrak{r}_o. \lambda \eta_o. \sim^{\mathcal{Q}} (\sim^{\mathcal{Q}} \mathfrak{r}_o \wedge^{\mathcal{Q}} \sim^{\mathcal{Q}} \eta_o)$

definition *disj-op* :: form \Rightarrow form \Rightarrow form (**infixl** $\vee^{\mathcal{Q}}$ 126) **where**

[simp]: $A \vee^{\mathcal{Q}} B = \vee_{o \rightarrow o \rightarrow o} \cdot A \cdot B$

definition *exists* :: nat \Rightarrow type \Rightarrow form \Rightarrow form ($(\exists - ./ -)$ [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* $(\exists x_\alpha. A) = \text{free-vars } A - \{(x, \alpha)\}$

by *simp*

definition *inequality-of-type* :: form \Rightarrow type \Rightarrow form \Rightarrow form ($(- \neq_\alpha -)$ [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 \Rightarrow form \Rightarrow 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* $(\alpha \rightarrow \beta)$ A **and** *is-wff-of-type* α B

| *abs-is-wff*: *is-wff-of-type* $(\alpha \rightarrow \beta)$ $(\lambda x_\alpha. A)$ **if** *is-wff-of-type* β A

definition *wffs-of-type* :: type \Rightarrow form set (*wffs*- [0]) **where**

$wffs_\alpha = \{f :: \text{form. } \textit{is-wff-of-type } \alpha f\}$

abbreviation $wffs$:: form set where

$$wffs \equiv \bigcup \alpha. wffs_\alpha$$

lemma *is-wff-of-type-wffs-of-type-eq* [pred-set-conv]:

shows *is-wff-of-type* $\alpha = (\lambda f. f \in wffs_\alpha)$

unfolding *wffs-of-type-def* **by** *simp*

lemmas *wffs-of-type-intros* [intro!] = *is-wff-of-type.intros*[to-set]

lemmas *wffs-of-type-induct* [consumes 1, induct set: *wffs-of-type*] = *is-wff-of-type.induct*[to-set]

lemmas *wffs-of-type-cases* [consumes 1, cases set: *wffs-of-type*] = *is-wff-of-type.cases*[to-set]

lemmas *wffs-of-type-simps* = *is-wff-of-type.simps*[to-set]

lemma *generalized-app-wff* [intro]:

assumes $length\ As = length\ ts$

and $\forall k < length\ As. As\ !\ k \in wffs_{ts\ !\ k}$

and $B \in wffs_{foldr\ (\rightarrow)\ ts\ \beta}$

shows $\cdot^{\mathcal{Q}_*} B\ As \in wffs_\beta$

using *assms* **proof** (*induction* *As* *ts* *arbitrary*; *B* *rule*: *list-induct2*)

case *Nil*

then show *?case*

by *simp*

next

case (*Cons* *A* *As* *t* *ts*)

from *Cons.prem1* **have** $A \in wffs_t$

by *fastforce*

moreover from *Cons.prem2* **have** $B \in wffs_{t \rightarrow foldr\ (\rightarrow)\ ts\ \beta}$

by *auto*

ultimately have $B \cdot A \in wffs_{foldr\ (\rightarrow)\ ts\ \beta}$

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.prem1* **have** $\forall k < length\ As. As\ !\ k \in wffs_{ts\ !\ k}$

by *fastforce*

ultimately have $\cdot^{\mathcal{Q}_*} (B \cdot A)\ As \in wffs_\beta$

using *Cons.IH* **by** (*simp* *only*:)

moreover have $\cdot^{\mathcal{Q}_*} B\ (A\ \#\ As) = \cdot^{\mathcal{Q}_*} (B \cdot A)\ As$

by *simp*

ultimately show *?case*

by (*simp* *only*:)

qed

lemma *generalized-abs-wff* [intro]:

assumes $B \in wffs_\beta$

shows $\lambda^{\mathcal{Q}_*} vs\ B \in wffs_{foldr\ (\rightarrow)\ (map\ snd\ vs)\ \beta}$

using *assms* **proof** (*induction* *vs*)

case *Nil*

then show *?case*

by *simp*
next
 case $(Cons\ v\ vs)$
 let $?δ = foldr\ (\rightarrow)\ (map\ snd\ vs)\ β$
obtain x **and** $α$ **where** $v = (x, α)$
 by *fastforce*
then have $FVar\ v \in wffs_α$
 by *auto*
from $Cons.prem\ s$ **have** $λ^{Q_*}\ vs\ B \in wffs_{?δ}$
 by $(fact\ Cons.IH)$
with $\langle v = (x, α) \rangle$ **have** $FAbs\ v\ (λ^{Q_*}\ vs\ B) \in wffs_{α \rightarrow ?δ}$
 by *blast*
moreover from $\langle v = (x, α) \rangle$ **have** $foldr\ (\rightarrow)\ (map\ snd\ (v\ \# \ vs))\ β = α \rightarrow ?δ$
 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 $Q\text{-wff}$ [*intro*]:
shows $Q_α \in wffs_{α \rightarrow α \rightarrow o}$
by *auto*

lemma $iota\text{-wff}$ [*intro*]:
shows $ι \in wffs_{(i \rightarrow o) \rightarrow i}$
by *auto*

lemma $equality\text{-wff}$ [*intro*]:
assumes $A \in wffs_α$ **and** $B \in wffs_α$
shows $A =_α B \in wffs_o$
using *assms* **by** *auto*

lemma $equivalence\text{-wff}$ [*intro*]:
assumes $A \in wffs_o$ **and** $B \in wffs_o$
shows $A \equiv^Q B \in wffs_o$
using *assms* **unfolding** $equivalence\text{-def}$ **by** *blast*

lemma $true\text{-wff}$ [*intro*]:
shows $T_o \in wffs_o$
by *force*

lemma $false\text{-wff}$ [*intro*]:
shows $F_o \in wffs_o$
by *auto*

lemma $pi\text{-wff}$ [*intro*]:
shows $\prod α \in wffs_{(α \rightarrow o) \rightarrow o}$
using $PI\text{-def}$ **by** *fastforce*

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

lemma *generalized-forall-wff* [intro]:
assumes $B \in \text{wffs}_o$
shows $\forall^Q_* vs B \in \text{wffs}_o$
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 $\wedge_{o \rightarrow o \rightarrow o} \in \text{wffs}_{o \rightarrow o \rightarrow o}$
by *auto*

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

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

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

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

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

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

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

using *assms* by *fastforce*

lemma *inequality-wff* [*intro*]:
assumes $A \in \text{wffs}_\alpha$ and $B \in \text{wffs}_\alpha$
shows $A \neq_\alpha B \in \text{wffs}_0$
using *assms* by *fastforce*

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

lemma *wffs-from-generalized-app*:
assumes $\cdot^Q_* B \text{ As} \in \text{wffs}_\beta$
obtains *ts*
where $\text{length } ts = \text{length } As$
and $\forall k < \text{length } As. As ! k \in \text{wffs}_{ts ! k}$
and $B \in \text{wffs}_{\text{foldr } (\rightarrow) ts \beta}$
using *assms* **proof** (*induction As arbitrary: B thesis*)
case *Nil*
then show *?case*
by *simp*
next
case (*Cons A As*)
from *Cons.prem*s have $\cdot^Q_* (B \cdot A) \text{ As} \in \text{wffs}_\beta$
by *auto*
then obtain *ts*
where $\text{length } ts = \text{length } As$
and $\forall k < \text{length } As. As ! k \in \text{wffs}_{ts ! k}$
and $B \cdot A \in \text{wffs}_{\text{foldr } (\rightarrow) ts \beta}$
using *Cons.IH* by *blast*
moreover
from $\langle B \cdot A \in \text{wffs}_{\text{foldr } (\rightarrow) ts \beta} \rangle$ obtain *t* where $B \in \text{wffs}_{t \rightarrow \text{foldr } (\rightarrow) ts \beta}$ and $A \in \text{wffs}_t$
by (*elim wffs-from-app*)
moreover from $\langle \text{length } ts = \text{length } As \rangle$ have $\text{length } (t \# ts) = \text{length } (A \# As)$
by *simp*
moreover from $\langle A \in \text{wffs}_t \rangle$ and $\langle \forall k < \text{length } As. As ! k \in \text{wffs}_{ts ! k} \rangle$
have $\forall k < \text{length } (A \# As). (A \# As) ! k \in \text{wffs}_{(t \# ts) ! k}$
by (*simp add: nth-Cons'*)
moreover from $\langle B \in \text{wffs}_{t \rightarrow \text{foldr } (\rightarrow) ts \beta} \rangle$ have $B \in \text{wffs}_{\text{foldr } (\rightarrow) (t \# ts) \beta}$
by *simp*
ultimately show *?case*
using *Cons.prem*s(1) by *blast*
qed

lemma *wffs-from-abs*:
assumes $\lambda x_\alpha. A \in \text{wffs}_\gamma$
obtains β where $\gamma = \alpha \rightarrow \beta$ and $A \in \text{wffs}_\beta$

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

lemma *wffs-from-equality*:

assumes $A =_{\alpha} B \in \text{wffs}_o$

shows $A \in \text{wffs}_{\alpha}$ and $B \in \text{wffs}_{\alpha}$

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

lemma *wffs-from-equivalence*:

assumes $A \equiv^{\mathcal{Q}} B \in \text{wffs}_o$

shows $A \in \text{wffs}_o$ and $B \in \text{wffs}_o$

using *assms* **unfolding** *equivalence-def* by (*fact wffs-from-equality*)+

lemma *wffs-from-forall*:

assumes $\forall x_{\alpha}. A \in \text{wffs}_o$

shows $A \in \text{wffs}_o$

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 $\wedge_{o \rightarrow o \rightarrow o} \cdot A \cdot B \in \text{wffs}_o$

shows $A \in \text{wffs}_o$ and $B \in \text{wffs}_o$

using *assms* by (*auto elim: wffs-from-app wffs-from-abs*)

lemma *wffs-from-conj-op*:

assumes $A \wedge^{\mathcal{Q}} B \in \text{wffs}_o$

shows $A \in \text{wffs}_o$ and $B \in \text{wffs}_o$

using *assms* **unfolding** *conj-op-def* by (*elim wffs-from-conj-fun*)+

lemma *wffs-from-imp-fun*:

assumes $\supset_{o \rightarrow o \rightarrow o} \cdot A \cdot B \in \text{wffs}_o$

shows $A \in \text{wffs}_o$ and $B \in \text{wffs}_o$

using *assms* by (*auto elim: wffs-from-app wffs-from-abs*)

lemma *wffs-from-imp-op*:

assumes $A \supset^{\mathcal{Q}} B \in \text{wffs}_o$

shows $A \in \text{wffs}_o$ and $B \in \text{wffs}_o$

using *assms* **unfolding** *imp-op-def* by (*elim wffs-from-imp-fun*)+

lemma *wffs-from-neg*:

assumes $\sim^{\mathcal{Q}} A \in \text{wffs}_o$

shows $A \in \text{wffs}_o$

using *assms* **unfolding** *neg-def* by (*fold equality-of-type-def*) (*drule wffs-from-equality, blast*)

lemma *wffs-from-disj-fun*:

assumes $\vee_{o \rightarrow o \rightarrow o} \cdot A \cdot B \in \text{wffs}_o$

shows $A \in \text{wffs}_o$ and $B \in \text{wffs}_o$

using *assms* by (*auto elim: wffs-from-app wffs-from-abs*)

lemma *wffs-from-disj-op*:

assumes $A \vee^Q B \in \text{wffs}_0$
shows $A \in \text{wffs}_0$ **and** $B \in \text{wffs}_0$
using *assms* **and** *wffs-from-disj-fun* **unfolding** *disj-op-def* **by** *blast+*

lemma *wffs-from-exists*:
assumes $\exists x_\alpha. A \in \text{wffs}_0$
shows $A \in \text{wffs}_0$
using *assms* **unfolding** *exists-def* **using** *wffs-from-neg* **and** *wffs-from-forall* **by** *blast*

lemma *wffs-from-inequality*:
assumes $A \neq_\alpha B \in \text{wffs}_0$
shows $A \in \text{wffs}_\alpha$ **and** $B \in \text{wffs}_\alpha$
using *assms* **unfolding** *inequality-of-type-def* **using** *wffs-from-equality* **and** *wffs-from-neg* **by** *meson+*

lemma *wff-has-unique-type*:
assumes $A \in \text{wffs}_\alpha$ **and** $A \in \text{wffs}_\beta$
shows $\alpha = \beta$
using *assms* **proof** (*induction arbitrary: $\alpha \beta$ rule: form.induct*)
case (*FVar v*)
obtain x **and** γ **where** $v = (x, \gamma)$
by *fastforce*
with *FVar.prem*s **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.prem*s **have** $\alpha = \gamma$ **and** $\beta = \gamma$
by (*blast elim: wffs-of-type-cases*)
then show *?case ..*
next
case (*FApp A B*)
from *FApp.prem*s **obtain** α' **and** β' **where** $A \in \text{wffs}_{\alpha' \rightarrow \alpha}$ **and** $A \in \text{wffs}_{\beta' \rightarrow \beta}$
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.prem*s **obtain** α' **and** β'
where $\alpha = \gamma \rightarrow \alpha'$ **and** $\beta = \gamma \rightarrow \beta'$ **and** $A \in \text{wffs}_{\alpha'}$ **and** $A \in \text{wffs}_{\beta'}$
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 $A \in \text{wffs}_o$
and $\bigwedge x. \mathcal{P}(x_o)$
and $\bigwedge c. \mathcal{P}(\{c\}_o)$
and $\bigwedge A B \alpha. A \in \text{wffs}_{\alpha \rightarrow o} \implies B \in \text{wffs}_\alpha \implies \mathcal{P}(A \cdot B)$
shows $\mathcal{P} A$
using *assms* **by** (*cases rule: wffs-of-type-cases*) *simp-all*

lemma *diff-types-implies-diff-wffs*:

assumes $A \in \text{wffs}_\alpha$ **and** $B \in \text{wffs}_\beta$
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** $\forall C \in \text{lset } Cs. \text{is-free-for } A v C$
shows *is-free-for* $A v (\cdot^{\mathcal{Q}}_* B Cs)$
using *assms* **proof** (*induction Cs rule: rev-induct*)
case *Nil*
then show *?case*
by *simp*
next
case (*snoc C Cs*)
from *snoc.premis*(2) **have** *is-free-for* $A v C$ **and** $\forall C \in \text{lset } Cs. \text{is-free-for } A v C$
by *simp-all*
with *snoc.premis*(1) **have** *is-free-for* $A v (\cdot^{\mathcal{Q}}_* B Cs)$
using *snoc.IH* **by** *simp*
with $\langle \text{is-free-for } A v C \rangle$ **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** $(x, \alpha) \notin \text{free-vars } A$
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* $A \ v \ (\lambda x_\alpha. T_o)$
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* $A \ v \ (\lambda x_\alpha. T_o =_{\alpha \rightarrow o} \lambda x_\alpha. B)$
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** $\text{lset } vs \cap \text{free-vars } A = \{\}$
shows *is-free-for* $A \ v \ (\forall^{\mathcal{Q}_*} vs \ B)$
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.prem(2)* **have** $v' \notin \text{free-vars } A$ **and** $\text{lset } vs \cap \text{free-vars } A = \{\}$
by *simp-all*
from *Cons.prem(1)* **and** $\langle \text{lset } vs \cap \text{free-vars } A = \{\} \rangle$ **have** *is-free-for* $A \ v \ (\forall^{\mathcal{Q}_*} vs \ B)$
by (*fact Cons.IH*)
from *this* **and** $\langle v' \notin \text{free-vars } A \rangle$ [*unfolded* $\langle v' = (x, \alpha) \rangle$] **have** *is-free-for* $A \ v \ (\forall x_\alpha. \forall^{\mathcal{Q}_*} vs \ B)$
by (*intro is-free-for-in-forall*)
with $\langle v' = (x, \alpha) \rangle$ **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 $\text{free-vars } \wedge_{o \rightarrow o \rightarrow o} = \{\}$
by *force*
then have *is-free-for* $A \ v \ (\wedge_{o \rightarrow o \rightarrow o})$
using *is-free-for-closed-form* **by** *fast*
with *assms* **have** *is-free-for* $A \ v \ (\wedge_{o \rightarrow o \rightarrow o} \cdot B \cdot C)$
by (*intro is-free-for-to-app*)
then show *?thesis*
by (*fold conj-op-def*)
qed

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

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

lemma *replacement-preserves-typing*:
assumes $C \langle p \leftarrow B \rangle \triangleright D$
and $A \preceq_p C$
and $A \in \text{wffs}_\alpha$ **and** $B \in \text{wffs}_\alpha$
shows $C \in \text{wffs}_\beta \iff D \in \text{wffs}_\beta$
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 $C \langle p \leftarrow B \rangle \triangleright D$
and $A \preceq_p C$
and $A \in \text{wffs}_\alpha$ **and** $B \in \text{wffs}_\alpha$
and $C \in \text{wffs}_\beta$ **and** $D \in \text{wffs}_\gamma$

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* \Rightarrow *type* \Rightarrow *bool* **where**

[*iff*]: *is-closed-wff-of-type* $A \alpha \longleftrightarrow A \in \text{wffs}_\alpha \wedge \text{free-vars } A = \{\}$

definition *is-sentence* :: *form* \Rightarrow *bool* **where**

[*iff*]: *is-sentence* $A \longleftrightarrow \text{is-closed-wff-of-type } A o$

2.14 Substitutions

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

definition *is-substitution* :: *substitution* \Rightarrow *bool* **where**

[*iff*]: *is-substitution* $\vartheta \longleftrightarrow (\forall (x, \alpha) \in \text{fmdom}' \vartheta. \vartheta \ \$\$! (x, \alpha) \in \text{wffs}_\alpha)$

fun *substitute* :: *substitution* \Rightarrow *form* \Rightarrow *form* (**S** - - [51, 51]) **where**

S $\vartheta (x_\alpha) = (\text{case } \vartheta \ \$\$ (x, \alpha) \text{ of } \text{None} \Rightarrow x_\alpha \mid \text{Some } A \Rightarrow A)$

| **S** $\vartheta (\{c\}_\alpha) = \{c\}_\alpha$

| **S** $\vartheta (A \cdot B) = (\text{S } \vartheta A) \cdot (\text{S } \vartheta B)$

| **S** $\vartheta (\lambda x_\alpha. A) = (\text{if } (x, \alpha) \notin \text{fmdom}' \vartheta \text{ then } \lambda x_\alpha. \text{S } \vartheta A \text{ else } \lambda x_\alpha. \text{S } (\text{fmdrop } (x, \alpha) \vartheta) A)$

lemma *empty-substitution-neutrality*:

shows **S** $\{\$\$ \} A = A$

by (*induction* A) *auto*

lemma *substitution-preserves-typing*:

assumes *is-substitution* ϑ

and $A \in \text{wffs}_\alpha$

shows **S** $\vartheta A \in \text{wffs}_\alpha$

using *assms(2)* and *assms(1)*[*unfolded is-substitution-def*] **proof** (*induction arbitrary: \vartheta*)

case (*var-is-wff* αx)

then show *?case*

by (*cases* $(x, \alpha) \in \text{fmdom}' \vartheta$) (*use fmdom'-notI in <force+>*)

next

case (*abs-is-wff* $\beta A \alpha x$)

then show *?case*

proof (*cases* $(x, \alpha) \in \text{fmdom}' \vartheta$)

case *True*

then have **S** $\vartheta (\lambda x_\alpha. A) = \lambda x_\alpha. \text{S } (\text{fmdrop } (x, \alpha) \vartheta) A$

by *simp*

moreover from *abs-is-wff.prem*s **have** *is-substitution* $(\text{fmdrop } (x, \alpha) \vartheta)$

by *fastforce*

with *abs-is-wff.IH* **have** **S** $(\text{fmdrop } (x, \alpha) \vartheta) A \in \text{wffs}_\beta$

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** $\mathbf{S} \vartheta A \in \text{wffs}_\beta$
using *abs-is-wff.premis* **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 $\mathbf{S} \vartheta (\forall x_\alpha. B) = (\text{if } (x, \alpha) \notin \text{fmdom}' \vartheta \text{ then } \forall x_\alpha. \mathbf{S} \vartheta B \text{ else } \forall x_\alpha. \mathbf{S} (\text{fmdrop } (x, \alpha) \vartheta) B)$
and $\mathbf{S} \vartheta (\exists x_\alpha. B) = (\text{if } (x, \alpha) \notin \text{fmdom}' \vartheta \text{ then } \exists x_\alpha. \mathbf{S} \vartheta B \text{ else } \exists x_\alpha. \mathbf{S} (\text{fmdrop } (x, \alpha) \vartheta) B)$
by *auto*

lemma *generalized-app-substitution:*

shows $\mathbf{S} \vartheta (\cdot^\mathcal{Q}_* A Bs) = \cdot^\mathcal{Q}_* (\mathbf{S} \vartheta A) (\text{map } (\lambda B. \mathbf{S} \vartheta B) Bs)$
by (*induction Bs arbitrary: A*) *simp-all*

lemma *generalized-abs-substitution:*

shows $\mathbf{S} \vartheta (\lambda^\mathcal{Q}_* vs A) = \lambda^\mathcal{Q}_* vs (\mathbf{S} (\text{fmdrop-set } (\text{fmdom}' \vartheta \cap \text{lset } vs) \vartheta) A)$

proof (*induction vs arbitrary: \vartheta*)

case *Nil*

then show *?case*

by *simp*

next

case (*Cons v vs*)

obtain *x* **and** α **where** $v = (x, \alpha)$

by *fastforce*

then show *?case*

proof (*cases v \notin fmdom' \vartheta*)

case *True*

then have $*$: $\text{fmdom}' \vartheta \cap \text{lset } (v \# vs) = \text{fmdom}' \vartheta \cap \text{lset } vs$

by *simp*

from *True* **have** $\mathbf{S} \vartheta (\lambda^\mathcal{Q}_* (v \# vs) A) = \lambda x_\alpha. \mathbf{S} \vartheta (\lambda^\mathcal{Q}_* vs A)$

using $\langle v = (x, \alpha) \rangle$ **by** *auto*

also have $\dots = \lambda x_\alpha. \lambda^\mathcal{Q}_* vs (\mathbf{S} (\text{fmdrop-set } (\text{fmdom}' \vartheta \cap \text{lset } vs) \vartheta) A)$

using *Cons.IH* **by** (*simp only:*)

also have $\dots = \lambda^\mathcal{Q}_* (v \# vs) (\mathbf{S} (\text{fmdrop-set } (\text{fmdom}' \vartheta \cap \text{lset } (v \# vs)) \vartheta) A)$

```

    using ⟨v = (x, α)⟩ and * by auto
  finally show ?thesis .
next
case False
let ?∅' = fmdrop v ∅
have *: fmdrop-set (fmdom' ∅ ∩ lset (v # vs)) ∅ = fmdrop-set (fmdom' ?∅' ∩ lset vs) ?∅'
  using False by clarsimp (metis Int-Diff Int-commute fmdrop-set-insert insert-Diff-single)
from False have S ∅ (λQ* (v # vs) A) = λxα. S ?∅' (λQ* vs A)
  using ⟨v = (x, α)⟩ by auto
also have ... = λxα. λQ* vs (S (fmdrop-set (fmdom' ?∅' ∩ lset vs) ?∅') A)
  using Cons.IH by (simp only:)
also have ... = λQ* (v # vs) (S (fmdrop-set (fmdom' ∅ ∩ lset (v # vs)) ∅) A)
  using ⟨v = (x, α)⟩ and * by auto
  finally show ?thesis .
qed
qed

lemma generalized-forall-substitution:
  shows S ∅ (∀Q* vs A) = ∀Q* vs (S (fmdrop-set (fmdom' ∅ ∩ lset vs) ∅) A)
proof (induction vs arbitrary: ∅)
case Nil
then show ?case
  by simp
next
case (Cons v vs)
obtain x and α where v = (x, α)
  by fastforce
then show ?case
proof (cases v ∉ fmdom' ∅)
case True
then have *: fmdom' ∅ ∩ lset (v # vs) = fmdom' ∅ ∩ lset vs
  by simp
from True have S ∅ (∀Q* (v # vs) A) = ∀xα. S ∅ (∀Q* vs A)
  using ⟨v = (x, α)⟩ by auto
also have ... = ∀xα. ∀Q* vs (S (fmdrop-set (fmdom' ∅ ∩ lset vs) ∅) A)
  using Cons.IH by (simp only:)
also have ... = ∀Q* (v # vs) (S (fmdrop-set (fmdom' ∅ ∩ lset (v # vs)) ∅) A)
  using ⟨v = (x, α)⟩ and * by auto
  finally show ?thesis .
next
case False
let ?∅' = fmdrop v ∅
have *: fmdrop-set (fmdom' ∅ ∩ lset (v # vs)) ∅ = fmdrop-set (fmdom' ?∅' ∩ lset vs) ?∅'
  using False by clarsimp (metis Int-Diff Int-commute fmdrop-set-insert insert-Diff-single)
from False have S ∅ (∀Q* (v # vs) A) = ∀xα. S ?∅' (∀Q* vs A)
  using ⟨v = (x, α)⟩ by auto
also have ... = ∀xα. ∀Q* vs (S (fmdrop-set (fmdom' ?∅' ∩ lset vs) ?∅') A)
  using Cons.IH by (simp only:)
also have ... = ∀Q* (v # vs) (S (fmdrop-set (fmdom' ∅ ∩ lset (v # vs)) ∅) A)

```

using $\langle v = (x, \alpha) \rangle$ **and** $*$ **by** *auto*
finally show *?thesis* .
qed
qed

lemma *singleton-substitution-simps:*

shows $\mathbf{S} \{(x, \alpha) \mapsto A\} (y_\beta) = (\text{if } (x, \alpha) \neq (y, \beta) \text{ then } y_\beta \text{ else } A)$
and $\mathbf{S} \{(x, \alpha) \mapsto A\} (\{c\}_\alpha) = \{c\}_\alpha$
and $\mathbf{S} \{(x, \alpha) \mapsto A\} (B \cdot C) = (\mathbf{S} \{(x, \alpha) \mapsto A\} B) \cdot (\mathbf{S} \{(x, \alpha) \mapsto A\} C)$
and $\mathbf{S} \{(x, \alpha) \mapsto A\} (\lambda y_\beta. B) = \lambda y_\beta. (\text{if } (x, \alpha) = (y, \beta) \text{ then } B \text{ else } \mathbf{S} \{(x, \alpha) \mapsto A\} B)$
by (*simp-all add: empty-substitution-neutrality fmdrop-fmupd-same*)

lemma *substitution-preserves-freeness:*

assumes $y \notin \text{free-vars } A$ **and** $y \neq z$
shows $y \notin \text{free-vars } \mathbf{S} \{x \mapsto FVar z\} A$
using *assms(1)* **proof** (*induction A rule: free-vars-form.induct*)
case ($1 \ x' \ \alpha$)
with *assms(2)* **show** *?case*
using *surj-pair[of z]* **by** (*cases x = (x', \alpha)*) *force+*
next
case ($4 \ x' \ \alpha \ A$)
then show *?case*
using *surj-pair[of z]*
by (*cases x = (x', \alpha)*) (*use singleton-substitution-simps(4) in presburger, auto*)
qed *auto*

lemma *renaming-substitution-minimal-change:*

assumes $y \notin \text{vars } A$ **and** $y \neq z$
shows $y \notin \text{vars } (\mathbf{S} \{x \mapsto FVar z\} A)$
using *assms(1)* **proof** (*induction A rule: vars-form.induct*)
case ($1 \ x' \ \alpha$)
with *assms(2)* **show** *?case*
using *surj-pair[of z]* **by** (*cases x = (x', \alpha)*) *force+*
next
case ($4 \ x' \ \alpha \ A$)
then show *?case*
using *surj-pair[of z]*
by (*cases x = (x', \alpha)*) (*use singleton-substitution-simps(4) in presburger, auto*)
qed *auto*

lemma *free-var-singleton-substitution-neutrality:*

assumes $v \notin \text{free-vars } A$
shows $\mathbf{S} \{v \mapsto B\} A = A$
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 $\mathbf{S} \{v \mapsto FVar\ v\} A = A$
by
(induction A rule: free-vars-form.induct)
(simp-all add: empty-substitution-neutrality fmdrop-fmupd-same)

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

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

```


let $\vartheta_{zy} = \{z \mapsto FVar\ y\}$ **and** $\vartheta_{xz} = \{x \mapsto FVar\ z\}$ **and** $\vartheta_{xy} = \{x \mapsto FVar\ y\}$
show *?case*
proof (cases $x = w$)
 case *True*
 then show *?thesis*
 proof (cases $z = w$)
 case *True*
 with $\langle x = w \rangle$ **have** $x \notin \text{free-vars } (FAbs\ w\ B)$ **and** $z \notin \text{free-vars } (FAbs\ w\ B)$
 using *surj-pair[of w]* **by** *fastforce+*
 from $\langle x \notin \text{free-vars } (FAbs\ w\ B) \rangle$ **have** $\mathbf{S}\ \vartheta_{xy}\ (FAbs\ w\ B) = FAbs\ w\ B$
 by (*fact free-var-singleton-substitution-neutrality*)
 also from $\langle z \notin \text{free-vars } (FAbs\ w\ B) \rangle$ **have** $\dots = \mathbf{S}\ \vartheta_{zy}\ (FAbs\ w\ B)$
 by (*fact free-var-singleton-substitution-neutrality[symmetric]*)
 also from $\langle x \notin \text{free-vars } (FAbs\ w\ B) \rangle$ **have** $\dots = \mathbf{S}\ \vartheta_{zy}\ \mathbf{S}\ \vartheta_{xz}\ (FAbs\ w\ B)$
 using *free-var-singleton-substitution-neutrality* **by** *simp*
 finally show *?thesis ..*
 next
 case *False*
 with $\langle x = w \rangle$ **have** $z \notin \text{free-vars } B$ **and** $x \notin \text{free-vars } (FAbs\ w\ B)$
 using $\langle z \notin \text{free-vars } (FAbs\ w\ B) \rangle$ **and** *surj-pair[of w]* **by** *fastforce+*
 from $\langle z \notin \text{free-vars } B \rangle$ **have** $\mathbf{S}\ \vartheta_{zy}\ B = B$
 by (*fact free-var-singleton-substitution-neutrality*)
 from $\langle x \notin \text{free-vars } (FAbs\ w\ B) \rangle$ **have** $\mathbf{S}\ \vartheta_{xy}\ (FAbs\ w\ B) = FAbs\ w\ B$
 by (*fact free-var-singleton-substitution-neutrality*)
 also from $\langle \mathbf{S}\ \vartheta_{zy}\ B = B \rangle$ **have** $\dots = FAbs\ w\ (\mathbf{S}\ \vartheta_{zy}\ B)$
 by (*simp only:*)
 also from $\langle z \notin \text{free-vars } (FAbs\ w\ B) \rangle$ **have** $\dots = \mathbf{S}\ \vartheta_{zy}\ (FAbs\ w\ B)$
 by (*simp add: $\langle FAbs\ w\ B = FAbs\ w\ (\mathbf{S}\ \vartheta_{zy}\ B) \rangle$ free-var-singleton-substitution-neutrality*)
 also from $\langle x \notin \text{free-vars } (FAbs\ w\ B) \rangle$ **have** $\dots = \mathbf{S}\ \vartheta_{zy}\ \mathbf{S}\ \vartheta_{xz}\ (FAbs\ w\ B)$
 using *free-var-singleton-substitution-neutrality* **by** *simp*
 finally show *?thesis ..*
qed
next
 case *False*
 then show *?thesis*
 proof (cases $z = w$)
 case *True*
 have $x \notin \text{free-vars } B$
 proof (*rule ccontr*)
 assume $\neg x \notin \text{free-vars } B$
 with $\langle x \neq w \rangle$ **have** $x \in \text{free-vars } (FAbs\ w\ B)$
 using *surj-pair[of w]* **by** *fastforce*
 then obtain p **where** $p \in \text{positions } (FAbs\ w\ B)$ **and** *is-free-at x p* $(FAbs\ w\ B)$
 using *free-vars-in-is-free-at* **by** *blast*
 with $\langle \text{is-free-for } (FVar\ z)\ x\ (FAbs\ w\ B) \rangle$ **have** $\neg \text{in-scope-of-abs } z\ p\ (FAbs\ w\ B)$
 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 $\langle \text{is-free-at } x\ p\ (FAbs\ w\ B) \rangle$]* **by** *blast*
 ultimately have $z \neq w$

```

    using in-scope-of-abs-in-abs by blast
  with ⟨z = w⟩ show False
  by contradiction
qed
then have *: S ?∂xy B = S ?∂xz B
  using free-var-singleton-substitution-neutrality by auto
from ⟨x ≠ w⟩ have S ?∂xy (FAbs w B) = FAbs w (S ?∂xy B)
  using surj-pair[of w] by fastforce
also from * have ... = FAbs w (S ?∂xz B)
  by (simp only:)
also from FAbs.prem1 have ... = S ?∂zy (FAbs w (S ?∂xz B))
  using ⟨x ∉ free-vars B⟩ and free-var-singleton-substitution-neutrality by auto
also from ⟨x ≠ w⟩ have ... = S ?∂zy S ?∂xz (FAbs w B)
  using surj-pair[of w] by fastforce
finally show ?thesis ..
next
case False
obtain vw and α where w = (vw, α)
  by fastforce
with ⟨is-free-for (FVar z) x (FAbs w B)⟩ and ⟨x ≠ w⟩ have is-free-for (FVar z) x B
  using is-free-for-from-abs by iprover
moreover from ⟨z ∉ free-vars (FAbs w B)⟩ and ⟨z ≠ w⟩ and ⟨w = (vw, α)⟩ have z ∉ free-vars
B
  by simp
ultimately have *: S ?∂zy S ?∂xz B = S ?∂xy B
  using FAbs.IH by simp
from ⟨x ≠ w⟩ have S ?∂xy (FAbs w B) = FAbs w (S ?∂xy B)
  using ⟨w = (vw, α)⟩ and free-var-singleton-substitution-neutrality by simp
also from * have ... = FAbs w (S ?∂zy S ?∂xz B)
  by (simp only:)
also from ⟨z ≠ w⟩ have ... = S ?∂zy (FAbs w (S ?∂xz B))
  using ⟨w = (vw, α)⟩ and free-var-singleton-substitution-neutrality by simp
also from ⟨x ≠ w⟩ have ... = S ?∂zy S ?∂xz (FAbs w B)
  using ⟨w = (vw, α)⟩ and free-var-singleton-substitution-neutrality by simp
finally show ?thesis ..
qed
qed
qed

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

```

```

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

```

```

    using substitution-absorption and surj-pair[of w] by force
  also from True have ... =  $\vartheta$  $$$! w
    using surj-pair[of w] by (metis fmdom'-notI option.case-eq-if substitute.simps(1))
  finally have S ( $\{x \mapsto FVar\ y\} ++_f \vartheta$ ) (FVar w) =  $\vartheta$  $$$! w .
  moreover from True and FVar.prem(4) have  $y \notin vars\ (\vartheta\ \text{\$!\$!}\ w)$ 
    by blast
  ultimately show ?thesis
    using form-is-free-for-absent-var by presburger
next
case False
with FVar.prem(3) and  $\langle w \neq x \rangle$  have S ( $\{x \mapsto FVar\ y\} ++_f \vartheta$ ) (FVar w) = FVar w
  using surj-pair[of w] by fastforce
with FVar.prem(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.prem(2) have  $y \notin vars\ C$  and  $y \notin vars\ D$ 
  by simp-all
from FApp.prem(1) have is-free-for A x C and is-free-for A x D
  using is-free-for-from-app by iprover+
have  $\forall v \in fmdom'\ \vartheta. is-free-for\ (\vartheta\ \text{\$!\$!}\ v)\ v\ C \wedge is-free-for\ (\vartheta\ \text{\$!\$!}\ v)\ v\ D$ 
proof (rule ballI)
  fix v
  assume  $v \in fmdom'\ \vartheta$ 
  with FApp.prem(5) have is-free-for  $(\vartheta\ \text{\$!\$!}\ v)\ v\ (C \cdot D)$ 
    by blast
  then show  $is-free-for\ (\vartheta\ \text{\$!\$!}\ v)\ v\ C \wedge is-free-for\ (\vartheta\ \text{\$!\$!}\ v)\ v\ D$ 
    using is-free-for-from-app by iprover+
qed
then have
  *:  $\forall v \in fmdom'\ \vartheta. is-free-for\ (\vartheta\ \text{\$!\$!}\ v)\ v\ C$  and **:  $\forall v \in fmdom'\ \vartheta. is-free-for\ (\vartheta\ \text{\$!\$!}\ v)\ v\ D$ 
  by auto
have S ( $\{x \mapsto FVar\ y\} ++_f \vartheta$ ) (C · D) = (S ( $\{x \mapsto FVar\ y\} ++_f \vartheta$ ) C) · (S ( $\{x \mapsto FVar\ y\} ++_f \vartheta$ ) D)
  by simp
moreover have is-free-for A y (S ( $\{x \mapsto FVar\ y\} ++_f \vartheta$ ) C)
  by (rule FApp.IH(1)[OF  $\langle is-free-for\ A\ x\ C \rangle \langle y \notin vars\ C \rangle$  FApp.prem(3,4) *])
moreover have is-free-for A y (S ( $\{x \mapsto FVar\ y\} ++_f \vartheta$ ) D)
  by (rule FApp.IH(2)[OF  $\langle is-free-for\ A\ x\ D \rangle \langle y \notin vars\ D \rangle$  FApp.prem(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 $F\text{Abs.prem}(2)$ **have** $y \notin \text{vars } B$
using *vars-form.elims* **by** *auto*
then show *?case*
proof (*cases* $w = x$)
case *True*
from *True* **and** $\langle x \notin \text{fmdom}' \vartheta \rangle$ **have** $w \notin \text{fmdom}' \vartheta$ **and** $x \notin \text{free-vars } (F\text{Abs } w \ B)$
using $\langle w = (x_w, \alpha_w) \rangle$ **by** *fastforce+*
with *True* **have** $\mathbf{S} (\{x \mapsto F\text{Var } y\} ++_f \vartheta) (F\text{Abs } w \ B) = \mathbf{S} \vartheta (F\text{Abs } w \ B)$
using *substitution-free-absorption* **by** *blast*
also have $\dots = F\text{Abs } w \ (\mathbf{S} \vartheta \ B)$
using $\langle w = (x_w, \alpha_w) \rangle$ $\langle w \notin \text{fmdom}' \vartheta \rangle$ *substitute.simps(4)* **by** *presburger*
finally have $\mathbf{S} (\{x \mapsto F\text{Var } y\} ++_f \vartheta) (F\text{Abs } w \ B) = F\text{Abs } w \ (\mathbf{S} \vartheta \ B)$.
moreover from $\langle \mathbf{S} \vartheta (F\text{Abs } w \ B) = F\text{Abs } w \ (\mathbf{S} \vartheta \ B) \rangle$ **have** $y \notin \text{vars } (F\text{Abs } w \ (\mathbf{S} \vartheta \ B))$
using *absent-vars-substitution-preservation[OF FAbs.prem(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 $F\text{Abs.prem}(1)$ **and** $\langle w \neq x \rangle$ **and** $\langle w = (v_w, \alpha_w) \rangle$ **have** *is-free-for* $A \ x \ B$
using *is-free-for-from-abs* **by** *iprover*
then show *?thesis*
proof (*cases* $w \in \text{fmdom}' \vartheta$)
case *True*
then have $\mathbf{S} (\{x \mapsto F\text{Var } y\} ++_f \vartheta) (F\text{Abs } w \ B) = F\text{Abs } w \ (\mathbf{S} (\text{fmdrop } w \ (\{x \mapsto F\text{Var } y\} ++_f \vartheta)) \ B)$
using $\langle w = (v_w, \alpha_w) \rangle$ **by** (*simp add: fmdrop-idle'*)
also from $\langle w \neq x \rangle$ **and** *True* **have** $\dots = F\text{Abs } w \ (\mathbf{S} (\{x \mapsto F\text{Var } y\} ++_f \text{fmdrop } w \ \vartheta) \ B)$
by (*simp add: fmdrop-fmupd*)
finally
have $\ast: \mathbf{S} (\{x \mapsto F\text{Var } y\} ++_f \vartheta) (F\text{Abs } w \ B) = F\text{Abs } w \ (\mathbf{S} (\{x \mapsto F\text{Var } y\} ++_f \text{fmdrop } w \ \vartheta) \ B)$.
have $\forall v \in \text{fmdom}' (\text{fmdrop } w \ \vartheta)$. *is-free-for* $(\text{fmdrop } w \ \vartheta \ \mathbf{S}! \ v) \ v \ B$
proof
fix v
assume $v \in \text{fmdom}' (\text{fmdrop } w \ \vartheta)$
with $F\text{Abs.prem}(5)$ **have** *is-free-for* $(\text{fmdrop } w \ \vartheta \ \mathbf{S}! \ v) \ v \ (F\text{Abs } w \ B)$
by *auto*
moreover from $\langle v \in \text{fmdom}' (\text{fmdrop } w \ \vartheta) \rangle$ **have** $v \neq w$
by *auto*
ultimately show *is-free-for* $(\text{fmdrop } w \ \vartheta \ \mathbf{S}! \ v) \ v \ B$
unfolding $\langle w = (v_w, \alpha_w) \rangle$ **using** *is-free-for-from-abs* **by** *iprover*
qed
moreover from $F\text{Abs.prem}(3)$ **have** $x \notin \text{fmdom}' (\text{fmdrop } w \ \vartheta)$
by *simp*
moreover from $F\text{Abs.prem}(4)$ **have** $\forall v \in \text{fmdom}' (\text{fmdrop } w \ \vartheta)$. $y \notin \text{vars } (\text{fmdrop } w \ \vartheta \ \mathbf{S}! \ v)$

by *simp*
 ultimately have *is-free-for* $A\ y$ ($\mathbf{S}\ (\{x \mapsto FVar\ y\} ++_f\ fmdrop\ w\ \vartheta)\ B$)
 using $\langle is-free-for\ A\ x\ B \rangle$ and $\langle y \notin vars\ B \rangle$ and *FAbs.IH* by *iprover*
 then show *?thesis*
 proof (cases $x \notin free-vars\ B$)
 case *True*
 have $y \notin vars\ (\mathbf{S}\ (\{x \mapsto FVar\ y\} ++_f\ \vartheta)\ (FAbs\ w\ B))$
 proof –
 have $\mathbf{S}\ (\{x \mapsto FVar\ y\} ++_f\ \vartheta)\ (FAbs\ w\ B) = FAbs\ w\ (\mathbf{S}\ (\{x \mapsto FVar\ y\} ++_f\ fmdrop\ w\ \vartheta)\ B)$
 B) using * .
 also from $\langle x \notin free-vars\ B \rangle$ and *FAbs.prem3* have $\dots = FAbs\ w\ (\mathbf{S}\ (fmdrop\ w\ \vartheta)\ B)$
 using *substitution-free-absorption* by (*simp add: fmdom'-notD*)
 finally have $\mathbf{S}\ (\{x \mapsto FVar\ y\} ++_f\ \vartheta)\ (FAbs\ w\ B) = FAbs\ w\ (\mathbf{S}\ (fmdrop\ w\ \vartheta)\ B)$.
 with *FAbs.prem2* and $\langle w = (v_w, \alpha_w) \rangle$ and *FAbs.prem4* 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 $\langle w \neq x \rangle$ have $\neg is-free-for\ A\ x\ (FAbs\ w\ B)$
 using *form-with-free-binder-not-free-for* by *simp*
 with *FAbs.prem1* show *False*
 by *contradiction*
 qed
 with $\langle is-free-for\ A\ y\ (\mathbf{S}\ (\{x \mapsto FVar\ y\} ++_f\ fmdrop\ w\ \vartheta)\ B) \rangle$
 have *is-free-for* $A\ y\ (FAbs\ w\ (\mathbf{S}\ (\{x \mapsto FVar\ y\} ++_f\ fmdrop\ w\ \vartheta)\ B))$
 unfolding $\langle w = (v_w, \alpha_w) \rangle$ using *is-free-for-to-abs* by *iprover*
 with * show *?thesis*
 by (*simp only:*)
 qed
 next
 case *False*
 have $\forall v \in fmdom'\ \vartheta. is-free-for\ (\vartheta\ \$\$!\ v)\ v\ B$
 proof (rule *ballI*)
 fix v
 assume $v \in fmdom'\ \vartheta$
 with *FAbs.prem5* have *is-free-for* $(\vartheta\ \$\$!\ v)\ v\ (FAbs\ w\ B)$
 by *blast*
 moreover from $\langle v \in fmdom'\ \vartheta \rangle$ and $\langle w \notin fmdom'\ \vartheta \rangle$ have $v \neq w$
 by *blast*
 ultimately show *is-free-for* $(\vartheta\ \$\$!\ v)\ v\ B$
 unfolding $\langle w = (v_w, \alpha_w) \rangle$ using *is-free-for-from-abs* by *iprover*
 qed
 with $\langle is-free-for\ A\ x\ B \rangle$ and $\langle y \notin vars\ B \rangle$ and *FAbs.prem3,4*

```

have is-free-for A y (S ({x ↦ FVar y} ++f ∅) B)
  using FAbs.IH by iprover
then show ?thesis
proof (cases x ∉ free-vars B)
  case True
  have y ∉ vars (S ({x ↦ FVar y} ++f ∅) (FAbs w B))
  proof -
    from False and ⟨w = (vw, αw)⟩ and ⟨w ≠ x⟩
    have S ({x ↦ FVar y} ++f ∅) (FAbs w B) = FAbs w (S ({x ↦ FVar y} ++f ∅) B)
    by auto
    also from ⟨x ∉ free-vars B⟩ and FAbs.prem3(3) have ... = FAbs w (S ∅ B)
    using substitution-free-absorption by (simp add: fmdom'-notD)
    finally have S ({x ↦ FVar y} ++f ∅) (FAbs w B) = FAbs w (S ∅ B) .
    with FAbs.prem3(2,4) and ⟨w = (vw, αw)⟩ show ?thesis
    using absent-vars-substitution-preservation by auto
  qed
then show ?thesis
  using is-free-for-absent-var by simp
next
case False
have w ∉ free-vars A
proof (rule ccontr)
  assume ¬ w ∉ free-vars A
  with False and ⟨w ≠ x⟩ have ¬ is-free-for A x (FAbs w B)
  using form-with-free-binder-not-free-for by simp
  with FAbs.prem1(1) show False
  by contradiction
qed
with ⟨is-free-for A y (S ({x ↦ FVar y} ++f ∅) B)⟩
have is-free-for A y (FAbs w (S ({x ↦ FVar y} ++f ∅) B))
  unfolding ⟨w = (vw, αw)⟩ using is-free-for-to-abs by iprover
moreover from ⟨w ∉ fmdom' ∅⟩ and ⟨w ≠ x⟩ and FAbs.prem3(3)
have S ({x ↦ FVar y} ++f ∅) (FAbs w B) = FAbs w (S ({x ↦ FVar y} ++f ∅) 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 ∅ and is-substitution {v ↦ A}
and ∅ $$ v = None and ∀ v' ∈ fmdom' ∅. v ∉ vars (∅ $$! v')
shows S {v ↦ A} S ∅ B = S ({v ↦ A} ++f ∅) B
using assms(1,3,4) proof (induction B arbitrary: ∅)
  case (FVar v')

```



```

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

```

by (*simp add*: $\langle w = (v_w, \alpha) \rangle$)
 also from *FAbs.IH* have $\dots = \text{FAbs } w \ (\mathbf{S} (\{v \mapsto A\} ++_f \vartheta) \ C)$
 using *FAbs.prem*s(1,2,3) by *blast*
 also from $\langle v \neq w \rangle$ and *True* have $\dots = \mathbf{S} (\{v \mapsto A\} ++_f \vartheta) \ (\text{FAbs } w \ C)$
 by (*simp add*: $\langle w = (v_w, \alpha) \rangle$)
 finally show *?thesis* .
 next
 case *False*
 then have $\mathbf{S} \{v \mapsto A\} \ \mathbf{S} \ \vartheta \ (\text{FAbs } w \ C) = \mathbf{S} \{v \mapsto A\} \ (\text{FAbs } w \ (\mathbf{S} \ (\text{fmdrop } w \ \vartheta) \ C))$
 by (*simp add*: $\langle w = (v_w, \alpha) \rangle$)
 also from $\langle v \neq w \rangle$ have $\dots = \text{FAbs } w \ (\mathbf{S} \{v \mapsto A\} \ \mathbf{S} \ (\text{fmdrop } w \ \vartheta) \ C)$
 by (*simp add*: $\langle w = (v_w, \alpha) \rangle$)
 also have $\dots = \text{FAbs } w \ (\mathbf{S} (\{v \mapsto A\} ++_f \text{fmdrop } w \ \vartheta) \ C)$
 proof –
 from $\langle \text{is-substitution } \vartheta \rangle$ have *is-substitution* (*fmdrop* $w \ \vartheta$)
 by *fastforce*
 moreover from $\langle \vartheta \ \S\S \ v = \text{None} \rangle$ have (*fmdrop* $w \ \vartheta$) $\S\S \ v = \text{None}$
 by *force*
 moreover from *FAbs.prem*s(3) have $\forall v' \in \text{fmdom}' \ (\text{fmdrop } w \ \vartheta). \ v \notin \text{vars} \ ((\text{fmdrop } w \ \vartheta) \ \S\S!$
 $v')$
 by *force*
 ultimately show *?thesis*
 using *FAbs.IH* by *blast*
 qed
 also from $\langle v \neq w \rangle$ have $\dots = \mathbf{S} (\{v \mapsto A\} ++_f \vartheta) \ (\text{FAbs } w \ C)$
 by (*simp add*: $\langle w = (v_w, \alpha) \rangle$ *fmdrop-idle'*)
 finally show *?thesis* .
 qed
 next
 case *False*
 then show *?thesis*
 proof (*cases* $w \notin \text{fmdom}' \ \vartheta$)
 case *True*
 then have $\mathbf{S} \{v \mapsto A\} \ \mathbf{S} \ \vartheta \ (\text{FAbs } w \ C) = \mathbf{S} \{v \mapsto A\} \ (\text{FAbs } w \ (\mathbf{S} \ \vartheta \ C))$
 by (*simp add*: $\langle w = (v_w, \alpha) \rangle$)
 also from $\langle \neg v \neq w \rangle$ have $\dots = \text{FAbs } w \ (\mathbf{S} \ \vartheta \ C)$
 using $\langle w = (v_w, \alpha) \rangle$ and *singleton-substitution-simps*(4) by *presburger*
 also from $\langle \neg v \neq w \rangle$ and *True* have $\dots = \text{FAbs } w \ (\mathbf{S} \ (\text{fmdrop } w \ (\{v \mapsto A\} ++_f \vartheta)) \ C)$
 by (*simp add*: *fmdrop-fmupd-same fmdrop-idle'*)
 also from $\langle \neg v \neq w \rangle$ have $\dots = \mathbf{S} (\{v \mapsto A\} ++_f \vartheta) \ (\text{FAbs } w \ C)$
 by (*simp add*: $\langle w = (v_w, \alpha) \rangle$)
 finally show *?thesis* .
 next
 case *False*
 then have $\mathbf{S} \{v \mapsto A\} \ \mathbf{S} \ \vartheta \ (\text{FAbs } w \ C) = \mathbf{S} \{v \mapsto A\} \ (\text{FAbs } w \ (\mathbf{S} \ (\text{fmdrop } w \ \vartheta) \ C))$
 by (*simp add*: $\langle w = (v_w, \alpha) \rangle$)
 also from $\langle \neg v \neq w \rangle$ have $\dots = \text{FAbs } w \ (\mathbf{S} \ (\text{fmdrop } w \ \vartheta) \ C)$
 using $\langle \vartheta \ \S\S \ v = \text{None} \rangle$ and *False* by (*simp add*: *fmdom'-notI*)
 also from $\langle \neg v \neq w \rangle$ have $\dots = \text{FAbs } w \ (\mathbf{S} \ (\text{fmdrop } w \ (\{v \mapsto A\} ++_f \vartheta)) \ C)$

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

lemma *updated-substitution-is-substitution*:

assumes $v \notin \text{fmdom}' \vartheta$ and *is-substitution* ($\vartheta(v \mapsto A)$)
 shows *is-substitution* ϑ
unfolding *is-substitution-def* **proof** (*intro ballI*)
 fix $v' :: \text{var}$
 obtain x and α where $v' = (x, \alpha)$
 by *fastforce*
 assume $v' \in \text{fmdom}' \vartheta$
 with *assms*(2)[*unfolded is-substitution-def*] have $v' \in \text{fmdom}' (\vartheta(v \mapsto A))$
 by *simp*
 with *assms*(2)[*unfolded is-substitution-def*] have $\vartheta(v \mapsto A) \ \S\ \$ (x, \alpha) \in \text{wffs}_\alpha$
 using $\langle v' = (x, \alpha) \rangle$ by *fastforce*
 with *assms*(1) and $\langle v' \in \text{fmdom}' \vartheta \rangle$ and $\langle v' = (x, \alpha) \rangle$ have $\vartheta \ \S\ \$ (x, \alpha) \in \text{wffs}_\alpha$
 by (*metis fmupd-lookup*)
 then show *case v' of* $(x, \alpha) \Rightarrow \vartheta \ \S\ \$ (x, \alpha) \in \text{wffs}_\alpha$
 by (*simp add: \langle v' = (x, \alpha) \rangle*)
 qed

definition *is-renaming-substitution* where

[*iff*]: *is-renaming-substitution* $\vartheta \longleftrightarrow$ *is-substitution* $\vartheta \wedge \text{fmpr}$ ed $(\lambda. A. \exists v. A = \text{FVar } v) \vartheta$

The following lemma proves that $\S_{y_{\alpha_1}^1 \dots y_{\alpha_n}^n} B = \S_{y_{\alpha_1}^1} \dots \S_{y_{\alpha_n}^n} B$ provided that

- $x_{\alpha_1}^1 \dots x_{\alpha_n}^n$ are distinct variables
- $y_{\alpha_1}^1 \dots y_{\alpha_n}^n$ are distinct variables, distinct from $x_{\alpha_1}^1 \dots x_{\alpha_n}^n$ 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 $ps :: (\text{var} \times \text{form}) \text{ list}$
 assumes $\vartheta = \text{fmap-of-list } ps$ and *is-renaming-substitution* ϑ
 and *distinct* ($\text{map fst } ps$) and *distinct* ($\text{map snd } ps$)
 and $\text{vars } (\text{fmrans}' \vartheta) \cap (\text{fmdom}' \vartheta \cup \text{vars } B) = \{\}$
 shows **S** $\vartheta B = \text{foldr } (\lambda(x, y) C. \mathbf{S} \{x \mapsto y\} C) ps B$
using *assms* **proof** (*induction ps arbitrary: \vartheta*)
 case *Nil*
 then have $\vartheta = \{\ \S\ \$ \}$

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.prem*s(1,2) obtain x and y where $\vartheta \ \$\$ (fst\ p) = Some\ (FVar\ y)$ and $p = (x, FVar\ y)$
 using *surj-pair*[of p] by *fastforce*
 let $\vartheta' = fmap\ of\ list\ ps$
 from *Cons.prem*s(1) and $\langle p = (x, FVar\ y) \rangle$ have $\vartheta = fmupd\ x\ (FVar\ y)\ \vartheta'$
 by *simp*
 moreover from *Cons.prem*s(3) and $\langle p = (x, FVar\ y) \rangle$ have $x \notin fmdom'\ \vartheta'$
 by *simp*
 ultimately have $\vartheta = \{x \mapsto FVar\ y\} ++_f\ \vartheta'$
 using *fmap-singleton-comm* by *fastforce*
 with *Cons.prem*s(2) and $\langle x \notin fmdom'\ \vartheta' \rangle$ have *is-renaming-substitution* ϑ'
 unfolding *is-renaming-substitution-def* and $\langle \vartheta = fmupd\ x\ (FVar\ y)\ \vartheta' \rangle$
 using *updated-substitution-is-substitution* by (*metis fmdiff-fmupd fmdom'-notD fmpred-filter*)
 from *Cons.prem*s(2) and $\langle \vartheta = fmupd\ x\ (FVar\ y)\ \vartheta' \rangle$ have *is-renaming-substitution* $\{x \mapsto FVar\ y\}$
 by *auto*
 have
 $foldr\ (\lambda(x, y)\ C.\ \mathbf{S}\ \{x \mapsto y\}\ C)\ (p\ \# \ ps)\ B$
 $=$
 $\mathbf{S}\ \{x \mapsto FVar\ y\}\ (foldr\ (\lambda(x, y)\ C.\ \mathbf{S}\ \{x \mapsto y\}\ C)\ ps\ B)$
 by (*simp add: \langle p = (x, FVar\ y) \rangle*)
 also have $\dots = \mathbf{S}\ \{x \mapsto FVar\ y\}\ \mathbf{S}\ \vartheta'\ B$
 proof -
 from *Cons.prem*s(3,4) have *distinct* (*map fst ps*) and *distinct* (*map snd ps*)
 by *fastforce+*
 moreover have $vars\ (fmrans'\ \vartheta') \cap (fmdom'\ \vartheta' \cup vars\ B) = \{\}$
 proof -
 have $vars\ (fmrans'\ \vartheta) = vars\ (\{FVar\ y\} \cup fmrans'\ \vartheta')$
 using $\langle \vartheta = fmupd\ x\ (FVar\ y)\ \vartheta' \rangle$ and $\langle x \notin fmdom'\ \vartheta' \rangle$ by (*metis fmdom'-notD fmrans'-fmupd*)
 then have $vars\ (fmrans'\ \vartheta) = \{y\} \cup vars\ (fmrans'\ \vartheta')$
 using *singleton-form-set-vars* by *auto*
 moreover have $fmdom'\ \vartheta = \{x\} \cup fmdom'\ \vartheta'$
 by (*simp add: \langle \vartheta = \{x \mapsto FVar\ y\} ++_f\ \vartheta' \rangle*)
 ultimately show *?thesis*
 using *Cons.prem*s(5) by *auto*
 qed
 ultimately show *?thesis*
 using *Cons.IH* and *\langle is-renaming-substitution \vartheta' \rangle* by *simp*
 qed
 also have $\dots = \mathbf{S}\ (\{x \mapsto FVar\ y\} ++_f\ \vartheta')\ B$
 proof (*rule substitution-fusion*)
 show *is-substitution* ϑ'
 using *\langle is-renaming-substitution \vartheta' \rangle* by *simp*

```

show is-substitution {x ↦ FVar y}
  using ⟨is-renaming-substitution {x ↦ FVar y}⟩ by simp
show ?∅' $$ x = None
  using ⟨x ∉ fmdom' ?∅'⟩ by blast
show ∀ v' ∈ fmdom' ?∅'. x ∉ vars (?∅' $$! v')
proof -
  have x ∈ fmdom' ∅
  using ⟨∅ = {x ↦ FVar y} ++f ?∅'⟩ by simp
  then have x ∉ vars (fmdom' ∅)
  using Cons.prem5 by blast
  moreover have {?∅' $$! v' | v'. v' ∈ fmdom' ?∅'} ⊆ fmdom' ∅
  unfolding ⟨∅ = ?∅'(x ↦ FVar y)⟩ using ⟨?∅' $$ x = None⟩
  by (auto simp add: fmllookup-of-list fmllookup-dom'-iff fmdom'I weak-map-of-SomeI)
  ultimately show ?thesis
  by force
qed
qed
also from ⟨∅ = {x ↦ FVar y} ++f ?∅'⟩ have ... = S ∅ B
  by (simp only:)
  finally show ?case ..
qed

lemma free-vars-agreement-substitution-equality:
  assumes fmdom' ∅ = fmdom' ∅'
  and ∀ v ∈ free-vars A ∩ fmdom' ∅. ∅ $$! v = ∅' $$! v
  shows S ∅ A = S ∅' A
using assms proof (induction A arbitrary: ∅ ∅')
  case (FVar v)
  have free-vars (FVar v) = {v}
  using surj-pair[of v] by fastforce
  with FVar have ∅ $$! v = ∅' $$! v
  by force
  with FVar.prem1 show ?case
  using surj-pair[of v] by (metis fmdom'-notD fmdom'-notI option.collapse substitute.simps(1))
next
  case FCon
  then show ?case
  by (metis prod.exhaust-sel substitute.simps(2))
next
  case (FApp B C)
  have S ∅ (B • C) = (S ∅ B) • (S ∅ C)
  by simp
  also have ... = (S ∅' B) • (S ∅' C)
proof -
  have ∀ v ∈ free-vars B ∩ fmdom' ∅. ∅ $$! v = ∅' $$! v
  and ∀ v ∈ free-vars C ∩ fmdom' ∅. ∅ $$! v = ∅' $$! v
  using FApp.prem2 by auto
  with FApp.IH(1,2) and FApp.prem1 show ?thesis
  by blast

```

```

qed
finally show ?case
  by simp
next
case (FAbs w B)
from FAbs.prem1(1,2) have *:  $\forall v \in \text{free-vars } B - \{w\} \cap \text{fmdom}' \vartheta. \vartheta \text{ $$$ } v = \vartheta' \text{ $$$ } v$ 
  using surj-pair[of w] by fastforce
show ?case
proof (cases  $w \in \text{fmdom}' \vartheta$ )
  case True
  then have S  $\vartheta (FAbs w B) = FAbs w (\mathbf{S} (\text{fmdrop } w \vartheta) B)$ 
    using surj-pair[of w] by fastforce
  also have ... =  $FAbs w (\mathbf{S} (\text{fmdrop } w \vartheta') B)$ 
  proof -
    from * have  $\forall v \in \text{free-vars } B \cap \text{fmdom}' (\text{fmdrop } w \vartheta). (\text{fmdrop } w \vartheta) \text{ $$$ } v = (\text{fmdrop } w \vartheta') \text{ $$$ } v$ 
    v
    by simp
    moreover have  $\text{fmdom}' (\text{fmdrop } w \vartheta) = \text{fmdom}' (\text{fmdrop } w \vartheta')$ 
      by (simp add: FAbs.prem1(1))
    ultimately show ?thesis
      using FAbs.IH by blast
  qed
  finally show ?thesis
    using FAbs.prem1(1) and True and surj-pair[of w] by fastforce
next
case False
then have S  $\vartheta (FAbs w B) = FAbs w (\mathbf{S} \vartheta B)$ 
  using surj-pair[of w] by fastforce
also have ... =  $FAbs w (\mathbf{S} \vartheta' B)$ 
proof -
  from * have  $\forall v \in \text{free-vars } B \cap \text{fmdom}' \vartheta. \vartheta \text{ $$$ } v = \vartheta' \text{ $$$ } v$ 
  using False by blast
  with FAbs.prem1(1) show ?thesis
  using FAbs.IH by blast
qed
finally show ?thesis
  using FAbs.prem1(1) and False and surj-pair[of w] by fastforce
qed
qed

```

The following lemma proves that $\S_{A_\alpha}^{x_\alpha} \S_{A_{\alpha_1}^1 \dots A_{\alpha_n}^n} B = \S_{A_\alpha}^{x_\alpha} \S_{A_\alpha A_{\alpha_1}^1}^{x_{\alpha_1}^1} \dots \S_{A_\alpha A_{\alpha_n}^n}^{x_{\alpha_n}^n} B$ provided that x_α is distinct from $x_{\alpha_1}^1, \dots, x_{\alpha_n}^n$ and $A_{\alpha_i}^i$ is free for $x_{\alpha_i}^i$ in B :

lemma substitution-consolidation:

```

assumes  $v \notin \text{fmdom}' \vartheta$ 
and  $\forall v' \in \text{fmdom}' \vartheta. \text{is-free-for } (\vartheta \text{ $$$ } v') v' B$ 
shows  $\mathbf{S} \{v \mapsto A\} \mathbf{S} \vartheta B = \mathbf{S} (\{v \mapsto A\} ++_f \text{fmap } (\lambda A'. \mathbf{S} \{v \mapsto A\} A') \vartheta) B$ 
using assms proof (induction B arbitrary:  $\vartheta$ )
  case (FApp B C)

```

have $\forall v' \in \text{fndom}' \vartheta. \text{is-free-for } (\vartheta \text{ \&\&! } v') v' B \wedge \text{is-free-for } (\vartheta \text{ \&\&! } v') v' C$
proof
fix v'
assume $v' \in \text{fndom}' \vartheta$
with $\text{FApp.prem}(2)$ **have** $\text{is-free-for } (\vartheta \text{ \&\&! } v') v' (B \cdot C)$
by *blast*
then show $\text{is-free-for } (\vartheta \text{ \&\&! } v') v' B \wedge \text{is-free-for } (\vartheta \text{ \&\&! } v') v' C$
using $\text{is-free-for-from-app}$ **by** *iprover*
qed
with FApp.IH **and** $\text{FApp.prem}(1)$ **show** $?case$
by *simp*
next
case $(\text{FAbs } w B)$
let $?\vartheta' = \text{fmmap } (\lambda A'. \mathbf{S} \{v \mapsto A\} A') \vartheta$
show $?case$
proof $(\text{cases } w \in \text{fndom}' \vartheta)$
case *True*
then have $w \in \text{fndom}' ?\vartheta'$
by *simp*
with *True* **and** FAbs.prem **have** $v \neq w$
by *blast*
from *True* **have** $\mathbf{S} \{v \mapsto A\} \mathbf{S} \vartheta (\text{FAbs } w B) = \mathbf{S} \{v \mapsto A\} (\text{FAbs } w (\mathbf{S} (\text{fmdrop } w \vartheta) B))$
using $\text{surj-pair}[of w]$ **by** *fastforce*
also from $\langle v \neq w \rangle$ **have** $\dots = \text{FAbs } w (\mathbf{S} \{v \mapsto A\} \mathbf{S} (\text{fmdrop } w \vartheta) B)$
using $\text{surj-pair}[of w]$ **by** *fastforce*
also have $\dots = \text{FAbs } w (\mathbf{S} (\text{fmdrop } w (\{v \mapsto A\} ++_f ?\vartheta')) B)$
proof $-$
obtain x_w **and** α_w **where** $w = (x_w, \alpha_w)$
by *fastforce*
have $\forall v' \in \text{fndom}' (\text{fmdrop } w \vartheta). \text{is-free-for } ((\text{fmdrop } w \vartheta) \text{ \&\&! } v') v' B$
proof
fix v'
assume $v' \in \text{fndom}' (\text{fmdrop } w \vartheta)$
with $\text{FAbs.prem}(2)$ **have** $\text{is-free-for } (\vartheta \text{ \&\&! } v') v' (\text{FAbs } w B)$
by *auto*
with $\langle w = (x_w, \alpha_w) \rangle$ **and** $\langle v' \in \text{fndom}' (\text{fmdrop } w \vartheta) \rangle$
have $\text{is-free-for } (\vartheta \text{ \&\&! } v') v' (\lambda x_w \alpha_w. B)$ **and** $v' \neq (x_w, \alpha_w)$
by *auto*
then have $\text{is-free-for } (\vartheta \text{ \&\&! } v') v' B$
using $\text{is-free-for-from-abs}$ **by** *presburger*
with $\langle v' \neq (x_w, \alpha_w) \rangle$ **and** $\langle w = (x_w, \alpha_w) \rangle$ **show** $\text{is-free-for } (\text{fmdrop } w \vartheta \text{ \&\&! } v') v' B$
by *simp*
qed
moreover have $v \notin \text{fndom}' (\text{fmdrop } w \vartheta)$
by $(\text{simp add: } \text{FAbs.prem}(1))$
ultimately show $?thesis$
using FAbs.IH **and** $\langle v \neq w \rangle$ **by** $(\text{simp add: } \text{fmdrop-fmupd})$
qed
finally show $?thesis$

```

    using  $\langle w \in \text{fmdom}' \vartheta' \rangle$  and surj-pair[of w] by fastforce
next
case False
then have  $w \notin \text{fmdom}' \vartheta'$ 
  by simp
from FAbs.prems have  $v \notin \text{fmdom}' \vartheta'$ 
  by simp
from False have *:  $\mathbf{S} \{v \mapsto A\} \mathbf{S} \vartheta (FAbs\ w\ B) = \mathbf{S} \{v \mapsto A\} (FAbs\ w\ (\mathbf{S} \vartheta\ B))$ 
  using surj-pair[of w] by fastforce
then show ?thesis
proof (cases  $v \neq w$ )
  case True
  then have  $\mathbf{S} \{v \mapsto A\} (FAbs\ w\ (\mathbf{S} \vartheta\ B)) = FAbs\ w\ (\mathbf{S} \{v \mapsto A\} (\mathbf{S} \vartheta\ B))$ 
    using surj-pair[of w] by fastforce
  also have  $\dots = FAbs\ w\ (\mathbf{S} (\{v \mapsto A\} ++_f \vartheta') B)$ 
  proof -
    obtain  $x_w$  and  $\alpha_w$  where  $w = (x_w, \alpha_w)$ 
    by fastforce
    have  $\forall v' \in \text{fmdom}' \vartheta. \text{is-free-for } (\vartheta \ \$\$! \ v') \ v' \ B$ 
  proof
    fix  $v'$ 
    assume  $v' \in \text{fmdom}' \vartheta$ 
    with FAbs.prems(2) have is-free-for  $(\vartheta \ \$\$! \ v') \ v' (FAbs\ w\ B)$ 
      by auto
    with  $\langle w = (x_w, \alpha_w) \rangle$  and  $\langle v' \in \text{fmdom}' \vartheta \rangle$  and False
    have is-free-for  $(\vartheta \ \$\$! \ v') \ v' (\lambda x_w \alpha_w. B)$  and  $v' \neq (x_w, \alpha_w)$ 
      by fastforce+
    then have is-free-for  $(\vartheta \ \$\$! \ v') \ v' B$ 
      using is-free-for-from-abs by presburger
    with  $\langle v' \neq (x_w, \alpha_w) \rangle$  and  $\langle w = (x_w, \alpha_w) \rangle$  show is-free-for  $(\vartheta \ \$\$! \ v') \ v' B$ 
      by simp
  qed
  with FAbs.IH show ?thesis
  using FAbs.prems(1) by blast
qed
finally show ?thesis
proof -
  assume
     $\mathbf{S} \{v \mapsto A\} (FAbs\ w\ (\mathbf{S} \vartheta\ B)) = FAbs\ w\ (\mathbf{S} (\{v \mapsto A\} ++_f \text{fmap} (\text{substitute } \{v \mapsto A\}) \vartheta))$ 
B)
  moreover have  $w \notin \text{fmdom}' (\{v \mapsto A\} ++_f \text{fmap} (\text{substitute } \{v \mapsto A\}) \vartheta)$ 
    using False and True by auto
  ultimately show ?thesis
    using * and surj-pair[of w] by fastforce
  qed
next
case False
then have  $v \notin \text{free-vars } (FAbs\ w\ (\mathbf{S} \vartheta\ B))$ 
  using surj-pair[of w] by fastforce

```



```

then have **:  $\mathbf{S} \{v \mapsto A\} (FAbs\ w\ (\mathbf{S}\ \vartheta\ B)) = FAbs\ w\ (\mathbf{S}\ \vartheta\ B)$ 
  using free-var-singleton-substitution-neutrality by blast
also have ... =  $FAbs\ w\ (\mathbf{S}\ \vartheta' B)$ 
proof -
{
  fix  $v'$ 
  assume  $v' \in fmdom'\ \vartheta$ 
  with FAbs.premis(1) have  $v' \neq v$ 
    by blast
  assume  $v \in free\ vars\ (\vartheta\ \$\$!\ v')$  and  $v' \in free\ vars\ B$ 
  with  $\langle v' \neq v \rangle$  have  $\neg is\ free\ for\ (\vartheta\ \$\$!\ v')\ v' (FAbs\ v\ B)$ 
    using form-with-free-binder-not-free-for by blast
  with FAbs.premis(2) and  $\langle v' \in fmdom'\ \vartheta \rangle$  and False have False
    by blast
}
then have  $\forall v' \in fmdom'\ \vartheta. v \notin free\ vars\ (\vartheta\ \$\$!\ v') \vee v' \notin free\ vars\ B$ 
  by blast
then have  $\forall v' \in fmdom'\ \vartheta. v' \in free\ vars\ B \longrightarrow \mathbf{S} \{v \mapsto A\} (\vartheta\ \$\$!\ v') = \vartheta\ \$\$!\ v'$ 
  using free-var-singleton-substitution-neutrality by blast
then have  $\forall v' \in free\ vars\ B. \vartheta\ \$\$!\ v' = \vartheta'\ \$\$!\ v'$ 
  by (metis fmdom'-map fmdom'-notD fmdom'-notI fmlookup-map option.map-sel)
then have  $\mathbf{S}\ \vartheta\ B = \mathbf{S}\ \vartheta' B$ 
  using free-vars-agreement-substitution-equality by (metis IntD1 fmdom'-map)
then show ?thesis
  by simp
qed
also from False and FAbs.premis(1) have ... =  $FAbs\ w\ (\mathbf{S}\ (fmdrop\ w\ (\{v \mapsto A\} ++_f\ \vartheta'))\ B)$ 
  by (simp add: fmdrop-fmupd-same fmdrop-idle')
also from False have ... =  $\mathbf{S}\ (\{v \mapsto A\} ++_f\ \vartheta') (FAbs\ w\ B)$ 
  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  $\vartheta$ 
  and  $v \notin vars\ (fmrans'\ \vartheta)$ 
  shows  $v \notin vars\ (fmrans'\ (fmdrop\ w\ \vartheta))$ 
using assms proof (induction  $\vartheta$ )
  case fmempty
  then show ?case
    by simp
next
  case (fmupd  $v'\ A\ \vartheta$ )
  from fmdom'-notI[OF fmupd.hyps] and fmupd.premis(1) have is-substitution  $\vartheta$ 
    by (rule updated-substitution-is-substitution)
  moreover from fmupd.premis(2) and fmupd.hyps have  $v \notin vars\ (fmrans'\ \vartheta)$ 

```

```

  by simp
  ultimately have  $v \notin \text{vars } (f\text{mran}' (f\text{mdrop } w \vartheta))$ 
  by (rule fmupd.IH)
  with fmupd.hyps and fmupd.prem(2) show ?case
  by (simp add: fmdrop-fmupd)
qed

lemma excluded-var-from-substitution:
  assumes is-substitution  $\vartheta$ 
  and  $v \notin f\text{mdom}' \vartheta$ 
  and  $v \notin \text{vars } (f\text{mran}' \vartheta)$ 
  and  $v \notin \text{vars } A$ 
  shows  $v \notin \text{vars } (\mathbf{S} \vartheta A)$ 
using assms proof (induction A arbitrary:  $\vartheta$ )
  case (FVar  $v'$ )
  then show ?case
  proof (cases  $v' \in f\text{mdom}' \vartheta$ )
    case True
    then have  $\vartheta \ \$\$! \ v' \in f\text{mran}' \vartheta$ 
    by (simp add: fmlookup-dom'-iff fmrans'I)
    with FVar(3) have  $v \notin \text{vars } (\vartheta \ \$\$! \ v')$ 
    by simp
    with True show ?thesis
    using surj-pair[of  $v'$ ] and fmdom'-notI by force
  next
    case False
    with FVar.prem(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  $v \notin \text{vars } B$  and  $v \neq w$ 
  using surj-pair[of  $w$ ] and FAbs.prem(4) by fastforce+
  then show ?case
  proof (cases  $w \notin f\text{mdom}' \vartheta$ )
    case True
    then have  $\mathbf{S} \vartheta (FAbs \ w \ B) = FAbs \ w \ (\mathbf{S} \vartheta \ B)$ 
    using surj-pair[of  $w$ ] by fastforce
    moreover from FAbs.IH have  $v \notin \text{vars } (\mathbf{S} \vartheta \ B)$ 
    using FAbs.prem(1-3) and  $\langle v \notin \text{vars } B \rangle$  by blast
    ultimately show ?thesis
  
```

```

    using ⟨v ≠ w⟩ and surj-pair[of w] by fastforce
next
case False
then have S ∅ (FAbs w B) = FAbs w (S (fmdrop w ∅) B)
  using surj-pair[of w] by fastforce
moreover have v ∉ vars (S (fmdrop w ∅) B)
proof -
  from FAbs.prem1 have is-substitution (fmdrop w ∅)
  by fastforce
  moreover from FAbs.prem2 have v ∉ fmdom' (fmdrop w ∅)
  by simp
  moreover from FAbs.prem3 have v ∉ vars (fmran' (fmdrop w ∅))
  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 (xα) = xα
| 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 (λxα. B) =
  (
    if (x, α) = v then
      λyα. S {(x, α) ↦ yα} (rename-bound-var v y B)
    else
      λxα. (rename-bound-var v y B)
  )

```

lemma rename-bound-var-preserves-typing:

```

assumes A ∈ wffsα
shows rename-bound-var (y, γ) z A ∈ wffsα
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) ↦ zγ} (rename-bound-var (y, γ) z A) ∈ wffsβ
  using substitution-preserves-typing by (simp add: wffs-of-type-intros(1))
then have λzγ. S {(y, γ) ↦ zγ} (rename-bound-var (y, γ) z A) ∈ wffsγ→β
  by blast
with True show ?thesis
by simp

```

```

next
  case False
  from abs-is-wff.IH have  $\lambda x_\delta. \text{rename-bound-var } (y, \gamma) z A \in \text{wffs}_{\delta \rightarrow \beta}$ 
  by blast
  with False show ?thesis
  by auto
qed
qed auto

```

```

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

```

```

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

```

```

lemma old-bound-var-not-free-after-renaming:
  assumes  $A \in \text{wffs}_\alpha$ 
  and  $z_\gamma \neq y_\gamma$ 
  and  $(z, \gamma) \notin \text{vars } A$ 
  and  $(y, \gamma) \notin \text{free-vars } A$ 
  shows  $(y, \gamma) \notin \text{free-vars } (\text{rename-bound-var } (y, \gamma) z A)$ 
using assms proof induction
  case (abs-is-wff  $\beta A \alpha x$ )
  then show ?case
  proof (cases  $(x, \alpha) = (y, \gamma)$ )
    case True
    with abs-is-wff.hyps and abs-is-wff.premis(2) show ?thesis
    using old-bound-var-not-free-in-abs-after-renaming by auto
  next
    case False
    with abs-is-wff.premis(2,3) and assms(2) show ?thesis
    using abs-is-wff.IH by force
  qed
qed fastforce+

```

```

lemma old-bound-var-not-occurring-after-renaming:
  assumes  $A \in \text{wffs}_\alpha$ 
  and  $z_\gamma \neq y_\gamma$ 
  shows  $\neg \text{occurs-at } (y, \gamma) p (\mathbf{S} \{(y, \gamma) \mapsto z_\gamma\} (\text{rename-bound-var } (y, \gamma) z A))$ 
using assms(1) proof (induction A arbitrary: p)
  case (var-is-wff  $\alpha x$ )

```

```

from assms(2) show ?case
  using subform-size-decrease by (cases ( $x, \alpha = (y, \gamma)$ ) fastforce+)
next
  case (con-is-wff  $\alpha c$ )
  then show ?case
    using occurs-at-alt-def(2) by auto
next
  case (app-is-wff  $\alpha \beta A B$ )
  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  $\beta A \alpha x$ )
  then show ?case
  proof (cases  $p$ )
    case (Cons  $d p'$ )
    then show ?thesis
  proof (cases  $d$ )
    case Left
    have  $*$ :  $\neg$  occurs-at ( $y, \gamma$ )  $p$  ( $\lambda x_\alpha. \mathbf{S} \{(y, \gamma) \mapsto z_\gamma\}$  (rename-bound-var ( $y, \gamma$ )  $z A$ ))
      for  $x$  and  $\alpha$ 
      using Left and Cons and abs-is-wff.IH by simp
    then show ?thesis
  proof (cases ( $x, \alpha = (y, \gamma)$ ))
    case True
    with assms(2) have
       $\mathbf{S} \{(y, \gamma) \mapsto z_\gamma\}$  (rename-bound-var ( $y, \gamma$ )  $z$  ( $\lambda x_\alpha. A$ ))
      =
       $\lambda z_\gamma. \mathbf{S} \{(y, \gamma) \mapsto z_\gamma\}$  (rename-bound-var ( $y, \gamma$ )  $z A$ )
      using free-var-in-renaming-substitution and free-var-singleton-substitution-neutrality
      by simp
    moreover have  $\neg$  occurs-at ( $y, \gamma$ )  $p$  ( $\lambda z_\gamma. \mathbf{S} \{(y, \gamma) \mapsto z_\gamma\}$  (rename-bound-var ( $y, \gamma$ )  $z A$ ))
      using Left and Cons and  $*$  by simp
    ultimately show ?thesis
      by simp
  next
  case False
  with assms(2) have
     $\mathbf{S} \{(y, \gamma) \mapsto z_\gamma\}$  (rename-bound-var ( $y, \gamma$ )  $z$  ( $\lambda x_\alpha. A$ ))
    =
     $\lambda x_\alpha. \mathbf{S} \{(y, \gamma) \mapsto z_\gamma\}$  (rename-bound-var ( $y, \gamma$ )  $z A$ )
    by simp
  moreover have  $\neg$  occurs-at ( $y, \gamma$ )  $p$  ( $\lambda x_\alpha. \mathbf{S} \{(y, \gamma) \mapsto z_\gamma\}$  (rename-bound-var ( $y, \gamma$ )  $z A$ ))
    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  $A \in \text{wffs}_\alpha$ 
  and  $z_\gamma \neq y_\gamma$ 
  and  $(z, \gamma) \notin \text{vars } A$ 
  and  $\text{occurs-at } (y, \gamma) p (\text{rename-bound-var } (y, \gamma) z A)$ 
  shows  $\neg \text{in-scope-of-abs } (z, \gamma) p (\text{rename-bound-var } (y, \gamma) z A)$ 
using assms(1,3,4) proof (induction arbitrary: p)
  case (var-is-wff  $\alpha x$ )
  then show ?case
    by (simp add: subforms-from-var(2))
next
  case (con-is-wff  $\alpha c$ )
  then show ?case
    using occurs-at-alt-def(2) by auto
next
  case (app-is-wff  $\alpha \beta B C$ )
  from app-is-wff.prem(1) have  $(z, \gamma) \notin \text{vars } B$  and  $(z, \gamma) \notin \text{vars } C$ 
    by simp-all
  from app-is-wff.prem(2)
  have  $\text{occurs-at } (y, \gamma) p (\text{rename-bound-var } (y, \gamma) z B \cdot \text{rename-bound-var } (y, \gamma) z C)$ 
    by simp
  then consider
    (a)  $\exists p'. p = \ll \# p' \wedge \text{occurs-at } (y, \gamma) p' (\text{rename-bound-var } (y, \gamma) z B)$ 
  | (b)  $\exists p'. p = \gg \# p' \wedge \text{occurs-at } (y, \gamma) p' (\text{rename-bound-var } (y, \gamma) z C)$ 
    using subforms-from-app by force
  then show ?case
proof cases
  case a
  then obtain  $p'$  where  $p = \ll \# p'$  and  $\text{occurs-at } (y, \gamma) p' (\text{rename-bound-var } (y, \gamma) z B)$ 
    by blast
  then have  $\neg \text{in-scope-of-abs } (z, \gamma) p' (\text{rename-bound-var } (y, \gamma) z B)$ 
    using app-is-wff.IH(1)[OF <(z, \gamma) \notin vars B] by blast
  then have  $\neg \text{in-scope-of-abs } (z, \gamma) p (\text{rename-bound-var } (y, \gamma) z (B \cdot C))$  for  $C$ 
    using  $\langle p = \ll \# p' \rangle$  and in-scope-of-abs-in-left-app by simp
  then show ?thesis
    by blast
next
  case b
  then obtain  $p'$  where  $p = \gg \# p'$  and  $\text{occurs-at } (y, \gamma) p' (\text{rename-bound-var } (y, \gamma) z C)$ 
    by blast
  then have  $\neg \text{in-scope-of-abs } (z, \gamma) p' (\text{rename-bound-var } (y, \gamma) z C)$ 
    using app-is-wff.IH(2)[OF <(z, \gamma) \notin vars C] by blast

```

then have \neg *in-scope-of-abs* $(z, \gamma) p$ (*rename-bound-var* $(y, \gamma) z (B \cdot C)$) **for** B
using $\langle p = \rangle \# p'$ **and** *in-scope-of-abs-in-right-app* **by** *simp*
then show *?thesis*
by *blast*
qed
next
case (*abs-is-wff* $\beta A \alpha x$)
from *abs-is-wff.premis(1)* **have** $(z, \gamma) \notin \text{vars } A$ **and** $(z, \gamma) \neq (x, \alpha)$
by *fastforce+*
then show *?case*
proof (*cases* $(y, \gamma) = (x, \alpha)$)
case *True*
then have *occurs-at* $(y, \gamma) p$ $(\lambda z\gamma. \mathbf{S} \{(y, \gamma) \mapsto z\gamma\} (\text{rename-bound-var } (y, \gamma) z A))$
using *abs-is-wff.premis(2)* **by** *simp*
moreover have \neg *occurs-at* $(y, \gamma) p$ $(\lambda z\gamma. \mathbf{S} \{(y, \gamma) \mapsto z\gamma\} (\text{rename-bound-var } (y, \gamma) z A))$
using *old-bound-var-not-occurring-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* $(y, \gamma) z (\lambda x\alpha. A) = \lambda x\alpha. \text{rename-bound-var } (y, \gamma) z A$
by *auto*
with *abs-is-wff.premis(2)* **have** *occurs-at* $(y, \gamma) p$ $(\lambda x\alpha. \text{rename-bound-var } (y, \gamma) z A)$
by *auto*
then obtain p' **where** $p = \langle \# p' \rangle$ **and** *occurs-at* $(y, \gamma) p'$ $(\text{rename-bound-var } (y, \gamma) z A)$
using *subforms-from-abs* **by** *fastforce*
then have \neg *in-scope-of-abs* $(z, \gamma) p'$ $(\text{rename-bound-var } (y, \gamma) z A)$
using *abs-is-wff.IH[OF \langle(z, \gamma) \notin \text{vars } A\rangle]* **by** *blast*
then have \neg *in-scope-of-abs* $(z, \gamma) (\langle \# p' \rangle (\lambda x\alpha. \text{rename-bound-var } (y, \gamma) z A))$
using $\langle p = \langle \# p' \rangle \rangle$ **and** *in-scope-of-abs-in-abs* **and** $\langle (z, \gamma) \neq (x, \alpha) \rangle$ **by** *auto*
then show *?thesis*
using $*$ **and** $\langle p = \langle \# p' \rangle \rangle$ **by** *simp*
qed
qed

lemma *is-free-for-in-rename-bound-var*:
assumes $A \in \text{wffs}_\alpha$
and $z\gamma \neq y\gamma$
and $(z, \gamma) \notin \text{vars } A$
shows *is-free-for* $(z\gamma) (y, \gamma) (\text{rename-bound-var } (y, \gamma) z A)$
proof (*rule ccontr*)
assume \neg *is-free-for* $(z\gamma) (y, \gamma) (\text{rename-bound-var } (y, \gamma) z A)$
then obtain p
where *is-free-at* $(y, \gamma) p$ $(\text{rename-bound-var } (y, \gamma) z A)$
and *in-scope-of-abs* $(z, \gamma) p$ $(\text{rename-bound-var } (y, \gamma) z A)$
by *force*
then show *False*
using *rename-bound-var-not-bound-occurrences[OF assms]* **by** *fastforce*

qed

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

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

lemma *old-var-not-free-not-occurring-after-rename*:
 assumes $A \in \text{wffs}_\alpha$
 and $z_\gamma \neq y_\gamma$
 and $(y, \gamma) \notin \text{free-vars } A$
 and $(z, \gamma) \notin \text{vars } A$
 shows $(y, \gamma) \notin \text{vars } (\text{rename-bound-var } (y, \gamma) z A)$
 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* :: $\text{bool} \Rightarrow V$ **where**
 $\text{bool-to-V} = (\text{SOME } f. \text{inj } f)$

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* :: $V \Rightarrow \text{bool}$ **where**
 [*simp*]: $\text{bool-from-V} = \text{inv } \text{bool-to-V}$

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

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

definition *bottom* :: V (\mathbf{F}) **where**

[simp]: $\mathbf{F} = \text{bool-to-}V \text{ False}$

definition *two-valued-boolean-algebra-universe* :: V (\mathbb{B}) **where**

[simp]: $\mathbb{B} = \text{set } \{\mathbf{T}, \mathbf{F}\}$

definition *negation* :: $V \Rightarrow V$ (\sim - [141] 141) **where**

[simp]: $\sim p = \text{bool-to-}V (\neg \text{bool-from-}V p)$

definition *conjunction* :: $V \Rightarrow V \Rightarrow V$ (**infixr** \wedge 136) **where**

[simp]: $p \wedge q = \text{bool-to-}V (\text{bool-from-}V p \wedge \text{bool-from-}V q)$

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 $p \in \text{elts } \mathbb{B}$ **and** $q \in \text{elts } \mathbb{B}$ **and** $r \in \text{elts } \mathbb{B}$

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

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

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

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

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

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

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

lemma *pwffs-subset-of-wffso*:
shows $\text{pwffs} \subseteq \text{wffso}$

proof

fix A

assume $A \in \text{pwffs}$

then show $A \in \text{wffso}$

by *induction auto*

qed

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

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

lemma *pwffs-free-vars-are-propositional*:

assumes $A \in \text{pwffs}$
and $v \in \text{free-vars } A$
obtains p **where** $v = (p, o)$
using *assms* **by** (*induction A arbitrary: thesis*) *auto*

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

assumes $A \in \text{pwffs}$
and $v \in \text{free-vars } A$
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.prem*s **consider**
 (a) $v \in \text{free-vars } C$ **and** $v \in \text{free-vars } D$
 | (b) $v \in \text{free-vars } C$ **and** $v \notin \text{free-vars } D$
 | (c) $v \notin \text{free-vars } C$ **and** $v \in \text{free-vars } D$
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)  $v \in \text{free-vars } C$  and  $v \in \text{free-vars } D$ 
| (b)  $v \in \text{free-vars } C$  and  $v \notin \text{free-vars } D$ 
| (c)  $v \notin \text{free-vars } C$  and  $v \in \text{free-vars } D$ 
  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)  $v \in \text{free-vars } C$  and  $v \in \text{free-vars } D$ 
| (b)  $v \in \text{free-vars } C$  and  $v \notin \text{free-vars } D$ 
| (c)  $v \notin \text{free-vars } C$  and  $v \in \text{free-vars } D$ 
    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 (equiv-pwff C D)
from equiv-pwff.premis consider
  (a) v ∈ free-vars C and v ∈ free-vars D
| (b) v ∈ free-vars C and v ∉ free-vars D
| (c) v ∉ free-vars C and v ∈ free-vars D
  by auto
then show ?case
proof cases
  case a
  then show ?thesis
    using equiv-pwff.IH by (intro is-free-for-in-equivalence)
next
  case b
  have is-free-for B v C
    by (fact equiv-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 equiv-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* :: $(\text{nat} \Rightarrow V) \Rightarrow \text{bool}$ **where**
[iff]: *is-tv-assignment* $\varphi \longleftrightarrow (\forall p. \varphi p \in \text{elts } \mathbb{B})$

Denotation of a pwff:

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

```

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

```

lemma *pwff-denotation-is-truth-value*:

```

assumes  $A \in \text{pwffs}$ 
and is-tv-assignment  $\varphi$ 
and is-pwff-denotation-function  $\mathcal{V}$ 
shows  $\mathcal{V} \varphi A \in \text{elts } \mathbb{B}$ 
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 (equiv-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 equiv-pwff.IH by auto
qed (use assms(2,3) in auto)

```


lemma *closed-pwff-is-meaningful-regardless-of-assignment*:

assumes $A \in \text{pwffs}$
and $\text{free-vars } A = \{\}$
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 $\dots = \mathcal{V} \psi T_o$
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 $\dots = \mathcal{V} \psi F_o$
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 $\langle \text{free-vars } (\sim^Q A) = \{\} \rangle$ **have** $\text{free-vars } A = \{\}$
by *simp*
have $\mathcal{V} \varphi (\sim^Q A) = \sim \mathcal{V} \varphi A$
using *assms(3,5)* **and** *neg-pwff.hyps* **by** *auto*
also from $\langle \text{free-vars } A = \{\} \rangle$ **have** $\dots = \sim \mathcal{V} \psi A$
using *assms(3-5)* **and** *neg-pwff.IH* **by** *presburger*
also have $\dots = \mathcal{V} \psi (\sim^Q A)$
using *assms(4,5)* **and** *neg-pwff.hyps* **by** *simp*
finally show *?case* .

next
case (*conj-pwff* $A B$)
from $\langle \text{free-vars } (A \wedge^Q B) = \{\} \rangle$ **have** $\text{free-vars } A = \{\}$ **and** $\text{free-vars } B = \{\}$
by *simp-all*
have $\mathcal{V} \varphi (A \wedge^Q B) = \mathcal{V} \varphi A \wedge \mathcal{V} \varphi B$
using *assms(3,5)* **and** *conj-pwff.hyps(1,2)* **by** *auto*
also from $\langle \text{free-vars } A = \{\} \rangle$ **and** $\langle \text{free-vars } B = \{\} \rangle$ **have** $\dots = \mathcal{V} \psi A \wedge \mathcal{V} \psi B$
using *conj-pwff.IH(1,2)* **by** *presburger*
also have $\dots = \mathcal{V} \psi (A \wedge^Q B)$
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 ∨Q B) = {}⟩ have free-vars A = {} and free-vars B = {}
  by simp-all
have V φ (A ∨Q B) = V φ A ∨ V φ B
  using assms(3,5) and disj-pwff.hyps(1,2) by auto
also from ⟨free-vars A = {}⟩ and ⟨free-vars B = {}⟩ have ... = V ψ A ∨ V ψ B
  using disj-pwff.IH(1,2) by presburger
also have ... = V ψ (A ∨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 ⊃Q B) = {}⟩ have free-vars A = {} and free-vars B = {}
  by simp-all
have V φ (A ⊃Q B) = V φ A ⊃ V φ B
  using assms(3,5) and imp-pwff.hyps(1,2) by auto
also from ⟨free-vars A = {}⟩ and ⟨free-vars B = {}⟩ have ... = V ψ A ⊃ V ψ B
  using imp-pwff.IH(1,2) by presburger
also have ... = V ψ (A ⊃Q B)
  using assms(4,5) and imp-pwff.hyps(1,2) by fastforce
finally show ?case .
next
case (equiv-pwff A B)
from ⟨free-vars (A ≡Q B) = {}⟩ have free-vars A = {} and free-vars B = {}
  by simp-all
have V φ (A ≡Q B) = V φ A ≡ V φ B
  using assms(3,5) and equiv-pwff.hyps(1,2) by auto
also from ⟨free-vars A = {}⟩ and ⟨free-vars B = {}⟩ have ... = V ψ A ≡ V ψ B
  using equiv-pwff.IH(1,2) by presburger
also have ... = V ψ (A ≡Q B)
  using assms(4,5) and equiv-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  $\varphi$   $T_o$   $\mathbf{T}$ 
|  $\mathcal{V}_B$ -graph-F:  $\mathcal{V}_B$ -graph  $\varphi$   $F_o$   $\mathbf{F}$ 
|  $\mathcal{V}_B$ -graph-var:  $\mathcal{V}_B$ -graph  $\varphi$  ( $p_o$ ) ( $\varphi$   $p$ )
|  $\mathcal{V}_B$ -graph-neg:  $\mathcal{V}_B$ -graph  $\varphi$  ( $\sim^Q A$ ) ( $\sim b_A$ ) if  $\mathcal{V}_B$ -graph  $\varphi$  A  $b_A$ 
|  $\mathcal{V}_B$ -graph-conj:  $\mathcal{V}_B$ -graph  $\varphi$  (A ∧Q B) ( $b_A$  ∧  $b_B$ ) 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 ∨Q B) ( $b_A$  ∨  $b_B$ ) 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 ⊃Q B) ( $b_A$  ⊃  $b_B$ ) if  $\mathcal{V}_B$ -graph  $\varphi$  A  $b_A$  and  $\mathcal{V}_B$ -graph  $\varphi$  B  $b_B$ 
|  $\mathcal{V}_B$ -graph-equiv:  $\mathcal{V}_B$ -graph  $\varphi$  (A ≡Q B) ( $b_A$  ≡  $b_B$ ) if  $\mathcal{V}_B$ -graph  $\varphi$  A  $b_A$  and  $\mathcal{V}_B$ -graph  $\varphi$  B  $b_B$  and A
≠  $F_o$ 

```

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  $\varphi$ 

```

```

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

```

```

lemma  $\mathcal{V}_B\text{-graph-denotation-uniqueness}$ :
  assumes  $A \in \text{pwffs}$ 
  and is-tv-assignment  $\varphi$ 
  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  $\mathcal{V}_B\text{-graph-T}$ 
    from  $\langle \mathcal{V}_B\text{-graph } \varphi \ T_o \ b' \rangle$  show ?case
    by (cases rule:  $\mathcal{V}_B\text{-graph.cases}$ ) simp-all
next
  case  $\mathcal{V}_B\text{-graph-F}$ 
    from  $\langle \mathcal{V}_B\text{-graph } \varphi \ F_o \ b' \rangle$  show ?case
    by (cases rule:  $\mathcal{V}_B\text{-graph.cases}$ ) simp-all
next
  case ( $\mathcal{V}_B\text{-graph-var } p$ )
    from  $\langle \mathcal{V}_B\text{-graph } \varphi \ (p_o) \ b' \rangle$  show ?case
    by (cases rule:  $\mathcal{V}_B\text{-graph.cases}$ ) simp-all
next
  case ( $\mathcal{V}_B\text{-graph-neg } A \ b_A$ )
    with  $\langle \mathcal{V}_B\text{-graph } \varphi \ (\sim^{\mathcal{Q}} A) \ b' \rangle$  have  $\mathcal{V}_B\text{-graph } \varphi \ A \ (\sim b')$ 
    proof (cases rule:  $\mathcal{V}_B\text{-graph.cases}$ )
      case ( $\mathcal{V}_B\text{-graph-neg } A' \ b_A$ )
        from  $\langle \sim^{\mathcal{Q}} A = \sim^{\mathcal{Q}} A' \rangle$  have  $A = A'$ 
        by simp
      with  $\langle \mathcal{V}_B\text{-graph } \varphi \ A' \ b_A \rangle$  have  $\mathcal{V}_B\text{-graph } \varphi \ A \ b_A$ 
      by simp

```

moreover have $b_A = \sim b'$
proof –
 have $b_A \in \text{elts } \mathbb{B}$
 by (fact $\mathcal{V}_B\text{-graph-denotation-is-truth-value}[OF \ \mathcal{V}_B\text{-graph-neg}(3) \ \text{assms}(2)]$)
moreover from $\langle b_A \in \text{elts } \mathbb{B} \rangle$ **and** $\mathcal{V}_B\text{-graph-neg}(2)$ **have** $\sim b' \in \text{elts } \mathbb{B}$
 by *fastforce*
 ultimately show *?thesis*
 using $\mathcal{V}_B\text{-graph-neg}(2)$ **by** *fastforce*
qed
 ultimately show *?thesis*
 by *blast*
qed *simp-all*
moreover from $\mathcal{V}_B\text{-graph-neg.prem}(1)$ **have** $A \in \text{pwffs}$
 by (force *elim: pwffs.cases*)
moreover have $b_A \in \text{elts } \mathbb{B}$ **and** $b' \in \text{elts } \mathbb{B}$ **and** $b_A = \sim b'$
proof –
 show $b_A \in \text{elts } \mathbb{B}$
 by (fact $\mathcal{V}_B\text{-graph-denotation-is-truth-value}[OF \ \langle \mathcal{V}_B\text{-graph } \varphi \ A \ b_A \rangle \ \text{assms}(2)]$)
 show $b' \in \text{elts } \mathbb{B}$
 by (fact $\mathcal{V}_B\text{-graph-denotation-is-truth-value}[OF \ \langle \mathcal{V}_B\text{-graph } \varphi \ (\sim^Q A) \ b' \rangle \ \text{assms}(2)]$)
 show $b_A = \sim b'$
 by (fact $\mathcal{V}_B\text{-graph-neg}(2)[OF \ \langle A \in \text{pwffs} \rangle \ \langle \mathcal{V}_B\text{-graph } \varphi \ A \ (\sim b') \rangle]$)
qed
 ultimately show *?case*
 by *force*
next
 case ($\mathcal{V}_B\text{-graph-conj } A \ b_A \ B \ b_B$)
 with $\langle \mathcal{V}_B\text{-graph } \varphi \ (A \wedge^Q B) \ b' \rangle$ **obtain** $b_{A'}$ **and** $b_{B'}$
 where $b' = b_{A'} \wedge b_{B'}$ **and** $\mathcal{V}_B\text{-graph } \varphi \ A \ b_{A'}$ **and** $\mathcal{V}_B\text{-graph } \varphi \ B \ b_{B'}$
 by (*cases rule: $\mathcal{V}_B\text{-graph.cases}$*) *simp-all*
 moreover have $A \in \text{pwffs}$ **and** $B \in \text{pwffs}$
 using *pwffs-from-conj-pwff*[$OF \ \mathcal{V}_B\text{-graph-conj.prem}(1)$] **by** *blast+*
 ultimately show *?case*
 using $\mathcal{V}_B\text{-graph-conj.IH}$ **and** $\mathcal{V}_B\text{-graph-conj.prem}(2)$ **by** *blast*
next
 case ($\mathcal{V}_B\text{-graph-disj } A \ b_A \ B \ b_B$)
 from $\langle \mathcal{V}_B\text{-graph } \varphi \ (A \vee^Q B) \ b' \rangle$ **obtain** $b_{A'}$ **and** $b_{B'}$
 where $b' = b_{A'} \vee b_{B'}$ **and** $\mathcal{V}_B\text{-graph } \varphi \ A \ b_{A'}$ **and** $\mathcal{V}_B\text{-graph } \varphi \ B \ b_{B'}$
 by (*cases rule: $\mathcal{V}_B\text{-graph.cases}$*) *simp-all*
 moreover have $A \in \text{pwffs}$ **and** $B \in \text{pwffs}$
 using *pwffs-from-disj-pwff*[$OF \ \mathcal{V}_B\text{-graph-disj.prem}(1)$] **by** *blast+*
 ultimately show *?case*
 using $\mathcal{V}_B\text{-graph-disj.IH}$ **and** $\mathcal{V}_B\text{-graph-disj.prem}(2)$ **by** *blast*
next
 case ($\mathcal{V}_B\text{-graph-imp } A \ b_A \ B \ b_B$)
 from $\langle \mathcal{V}_B\text{-graph } \varphi \ (A \supset^Q B) \ b' \rangle$ **obtain** $b_{A'}$ **and** $b_{B'}$
 where $b' = b_{A'} \supset b_{B'}$ **and** $\mathcal{V}_B\text{-graph } \varphi \ A \ b_{A'}$ **and** $\mathcal{V}_B\text{-graph } \varphi \ B \ b_{B'}$
 by (*cases rule: $\mathcal{V}_B\text{-graph.cases}$*) *simp-all*
 moreover have $A \in \text{pwffs}$ **and** $B \in \text{pwffs}$

```

    using pwffs-from-imp-pwff[OF  $\mathcal{V}_B$ -graph-imp.prem(1)] by blast+
    ultimately show ?case
    using  $\mathcal{V}_B$ -graph-imp.IH and  $\mathcal{V}_B$ -graph-imp.prem(2) by blast
next
case ( $\mathcal{V}_B$ -graph-equiv A bA B bB)
with  $\langle \mathcal{V}_B$ -graph  $\varphi$  (A  $\equiv^Q$  B) b'  $\rangle$  obtain bA' and bB'
  where b' = bA'  $\equiv$  bB' and  $\mathcal{V}_B$ -graph  $\varphi$  A bA' and  $\mathcal{V}_B$ -graph  $\varphi$  B bB'
  by (cases rule:  $\mathcal{V}_B$ -graph.cases) simp-all
moreover have A  $\in$  pwffs and B  $\in$  pwffs
  using pwffs-from-equiv-pwff[OF  $\mathcal{V}_B$ -graph-equiv.prem(1)] by blast+
  ultimately show ?case
  using  $\mathcal{V}_B$ -graph-equiv.IH and  $\mathcal{V}_B$ -graph-equiv.prem(2) by blast
qed

```

```

lemma  $\mathcal{V}_B$ -graph-denotation-existence:
  assumes A  $\in$  pwffs
  and is-tv-assignment  $\varphi$ 
  shows  $\exists b. \mathcal{V}_B$ -graph  $\varphi$  A b
using assms proof induction
  case (equiv-pwff A B)
  then obtain bA and bB where  $\mathcal{V}_B$ -graph  $\varphi$  A bA and  $\mathcal{V}_B$ -graph  $\varphi$  B bB
  by blast
  then show ?case
  proof (cases A  $\neq$  Fo)
  case True
  then show ?thesis
  using equiv-pwff.IH and equiv-pwff.prem by blast
  next
  case False
  then have A = Fo
  by blast
  then show ?thesis
  using  $\mathcal{V}_B$ -graph-neg[OF  $\langle \mathcal{V}_B$ -graph  $\varphi$  B bB  $\rangle$ ] by auto
qed
qed blast+

```

```

lemma  $\mathcal{V}_B$ -graph-is-functional:
  assumes A  $\in$  pwffs
  and is-tv-assignment  $\varphi$ 
  shows  $\exists! b. \mathcal{V}_B$ -graph  $\varphi$  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]:  $\mathcal{V}_B \varphi A = (THE b. \mathcal{V}_B$ -graph  $\varphi$  A b)

```

```

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

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

lemma $\mathcal{V}_B\text{-equiv}$ [$\mathcal{V}_B\text{-simps}$]:
assumes $A \in \text{pwffs}$ **and** $B \in \text{pwffs}$
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 $\mathcal{V}_B\text{-F}$ [*OF assms(3)*] **and** $\mathcal{V}_B\text{-neg}$ [*OF assms(2,3)*] **by force**
next
case *False*
from *assms(1,3)* **have** $\mathcal{V}_B\text{-graph } \varphi A (\mathcal{V}_B \varphi A)$
by (*intro* $\mathcal{V}_B\text{-graph-}\mathcal{V}_B$)
moreover from *assms(2,3)* **have** $\mathcal{V}_B\text{-graph } \varphi B (\mathcal{V}_B \varphi B)$
by (*intro* $\mathcal{V}_B\text{-graph-}\mathcal{V}_B$)
ultimately have $\mathcal{V}_B\text{-graph } \varphi (A \equiv^{\mathcal{Q}} B) (\mathcal{V}_B \varphi A \equiv \mathcal{V}_B \varphi B)$
using *False* **by** (*intro* $\mathcal{V}_B\text{-graph-equiv}$)
with *assms* **show** *?thesis*
using *equiv-pwff* **by** (*intro* $\mathcal{V}_B\text{-equality}$)
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* $A \iff A \in \text{pwffs} \wedge (\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi A = \mathbf{T})$

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

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

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

```

```

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

```

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

lemma *closed-pwff-substitution-free-vars*:

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

finally have *: $\text{free-vars } (\mathbf{S} \ ?\vartheta (C \supset^{\mathcal{Q}} D)) = \text{free-vars } (\mathbf{S} \ ?\vartheta C) \cup \text{free-vars } (\mathbf{S} \ ?\vartheta D)$.
from *imp-pwff.prem*s **consider**
| (a) $(p, o) \in \text{free-vars } C$ **and** $(p, o) \in \text{free-vars } D$
| (b) $(p, o) \in \text{free-vars } C$ **and** $(p, o) \notin \text{free-vars } D$
| (c) $(p, o) \notin \text{free-vars } C$ **and** $(p, o) \in \text{free-vars } D$
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 $\text{free-vars } (\mathbf{S} \ ?\vartheta (C \equiv^{\mathcal{Q}} D)) = \text{free-vars } ((\mathbf{S} \ ?\vartheta C) \equiv^{\mathcal{Q}} (\mathbf{S} \ ?\vartheta D))$
by *simp*
also have $\dots = \text{free-vars } (\mathbf{S} \ ?\vartheta C) \cup \text{free-vars } (\mathbf{S} \ ?\vartheta D)$
by (*fact eqv-fv*)
finally have *: $\text{free-vars } (\mathbf{S} \ ?\vartheta (C \equiv^{\mathcal{Q}} D)) = \text{free-vars } (\mathbf{S} \ ?\vartheta C) \cup \text{free-vars } (\mathbf{S} \ ?\vartheta D)$.
from *eqv-pwff.prem*s **consider**
| (a) $(p, o) \in \text{free-vars } C$ **and** $(p, o) \in \text{free-vars } D$
| (b) $(p, o) \in \text{free-vars } C$ **and** $(p, o) \notin \text{free-vars } D$
| (c) $(p, o) \notin \text{free-vars } C$ **and** $(p, o) \in \text{free-vars } D$
by *auto*
from *this* **and** * **and** *eqv-pwff.IH* **show** ?*case*
using *free-var-singleton-substitution-neutrality* **by** *cases auto*
qed (*use assms*(β) **in** $\langle \text{force+} \rangle$)

Substitution in a pwff:

definition *is-pwff-substitution* **where**

[*iff*]: *is-pwff-substitution* $\vartheta \longleftrightarrow \text{is-substitution } \vartheta \wedge (\forall (x, \alpha) \in \text{fmdom}' \vartheta. \alpha = o)$

Tautologous pwff:

definition *is-tautologous* :: *form* \Rightarrow *bool* **where**

[*iff*]: *is-tautologous* $B \longleftrightarrow (\exists \vartheta A. \text{is-tautology } A \wedge \text{is-pwff-substitution } \vartheta \wedge B = \mathbf{S} \ \vartheta \ A)$

lemma *tautologous-is-wffo*:

assumes *is-tautologous* A

shows $A \in \text{wffs}_o$

using *assms* **and** *substitution-preserves-typing* **and** *tautology-is-wffo* **by** *blast*

lemma *implication-reflexivity-is-tautologous*:

assumes $A \in \text{wffs}_o$

shows *is-tautologous* $(A \supset^{\mathcal{Q}} A)$

proof –

let $\ ?\vartheta = \{(\mathfrak{r}, o) \mapsto A\}$

have *is-tautology* $(\mathfrak{r}_o \supset^{\mathcal{Q}} \mathfrak{r}_o)$

by (*fact propositional-implication-reflexivity-is-tautology*)

moreover have *is-pwff-substitution* $\ ?\vartheta$

using *assms* **by** *auto*

moreover have $A \supset^{\mathcal{Q}} A = \mathbf{S} \ ?\vartheta (\mathfrak{r}_o \supset^{\mathcal{Q}} \mathfrak{r}_o)$

by *simp*

ultimately show ?*thesis*

by *blast*
qed

lemma *principle-of-simplification-is-tautologous*:

assumes $A \in \text{wffs}_o$ and $B \in \text{wffs}_o$
shows *is-tautologous* $(A \supset^{\mathcal{Q}} (B \supset^{\mathcal{Q}} A))$

proof –

let $?\vartheta = \{(\mathfrak{x}, o) \mapsto A, (\mathfrak{y}, o) \mapsto B\}$
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 $A \in \text{wffs}_o$ and $B \in \text{wffs}_o$
shows *is-tautologous* $((A \supset^{\mathcal{Q}} \sim^{\mathcal{Q}} B) \supset^{\mathcal{Q}} (B \supset^{\mathcal{Q}} \sim^{\mathcal{Q}} A))$

proof –

let $?\vartheta = \{(\mathfrak{x}, o) \mapsto A, (\mathfrak{y}, o) \mapsto B\}$
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:

$\mathfrak{g}_{o \rightarrow o} \cdot T_o \wedge^{\mathcal{Q}} \mathfrak{g}_{o \rightarrow o} \cdot F_o \equiv^{\mathcal{Q}} \forall \mathfrak{x}_o. \mathfrak{g}_{o \rightarrow o} \cdot \mathfrak{x}_o \in \text{axioms}$

| *axiom-2*:
 $(\mathfrak{r}_\alpha =_\alpha \mathfrak{h}_\alpha) \supset^\mathcal{Q} (\mathfrak{h}_{\alpha \rightarrow o} \cdot \mathfrak{r}_\alpha \equiv^\mathcal{Q} \mathfrak{h}_{\alpha \rightarrow o} \cdot \mathfrak{h}_\alpha) \in \text{axioms}$

| *axiom-3*:
 $(\mathfrak{f}_{\alpha \rightarrow \beta} =_{\alpha \rightarrow \beta} \mathfrak{g}_{\alpha \rightarrow \beta}) \equiv^\mathcal{Q} \forall \mathfrak{r}_\alpha. (\mathfrak{f}_{\alpha \rightarrow \beta} \cdot \mathfrak{r}_\alpha =_\beta \mathfrak{g}_{\alpha \rightarrow \beta} \cdot \mathfrak{r}_\alpha) \in \text{axioms}$

| *axiom-4-1-con*:
 $(\lambda x_\alpha. \{\!|c\!\}_\beta) \cdot A =_\beta \{\!|c\!\}_\beta \in \text{axioms}$ **if** $A \in \text{wffs}_\alpha$

| *axiom-4-1-var*:
 $(\lambda x_\alpha. y_\beta) \cdot A =_\beta y_\beta \in \text{axioms}$ **if** $A \in \text{wffs}_\alpha$ **and** $y_\beta \neq x_\alpha$

| *axiom-4-2*:
 $(\lambda x_\alpha. x_\alpha) \cdot A =_\alpha A \in \text{axioms}$ **if** $A \in \text{wffs}_\alpha$

| *axiom-4-3*:
 $(\lambda x_\alpha. B \cdot C) \cdot A =_\beta ((\lambda x_\alpha. B) \cdot A) \cdot ((\lambda x_\alpha. C) \cdot A) \in \text{axioms}$
if $A \in \text{wffs}_\alpha$ **and** $B \in \text{wffs}_{\gamma \rightarrow \beta}$ **and** $C \in \text{wffs}_\gamma$

| *axiom-4-4*:
 $(\lambda x_\alpha. \lambda y_\gamma. B) \cdot A =_{\gamma \rightarrow \delta} (\lambda y_\gamma. (\lambda x_\alpha. B) \cdot A) \in \text{axioms}$
if $A \in \text{wffs}_\alpha$ **and** $B \in \text{wffs}_\delta$ **and** $(y, \gamma) \notin \{(x, \alpha)\} \cup \text{vars } A$

| *axiom-4-5*:
 $(\lambda x_\alpha. \lambda x_\alpha. B) \cdot A =_{\alpha \rightarrow \delta} (\lambda x_\alpha. B) \in \text{axioms}$ **if** $A \in \text{wffs}_\alpha$ **and** $B \in \text{wffs}_\delta$

| *axiom-5*:
 $\iota \cdot (Q_i \cdot \mathfrak{h}_i) =_i \mathfrak{h}_i \in \text{axioms}$

lemma *axioms-are-wffs-of-type-o*:

shows $\text{axioms} \subseteq \text{wffs}_o$

by (*intro subsetI*, *cases rule: axioms.cases*) *auto*

5.2 Inference rule R

definition *is-rule-R-app* :: *position* \Rightarrow *form* \Rightarrow *form* \Rightarrow *form* \Rightarrow *bool* **where**

[*iff*]: *is-rule-R-app* p D C $E \longleftrightarrow$

(
 $\exists \alpha$ A B .
 $E = A =_\alpha B \wedge A \in \text{wffs}_\alpha \wedge B \in \text{wffs}_\alpha \wedge \text{--- } E$ is a well-formed equality
 $A \preceq_p C \wedge$
 $D \in \text{wffs}_o \wedge$
 $C \langle p \leftarrow B \rangle \triangleright D$
)

lemma *rule-R-original-form-is-wffo*:

assumes *is-rule-R-app* p D C E

shows $C \in \text{wffs}_o$

using *assms* **and** *replacement-preserves-typing* **by** *fastforce*

5.3 Proof and derivability

inductive *is-derivable* :: *form* \Rightarrow *bool* **where**

dv-axiom: *is-derivable* A **if** $A \in \text{axioms}$

| *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 $A \in \text{wffs}_0$
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
 $\mathcal{S} ! i' \in \text{axioms} \vee$
 $(\exists p j k. \{j, k\} \subseteq \{0..<i'\} \wedge \text{is-rule-R-app } p (\mathcal{S} ! i') (\mathcal{S} ! j) (\mathcal{S} ! k))$

definition *is-proof* :: *form list* \Rightarrow *bool* **where**

[iff]: *is-proof* $\mathcal{S} \longleftrightarrow (\forall i' < \text{length } \mathcal{S}. \text{is-proof-step } \mathcal{S} i')$

lemma *common-prefix-is-subproof*:

assumes *is-proof* $(\mathcal{S} @ \mathcal{S}_1)$
and $i' < \text{length } \mathcal{S}$
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** $i' < \text{length } (\mathcal{S} @ \mathcal{S}_1)$
by *simp*

ultimately obtain p and j and k **where** **:

$(\mathcal{S} @ \mathcal{S}_1) ! i' \in \text{axioms} \vee$
 $\{j, k\} \subseteq \{0..<i'\} \wedge \text{is-rule-R-app } p ((\mathcal{S} @ \mathcal{S}_1) ! i') ((\mathcal{S} @ \mathcal{S}_1) ! j) ((\mathcal{S} @ \mathcal{S}_1) ! k)$
using *assms*(1) **by** *fastforce*

then consider

$(\text{axiom}) (\mathcal{S} @ \mathcal{S}_1) ! i' \in \text{axioms}$
 $| (\text{rule-R}) \{j, k\} \subseteq \{0..<i'\} \wedge \text{is-rule-R-app } p ((\mathcal{S} @ \mathcal{S}_1) ! i') ((\mathcal{S} @ \mathcal{S}_1) ! j) ((\mathcal{S} @ \mathcal{S}_1) ! k)$
by *blast*

then have

$(\mathcal{S} @ \mathcal{S}_2) ! i' \in \text{axioms} \vee$
 $\{j, k\} \subseteq \{0..<i'\} \wedge \text{is-rule-R-app } p ((\mathcal{S} @ \mathcal{S}_2) ! i') ((\mathcal{S} @ \mathcal{S}_2) ! j) ((\mathcal{S} @ \mathcal{S}_2) ! k)$

proof cases

case *axiom*

with * **have** $(\mathcal{S} @ \mathcal{S}_2) ! i' \in \text{axioms}$

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 $\{j, k\} \subseteq \{0..<i'\} \wedge \text{is-rule-R-app } p ((\mathcal{S} @ \mathcal{S}_2) ! i') ((\mathcal{S} @ \mathcal{S}_2) ! j) ((\mathcal{S} @ \mathcal{S}_2) ! k)$

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 $i' < \text{length } (\mathcal{S} @ \mathcal{S}') - \text{length } \mathcal{S}'$

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* $(\mathcal{S} @ [A]) (\text{length } (\mathcal{S} @ [A]) - 1)$

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 $i' \in \{\text{length } \mathcal{S} .. < \text{length } (\mathcal{S} @ \mathcal{S}')\}$

shows *is-proof-step* $(\mathcal{S} @ \mathcal{S}') i'$

proof –

let $?S = \mathcal{S} @ \mathcal{S}'$

let $?i = i' - \text{length } \mathcal{S}$

from *assms*(2) **have** $?S ! i' = \mathcal{S}' ! ?i$ **and** $?i < \text{length } \mathcal{S}'$

by (*simp-all add: nth-append less-diff-conv2*)

then have *is-proof-step* $?S i' = \text{is-proof-step } \mathcal{S}' ?i$

proof –

from *assms*(1) **and** $\langle ?i < \text{length } \mathcal{S}' \rangle$ **obtain** j **and** k **and** p **where** *:

$\mathcal{S}' ! ?i \in \text{axioms} \vee (\{j, k\} \subseteq \{0 .. < ?i\} \wedge \text{is-rule-R-app } p (\mathcal{S}' ! ?i) (\mathcal{S}' ! j) (\mathcal{S}' ! k))$

by *fastforce*

then consider

(*axiom*) $\mathcal{S}' ! ?i \in \text{axioms}$

| (*rule-R*) $\{j, k\} \subseteq \{0 .. < ?i\} \wedge \text{is-rule-R-app } p (\mathcal{S}' ! ?i) (\mathcal{S}' ! j) (\mathcal{S}' ! k)$

by *blast*

then have

$?S ! i' \in \text{axioms} \vee$

(

$\{j + \text{length } \mathcal{S}, k + \text{length } \mathcal{S}\} \subseteq \{0 .. < i'\} \wedge$

$\text{is-rule-R-app } p (?S ! i') (?S ! (j + \text{length } \mathcal{S})) (?S ! (k + \text{length } \mathcal{S}))$

)

proof *cases*

case *axiom*

with $\langle ?S ! i' = \mathcal{S}' ! ?i \rangle$ **have** $?S ! i' \in \text{axioms}$

by (*simp only:*)

then show *thesis* ..

next

case *rule-R*

with *assms*(2) **have** $?S ! (j + \text{length } \mathcal{S}) = \mathcal{S}' ! j$ **and** $?S ! (k + \text{length } \mathcal{S}) = \mathcal{S}' ! k$

by (*simp-all add: nth-append*)

with $\langle ?S ! i' = \mathcal{S}' ! ?i \rangle$ **and** *rule-R* **have**

$\{j + \text{length } \mathcal{S}, k + \text{length } \mathcal{S}\} \subseteq \{0 .. < i'\} \wedge$

$\text{is-rule-R-app } p (?S ! i') (?S ! (j + \text{length } \mathcal{S})) (?S ! (k + \text{length } \mathcal{S}))$

by *auto*

```

    then show ?thesis ..
  qed
  with * show ?thesis
    by fastforce
  qed
  with assms(1) and ⟨?i < length S'⟩ show ?thesis
    by simp
  qed

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```


lemma *axiom-appended-to-proof-is-proof*:

assumes *is-proof* \mathcal{S}
and $A \in \text{axioms}$
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 $i_C < \text{length } \mathcal{S}$ **and** $\mathcal{S} ! i_C = C$
and $i_E < \text{length } \mathcal{S}$ **and** $\mathcal{S} ! i_E = E$
and *is-rule-R-app* $p D C E$
shows *is-proof* $(\mathcal{S} @ [D])$

proof –

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

qed

definition *is-proof-of* :: *form list* \Rightarrow *form* \Rightarrow *bool* **where**

[*iff*]: *is-proof-of* $\mathcal{S} A \longleftrightarrow \mathcal{S} \neq [] \wedge \text{is-proof } \mathcal{S} \wedge \text{last } \mathcal{S} = A$

lemma *proof-prefix-is-proof-of-last*:

assumes *is-proof* $(\mathcal{S} @ \mathcal{S}')$ **and** $\mathcal{S} \neq []$
shows *is-proof-of* $\mathcal{S} (\text{last } \mathcal{S})$

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* \Rightarrow *bool* **where**

[*iff*]: *is-theorem* $A \longleftrightarrow (\exists \mathcal{S}. \text{is-proof-of } \mathcal{S} A)$

lemma *proof-form-is-woff*:

assumes *is-proof-of* $\mathcal{S} A$
and $B \in \text{lset } \mathcal{S}$

shows $B \in \text{wffs}_0$
using *assms* and *elem-of-proof-is-wffo* **by** *blast*

lemma *proof-form-is-theorem*:

assumes *is-proof* \mathcal{S} and $\mathcal{S} \neq []$

and $i' < \text{length } \mathcal{S}$

shows *is-theorem* $(\mathcal{S} ! i')$

proof –

let $?S_1 = \text{take } (\text{Suc } i') \mathcal{S}$

from *assms*(1) **obtain** \mathcal{S}_2 **where** *is-proof* $(?S_1 @ \mathcal{S}_2)$

by (*metis append-take-drop-id*)

then have *is-proof* $?S_1$

by (*fact proof-prefix-is-proof*)

moreover from *assms*(3) **have** $\text{last } ?S_1 = \mathcal{S} ! i'$

by (*simp add: take-Suc-conv-app-nth*)

ultimately show *?thesis*

using *assms*(2) **unfolding** *is-proof-of-def* and *is-theorem-def* **by** (*metis Zero-neq-Suc take-eq-Nil2*)

qed

theorem *derivable-form-is-theorem*:

assumes *is-derivable* A

shows *is-theorem* A

using *assms* **proof** (*induction rule: is-derivable.induct*)

case (*dv-axiom* A)

then have *is-proof* $[A]$

by (*fact single-axiom-is-proof*)

moreover have $\text{last } [A] = A$

by *simp*

ultimately show *?case*

by *blast*

next

case (*dv-rule-R* $C E p D$)

obtain \mathcal{S}_C and \mathcal{S}_E **where**

is-proof \mathcal{S}_C and $\mathcal{S}_C \neq []$ and $\text{last } \mathcal{S}_C = C$ and

is-proof \mathcal{S}_E and $\mathcal{S}_E \neq []$ and $\text{last } \mathcal{S}_E = E$

using *dv-rule-R.IH* **by** *fastforce*

let $?i_C = \text{length } \mathcal{S}_C - 1$ and $?i_E = \text{length } \mathcal{S}_C + \text{length } \mathcal{S}_E - 1$ and $?i_D = \text{length } \mathcal{S}_C + \text{length } \mathcal{S}_E$

let $?S = \mathcal{S}_C @ \mathcal{S}_E @ [D]$

from $\langle \mathcal{S}_C \neq [] \rangle$ **have** $?i_C < \text{length } (\mathcal{S}_C @ \mathcal{S}_E)$ and $?i_E < \text{length } (\mathcal{S}_C @ \mathcal{S}_E)$

using *linorder-not-le* **by** *fastforce+*

moreover have $(\mathcal{S}_C @ \mathcal{S}_E) ! ?i_C = C$ and $(\mathcal{S}_C @ \mathcal{S}_E) ! ?i_E = E$

using $\langle \mathcal{S}_C \neq [] \rangle$ and $\langle \text{last } \mathcal{S}_C = C \rangle$

by

(
simp add: last-conv-nth nth-append,
metis $\langle \text{last } \mathcal{S}_E = E \rangle \langle \mathcal{S}_E \neq [] \rangle$ *append-is-Nil-conv last-appendR last-conv-nth length-append*
)

with $\langle \text{is-rule-R-app } p D C E \rangle$ **have** *is-rule-R-app* $p D ((\mathcal{S}_C @ \mathcal{S}_E) ! ?i_C) ((\mathcal{S}_C @ \mathcal{S}_E) ! ?i_E)$

using $\langle \mathcal{S}_C @ \mathcal{S}_E \rangle ! ?i_C = C$ **by** *fastforce*
moreover from $\langle \text{is-proof } \mathcal{S}_C \rangle$ **and** $\langle \text{is-proof } \mathcal{S}_E \rangle$ **have** $\text{is-proof } (\mathcal{S}_C @ \mathcal{S}_E)$
by *(fact proofs-concatenation-is-proof)*
ultimately have $\text{is-proof } ((\mathcal{S}_C @ \mathcal{S}_E) @ [D])$
using *rule-R-app-appended-to-proof-is-proof* **by** *presburger*
with $\langle \mathcal{S}_C \neq [] \rangle$ **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 S* **and** $\mathcal{S} \neq []$ **and** *last S = A*

by *fastforce*

then show $?thesis$

proof (*induction length S arbitrary: S A rule: less-induct*)

case *less*

let $?i' = \text{length } \mathcal{S} - 1$

from $\langle \mathcal{S} \neq [] \rangle$ **and** $\langle \text{last } \mathcal{S} = A \rangle$ **have** $\mathcal{S} ! ?i' = A$

by *(simp add: last-conv-nth)*

from $\langle \text{is-proof } \mathcal{S} \rangle$ **and** $\langle \mathcal{S} \neq [] \rangle$ **and** $\langle \text{last } \mathcal{S} = A \rangle$ **have** *is-proof-step S ?i'*

using *added-suffix-proof-preservation* **[where** $\mathcal{S}' = []$ **by** *simp*

then consider

(axiom) S ! ?i' ∈ axioms

| *(rule-R) ∃ p j k. {j, k} ⊆ {0..<?i'} ∧ is-rule-R-app p (S ! ?i') (S ! j) (S ! k)*

by *fastforce*

then show $?case$

proof *cases*

case *axiom*

with $\langle \mathcal{S} ! ?i' = A \rangle$ **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 (S ! ?i') (S ! j) (S ! k)*

by *force*

let $?S_j = \text{take } (\text{Suc } j) \mathcal{S}$

let $?S_k = \text{take } (\text{Suc } k) \mathcal{S}$

obtain \mathcal{S}_j' **and** \mathcal{S}_k' **where** $\mathcal{S} = ?S_j @ \mathcal{S}_j'$ **and** $\mathcal{S} = ?S_k @ \mathcal{S}_k'$

by *(metis append-take-drop-id)*

with $\langle \text{is-proof } \mathcal{S} \rangle$ **have** *is-proof (?S_j @ S_j')* **and** *is-proof (?S_k @ S_k')*

by *(simp-all only:)*

moreover

from $\langle \mathcal{S} = ?S_j @ \mathcal{S}_j' \rangle$ **and** $\langle \mathcal{S} = ?S_k @ \mathcal{S}_k' \rangle$ **and** $\langle \text{last } \mathcal{S} = A \rangle$ **and** $\langle \{j, k\} \subseteq \{0..<\text{length } \mathcal{S} - 1\} \rangle$

have $\text{last } \mathcal{S}_j' = A$ **and** $\text{last } \mathcal{S}_k' = A$

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 $\langle \mathcal{S} \neq [] \rangle$ **have** $?S_j \neq []$ **and** $?S_k \neq []$
by *simp-all*
ultimately have *is-proof-of* $?S_j$ (*last* $?S_j$) **and** *is-proof-of* $?S_k$ (*last* $?S_k$)
using *proof-prefix-is-proof-of-last* [where $\mathcal{S} = ?S_j$ and $\mathcal{S}' = S_j$]
and *proof-prefix-is-proof-of-last* [where $\mathcal{S} = ?S_k$ and $\mathcal{S}' = S_k$]
by *fastforce+*
moreover from $\langle \text{last } S_j' = A \rangle$ **and** $\langle \text{last } S_k' = A \rangle$
have *length* $?S_j < \text{length } \mathcal{S}$ **and** *length* $?S_k < \text{length } \mathcal{S}$
using $\langle \{j, k\} \subseteq \{0..<\text{length } \mathcal{S} - 1\} \rangle$ **by** *force+*
moreover from *calculation*(3,4) **have** *last* $?S_j = \mathcal{S} ! j$ **and** *last* $?S_k = \mathcal{S} ! k$
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 $\langle ?S_j \neq [] \rangle$ **and** $\langle ?S_k \neq [] \rangle$ **and** *less*(1) **by** *blast+*
with $\langle \text{is-rule-R-app } p (\mathcal{S} ! ?i') (\mathcal{S} ! j) (\mathcal{S} ! k) \rangle$ **and** $\langle \mathcal{S} ! ?i' = A \rangle$ **show** *?thesis*
by (*blast intro: dv-rule-R*)
qed
qed
qed

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

lemma *theorem-is-wffo*:
assumes *is-theorem* A
shows $A \in \text{wffs}_0$
proof –
from *assms* **obtain** \mathcal{S} **where** *is-proof-of* $\mathcal{S} A$
by *blast*
then have $A \in \text{lset } \mathcal{S}$
by *auto*
with $\langle \text{is-proof-of } \mathcal{S} A \rangle$ **show** *?thesis*
using *proof-form-is-wffo* **by** *blast*
qed

lemma *equality-reflexivity*:
assumes $A \in \text{wffs}_\alpha$
shows *is-theorem* ($A =_\alpha A$) (**is** *is-theorem* $?A_2$)
proof –
let $?A_1 = (\lambda \mathfrak{r}_\alpha. \mathfrak{r}_\alpha) \cdot A =_\alpha A$
let $?S = [?A_1, ?A_2]$
– (.1) Axiom 4.2
have *is-proof-step* $?S 0$
proof –
from *assms* **have** $?A_1 \in \text{axioms}$
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 $\exists p j k. \{j::nat, k\} \subseteq \{0..<1\} \wedge is-rule-R-app \ ?p \ ?A_2 \ (\ ?S \ ! \ j) \ (\ ?S \ ! \ k)$
 proof –
 let ?D = ?A₂ **and** ?j = 0::nat **and** ?k = 0
 have {?j, ?k} $\subseteq \{0..<1\}$
 by *simp*
 moreover have *is-rule-R-app* ?p ?A₂ (?S ! ?j) (?S ! ?k)
 proof –
 have $(\lambda \mathfrak{r}_\alpha. \mathfrak{r}_\alpha) \cdot A \preceq_{?p} (\ ?S \ ! \ ?j)$
 by *force*
 moreover have $(\ ?S \ ! \ ?j) \langle ?p \leftarrow A \rangle \triangleright ?D$
 by *force*
 moreover from $\langle A \in wffs_\alpha \rangle$ **have** ?D $\in wffs_o$
 by (*intro equality-wff*)
 moreover from $\langle A \in wffs_\alpha \rangle$ **have** $(\lambda \mathfrak{r}_\alpha. \mathfrak{r}_\alpha) \cdot A \in wffs_\alpha$
 by (*meson wffs-of-type-simps*)
 ultimately show ?thesis
 using $\langle A \in wffs_\alpha \rangle$ **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 $A \in wffs_\alpha$
 shows *is-theorem* $(A =_\alpha A)$ (**is** *is-theorem* ?A₂)
proof (*intro derivable-form-is-theorem*)
 let ?A₁ = $(\lambda \mathfrak{r}_\alpha. \mathfrak{r}_\alpha) \cdot A =_\alpha A$
 — (.1) Axiom 4.2
 from *assms* **have** ?A₁ $\in axioms$
 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 $?p = [\langle, \rangle]$ **and** $?C = ?A_1$ **and** $?E = ?A_1$ **and** $?D = ?A_2$
have *is-rule-R-app* $?p$ $?D$ $?C$ $?E$
proof –
have $(\lambda \mathfrak{x}_\alpha. \mathfrak{r}_\alpha) \cdot A \preceq_{?p} ?C$
by *force*
moreover have $?C \langle ?p \leftarrow A \rangle \triangleright ?D$
by *force*
moreover from $\langle A \in \text{wffs}_\alpha \rangle$ **have** $?D \in \text{wffs}_o$
by (*intro equality-wff*)
moreover from $\langle A \in \text{wffs}_\alpha \rangle$ **have** $(\lambda \mathfrak{x}_\alpha. \mathfrak{r}_\alpha) \cdot A \in \text{wffs}_\alpha$
by (*meson wffs-of-type-simps*)
ultimately show *?thesis*
using $\langle A \in \text{wffs}_\alpha \rangle$ **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* \Rightarrow *form* \Rightarrow *'a* \Rightarrow *var set* **where**
 $[simp]:$ *capture-exposed-vars-at* p A $\mathcal{X} =$
 $\{(x, \beta) \mid x \beta p' E. \text{strict-prefix } p' p \wedge \lambda x_\beta. E \preceq_{p'} A \wedge (x, \beta) \in \text{free-vars } \mathcal{X}\}$

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

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

$(x, \beta) \in \text{free-vars } (A =_{\alpha} B) \wedge$
 $(\exists H \in \mathcal{H}. (x, \beta) \in \text{free-vars } H)$
 $)$
proof –
have
 $\text{capture-exposed-vars-at } p \ C \ (A =_{\alpha} B)$
 $=$
 $\{(x, \beta) \mid x \ \beta \ p' \ E. \text{strict-prefix } p' \ p \wedge \lambda x_{\beta}. E \preceq_{p'} C \wedge (x, \beta) \in \text{free-vars } (A =_{\alpha} B)\}$
using *assms* **and** *capture-exposed-vars-at-alt-def* **unfolding** *capture-exposed-vars-at-def* **by** *fast*
moreover have
 $\text{capture-exposed-vars-at } p \ C \ \mathcal{H}$
 $=$
 $\{(x, \beta) \mid x \ \beta \ p' \ E. \text{strict-prefix } p' \ p \wedge \lambda x_{\beta}. E \preceq_{p'} C \wedge (x, \beta) \in \text{free-vars } \mathcal{H}\}$
using *assms* **and** *capture-exposed-vars-at-alt-def* **unfolding** *capture-exposed-vars-at-def* **by** *fast*
ultimately have
 $\text{capture-exposed-vars-at } p \ C \ (A =_{\alpha} B) \cap \text{capture-exposed-vars-at } p \ C \ \mathcal{H}$
 $=$
 $\{(x, \beta) \mid x \ \beta \ p' \ E. \text{strict-prefix } p' \ p \wedge \lambda x_{\beta}. E \preceq_{p'} C \wedge (x, \beta) \in \text{free-vars } (A =_{\alpha} B) \wedge$
 $(x, \beta) \in \text{free-vars } \mathcal{H}\}$
by *auto*
also have
 \dots
 $=$
 $\{(x, \beta) \mid x \ \beta \ p' \ E. \text{strict-prefix } p' \ p \wedge \lambda x_{\beta}. E \preceq_{p'} C \wedge (x, \beta) \in \text{free-vars } (A =_{\alpha} B) \wedge$
 $(\exists H \in \mathcal{H}. (x, \beta) \in \text{free-vars } H)\}$
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**
 $[iff]: \text{is-rule-R'-app } \mathcal{H} \ p \ D \ C \ E \longleftrightarrow \text{is-rule-R-app } p \ D \ C \ E \wedge \text{rule-R'-side-condition } \mathcal{H} \ p \ D \ C \ E$

lemma *is-rule-R'-app-alt-def*:

shows

$\text{is-rule-R'-app } \mathcal{H} \ p \ D \ C \ E$

\longleftrightarrow

$($

$\exists \alpha \ A \ B.$

$E = A =_{\alpha} B \wedge A \in \text{wffs}_{\alpha} \wedge B \in \text{wffs}_{\alpha} \wedge \text{--- } E$ is a well-formed equality

$A \preceq_p C \wedge D \in \text{wffs}_0 \wedge$

$C \langle p \leftarrow B \rangle \triangleright D \wedge$

$($

$\nexists x \ \beta \ E \ p'.$

$\text{strict-prefix } p' \ p \wedge$

$\lambda x_{\beta}. E \preceq_{p'} C \wedge$

$(x, \beta) \in \text{free-vars } (A =_{\alpha} B) \wedge$

$(\exists H \in \mathcal{H}. (x, \beta) \in \text{free-vars } H)$

$)$

)
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 $C \in \text{wffs}_o \longleftrightarrow D \in \text{wffs}_o$
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* \Rightarrow *bool* **where**
is-hyps $\mathcal{H} \equiv \mathcal{H} \subseteq \text{wffs}_o \wedge \text{finite } \mathcal{H}$

inductive *is-derivable-from-hyps* :: *form set* \Rightarrow *form* \Rightarrow *bool* ($- \vdash - [50, 50] \ 50$) **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 $A \in \text{wffs}_o$
using *assms* **and** *theorem-is-wffo* **by** (*cases rule: is-derivable-from-hyps.cases*) *auto*

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

type-synonym *hyp-proof* = *form list* \times *form list*

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

lemma *common-prefix-is-hyp-subproof-from*:
assumes *is-hyp-proof* $\mathcal{H} \ \mathcal{S}_1 \ (\mathcal{S}_2 @ \mathcal{S}_2')$
and $i' < \text{length } \mathcal{S}_2$
shows *is-hyp-proof-step* $\mathcal{H} \ \mathcal{S}_1 \ (\mathcal{S}_2 @ \mathcal{S}_2'') \ i'$

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


```

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

```

lemma *added-suffix-thms-hyp-proof-preservation:*

```

  assumes is-hyp-proof ℋ S1 S2
  shows is-hyp-proof ℋ (S1 @ S1') S2
  using assms by auto

```

lemma *added-suffix-hyp-proof-preservation:*

```

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

```

lemma *appended-hyp-proof-step-is-hyp-proof:*

```

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

```

```

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

```

lemma *added-prefix-hyp-proof-preservation:*

```

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

```

```

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

```

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

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

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

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

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

```

fix  $i'$ 
assume  $i' < \text{length } ?\mathcal{S}'$ 
then consider (a)  $i' < \text{length } \mathcal{S}_1'$  | (b)  $i' \in \{\text{length } \mathcal{S}_1' .. < \text{length } ?\mathcal{S}'\}$ 
  by fastforce
then show is-hyp-proof-step  $\mathcal{H} ?\mathcal{S} ?\mathcal{S}' i'$ 
proof cases
  case a
    from  $\langle \text{is-hyp-proof } \mathcal{H} \mathcal{S}_1 \mathcal{S}_1' \rangle$  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  $\mathcal{S}_1 \subseteq \text{wffs}_0$ 
and is-hyp-proof  $\mathcal{H} \mathcal{S}_1 \mathcal{S}_2$ 
and  $A \in \text{lset } \mathcal{S}_2$ 
shows  $A \in \text{wffs}_0$ 
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  $\langle \text{is-hyp-proof } \mathcal{H} \mathcal{S}_1 (\mathcal{S}_2 @ [A']) \rangle$  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  $A \in \text{lset } \mathcal{S}_2$ )
      case True
        with snoc.prem(1,2) and  $\langle \text{is-hyp-proof } \mathcal{H} \mathcal{S}_1 \mathcal{S}_2 \rangle$  show ?thesis
        by (fact snoc.IH)
      next
      case False
        with snoc.prem(4) have  $A' = A$ 
        by simp
        with snoc.prem(3) have
           $(\mathcal{S}_2 @ [A]) ! i' \in \mathcal{H} \vee$ 
           $(\mathcal{S}_2 @ [A]) ! i' \in \text{lset } \mathcal{S}_1 \vee$ 
           $(\mathcal{S}_2 @ [A]) ! i' \in \text{wffs}_0$  if  $i' \in \{0 .. < \text{length } (\mathcal{S}_2 @ [A])\}$  for  $i'$ 
        using that by auto
        then have  $A \in \text{wffs}_0 \vee A \in \mathcal{H} \vee A \in \text{lset } \mathcal{S}_1 \vee \text{length } \mathcal{S}_2 \notin \{0 .. < \text{Suc } (\text{length } \mathcal{S}_2)\}$ 

```

by (*metis* (*no-types*) *length-append-singleton nth-append-length*)
 with *assms(1)* and $\langle \text{lset } \mathcal{S}_1 \subseteq \text{wffs}_0 \rangle$ **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 $A \in \text{lset } \mathcal{S}_1$
 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 $A \in \text{lset } \mathcal{S}_1$
 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 $A \in \text{lset } \mathcal{S}_1$
 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 $i_C < \text{length } \mathcal{S}$ **and** $\mathcal{S} ! i_C = C$
and $i_E < \text{length } \mathcal{S}$ **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 $?S = \mathcal{S} @ [D]$
fix i'
assume $i' < \text{length } ?S$
then consider (a) $i' < \text{length } \mathcal{S}$ | (b) $i' = \text{length } \mathcal{S}$
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 $?i_D = \text{length } \mathcal{S}$
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* $\mathcal{H} p (?S ! ?i_D) (?S ! i_C) (?S ! i_E)$
by (*simp add: nth-append*)
with *assms(2,4)* **have**
 $\exists p j k. \{j, k\} \subseteq \{0..<?i_D\} \wedge \text{is-rule-R'-app } \mathcal{H} p (?S ! ?i_D) (?S ! j) (?S ! k)$
by (*intro exI*)+ *auto*
then have *is-hyp-proof-step* $\mathcal{H} \mathcal{S}' ?S (\text{length } ?S - 1)$
by *simp*
moreover from b have $i' = \text{length } ?S - 1$
by *simp*
ultimately show *?thesis*
by *fast*
qed
qed

definition *is-hyp-proof-of* :: *form set* \Rightarrow *form list* \Rightarrow *form list* \Rightarrow *form* \Rightarrow *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 $\exists \mathcal{S}_1 \mathcal{S}_2. \textit{is-hyp-proof-of} \mathcal{H} \mathcal{S}_1 \mathcal{S}_2 A$

using *assms* **proof** (*induction rule: is-derivable-from-hyps.induct*)

case (*dv-hyp* A)

from $\langle A \in \mathcal{H} \rangle$ **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 $\langle \textit{is-hyps} \mathcal{H} \rangle$ **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 $\langle \textit{is-hyps} \mathcal{H} \rangle$ and $\langle \textit{is-proof} \mathcal{S} \rangle$ **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 $?i_C = \textit{length} \mathcal{S}_C - 1$ and $?i_E = \textit{length} \mathcal{S}_C + \textit{length} \mathcal{S}_E - 1$ and $?i_D = \textit{length} \mathcal{S}_C + \textit{length} \mathcal{S}_E$

let $?S = \mathcal{S}_C @ \mathcal{S}_E @ [D]$

from $\langle \mathcal{S}_C \neq [] \rangle$ **have** $?i_C < \textit{length} (\mathcal{S}_C @ \mathcal{S}_E)$ and $?i_E < \textit{length} (\mathcal{S}_C @ \mathcal{S}_E)$

using *linorder-not-le* **by** *fastforce+*

moreover **have** $(\mathcal{S}_C @ \mathcal{S}_E) ! ?i_C = C$ and $(\mathcal{S}_C @ \mathcal{S}_E) ! ?i_E = E$

using $\langle \mathcal{S}_C \neq [] \rangle$ and $\langle \textit{last} \mathcal{S}_C = C \rangle$ and $\langle \mathcal{S}_E \neq [] \rangle$ and $\langle \textit{last} \mathcal{S}_E = E \rangle$

by

(
simp add: last-conv-nth nth-append,
metis append-is-Nil-conv last-appendR last-conv-nth length-append
)

with $\langle \textit{is-rule-R'-app} \mathcal{H} p D C E \rangle$ **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** $\langle \textit{is-hyp-proof} \mathcal{H} \mathcal{S}_C' \mathcal{S}_C \rangle$ and $\langle \textit{is-hyp-proof} \mathcal{H} \mathcal{S}_E' \mathcal{S}_E \rangle$

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 $\langle \text{is-proof } \mathcal{S}_C' \rangle$ **and** $\langle \text{is-proof } \mathcal{S}_E' \rangle$ **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 $\langle \text{is-hyps } \mathcal{H} \rangle$ **by** *fastforce*
then show *?case*
by *(intro exI)*
qed

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

using *is-theorem-def* by *blast*
 with $\langle \text{is-hyps } \mathcal{H} \rangle$ 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 $?S_j = \text{take } (\text{Suc } j) \mathcal{S}_2$ and $?S_k = \text{take } (\text{Suc } k) \mathcal{S}_2$
 obtain \mathcal{S}_j' and \mathcal{S}_k' where $\mathcal{S}_2 = ?S_j @ \mathcal{S}_j'$ and $\mathcal{S}_2 = ?S_k @ \mathcal{S}_k'$
 by (*metis append-take-drop-id*)
 then have *is-hyp-proof* $\mathcal{H} \mathcal{S}_1 (?S_j @ \mathcal{S}_j')$ and *is-hyp-proof* $\mathcal{H} \mathcal{S}_1 (?S_k @ \mathcal{S}_k')$
 by (*simp-all only: <is-hyp-proof H S1 S2>*)
 moreover from $\langle \mathcal{S}_2 \neq [] \rangle$ and $\langle \mathcal{S}_2 = ?S_j @ \mathcal{S}_j' \rangle$ and $\langle \mathcal{S}_2 = ?S_k @ \mathcal{S}_k' \rangle$ and $\langle \text{last } \mathcal{S}_2 = A \rangle$
 have $\text{last } \mathcal{S}_j' = A$ and $\text{last } \mathcal{S}_k' = A$
 using $\langle \{j, k\} \subseteq \{0..<\text{length } \mathcal{S}_2 - 1\} \rangle$ 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 $\langle \mathcal{S}_2 \neq [] \rangle$ have $?S_j \neq []$ and $?S_k \neq []$
 by *simp-all*
 ultimately have *is-hyp-proof-of* $\mathcal{H} \mathcal{S}_1 ?S_j (\text{last } ?S_j)$ and *is-hyp-proof-of* $\mathcal{H} \mathcal{S}_1 ?S_k (\text{last } ?S_k)$
 using *hyp-proof-prefix-is-hyp-proof-of-last*
 [*OF <is-hyps H> <is-proof S1> <is-hyp-proof H S1 (?Sj @ Sj')> <?Sj ≠ []>*]
 and *hyp-proof-prefix-is-hyp-proof-of-last*
 [*OF <is-hyps H> <is-proof S1> <is-hyp-proof H S1 (?Sk @ Sk')> <?Sk ≠ []>*]
 by *fastforce+*
 moreover from $\langle \text{last } \mathcal{S}_j' = A \rangle$ and $\langle \text{last } \mathcal{S}_k' = A \rangle$
 have $\text{length } ?S_j < \text{length } \mathcal{S}_2$ and $\text{length } ?S_k < \text{length } \mathcal{S}_2$
 using $\langle \{j, k\} \subseteq \{0..<\text{length } \mathcal{S}_2 - 1\} \rangle$ by *force+*
 moreover from *calculation(3,4)* have $\text{last } ?S_j = \mathcal{S}_2 ! j$ and $\text{last } ?S_k = \mathcal{S}_2 ! k$
 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 $\langle \text{is-hyps } \mathcal{H} \rangle$
 and *less(1)[OF <length ?Sj < length S2>]* and *less(1)[OF <length ?Sk < length S2>]*
 by *fast+*
 with $\langle \text{is-hyps } \mathcal{H} \rangle$ and $\langle \mathcal{S}_2 ! ?i' = A \rangle$ 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 $\mathcal{H} \vdash A \iff (\exists \mathcal{S}_1 \mathcal{S}_2. \text{is-hyp-proof-of } \mathcal{H} \mathcal{S}_1 \mathcal{S}_2 A)$

using *hyp-derivability-implies-hyp-proof-existence* and *hyp-proof-existence-implies-hyp-derivability ..*

proposition *derivability-from-no-hyps-theoremhood-equivalence:*

shows $\{\} \vdash A \iff \text{is-theorem } A$

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 $\langle \text{is-rule-R'-app } \{\} p D C E \rangle$ **have** *is-rule-R-app p D C E*
by *simp*
moreover from $\langle \text{is-theorem } C \rangle$ **and** $\langle \text{is-theorem } E \rangle$ **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 $\{\} \vdash A$
by (*blast intro: dv-thm*)
qed

abbreviation *is-derivable-from-no-hyps* ($\vdash - [50] 50$) **where**
 $\vdash A \equiv \{\} \vdash A$

corollary *derivability-implies-hyp-derivability*:
assumes $\vdash A$ **and** *is-hyps \mathcal{H}*
shows $\mathcal{H} \vdash A$
using *assms and derivability-from-no-hyps-theoremhood-equivalence and dv-thm* **by** *simp*

lemma *axiom-is-derivable-from-no-hyps*:
assumes $A \in \text{axioms}$
shows $\vdash A$
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 $A \in \text{axioms}$ **and** *is-hyps \mathcal{H}*
shows $\mathcal{H} \vdash A$
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 $\vdash C$ **and** $\vdash A =_{\alpha} B$
and $A \preceq_p C$ **and** $C \langle p \leftarrow B \rangle \triangleright D$
shows $\vdash D$

proof –
from *assms(1,2)* **have** *is-derivable C and is-derivable ($A =_{\alpha} B$)*
using *derivability-from-no-hyps-theoremhood-equivalence*
and *theoremhood-derivability-equivalence* **by** *blast+*
moreover have *is-rule-R-app p D C ($A =_{\alpha} B$)*
proof –
from *assms(1-4)* **have** $D \in \text{wffs}_o$ **and** $A \in \text{wffs}_{\alpha}$ **and** $B \in \text{wffs}_{\alpha}$
by (*meson hyp-derivable-form-is-wffso replacement-preserves-typing wffs-from-equality*)
with *assms(3,4)* **show** *?thesis*

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

lemma *rule-R'* [*consumes 2, case-names occ-subform replacement no-capture*]:
assumes $\mathcal{H} \vdash C$ **and** $\mathcal{H} \vdash A =_\alpha B$
and $A \preceq_p C$ **and** $C \langle p \leftarrow B \rangle \triangleright D$
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** $D \in \text{wffs}_o$
 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** \rightarrow 60)

6.1 Proposition 5200

proposition *prop-5200*:
assumes $A \in \text{wffs}_\alpha$
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** $A \in \text{wffs}_\alpha$
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 $A \in \text{wffs}_{\alpha}$

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 $\mathcal{H} \vdash A =_{\alpha \rightarrow \beta} B$ **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** $A \in \text{wffs}_{\alpha \rightarrow \beta}$ **and** $C \in \text{wffs}_{\alpha}$

using *hyp-derivable-form-is-wffso* **and** *wffs-from-equality* **by** *blast+*

then show $A \cdot C \in \text{wffs}_{\beta}$

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 $\mathcal{H} \vdash A =_{\alpha \rightarrow \beta} B$ and $C \in \text{wffs}_{\alpha}$

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 $A \in \text{wffs}_{\alpha \rightarrow \beta}$

by (*fact hyp-derivable-form-is-wffso*[*OF assms(1)*, *THEN wffs-from-equality(1)*])

with *assms(2)* show $A \cdot C \in \text{wffs}_{\beta}$

by *auto*

qed

from *this* and *assms(1)* show *?thesis*

by (*rule rule-R'*[**where** $p = [\rangle, \langle]$]) (*force+*, *fastforce dest: subforms-from-app*)

qed

proposition *prop-5201-6*:

assumes $\mathcal{H} \vdash C =_{\alpha} D$ and $A \in \text{wffs}_{\alpha \rightarrow \beta}$

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 $C \in \text{wffs}_{\alpha}$

by (*fact hyp-derivable-form-is-wffso*[*OF assms(1)*, *THEN wffs-from-equality(1)*])

with *assms(2)* show $A \cdot C \in \text{wffs}_{\beta}$

by *auto*

qed

from *this* and *assms(1)* show *?thesis*

by (*rule rule-R'*[**where** $p = [\rangle, \rangle]$]) (*force+*, *fastforce dest: subforms-from-app*)

qed

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

6.3 Proposition 5202 (Rule RR)

proposition *prop-5202*:

assumes $\vdash A =_{\alpha} B \vee \vdash B =_{\alpha} A$

and $p \in \text{positions } C$ and $A \preceq_p C$ and $C \langle p \leftarrow B \rangle \triangleright D$

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 $(a) \vdash A =_{\alpha} B \mid (b) \vdash B =_{\alpha} A$

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

lemmas *rule-RR = prop-5202*

6.4 Proposition 5203

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

moreover have $\mathbf{S} \{(x, \alpha) \mapsto A\} (\llbracket c \rrbracket_\beta) = \llbracket c \rrbracket_\beta$
by *auto*
ultimately show *?case*
by (*simp only*:)

next
case (*app-is-wff* $\gamma \beta D C$)
from *app-is-wff.prem*s(2) **have** *not-bound-subforms*: $\forall v \in \text{vars } A. \neg \text{is-bound } v D \wedge \neg \text{is-bound } v C$
using *is-bound-in-app-homomorphism* **by** *fast*
from $\langle D \in \text{wffs}_{\gamma \rightarrow \beta} \rangle$ **have** $\vdash (\lambda x_\alpha. D) \cdot A =_{\gamma \rightarrow \beta} \mathbf{S} \{(x, \alpha) \mapsto A\} D$
using *app-is-wff.IH*(1)[*OF assms*(1)] **and** *not-bound-subforms* **by** *simp*
moreover from $\langle C \in \text{wffs}_\gamma \rangle$ **have** $\vdash (\lambda x_\alpha. C) \cdot A =_\gamma \mathbf{S} \{(x, \alpha) \mapsto A\} C$
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) $\langle D \in \text{wffs}_{\gamma \rightarrow \beta} \rangle \langle C \in \text{wffs}_\gamma \rangle$]].
ultimately show *?case*
using *Equality-Rules*(3,4) **and** *substitute.simps*(3) **by** *presburger*

next
case (*abs-is-wff* $\beta D \gamma y$)
then show *?case*
proof (*cases* $y_\gamma = x_\alpha$)
case *True*
then have $\vdash (\lambda x_\alpha. \lambda y_\gamma. D) \cdot A =_{\gamma \rightarrow \beta} \lambda y_\gamma. D$
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** $\mathbf{S} \{(x, \alpha) \mapsto A\} (\lambda y_\gamma. D) = \lambda y_\gamma. D$
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* $(\lambda y_\gamma. D) [\llbracket \cdot \rrbracket] = \{(y, \gamma)\}$
by *simp*
then have *is-bound* $(y, \gamma) (\lambda y_\gamma. D)$
by *fastforce*
with *abs-is-wff.prem*s(2) **have** $(y, \gamma) \notin \text{vars } A$
by *blast*
with $\langle y_\gamma \neq x_\alpha \rangle$ **have** $\vdash (\lambda x_\alpha. \lambda y_\gamma. D) \cdot A =_{\gamma \rightarrow \beta} \lambda y_\gamma. (\lambda x_\alpha. D) \cdot A$
using *axiom-4-4*[*OF assms*(1) *abs-is-wff.hyps*(1)] **and** *axiom-is-derivable-from-no-hyps* **by** *blast*
moreover have $\vdash (\lambda x_\alpha. D) \cdot A =_\beta \mathbf{S} \{(x, \alpha) \mapsto A\} D$
proof –
have $\forall p. y_\gamma \preceq_\llbracket \cdot \rrbracket p \lambda y_\gamma. D \longrightarrow y_\gamma \preceq_p D$
using *subforms-from-abs* **by** *fastforce*
from *abs-is-wff.prem*s(2) **have** $\forall v \in \text{vars } A. \neg \text{is-bound } v D$
using *is-bound-in-abs-body* **by** *fast*
then show *?thesis*
by (*fact* *abs-is-wff.IH*[*OF assms*(1)])

qed
ultimately have $\vdash (\lambda x_\alpha. \lambda y_\gamma. D) \cdot A =_{\gamma \rightarrow \beta} \lambda y_\gamma. \mathbf{S} \{(x, \alpha) \mapsto A\} D$
by (*rule* *rule-R*[*where* $p = [\llbracket \cdot \rrbracket, \llbracket \cdot \rrbracket]$]) *force+*

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

6.5 Proposition 5204

proposition *prop-5204*:

assumes $A \in \text{wffs}_\alpha$ and $B \in \text{wffs}_\beta$ and $C \in \text{wffs}_\beta$
 and $\vdash B =_\beta C$
 and $\forall v \in \text{vars } A. \neg \text{is-bound } v \ B \wedge \neg \text{is-bound } v \ C$
 shows $\vdash \mathbf{S} \{(x, \alpha) \mapsto A\} (B =_\beta C)$

proof –

have $\vdash (\lambda x_\alpha. B) \cdot A =_\beta (\lambda x_\alpha. B) \cdot A$

proof –

have $(\lambda x_\alpha. B) \cdot A \in \text{wffs}_\beta$

using *assms(1,2)* by *auto*

then show *?thesis*

by (*fact prop-5200*)

qed

from *this* and *assms(4)* have $\vdash (\lambda x_\alpha. B) \cdot A =_\beta (\lambda x_\alpha. C) \cdot A$

by (*rule rule-R[where p = [»,«,«]]*) *force+*

moreover from *assms(1,2,5)* have $\vdash (\lambda x_\alpha. B) \cdot A =_\beta \mathbf{S} \{(x, \alpha) \mapsto A\} B$

using *prop-5203* by *auto*

moreover from *assms(1,3,5)* have $\vdash (\lambda x_\alpha. C) \cdot A =_\beta \mathbf{S} \{(x, \alpha) \mapsto A\} C$

using *prop-5203* by *auto*

ultimately have $\vdash (\mathbf{S} \{(x, \alpha) \mapsto A\} B) =_\beta (\mathbf{S} \{(x, \alpha) \mapsto A\} C)$

using *Equality-Rules(2,3)* by *blast*

then show *?thesis*

by *simp*

qed

6.6 Proposition 5205 (η -conversion)

proposition *prop-5205*:

shows $\vdash \mathbf{f}_{\alpha \rightarrow \beta} =_{\alpha \rightarrow \beta} (\lambda y_\alpha. \mathbf{f}_{\alpha \rightarrow \beta} \cdot y_\alpha)$

proof –

{

fix y

assume $y_\alpha \neq \mathbf{r}_\alpha$

let $?A = \lambda y_\alpha. \mathbf{f}_{\alpha \rightarrow \beta} \cdot y_\alpha$

have $\vdash (\mathbf{f}_{\alpha \rightarrow \beta} =_{\alpha \rightarrow \beta} ?A) =_o \forall \mathbf{r}_\alpha. (\mathbf{f}_{\alpha \rightarrow \beta} \cdot \mathbf{r}_\alpha =_\beta ?A \cdot \mathbf{r}_\alpha)$

proof –

have $\vdash (\mathbf{f}_{\alpha \rightarrow \beta} =_{\alpha \rightarrow \beta} \mathbf{g}_{\alpha \rightarrow \beta}) =_o \forall \mathbf{r}_\alpha. (\mathbf{f}_{\alpha \rightarrow \beta} \cdot \mathbf{r}_\alpha =_\beta \mathbf{g}_{\alpha \rightarrow \beta} \cdot \mathbf{r}_\alpha)$ (*is* $\vdash ?B =_o ?C$)

using *axiom-3[unfolded equivalence-def]* by (*rule axiom-is-derivable-from-no-hyps*)

have $\vdash \mathbf{S} \{(\mathbf{g}, \alpha \rightarrow \beta) \mapsto ?A\} (?B =_o ?C)$

proof –

have $?A \in \text{wffs}_{\alpha \rightarrow \beta}$ and $?B \in \text{wffs}_o$ and $?C \in \text{wffs}_o$

by *auto*

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

have vars $?B = \{(f, \alpha \rightarrow \beta), (g, \alpha \rightarrow \beta)\}$ **and** vars $?C = \{(f, \alpha \rightarrow \beta), (x, \alpha), (g, \alpha \rightarrow \beta)\}$
by force+
with $\langle y_\alpha \neq x_\alpha \rangle$ **have** $(y, \alpha) \notin \text{vars } ?B$ **and** $(y, \alpha) \notin \text{vars } ?C$
by force+
then **have** $\neg \text{is-bound } (y, \alpha) ?B$ **and** $\neg \text{is-bound } (y, \alpha) ?C$
using absent-var-is-not-bound **by** blast+
moreover **have** $\neg \text{is-bound } (f, \alpha \rightarrow \beta) ?B$ **and** $\neg \text{is-bound } (f, \alpha \rightarrow \beta) ?C$
by code-simp+
moreover from $\langle v \in \text{vars } ?A \rangle$ **have** $v \in \{(y, \alpha), (f, \alpha \rightarrow \beta)\}$
by auto
ultimately show $\neg \text{is-bound } v ?B \wedge \neg \text{is-bound } v ?C$
by fast
qed
ultimately show ?thesis
using $\langle ?B =_o ?C \rangle$ **and** prop-5204 **by** presburger
qed
then show ?thesis
by simp
qed
ultimately have $\vdash f_{\alpha \rightarrow \beta} =_{\alpha \rightarrow \beta} (\lambda y \alpha. f_{\alpha \rightarrow \beta} \cdot y_\alpha)$
using Equality-Rules(1)[unfolded equivalence-def] **and** Equality-Rules(2) **and** prop-5200
by (metis wffs-of-type-intros(1))
}
note $x\text{-neg-}y = \text{this}$
then have §6: $\vdash f_{\alpha \rightarrow \beta} =_{\alpha \rightarrow \beta} \lambda \eta \alpha. f_{\alpha \rightarrow \beta} \cdot \eta_\alpha$ (**is** $\vdash ?B =_ . ?C$)
by simp
then have §7: $\vdash (\lambda x \alpha. f_{\alpha \rightarrow \beta} \cdot x_\alpha) =_{\alpha \rightarrow \beta} (\lambda \eta \alpha. (\lambda x \alpha. f_{\alpha \rightarrow \beta} \cdot x_\alpha) \cdot \eta_\alpha)$
proof –
let $?A = \lambda x \alpha. f_{\alpha \rightarrow \beta} \cdot x_\alpha$
have $?A \in \text{wffs}_{\alpha \rightarrow \beta}$ **and** $?B \in \text{wffs}_{\alpha \rightarrow \beta}$ **and** $?C \in \text{wffs}_{\alpha \rightarrow \beta}$
by auto
moreover have $\forall v \in \text{vars } ?A. \neg \text{is-bound } v ?B \wedge \neg \text{is-bound } v ?C$
proof
fix v
assume $v \in \text{vars } ?A$
have $\neg \text{is-bound } (x, \alpha) ?B$ **and** $\neg \text{is-bound } (x, \alpha) ?C$
by code-simp+
moreover have $\neg \text{is-bound } (f, \alpha \rightarrow \beta) ?B$ **and** $\neg \text{is-bound } (f, \alpha \rightarrow \beta) ?C$
by code-simp+
moreover from $\langle v \in \text{vars } ?A \rangle$ **have** $v \in \{(x, \alpha), (f, \alpha \rightarrow \beta)\}$
by auto
ultimately show $\neg \text{is-bound } v ?B \wedge \neg \text{is-bound } v ?C$
by fast
qed
ultimately have $\vdash \mathbf{S} \{(f, \alpha \rightarrow \beta) \mapsto ?A\} (?B =_{\alpha \rightarrow \beta} ?C)$
using §6 **and** prop-5204 **by** presburger
then show ?thesis
by simp
qed

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

6.7 Proposition 5206 (α -conversion)

proposition *prop-5206*:
assumes $A \in \text{wffs}_\alpha$
and $(z, \beta) \notin \text{free-vars } A$
and *is-free-for* $(z_\beta)(x, \beta) A$
shows $\vdash (\lambda x_\beta. A) =_{\beta \rightarrow \alpha} (\lambda z_\beta. \mathbf{S} \{(x, \beta) \rightsquigarrow z_\beta\} A)$
proof –
have *is-substitution* $\{(x, \beta) \rightsquigarrow z_\beta\}$
by *auto*
from this and *assms(1)* **have** $\mathbf{S} \{(x, \beta) \rightsquigarrow z_\beta\} A \in \text{wffs}_\alpha$
by (*fact substitution-preserves-typing*)
obtain y **where** $(y, \beta) \notin \{(x, \beta), (z, \beta)\} \cup \text{vars } A$
proof –
have *finite* $(\{(x, \beta), (z, \beta)\} \cup \text{vars } A)$
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** $(y, \beta) \notin \text{vars } A$ **and** $(y, \beta) \notin \text{free-vars } A$
using *free-vars-in-all-vars* **by** *auto*
have §1: $\vdash (\lambda x_\beta. A) =_{\beta \rightarrow \alpha} (\lambda y_\beta. (\lambda x_\beta. A) \cdot y_\beta)$
proof –
let $?A = \lambda x_\beta. A$
have $*$: $\vdash \mathfrak{f}_{\beta \rightarrow \alpha} =_{\beta \rightarrow \alpha} (\lambda y_\beta. \mathfrak{f}_{\beta \rightarrow \alpha} \cdot y_\beta)$ (**is** $\vdash ?B =. ?C$)
by (*fact prop-5205*)
moreover have $\vdash \mathbf{S} \{(\mathfrak{f}, \beta \rightarrow \alpha) \rightsquigarrow ?A\} (?B =_{\beta \rightarrow \alpha} ?C)$
proof –
from *assms(1)* **have** $?A \in \text{wffs}_{\beta \rightarrow \alpha}$ **and** $?B \in \text{wffs}_{\beta \rightarrow \alpha}$ **and** $?C \in \text{wffs}_{\beta \rightarrow \alpha}$

by *auto*
moreover have $\forall v \in \text{vars } ?A. \neg \text{is-bound } v ?B \wedge \neg \text{is-bound } v ?C$
proof
 fix v
 assume $v \in \text{vars } ?A$
then consider $(a) v = (x, \beta) \mid (b) v \in \text{vars } A$
 by *fastforce*
then show $\neg \text{is-bound } v ?B \wedge \neg \text{is-bound } v ?C$
proof cases
 case a
then show *?thesis*
 using $\langle (y, \beta) \neq (x, \beta) \rangle$ by *force*
next
 case b
then have $\neg \text{is-bound } v ?B$
 by *simp*
moreover have $\neg \text{is-bound } v ?C$
 using b and $\langle (y, \beta) \notin \text{vars } A \rangle$ 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 $\S 2: \vdash (\lambda x_{\beta}. A) =_{\beta \rightarrow \alpha} (\lambda y_{\beta}. \mathbf{S} \{(x, \beta) \mapsto y_{\beta}\} A)$
proof $-$
have $\vdash (\lambda x_{\beta}. A) \cdot y_{\beta} =_{\alpha} \mathbf{S} \{(x, \beta) \mapsto y_{\beta}\} A$ (**is** $\vdash (\lambda x_{\beta}. ?B) \cdot ?A =_{-} -$)
proof $-$
have $?A \in \text{wffs}_{\beta}$ and $?B \in \text{wffs}_{\alpha}$
 by *blast fact*
moreover have $\forall v \in \text{vars } ?A. \neg \text{is-bound } v ?B$
 using $\langle (y, \beta) \notin \text{vars } A \rangle$ and *absent-var-is-not-bound* by *auto*
ultimately show *?thesis*
 by (*fact prop-5203*)
qed
with $\S 1$ **show** *?thesis*
 by (*rule rule-R* [**where** $p = [\rangle, \langle]$]) *force+*
qed
moreover
have $\S 3: \vdash (\lambda z_{\beta}. \mathbf{S} \{(x, \beta) \mapsto z_{\beta}\} A) =_{\beta \rightarrow \alpha} (\lambda y_{\beta}. (\lambda z_{\beta}. \mathbf{S} \{(x, \beta) \mapsto z_{\beta}\} A) \cdot y_{\beta})$
proof $-$
let $?A = \lambda z_{\beta}. \mathbf{S} \{(x, \beta) \mapsto z_{\beta}\} A$
have $*$: $\vdash \mathfrak{f}_{\beta \rightarrow \alpha} =_{\beta \rightarrow \alpha} (\lambda y_{\beta}. \mathfrak{f}_{\beta \rightarrow \alpha} \cdot y_{\beta})$ (**is** $\vdash ?B =_{-} ?C$)
 by (*fact prop-5205*)
moreover have $\vdash \mathbf{S} \{(\mathfrak{f}, \beta \rightarrow \alpha) \mapsto ?A\} (?B =_{\beta \rightarrow \alpha} ?C)$

proof –
have $?A \in \text{wffs}_{\beta \rightarrow \alpha}$ **and** $?B \in \text{wffs}_{\beta \rightarrow \alpha}$ **and** $?C \in \text{wffs}_{\beta \rightarrow \alpha}$
using $\langle \mathbf{S} \{(x, \beta) \mapsto z_\beta\} A \in \text{wffs}_\alpha \rangle$ **by** *auto*
moreover have $\forall v \in \text{vars } ?A. \neg \text{is-bound } v ?B \wedge \neg \text{is-bound } v ?C$
proof
fix v
assume $v \in \text{vars } ?A$
then consider $(a) v = (z, \beta) \mid (b) v \in \text{vars } (\mathbf{S} \{(x, \beta) \mapsto z_\beta\} A)$
by *fastforce*
then show $\neg \text{is-bound } v ?B \wedge \neg \text{is-bound } v ?C$
proof cases
case a
then show *?thesis*
using $\langle (y, \beta) \neq (z, \beta) \rangle$ **by** *auto*
next
case b
then have $\neg \text{is-bound } v ?B$
by *simp*
moreover from b **and** $\langle (y, \beta) \notin \text{vars } A \rangle$ **and** $\langle (y, \beta) \neq (z, \beta) \rangle$ **have** $v \neq (y, \beta)$
using *renaming-substitution-minimal-change* **by** *blast*
then have $\neg \text{is-bound } v ?C$
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 $\S 4: \vdash (\lambda z_\beta. \mathbf{S} \{(x, \beta) \mapsto z_\beta\} A) =_{\beta \rightarrow \alpha} (\lambda y_\beta. \mathbf{S} \{(x, \beta) \mapsto y_\beta\} A)$
proof –
have $\vdash (\lambda z_\beta. \mathbf{S} \{(x, \beta) \mapsto z_\beta\} A) \cdot y_\beta =_\alpha \mathbf{S} \{(x, \beta) \mapsto y_\beta\} A$ (**is** $\vdash (\lambda z_\beta. ?B) \cdot ?A =. -$)
proof –
have $?A \in \text{wffs}_\beta$ **and** $?B \in \text{wffs}_\alpha$
by *blast fact*
moreover from $\langle (y, \beta) \notin \text{vars } A \rangle$ **and** $\langle (y, \beta) \neq (z, \beta) \rangle$ **have** $\forall v \in \text{vars } ?A. \neg \text{is-bound } v ?B$
using *absent-var-is-not-bound* **and** *renaming-substitution-minimal-change* **by** *auto*
ultimately have $\vdash (\lambda z_\beta. \mathbf{S} \{(x, \beta) \mapsto z_\beta\} A) \cdot y_\beta =_\alpha \mathbf{S} \{(z, \beta) \mapsto y_\beta\} \mathbf{S} \{(x, \beta) \mapsto z_\beta\} A$
using *prop-5203* **by** *fast*
moreover have $\mathbf{S} \{(z, \beta) \mapsto y_\beta\} \mathbf{S} \{(x, \beta) \mapsto z_\beta\} A = \mathbf{S} \{(x, \beta) \mapsto y_\beta\} A$
by (*fact renaming-substitution-composability*[*OF assms(2,3)*])
ultimately show *?thesis*
by (*simp only*:)
qed
with $\S 3$ **show** *?thesis*
by (*rule rule-R* [**where** $p = [\gg, \ll]$]) *auto*

```

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

```

```
lemmas  $\alpha = \text{prop-5206}$ 
```

6.8 Proposition 5207 (β -conversion)

```
context
```

```
begin
```

```
private lemma bound-var-renaming-equality:
```

```
  assumes  $A \in \text{wffs}_\alpha$ 
```

```
  and  $z_\gamma \neq y_\gamma$ 
```

```
  and  $(z, \gamma) \notin \text{vars } A$ 
```

```
  shows  $\vdash A =_\alpha \text{rename-bound-var } (y, \gamma) z A$ 
```

```
using assms proof induction
```

```
  case (var-is-wff  $\alpha x$ )
```

```
  then show ?case
```

```
    using prop-5200 by force
```

```
next
```

```
  case (con-is-wff  $\alpha c$ )
```

```
  then show ?case
```

```
    using prop-5200 by force
```

```
next
```

```
  case (app-is-wff  $\alpha \beta A B$ )
```

```
  then show ?case
```

```
    using Equality-Rules(4) by auto
```

```
next
```

```
  case (abs-is-wff  $\beta A \alpha x$ )
```

```
  then show ?case
```

```
proof (cases  $(y, \gamma) = (x, \alpha)$ )
```

```
  case True
```

```
  have  $\vdash \lambda y_\gamma. A =_{\gamma \rightarrow \beta} \lambda y_\gamma. A$ 
```

```
    by (fact abs-is-wff.hyps[THEN prop-5200[OF wffs-of-type-intros(4)]])
```

```
  moreover have  $\vdash A =_\beta \text{rename-bound-var } (y, \gamma) z A$ 
```

```
    using abs-is-wff.IH[OF assms(2)] and abs-is-wff.prem(2) by fastforce
```

```
  ultimately have  $\vdash \lambda y_\gamma. A =_{\gamma \rightarrow \beta} \lambda y_\gamma. \text{rename-bound-var } (y, \gamma) z A$ 
```

```
    by (rule rule-R[where  $p = [\rangle, \langle]$ ]) force+
```

```
  moreover
```

```
  have
```

```
     $\vdash \lambda y_\gamma. \text{rename-bound-var } (y, \gamma) z A$ 
```

```
       $=_{\gamma \rightarrow \beta}$ 
```

```
       $\lambda z_\gamma. \mathbf{S} \{(y, \gamma) \mapsto z_\gamma\} (\text{rename-bound-var } (y, \gamma) z A)$ 
```

```
proof -
```

```
  have  $\text{rename-bound-var } (y, \gamma) z A \in \text{wffs}_\beta$ 
```

```
    using hyp-derivable-form-is-wffso[OF  $\vdash A =_\beta \text{rename-bound-var } (y, \gamma) z A$ ]
```

```
    by (blast dest: wffs-from-equality)
```

moreover from *abs-is-wff.premis(2)* **have** $(z, \gamma) \notin \text{free-vars}(\text{rename-bound-var}(y, \gamma) z A)$
using *rename-bound-var-free-vars[OF abs-is-wff.hyps assms(2)]* **by** *simp*
moreover from *abs-is-wff.premis(2)* **have** *is-free-for* $(z_\gamma)(y, \gamma)(\text{rename-bound-var}(y, \gamma) z A)$
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 $\vdash \lambda y_\gamma. A =_{\gamma \rightarrow \beta} \lambda z_\gamma. \mathbf{S} \{(y, \gamma) \mapsto z_\gamma\}(\text{rename-bound-var}(y, \gamma) z A)$
by (*rule Equality-Rules(3)*)
then show *?thesis*
using *True* **by** *auto*
next
case *False*
have $\vdash \lambda x_\alpha. A =_{\alpha \rightarrow \beta} \lambda x_\alpha. A$
by (*fact abs-is-wff.hyps[THEN prop-5200[OF wffs-of-type-intros(4)]]*)
moreover have $\vdash A =_\beta \text{rename-bound-var}(y, \gamma) z A$
using *abs-is-wff.IH[OF assms(2)]* **and** *abs-is-wff.premis(2)* **by** *fastforce*
ultimately have $\vdash \lambda x_\alpha. A =_{\alpha \rightarrow \beta} \lambda x_\alpha. \text{rename-bound-var}(y, \gamma) z A$
by (*rule rule-R[where p = [»«]]*) *force+*
then show *?thesis*
using *False* **by** *auto*
qed
qed

proposition prop-5207:
assumes $A \in \text{wffs}_\alpha$ **and** $B \in \text{wffs}_\beta$
and *is-free-for* $A(x, \alpha) B$
shows $\vdash (\lambda x_\alpha. B) \cdot A =_\beta \mathbf{S} \{(x, \alpha) \mapsto A\} B$
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** $\vdash (\lambda x_\alpha. x_\alpha) \cdot A =_\alpha A$
using *axiom-4-2* **by** (*intro axiom-is-derivable-from-no-hyps*)
moreover have $\mathbf{S} \{(x, \alpha) \mapsto A\} (x_\alpha) = A$
by *force*
ultimately show *?thesis*
unfolding *True* **and** *var-is-wff* **by** *simp*
next
case *False*
with *assms(1)* **have** $\vdash (\lambda x_\alpha. y_\beta) \cdot A =_\beta y_\beta$
using *axiom-4-1-var* **by** (*intro axiom-is-derivable-from-no-hyps*)
moreover from *False* **have** $\mathbf{S} \{(x, \alpha) \mapsto A\} (y_\beta) = y_\beta$

by *auto*
 ultimately show *?thesis*
 unfolding *False* and *var-is-wff* by *simp*
 qed
 next
 case (*con-is-wff* *c*)
 from *assms*(1) have $\vdash (\lambda x_\alpha. \llbracket c \rrbracket_\beta) \cdot A =_\beta \llbracket c \rrbracket_\beta$
 using *axiom-4-1-con* by (*intro axiom-is-derivable-from-no-hyps*)
 moreover have $\mathbf{S} \{(x, \alpha) \mapsto A\} (\llbracket c \rrbracket_\beta) = \llbracket c \rrbracket_\beta$
 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*, α) *D* and *is-free-for* *A* (*x*, α) *C*
 using *is-free-for-from-app* by *iprover+*
 from $\langle \text{is-free-for } A \text{ } (x, \alpha) \text{ } D \rangle$ have $\vdash (\lambda x_\alpha. D) \cdot A =_{\gamma \rightarrow \beta} \mathbf{S} \{(x, \alpha) \mapsto A\} D$
 by (*fact less*(1)[*OF* $\langle \text{form-size } D < \text{form-size } B \rangle$ *assms*(1) *app-is-wff*(2)])
 moreover from $\langle \text{is-free-for } A \text{ } (x, \alpha) \text{ } C \rangle$ have $\vdash (\lambda x_\alpha. C) \cdot A =_\gamma \mathbf{S} \{(x, \alpha) \mapsto A\} C$
 by (*fact less*(1)[*OF* $\langle \text{form-size } C < \text{form-size } B \rangle$ *assms*(1) *app-is-wff*(3)])
 moreover have $\vdash (\lambda x_\alpha. D \cdot C) \cdot A =_\beta ((\lambda x_\alpha. D) \cdot A) \cdot ((\lambda x_\alpha. C) \cdot A)$
 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 $\vdash (\lambda x_\alpha. \lambda y_\gamma. D) \cdot A =_\beta \lambda y_\gamma. D$
 using *axiom-4-5*[*OF* *assms*(1) *abs-is-wff*(3)] by (*simp add: axiom-is-derivable-from-no-hyps*)
 moreover have $\mathbf{S} \{(x, \alpha) \mapsto A\} (\lambda y_\gamma. D) = \lambda y_\gamma. D$
 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*, α) *D*
 using *is-free-for-from-abs*[*OF less*(4)[*unfolded abs-is-wff*(2)]] and $\langle y_\gamma \neq x_\alpha \rangle$ by *blast*
 have $\vdash (\lambda x_\alpha. (\lambda y_\gamma. D)) \cdot A =_\beta \lambda y_\gamma. D$ $\mathbf{S} \{(x, \alpha) \mapsto A\} D$
 proof (*cases* ($y, \gamma \notin \text{vars } A$))
 case *True*
 with $\langle y_\gamma \neq x_\alpha \rangle$ have $\vdash (\lambda x_\alpha. \lambda y_\gamma. D) \cdot A =_{\gamma \rightarrow \delta} \lambda y_\gamma. (\lambda x_\alpha. D) \cdot A$
 using *axiom-4-4*[*OF* *assms*(1) *abs-is-wff*(3)] and *axiom-is-derivable-from-no-hyps* by *auto*
 moreover have $\vdash (\lambda x_\alpha. D) \cdot A =_\delta \mathbf{S} \{(x, \alpha) \mapsto A\} D$

by
 (

- fact less(1)*
- $[OF \langle \text{form-size } D < \text{form-size } B \rangle \text{ assms}(1) \langle D \in \text{wffs}_\delta \rangle \langle \text{is-free-for } A(x, \alpha) D \rangle]$

)

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** $(z, \gamma) \notin (\{(x, \alpha), (y, \gamma)\} \cup \text{vars } \{A, D\})$
using *fresh-var-existence by (metis Un-insert-left finite.simps insert-is-Un)*
then have $z_\gamma \neq x_\alpha$ **and** $z_\gamma \neq y_\gamma$ **and** $(z, \gamma) \notin \text{vars } \{A, D\}$
by *simp-all*
then show *?thesis*
proof (cases $(x, \alpha) \notin \text{free-vars } D$)
case *True*
define *D'* **where** $D' = \mathbf{S} \{(y, \gamma) \mapsto z_\gamma\} D$
have *is-substitution* $\{(y, \gamma) \mapsto z_\gamma\}$
by *auto*
with $\langle D \in \text{wffs}_\delta \rangle$ **and** *D'-def* **have** $D' \in \text{wffs}_\delta$
using *substitution-preserves-typing by blast*
then have $\vdash (\lambda x_\alpha. \lambda z_\gamma. D') \cdot A =_{\gamma \rightarrow \delta} \lambda z_\gamma. (\lambda x_\alpha. D') \cdot A$
using $\langle z_\gamma \neq x_\alpha \rangle$ **and** $\langle (z, \gamma) \notin \text{vars } \{A, D\} \rangle$ **and** *axiom-4-4[OF assms(1)]*
and *axiom-is-derivable-from-no-hyps*
by *auto*
moreover have $\S 2: \vdash (\lambda x_\alpha. D') \cdot A =_\delta D'$
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 $\langle \text{form-size } D < \text{form-size } B \rangle$ **by** *simp*
moreover from $\langle z_\gamma \neq x_\alpha \rangle$ **have** *is-free-for A (x, alpha) 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 $\vdash (\lambda x_\alpha. D') \cdot A =_\delta \mathbf{S} \{(x, \alpha) \mapsto A\} D'$
using *less(1)* **and** *assms(1)* **and** $\langle D' \in \text{wffs}_\delta \rangle$ **by** *simp*
moreover from $\langle z_\gamma \neq x_\alpha \rangle$ **have** $(x, \alpha) \notin \text{free-vars } D'$
unfolding *D'-def* **using** *substitution-preserves-freeness[OF True]* **by** *fast*
then have $\mathbf{S} \{(x, \alpha) \mapsto A\} D' = D'$
by (*fact free-var-singleton-substitution-neutrality*)
ultimately show *?thesis*
by (*simp only:*)

qed
ultimately have $\S 3: \vdash (\lambda x_\alpha. \lambda z_\gamma. D') \cdot A =_{\gamma \rightarrow \delta} \lambda z_\gamma. D'$ (**is** $\langle \vdash ?A3 \rangle$)
by (rule *rule-R[where p = [»,«]]*) *force+*
moreover have $\S 4: \vdash (\lambda y_\gamma. D) =_{\gamma \rightarrow \delta} \lambda z_\gamma. D'$

proof –
have $\langle z, \gamma \rangle \notin \text{free-vars } D$
using $\langle z, \gamma \rangle \notin \text{vars } \{A, D\}$ **and** *free-vars-in-all-vars-set* **by** *auto*
moreover have *is-free-for* $\langle z, \gamma \rangle (y, \gamma) D$
using $\langle z, \gamma \rangle \notin \text{vars } \{A, D\}$ **and** *absent-var-is-free-for* **by** *force*
ultimately have $\vdash \lambda y \gamma. D =_{\gamma \rightarrow \delta} \lambda z \gamma. \mathbf{S} \{(y, \gamma) \mapsto z \gamma\} D$
using $\alpha[OF \langle D \in \text{wffs}_\delta \rangle]$ **by** *fast*
then show *?thesis*
using *D'-def* **by** *blast*
qed
ultimately have $\S 5: \vdash (\lambda x \alpha. \lambda y \gamma. D) \cdot A =_{\gamma \rightarrow \delta} \lambda y \gamma. D$
proof –
note *rule-RR' = rule-RR[OF disjI2]*
have $\S 5_1: \vdash (\lambda x \alpha. \lambda y \gamma. D) \cdot A =_{\gamma \rightarrow \delta} \lambda z \gamma. D'$ (**is** $\langle \vdash ?A5_1 \rangle$)
by (*rule rule-RR'[OF §4, where* $p = [\langle \langle \rangle, \langle \rangle, \langle \rangle, \langle \rangle \rangle]$ **and** $C = ?A3]$) (*use* $\S 3$ **in** $\langle \text{force+} \rangle$)
show *?thesis*
by (*rule rule-RR'[OF §4, where* $p = [\langle \rangle \rangle]$ **and** $C = ?A5_1]$) (*use* $\S 5_1$ **in** $\langle \text{force+} \rangle$)
qed
then show *?thesis*
using *free-var-singleton-substitution-neutrality[OF $\langle (x, \alpha) \notin \text{free-vars } D \rangle]$*
by (*simp only: $\langle \beta = \gamma \rightarrow \delta \rangle$*)
next
case *False*
have $\langle y, \gamma \rangle \notin \text{free-vars } A$
proof (*rule ccontr*)
assume $\neg \langle y, \gamma \rangle \notin \text{free-vars } A$
moreover from $\langle \neg (x, \alpha) \notin \text{free-vars } D \rangle$ **obtain** p
where $p \in \text{positions } D$ **and** *is-free-at* $\langle x, \alpha \rangle p D$
using *free-vars-in-is-free-at* **by** *blast*
then have $\langle \# p \in \text{positions } (\lambda y \gamma. D) \rangle$ **and** *is-free-at* $\langle x, \alpha \rangle (\langle \# p \rangle (\lambda y \gamma. D))$
using *is-free-at-to-abs[OF $\langle \text{is-free-at } \langle x, \alpha \rangle p D \rangle]$* **and** $\langle y_\gamma \neq x_\alpha \rangle$ **by** (*simp, fast*)
moreover have *in-scope-of-abs* $\langle y, \gamma \rangle (\langle \# p \rangle (\lambda y \gamma. D))$
by *force*
ultimately have $\neg \text{is-free-for } A \langle x, \alpha \rangle (\lambda y \gamma. D)$
by *blast*
with $\langle \text{is-free-for } A \langle x, \alpha \rangle B \rangle$ [*unfolded abs-is-wff(2)*] **show** *False*
by *contradiction*
qed
define A' **where** $A' = \text{rename-bound-var } \langle y, \gamma \rangle z A$
have $A' \in \text{wffs}_\alpha$
unfolding *A'-def* **by** (*fact rename-bound-var-preserves-typing[OF assms(1)]*)
from $\langle z, \gamma \rangle \notin \text{vars } \{A, D\}$ **have** $\langle y, \gamma \rangle \notin \text{vars } A'$
using
 $\text{old-var-not-free-not-occurring-after-rename}$
 $[$
 $OF \text{ assms}(1) \langle z_\gamma \neq y_\gamma \rangle \langle (y, \gamma) \notin \text{free-vars } A \rangle$
 $]$
unfolding *A'-def* **by** *simp*
from *A'-def* **have** $\S 6: \vdash A =_\alpha A'$

using *bound-var-renaming-equality*[*OF* *assms*(1) $\langle z\gamma \neq y\gamma \rangle$] **and** $\langle (z, \gamma) \notin \text{vars } \{A, D\} \rangle$
by *simp*
moreover have $\S 7: \vdash (\lambda x_\alpha. \lambda y_\gamma. D) \cdot A' =_{\gamma \rightarrow \delta} \lambda y_\gamma. (\lambda x_\alpha. D) \cdot A'$ (**is** $\langle \vdash ?A7 \rangle$)
using *axiom-4-4*[*OF* $\langle A' \in \text{wffs}_\alpha \rangle \langle D \in \text{wffs}_\delta \rangle$]
and $\langle (y, \gamma) \notin \text{vars } A' \rangle$ **and** $\langle y_\gamma \neq x_\alpha \rangle$ **and** *axiom-is-derivable-from-no-hyps*
by *auto*
ultimately have $\S 8: \vdash (\lambda x_\alpha. \lambda y_\gamma. D) \cdot A =_{\gamma \rightarrow \delta} \lambda y_\gamma. (\lambda x_\alpha. D) \cdot A$
proof –
note *rule-RR'* = *rule-RR*[*OF* *disjI2*]
have $\S 8_1: \vdash (\lambda x_\alpha. \lambda y_\gamma. D) \cdot A =_{\gamma \rightarrow \delta} \lambda y_\gamma. (\lambda x_\alpha. D) \cdot A'$ (**is** $\langle \vdash ?A8_1 \rangle$)
by (*rule rule-RR'*[*OF* $\S 6$, **where** $p = [\langle \langle, \rangle, \rangle]$ **and** $C = ?A7$]) (*use* $\S 7$ **in** $\langle \text{force+} \rangle$)
show *?thesis*
by (*rule rule-RR'*[*OF* $\S 6$, **where** $p = [\langle \langle, \rangle, \rangle]$ **and** $C = ?A8_1$]) (*use* $\S 8_1$ **in** $\langle \text{force+} \rangle$)
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** $\S 9: \vdash (\lambda x_\alpha. D) \cdot A =_\delta \mathbf{S} \{(x, \alpha) \mapsto A\} D$
using *less*(1) **and** $\langle D \in \text{wffs}_\delta \rangle$ **and** $\langle \text{is-free-for } A (x, \alpha) D \rangle$ **by** (*simp only:*)
ultimately show *?thesis*
unfolding $\langle \beta = \gamma \rightarrow \delta \rangle$ **by** (*rule rule-R*[**where** $p = [\langle \rangle, \langle \rangle]$]) *force+*
qed
qed
then show *?thesis*
unfolding *abs-is-wff*(2) **using** *False* **and** *singleton-substitution-simps*(4) **by** *simp*
qed
qed
qed
qed
end

6.9 Proposition 5208

proposition *prop-5208*:
assumes $vs \neq []$ **and** $B \in \text{wffs}_\beta$
shows $\vdash \cdot^{\mathcal{Q}_*} (\lambda^{\mathcal{Q}_*} vs B)$ (*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 $\cdot^{\mathcal{Q}_*} (\lambda^{\mathcal{Q}_*} [v] B)$ (*map FVar [v]*) $= (\lambda x_\alpha. B) \cdot x_\alpha$
by *simp*
moreover have $\vdash (\lambda x_\alpha. B) \cdot x_\alpha =_\beta B$
proof –
have *is-free-for* $(x_\alpha) (x, \alpha) B$
by *fastforce*
then have $\vdash (\lambda x_\alpha. B) \cdot x_\alpha =_\beta \mathbf{S} \{(x, \alpha) \mapsto x_\alpha\} B$
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 $\vdash \cdot^{\mathcal{Q}_*} (\lambda^{\mathcal{Q}_*} (v \# vs) B) (\text{map } FVar (v \# vs)) =_{\beta} \cdot^{\mathcal{Q}_*} (\lambda^{\mathcal{Q}_*} vs B) (\text{map } FVar vs)$
proof –
have $\cdot^{\mathcal{Q}_*} (\lambda^{\mathcal{Q}_*} (v \# vs) B) (\text{map } FVar (v \# vs)) \in \text{wffs}_{\beta}$
proof –
have $\lambda^{\mathcal{Q}_*} (v \# vs) B \in \text{wffs}_{\text{foldr}} (\rightarrow) (\text{map } \text{snd} (v \# vs)) \beta$
using *generalized-abs-wff [OF assms(2)]* **by** *blast*
moreover
have $\forall k < \text{length} (\text{map } FVar (v \# vs)). \text{map } FVar (v \# vs) ! k \in \text{wffs}_{\text{map } \text{snd} (v \# vs) ! k}$
proof *safe*
fix k
assume $*$: $k < \text{length} (\text{map } FVar (v \# vs))$
moreover obtain x **and** α **where** $(v \# vs) ! k = (x, \alpha)$
by *fastforce*
with $*$ **have** $\text{map } FVar (v \# vs) ! k = x_{\alpha}$ **and** $\text{map } \text{snd} (v \# vs) ! k = \alpha$
by (*metis length-map nth-map snd-conv*)+
ultimately show $\text{map } FVar (v \# vs) ! k \in \text{wffs}_{\text{map } \text{snd} (v \# vs) ! k}$
by *fastforce*
qed
ultimately show *?thesis*
using *generalized-app-wff*[**where** $As = \text{map } FVar (v \# vs)$ **and** $ts = \text{map } \text{snd} (v \# vs)$] **by**

simp
qed
then have
 $\vdash \cdot^{\mathcal{Q}_*} (\lambda^{\mathcal{Q}_*} (v \# vs) B) (\text{map } FVar (v \# vs)) =_{\beta} \cdot^{\mathcal{Q}_*} (\lambda^{\mathcal{Q}_*} (v \# vs) B) (\text{map } FVar (v \# vs))$
by (*fact prop-5200*)
then have
 $\vdash \cdot^{\mathcal{Q}_*} (\lambda^{\mathcal{Q}_*} (v \# vs) B) (\text{map } FVar (v \# vs)) =_{\beta} \cdot^{\mathcal{Q}_*} ((\lambda^{\mathcal{Q}_*} (v \# vs) B) \cdot FVar v) (\text{map } FVar$
 $vs)$

by *simp*
moreover have $\vdash (\lambda^{\mathcal{Q}_*} (v \# vs) B) \cdot FVar v = \text{foldr} (\rightarrow) (\text{map } \text{snd} vs) \beta (\lambda^{\mathcal{Q}_*} vs B)$
proof –
have $\vdash (\lambda^{\mathcal{Q}_*} (v \# vs) B) \cdot FVar v = \text{foldr} (\rightarrow) (\text{map } \text{snd} vs) \beta \mathbf{S} \{v \mapsto FVar v\} (\lambda^{\mathcal{Q}_*} vs B)$
proof –
from $\langle v = (x, \alpha) \rangle$ **have** $\lambda^{\mathcal{Q}_*} (v \# vs) B = \lambda x_{\alpha}. \lambda^{\mathcal{Q}_*} vs B$
by *simp*
have $\lambda^{\mathcal{Q}_*} vs B \in \text{wffs}_{\text{foldr}} (\rightarrow) (\text{map } \text{snd} vs) \beta$
using *generalized-abs-wff [OF assms(2)]* **by** *blast*
moreover have *is-free-for* $(x_{\alpha}) (x, \alpha) (\lambda^{\mathcal{Q}_*} vs B)$
by *fastforce*
ultimately
have $\vdash (\lambda x_{\alpha}. \lambda^{\mathcal{Q}_*} vs B) \cdot x_{\alpha} = \text{foldr} (\rightarrow) (\text{map } \text{snd} vs) \beta \mathbf{S} \{(x, \alpha) \mapsto x_{\alpha}\} \lambda^{\mathcal{Q}_*} vs B$

by (rule prop-5207 [OF wffs-of-type-intros(1)])
 with $\langle v = (x, \alpha) \rangle$ 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 $\vdash \cdot^{\mathcal{Q}}_{\star} (\lambda^{\mathcal{Q}}_{\star} vs B) (\text{map FVar } vs) =_{\beta} B$
 by (fact cons.IH)
 ultimately show ?case
 by (rule rule-R [where p = [»]]) auto
 qed

6.10 Proposition 5209

proposition prop-5209:
 assumes $A \in \text{wffs}_{\alpha}$ and $B \in \text{wffs}_{\beta}$ and $C \in \text{wffs}_{\beta}$
 and $\vdash B =_{\beta} C$
 and is-free-for $A (x, \alpha) (B =_{\beta} C)$
 shows $\vdash \mathbf{S} \{(x, \alpha) \mapsto A\} (B =_{\beta} C)$
proof –
 have $\vdash (\lambda x_{\alpha}. B) \cdot A =_{\beta} (\lambda x_{\alpha}. B) \cdot A$
proof –
 have $(\lambda x_{\alpha}. B) \cdot A \in \text{wffs}_{\beta}$
 using assms(1,2) by blast
 then show ?thesis
 by (fact prop-5200)
 qed
 from this and assms(4) have $\vdash (\lambda x_{\alpha}. B) \cdot A =_{\beta} (\lambda x_{\alpha}. C) \cdot A$
 by (rule rule-R [where p = [»,«,«]]) force+
 moreover have $\vdash (\lambda x_{\alpha}. B) \cdot A =_{\beta} \mathbf{S} \{(x, \alpha) \mapsto A\} B$
proof –
 from assms(5)[unfolded equality-of-type-def] have is-free-for $A (x, \alpha) (Q_{\beta} \cdot B)$
 by (rule is-free-for-from-app)
 then have is-free-for $A (x, \alpha) B$

by (rule is-free-for-from-app)
 with *assms(1,2)* show *?thesis*
 by (rule prop-5207)
qed
moreover have $\vdash (\lambda x_\alpha. C) \cdot A =_\beta \mathbf{S} \{(x, \alpha) \mapsto A\} C$
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 $\vdash (\mathbf{S} \{(x, \alpha) \mapsto A\} B) =_\beta (\mathbf{S} \{(x, \alpha) \mapsto A\} C)$
 using *Equality-Rules(2,3)* by *blast*
then show *?thesis*
 by *simp*
qed

6.11 Proposition 5210

proposition *prop-5210*:
assumes $B \in \text{wffs}_\beta$
shows $\vdash T_o =_o (B =_\beta B)$
proof –
have §1:
 \vdash
 $((\lambda \eta_\beta. \eta_\beta) =_{\beta \rightarrow \beta} (\lambda \eta_\beta. \eta_\beta))$
 $=_o$
 $\forall \mathfrak{r}_\beta. ((\lambda \eta_\beta. \eta_\beta) \cdot \mathfrak{r}_\beta =_\beta (\lambda \eta_\beta. \eta_\beta) \cdot \mathfrak{r}_\beta)$
proof –
have $\vdash (\mathfrak{f}_{\beta \rightarrow \beta} =_{\beta \rightarrow \beta} \mathfrak{g}_{\beta \rightarrow \beta}) =_o \forall \mathfrak{r}_\beta. (\mathfrak{f}_{\beta \rightarrow \beta} \cdot \mathfrak{r}_\beta =_\beta \mathfrak{g}_{\beta \rightarrow \beta} \cdot \mathfrak{r}_\beta)$ (*is* $\vdash ?B =_o ?C$)
 using *axiom-3[unfolded equivalence-def]* by (rule *axiom-is-derivable-from-no-hyps*)
moreover have $(\lambda \eta_\beta. \eta_\beta) \in \text{wffs}_{\beta \rightarrow \beta}$ **and** $?B \in \text{wffs}_o$ **and** $?C \in \text{wffs}_o$
 by *auto*
moreover have *is-free-for* $(\lambda \eta_\beta. \eta_\beta) (\mathfrak{f}, \beta \rightarrow \beta) (?B =_o ?C)$
 by *simp*
ultimately have $\vdash \mathbf{S} \{(\mathfrak{f}, \beta \rightarrow \beta) \mapsto (\lambda \eta_\beta. \eta_\beta)\} (?B =_o ?C)$ (*is* $\vdash ?S$)
 using *prop-5209* by *presburger*
moreover have $?S =$
 $($
 $(\lambda \eta_\beta. \eta_\beta) =_{\beta \rightarrow \beta} \mathfrak{g}_{\beta \rightarrow \beta}) =_o \forall \mathfrak{r}_\beta. ((\lambda \eta_\beta. \eta_\beta) \cdot \mathfrak{r}_\beta =_\beta \mathfrak{g}_{\beta \rightarrow \beta} \cdot \mathfrak{r}_\beta$
 $)$ (*is* $= ?B' =_o ?C'$)
 by *simp*
ultimately have $\vdash ?B' =_o ?C'$
 by (*simp only*:)
moreover from $\langle (\lambda \eta_\beta. \eta_\beta) \in \text{wffs}_{\beta \rightarrow \beta} \rangle$ **have** $?B' \in \text{wffs}_o$ **and** $?C' \in \text{wffs}_o$
 by *auto*
moreover have *is-free-for* $(\lambda \eta_\beta. \eta_\beta) (\mathfrak{g}, \beta \rightarrow \beta) (?B' =_o ?C')$
 by *simp*
ultimately have $\vdash \mathbf{S} \{(\mathfrak{g}, \beta \rightarrow \beta) \mapsto (\lambda \eta_\beta. \eta_\beta)\} (?B' =_o ?C')$ (*is* $\vdash ?S'$)

using *prop-5209*[*OF* $\langle (\lambda \eta_\beta. \eta_\beta) \in \text{wffs}_{\beta \rightarrow \beta} \rangle$] **by** *blast*
 then show *?thesis*
 by *simp*
qed
 then have $\vdash (\lambda \mathfrak{r}_\beta. T_o) =_{\beta \rightarrow o} (\lambda \mathfrak{r}_\beta. (\mathfrak{r}_\beta =_\beta \mathfrak{r}_\beta))$
proof –
 have $\lambda \eta_\beta. \eta_\beta \in \text{wffs}_{\beta \rightarrow \beta}$
 by *blast*
 then have $\vdash \lambda \eta_\beta. \eta_\beta =_{\beta \rightarrow \beta} \lambda \eta_\beta. \eta_\beta$
 by (*fact prop-5200*)
 with §1 have $\vdash \forall \mathfrak{r}_\beta. ((\lambda \eta_\beta. \eta_\beta) \cdot \mathfrak{r}_\beta =_\beta (\lambda \eta_\beta. \eta_\beta) \cdot \mathfrak{r}_\beta)$
 using *rule-R* and *is-subform-at.simps(1)* **by** *blast*
 moreover have $\vdash (\lambda \eta_\beta. \eta_\beta) \cdot \mathfrak{r}_\beta =_\beta \mathfrak{r}_\beta$
 using *axiom-4-2*[*OF* *wffs-of-type-intros(1)*] **by** (*rule axiom-is-derivable-from-no-hyps*)
 ultimately have $\vdash \forall \mathfrak{r}_\beta. (\mathfrak{r}_\beta =_\beta (\lambda \eta_\beta. \eta_\beta) \cdot \mathfrak{r}_\beta)$
 by (*rule rule-R*[**where** $p = [\rangle, \langle, \langle, \rangle]$]) *auto*
 from *this* and $\langle (\lambda \eta_\beta. \eta_\beta) \cdot \mathfrak{r}_\beta =_\beta \mathfrak{r}_\beta \rangle$ have $\vdash \forall \mathfrak{r}_\beta. (\mathfrak{r}_\beta =_\beta \mathfrak{r}_\beta)$
 by (*rule rule-R*[**where** $p = [\rangle, \langle, \rangle]$]) *auto*
 then show *?thesis*
 unfolding *forall-def* and *PI-def* **by** (*fold equality-of-type-def*)
qed
 from *this* and *assms* have $\exists: \vdash (\lambda \mathfrak{r}_\beta. T_o) \cdot B =_o (\lambda \mathfrak{r}_\beta. (\mathfrak{r}_\beta =_\beta \mathfrak{r}_\beta)) \cdot B$
 by (*rule Equality-Rules(5)*)
 then show *?thesis*
proof –
 have $\vdash (\lambda \mathfrak{r}_\beta. T_o) \cdot B =_o T_o$
 using *prop-5207*[*OF* *assms true-wff*] **by** *fastforce*
 from \exists and *this* have $\vdash T_o =_o (\lambda \mathfrak{r}_\beta. (\mathfrak{r}_\beta =_\beta \mathfrak{r}_\beta)) \cdot B$
 by (*rule rule-R*[**where** $p = [\langle, \rangle]$]) *auto*
 moreover have $\vdash (\lambda \mathfrak{r}_\beta. (\mathfrak{r}_\beta =_\beta \mathfrak{r}_\beta)) \cdot B =_o (B =_\beta B)$
proof –
 have $\mathfrak{r}_\beta =_\beta \mathfrak{r}_\beta \in \text{wffs}_o$ and *is-free-for* B (\mathfrak{r}, β) $(\mathfrak{r}_\beta =_\beta \mathfrak{r}_\beta)$
 by (*blast, intro is-free-for-in-equality is-free-for-in-var*)
 moreover have **S** $\{(\mathfrak{r}, \beta) \mapsto B\} (\mathfrak{r}_\beta =_\beta \mathfrak{r}_\beta) = (B =_\beta B)$
 by *simp*
 ultimately show *?thesis*
 using *prop-5207*[*OF* *assms*] **by** *metis*
qed
 ultimately show *?thesis*
 by (*rule rule-R* [**where** $p = [\rangle]$]) *auto*
qed
qed

6.12 Proposition 5211

proposition *prop-5211*:

shows $\vdash (T_o \wedge^Q T_o) =_o T_o$

proof –

have *const-T-wff*: $(\lambda x_o. T_o) \in \text{wffs}_{o \rightarrow o}$ **for** x

by *blast*
have §1: $\vdash (\lambda\eta_o. T_o) \cdot T_o \wedge^{\mathcal{Q}} (\lambda\eta_o. T_o) \cdot F_o =_o \forall \mathfrak{r}_o. (\lambda\eta_o. T_o) \cdot \mathfrak{r}_o$
proof –
 have $\vdash \mathfrak{g}_{o \rightarrow o} \cdot T_o \wedge^{\mathcal{Q}} \mathfrak{g}_{o \rightarrow o} \cdot F_o =_o \forall \mathfrak{r}_o. \mathfrak{g}_{o \rightarrow o} \cdot \mathfrak{r}_o$ (**is** $\vdash ?B =_o ?C$)
 using *axiom-1[unfolded equivalence-def]* **by** (*rule axiom-is-derivable-from-no-hyps*)
moreover have $?B \in \text{wffs}_o$ **and** $?C \in \text{wffs}_o$
 by *auto*
moreover have *is-free-for* $(\lambda\eta_o. T_o)$ $(\mathfrak{g}, o \rightarrow o)$ $(?B =_o ?C)$
 by *simp*
ultimately have $\vdash \mathbf{S} \{(\mathfrak{g}, o \rightarrow o) \mapsto (\lambda\eta_o. T_o)\} (?B =_o ?C)$
 using *const-T-wff* **and** *prop-5209* **by** *presburger*
then show *?thesis*
 by *simp*
qed
then have $\vdash T_o \wedge^{\mathcal{Q}} T_o =_o \forall \mathfrak{r}_o. T_o$
proof –
 have *T-β-redex*: $\vdash (\lambda\eta_o. T_o) \cdot A =_o T_o$ **if** $A \in \text{wffs}_o$ **for** A
 using *that* **and** *prop-5207[OF that true-wff]* **by** *fastforce*
from §1 **and** *T-β-redex[OF true-wff]*
 have $\vdash T_o \wedge^{\mathcal{Q}} (\lambda\eta_o. T_o) \cdot F_o =_o \forall \mathfrak{r}_o. (\lambda\eta_o. T_o) \cdot \mathfrak{r}_o$
 by (*rule rule-R[where* $p = [\langle\langle, \rangle, \langle, \rangle]\rangle]$) *force+*
from this **and** *T-β-redex[OF false-wff]* **have** $\vdash T_o \wedge^{\mathcal{Q}} T_o =_o \forall \mathfrak{r}_o. (\lambda\eta_o. T_o) \cdot \mathfrak{r}_o$
 by (*rule rule-R[where* $p = [\langle\langle, \rangle, \rangle]\rangle]$) *force+*
from this **and** *T-β-redex[OF wffs-of-type-intros(1)]* **show** *?thesis*
 by (*rule rule-R[where* $p = [\rangle, \rangle, \langle]\rangle]$) *force+*
qed
moreover have $\vdash T_o =_o \forall \mathfrak{r}_o. T_o$
 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** $A \in \text{wffs}_{\alpha}$ **and** $C \in \text{wffs}_{\beta}$

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* $\langle A \in \text{wffs}_{\alpha} \rangle$])

moreover **have** $\vdash A =_{\alpha} B$

by *fact*

ultimately **have** $\vdash T_o =_o (A =_{\alpha} B)$

by (*rule rule-R*[**where** $p = [\rangle, \rangle]$]) *force+*

have $\vdash T_o =_o (C =_{\beta} C)$

by (*fact prop-5210*[*OF* $\langle C \in \text{wffs}_{\beta} \rangle$])

moreover **have** $\vdash C =_{\beta} D$

by *fact*

ultimately **have** $\vdash T_o =_o (C =_{\beta} D)$

by (*rule rule-R*[**where** $p = [\rangle, \rangle]$]) *force+*

then **show** *?thesis*

proof –

have $\vdash T_o \wedge^{\mathcal{Q}} T_o$

by (*fact prop-5212*)

from *this* **and** $\langle \vdash T_o =_o (A =_{\alpha} B) \rangle$ **have** $\vdash (A =_{\alpha} B) \wedge^{\mathcal{Q}} T_o$

by (*rule rule-R*[**where** $p = [\langle, \rangle]$]) *force+*

from *this* **and** $\langle \vdash T_o =_o (C =_{\beta} D) \rangle$ **show** *?thesis*

by (*rule rule-R*[**where** $p = [\rangle]$]) *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*: $(\lambda \mathfrak{r}_o. \mathfrak{r}_o) \in \text{wffs}_{o \rightarrow o}$

by *blast*

have §1: $\vdash (\lambda \mathfrak{r}_o. \mathfrak{r}_o) \cdot T_o \wedge^{\mathcal{Q}} (\lambda \mathfrak{r}_o. \mathfrak{r}_o) \cdot F_o =_o \forall \mathfrak{r}_o. (\lambda \mathfrak{r}_o. \mathfrak{r}_o) \cdot \mathfrak{r}_o$

proof –

have $\vdash \mathfrak{g}_{o \rightarrow o} \cdot T_o \wedge^{\mathcal{Q}} \mathfrak{g}_{o \rightarrow o} \cdot F_o =_o \forall \mathfrak{r}_o. \mathfrak{g}_{o \rightarrow o} \cdot \mathfrak{r}_o$ (**is** $\vdash ?B =_o ?C$)

using *axiom-1*[*unfolded equivalence-def*] **by** (*rule axiom-is-derivable-from-no-hyps*)

moreover **have** $?B \in \text{wffs}_o$ **and** $?C \in \text{wffs}_o$ **and** *is-free-for* $(\lambda \mathfrak{r}_o. \mathfrak{r}_o)$ $(\mathfrak{g}, o \rightarrow o)$ $(?B =_o ?C)$

by *auto*

ultimately **have** $\vdash \mathbf{S} \{(\mathfrak{g}, o \rightarrow o) \mapsto (\lambda \mathfrak{r}_o. \mathfrak{r}_o)\} (?B =_o ?C)$

using *id-on-o-is-wff* **and** *prop-5209* **by** *presburger*

then **show** *?thesis*

by *simp*
qed
then have $\vdash T_o \wedge^{\mathcal{Q}} F_o =_o \forall \mathfrak{r}_o. \mathfrak{r}_o$
proof –
 have *id- β -redex*: $\vdash (\lambda \mathfrak{r}_o. \mathfrak{r}_o) \cdot A =_o A$ **if** $A \in \text{wffs}_o$ **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 $\vdash T_o \wedge^{\mathcal{Q}} (\lambda \mathfrak{r}_o. \mathfrak{r}_o) \cdot F_o =_o \forall \mathfrak{r}_o. (\lambda \mathfrak{r}_o. \mathfrak{r}_o) \cdot \mathfrak{r}_o$
 by (*rule rule-R*[**where** $p = [\langle \rangle, \rangle, \langle \rangle, \rangle]$]) *force+*
from this and *id- β -redex*[*OF false-wff*] **have** $\vdash T_o \wedge^{\mathcal{Q}} F_o =_o \forall \mathfrak{r}_o. (\lambda \mathfrak{r}_o. \mathfrak{r}_o) \cdot \mathfrak{r}_o$
 by (*rule rule-R*[**where** $p = [\langle \rangle, \rangle, \rangle]$]) *force+*
from this and *id- β -redex*[*OF wffs-of-type-intros(1)*] **show** *?thesis*
 by (*rule rule-R*[**where** $p = [\rangle, \rangle, \langle \rangle]$]) *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** $A \in \text{wffs}_\alpha$
and *is-free-for* $A(x, \alpha) B$
shows $\mathcal{H} \vdash \mathbf{S} \{(x, \alpha) \mapsto A\} B$
proof –
from *assms(1)* **have** *is-hyps* \mathcal{H}
 by (*blast elim: is-derivable-from-hyps.cases*)
from *assms(1)* **have** $\mathcal{H} \vdash (\lambda \mathfrak{r}_\alpha. T_o) =_{\alpha \rightarrow o} (\lambda x_\alpha. B)$
 by *simp*
with *assms(2)* **have** $\mathcal{H} \vdash (\lambda \mathfrak{r}_\alpha. T_o) \cdot A =_o (\lambda x_\alpha. B) \cdot A$
 by (*intro Equality-Rules(5)*)
then have $\mathcal{H} \vdash T_o =_o \mathbf{S} \{(x, \alpha) \mapsto A\} B$
proof –
have $\mathcal{H} \vdash (\lambda \mathfrak{r}_\alpha. T_o) \cdot A =_o T_o$
proof –
have $\vdash (\lambda \mathfrak{r}_\alpha. T_o) \cdot A =_o T_o$
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 $\mathcal{H} \vdash (\lambda x_\alpha. B) \cdot A =_o \mathbf{S} \{(x, \alpha) \mapsto A\} B$
proof –
have $B \in \text{wffs}_o$
using *hyp-derivable-form-is-wffso*[*OF assms(1)*] **by** (*fastforce elim: wffs-from-forall*)
with *assms(2,3)* **have** $\vdash (\lambda x_\alpha. B) \cdot A =_o \mathbf{S} \{(x, \alpha) \mapsto A\} B$
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 $\langle \mathcal{H} \vdash (\lambda x_\alpha. T_o) \cdot A =_o (\lambda x_\alpha. B) \cdot A \rangle$ **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* $\langle is-hyps \mathcal{H} \rangle$])
from *this* **and** $\langle \mathcal{H} \vdash T_o =_o \mathbf{S} \{(x, \alpha) \mapsto A\} B \rangle$ **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 $\vdash (T_o \wedge^Q A) =_o A$

proof –

let $?B = \lambda x_o. (T_o \wedge^Q x_o =_o x_o)$

have *B-is-wff*: $?B \in wffs_{o \rightarrow o}$

by *auto*

have §1: $\vdash ?B \cdot T_o \wedge^Q ?B \cdot F_o =_o \forall x_o. ?B \cdot x_o$

proof –

have $\vdash \mathbf{g}_{o \rightarrow o} \cdot T_o \wedge^Q \mathbf{g}_{o \rightarrow o} \cdot F_o =_o \forall x_o. \mathbf{g}_{o \rightarrow o} \cdot x_o$ (**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* $?B$ ($\mathbf{g}, o \rightarrow o$) ($?C =_o ?D$)

by *auto*

ultimately have $\vdash \mathbf{S} \{(\mathbf{g}, o \rightarrow o) \mapsto ?B\}$ ($?C =_o ?D$)

using *B-is-wff* **and** *prop-5209* **by** *presburger*

then show *?thesis*

by *simp*

qed

have *: *is-free-for* A (x, o) ($T_o \wedge^Q x_o =_o x_o$) **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 $\vdash (T_o \wedge^Q T_o =_o T_o) \wedge^Q (T_o \wedge^Q F_o =_o F_o)$

by (*fact prop-5213*[*OF prop-5211 prop-5214*])

moreover

have $\vdash (T_o \wedge^Q T_o =_o T_o) \wedge^Q (T_o \wedge^Q F_o =_o F_o) =_o \forall x_o. (T_o \wedge^Q x_o =_o x_o)$

proof –

have *B-β-redex*: $\vdash ?B \cdot A =_o (T_o \wedge^Q A =_o A)$ **if** $A \in wffs_o$ **for** A

proof –

have $T_o \wedge^Q x_o =_o x_o \in wffs_o$

by *blast*

moreover have $\mathbf{S} \{(x, o) \mapsto A\} (T_o \wedge^Q x_o =_o x_o) = (T_o \wedge^Q A =_o A)$

by *simp*

ultimately show *?thesis*

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

6.18 Proposition 5217

proposition prop-5217:

shows $\vdash (T_o =_o F_o) =_o F_o$

proof –

let $?B = \lambda \mathfrak{r}_o. (T_o =_o \mathfrak{r}_o)$

have B-is-wff: $?B \in \text{wffs}_{o \rightarrow o}$

by auto

have *: is-free-for A (\mathfrak{r}, o) ($T_o =_o \mathfrak{r}_o$) for A

by (intro is-free-for-in-equality is-free-for-in-true is-free-for-in-var)

have §1: $\vdash ?B \cdot T_o \wedge^Q ?B \cdot F_o =_o \forall \mathfrak{r}_o. ?B \cdot \mathfrak{r}_o$

proof –

have $\vdash \mathfrak{g}_{o \rightarrow o} \cdot T_o \wedge^Q \mathfrak{g}_{o \rightarrow o} \cdot F_o =_o \forall \mathfrak{r}_o. \mathfrak{g}_{o \rightarrow o} \cdot \mathfrak{r}_o$ (is $\vdash ?C =_o ?D$)

using axiom-1[unfolded equivalence-def] by (rule axiom-is-derivable-from-no-hyps)

moreover have $?C \in \text{wffs}_o$ and $?D \in \text{wffs}_o$ and is-free-for ?B ($\mathfrak{g}, o \rightarrow o$) ($?C =_o ?D$)

by auto

ultimately have $\vdash \mathbf{S} \{(\mathfrak{g}, o \rightarrow o) \mapsto ?B\} (?C =_o ?D)$

using B-is-wff and prop-5209 by presburger

then show ?thesis

by simp

qed

then have $\vdash (T_o =_o T_o) \wedge^Q (T_o =_o F_o) =_o \forall \mathfrak{r}_o. (T_o =_o \mathfrak{r}_o)$ (is $\vdash ?A$)

proof –

have B-β-redex: $\vdash ?B \cdot A =_o (T_o =_o A)$ if $A \in \text{wffs}_o$ for A

proof –

have $T_o =_o \mathfrak{r}_o \in \text{wffs}_o$

by auto

moreover have $\mathbf{S} \{(\mathfrak{r}, o) \mapsto A\} (T_o =_o \mathfrak{r}_o) = (T_o =_o A)$

by simp

ultimately show ?thesis

using * and prop-5207[OF that] by metis

qed

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

shows $\vdash (T_o =_o A) =_o A$

proof –

let $?B = \lambda \mathfrak{r}_o. ((T_o =_o \mathfrak{r}_o) =_o \mathfrak{r}_o)$

have *B-is-wff*: $?B \in \text{wffs}_{o \rightarrow o}$

by *auto*

have §1: $\vdash ?B \cdot T_o \wedge^Q ?B \cdot F_o =_o \forall \mathfrak{r}_o. ?B \cdot \mathfrak{r}_o$

proof –

have $\vdash \mathfrak{g}_{o \rightarrow o} \cdot T_o \wedge^Q \mathfrak{g}_{o \rightarrow o} \cdot F_o =_o \forall \mathfrak{r}_o. \mathfrak{g}_{o \rightarrow o} \cdot \mathfrak{r}_o$ (*is* $\vdash ?C =_o ?D$)

using *axiom-1[unfolding equivalence-def]* by (*rule axiom-is-derivable-from-no-hyps*)

moreover have $?C \in \text{wffs}_o$ and $?D \in \text{wffs}_o$ and *is-free-for* $?B$ ($\mathfrak{g}, o \rightarrow o$) ($?C =_o ?D$)

by *auto*

ultimately have $\vdash \mathbf{S} \{(\mathfrak{g}, o \rightarrow o) \mapsto ?B\}$ ($?C =_o ?D$)

using *prop-5209[OF B-is-wff]* by *presburger*

then show *?thesis*

by *simp*

qed

have *: *is-free-for* A (\mathfrak{r}, o) $((T_o =_o \mathfrak{r}_o) =_o \mathfrak{r}_o)$ for A

by (*intro is-free-for-in-equality is-free-for-in-true is-free-for-in-var*)

have §2:

\vdash

$((T_o =_o T_o) =_o T_o) \wedge^Q ((T_o =_o F_o) =_o F_o)$

$=_o$

$\forall \mathfrak{r}_o. ((T_o =_o \mathfrak{r}_o) =_o \mathfrak{r}_o)$

proof –

have *B-β-redex*: $\vdash ?B \cdot A =_o ((T_o =_o A) =_o A)$ if $A \in \text{wffs}_o$ for A

proof –

have $(T_o =_o \mathfrak{r}_o) =_o \mathfrak{r}_o \in \text{wffs}_o$

by *auto*

moreover have $\mathbf{S} \{(\mathfrak{r}, o) \mapsto A\}$ $((T_o =_o \mathfrak{r}_o) =_o \mathfrak{r}_o) = ((T_o =_o A) =_o A)$

by *simp*

ultimately show *?thesis*

using * and *prop-5207[OF that]* by *metis*

qed

from §1 and *B-β-redex[OF true-wff]*

have $\vdash ((T_o =_o T_o) =_o T_o) \wedge^Q ?B \cdot F_o =_o \forall \mathfrak{r}_o. ?B \cdot \mathfrak{r}_o$

by (*rule rule-R[where p = [«», «», «»]]*) *force+*

from *this* and *B-β-redex[OF false-wff]*

have $\vdash ((T_o =_o T_o) =_o T_o) \wedge^Q ((T_o =_o F_o) =_o F_o) =_o \forall \mathfrak{r}_o. ?B \cdot \mathfrak{r}_o$

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 $\vdash ((T_o =_o T_o) =_o T_o) \wedge^Q ((T_o =_o F_o) =_o F_o)$
 by (fact prop-5213[OF §3 prop-5217])
 from this and §2 have §4: $\vdash \forall \mathfrak{r}_o. ((T_o =_o \mathfrak{r}_o) =_o \mathfrak{r}_o)$
 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 $A \in \text{wffs}_o$
 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 $\langle \mathcal{H} \vdash A \rangle$ 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 $\langle \mathcal{H} \vdash T_o =_o A \rangle$ show $\mathcal{H} \vdash A$
 by (rule Equality-Rules(1)[unfolded equivalence-def])
 qed

proposition prop-5219-2:
 assumes $A \in \text{wffs}_o$
 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 $\vdash (\lambda x_\alpha. T_o) =_{\alpha \rightarrow o} (\lambda z_\alpha. T_o)$
proof –

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

proposition *prop-5220*:

assumes $\mathcal{H} \vdash A$

and $(x, \alpha) \notin \text{free-vars } \mathcal{H}$

shows $\mathcal{H} \vdash \forall x_\alpha. A$

proof –

from $\langle \mathcal{H} \vdash A \rangle$ **have** *is-hyps* \mathcal{H}

by (*blast dest: is-derivable-from-hyps.cases*)

have $\mathcal{H} \vdash A$

by *fact*

then have $\S 2: \mathcal{H} \vdash T_o =_o A$

using *rule-T(1)[OF hyp-derivable-form-is-wffso[OF $\langle \mathcal{H} \vdash A \rangle$]]* **by** *simp*

have $\S 3: \mathcal{H} \vdash (\lambda \mathfrak{r}_\alpha. T_o) =_{\alpha \rightarrow o} (\lambda x_\alpha. T_o)$

by (*fact derivability-implies-hyp-derivability[OF const-true- α -conversion $\langle \text{is-hyps } \mathcal{H} \rangle$]*)

from $\S 3$ **and** $\S 2$ **have** $\mathcal{H} \vdash \lambda \mathfrak{r}_\alpha. T_o =_{\alpha \rightarrow o} \lambda x_\alpha. A$

proof (*induction rule: rule-R'[where $p = [\rangle, \langle]]$*)

case *no-capture*

have $*$: $[\rangle, \langle] \in \text{positions } (\lambda \mathfrak{r}_\alpha. T_o =_{\alpha \rightarrow o} \lambda x_\alpha. T_o)$

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 $\text{lset } vs \cap \text{free-vars } \mathcal{H} = \{\}$

shows $\mathcal{H} \vdash \forall \mathcal{Q}_* vs A$

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.prem*s **have** $\text{lset } vs \cap \text{free-vars } \mathcal{H} = \{\}$ **and** $(x, \alpha) \notin \text{free-vars } \mathcal{H}$
 by *simp-all*
 from $\langle \text{lset } vs \cap \text{free-vars } \mathcal{H} = \{\} \rangle$ **have** $\mathcal{H} \vdash \forall^{\mathcal{Q}_*} vs A$
 by (*fact Cons.IH*)
 with $\langle (x, \alpha) \notin \text{free-vars } \mathcal{H} \rangle$ **and** $\langle v = (x, \alpha) \rangle$ **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 $(x, \alpha) \notin \text{free-vars } \mathcal{H}$
 and *is-free-for* $A (x, \alpha) B$
 and $A \in \text{wffs}_\alpha$
 shows $\mathcal{H} \vdash \mathbf{S} \{(x, \alpha) \mapsto A\} B$
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 $\forall v \in \text{fmdom}' \vartheta. \text{var-name } v \notin \text{free-var-names } \mathcal{H} \wedge \text{is-free-for } (\vartheta \ \$\$! \ v) \ v \ B$
 and $\vartheta \neq \{\$\$\}$
 shows $\mathcal{H} \vdash \mathbf{S} \vartheta B$
proof –
obtain *xs* **and** *As*
 where $\text{lset } xs = \text{fmdom}' \vartheta$ — i.e., $x_{\alpha_1}^1, \dots, x_{\alpha_n}^n$
 and $As = \text{map } ((\$\$!) \ \vartheta) \ xs$ — i.e., $A_{\alpha_1}^1, \dots, A_{\alpha_n}^n$
 and $\text{length } xs = \text{card } (\text{fmdom}' \ \vartheta)$
 by (*metis distinct-card finite-distinct-list finite-fmdom'*)
then have *distinct xs*
 by (*simp add: card-distinct*)
from $\langle \text{lset } xs = \text{fmdom}' \ \vartheta \rangle$ **and** $\langle As = \text{map } ((\$\$!) \ \vartheta) \ xs \rangle$ **have** $\text{lset } As = \text{fmdom}' \ \vartheta$
 by (*intro subset-antisym subsetI*) (*force simp add: fmllookup-dom'-iff fmllookup-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., $y_{\alpha_1}^1, \dots, y_{\alpha_n}^n$
 where $\text{length } ys = \text{length } xs$

and *distinct ys*
and *ys-fresh:*
 $(\text{var-name } \langle \text{lset } ys \rangle \cap (\text{var-name } \langle (\text{vars } B \cup \text{vars } (\text{lset } As) \cup \text{vars } \mathcal{H} \cup \text{lset } xs) \rangle) = \{\})$
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: \langle As = map ((\\$\\$!) \vartheta) xs \rangle)*
— $\mathcal{H} \vdash \S_{\substack{x_{\alpha_1}^1 \dots x_{\alpha_k}^k x_{\alpha_{k+1}}^{k+1} \dots x_{\alpha_n}^n \\ A_{\alpha_1}^1 \dots A_{\alpha_k}^k y_{\alpha_{k+1}}^{k+1} \dots y_{\alpha_n}^n}} B$
have $\mathcal{H} \vdash \mathbf{S}$ *(fmap-of-list (zip xs (take k As @ drop k (map FVar ys)))) B* **if** $k \leq \text{length } xs$ **for** k
using *that proof (induction k)*
case 0
have $\mathcal{H} \vdash \mathbf{S}$ *(fmap-of-list (zip xs (map FVar ys))) B*
using $\langle \text{length } ys = \text{length } xs \rangle$
and $\langle \text{length } xs = \text{length } As \rangle$
and $\langle (\text{var-name } \langle \text{lset } ys \rangle \cap (\text{var-name } \langle (\text{vars } B \cup \text{vars } (\text{lset } As) \cup \text{vars } \mathcal{H} \cup \text{lset } xs) \rangle) = \{\}) \rangle$
and $\langle \text{lset } xs = \text{fmdom}' \vartheta \rangle$
and $\langle \text{distinct } ys \rangle$
and *assms(3)*
and $\langle \text{map var-type } ys = \text{map var-type } xs \rangle$
and $\langle \text{distinct } xs \rangle$
and $\langle \text{length } xs = \text{card } (\text{fmdom}' \vartheta) \rangle$
proof *(induction ys xs As arbitrary: \vartheta rule: list-induct3)*
case *Nil*
with *assms(1)* **show** *?case*
using *empty-substitution-neutrality* **by** *auto*
next
— In the following:

- $\vartheta = \{x_{\alpha_1}^1 \mapsto y_{\alpha_1}^1, \dots, x_{\alpha_n}^n \mapsto y_{\alpha_n}^n\}$
- $?\vartheta = \{x_{\alpha_2}^2 \mapsto y_{\alpha_2}^2, \dots, x_{\alpha_n}^n \mapsto y_{\alpha_n}^n\}$
- $v_x = x_{\alpha_1}^1$, and $v_y = y_{\alpha_1}^1$

case *(Cons v_y ys v_x xs A' As')*
let $?\vartheta = \text{fmap-of-list } (\text{zip } xs \text{ (map FVar } ys))$
from *Cons.hyps(1)* **have** $\text{lset } xs = \text{fmdom}' ?\vartheta$
by *simp*
from *Cons.hyps(1)* **and** *Cons.prem(6)* **have** $\text{fmrans}' ?\vartheta = \text{FVar } \langle \text{lset } ys \rangle$
by *force*
have *is-substitution ?\vartheta*
unfolding *is-substitution-def* **proof**
fix v
assume $v \in \text{fmdom}' ?\vartheta$
with $\langle \text{lset } xs = \text{fmdom}' ?\vartheta \rangle$ **obtain** k **where** $v = xs ! k$ **and** $k < \text{length } xs$
by *(metis in-set-conv-nth)*
moreover **obtain** α **where** *var-type v = \alpha*
by *blast*
moreover **from** $\langle k < \text{length } xs \rangle$ **and** $\langle v = xs ! k \rangle$ **have** $?\vartheta \ \$\$! \ v = (\text{map FVar } ys) ! k$
using *Cons.hyps(1)* **and** *Cons.prem(6)* **by** *auto*

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

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

by (*fastforce intro: fmap.map-ident-strong*)
 with $\langle v_x \notin \text{fmdom}' \vartheta \rangle$ **show** *?thesis*
 using $\langle \forall v \in \text{fmdom}' \vartheta. \text{var-name } v \notin \text{free-var-names } \mathcal{H} \wedge \text{is-free-for } (\vartheta \text{ \&\&! } v) v B \rangle$
 and *substitution-consolidation* **by** *auto*
qed
 — $\mathcal{H} \vdash \S_{y_{\alpha_1}^1 \dots y_{\alpha_n}^n}^{x_{\alpha_1}^1 \dots x_{\alpha_n}^n} B$
 ultimately **show** *?case*
 using $\langle v_x = (x, \alpha) \rangle$ and $\langle v_x \notin \text{fmdom}' \vartheta \rangle$ and *fmap-singleton-comm* **by** *fastforce*
qed
 with *0* and *that* **show** *?case*
 by *auto*
next
 case (*Suc k*)
 let $?ps = \lambda k. \text{zip } xs \text{ (take } k \text{ As @ drop } k \text{ (map FVar ys))}$
 let $?y = ys ! k$ and $?A = As ! k$
 let $?d = \lambda k. \text{fmap-of-list } (?ps \ k)$
 let $?d' = \lambda k. \text{fmap-of-list } (\text{map } (\lambda(v', A'). (v', \mathbf{S} \{?y \mapsto ?A\} A')) (?ps \ k))$
 have $\text{fmdom}' (?d \ k) = \text{lset } xs$ **for** k'
 by (*simp add: <length xs = length As> <length ys = length xs>*)
 have $\text{fmdom}' (?d' \ k') = \text{lset } xs$ **for** k'
 using $\langle \text{length } xs = \text{length } As \rangle$ and $\langle \text{length } ys = \text{length } xs \rangle$ and *fmdom'-fmap-of-list* **by** *simp*
 have $?y \in \text{lset } ys$
 using *Suc.prem*s $\langle \text{length } ys = \text{length } xs \rangle$ **by** *simp*
 have $\forall j < \text{length } ys. ys ! j \notin \text{vars } (\mathcal{H}::\text{form set}) \wedge ys ! j \notin \text{vars } B$
 using $\langle (\text{var-name } ' \text{lset } ys) \cap (\text{var-name } ' (\text{vars } B \cup \text{vars } (\text{lset } As) \cup \text{vars } \mathcal{H} \cup \text{lset } xs)) = \{\} \rangle$
 by *force*
 obtain n_y and α_y where $(n_y, \alpha_y) = ?y$
 using *surj-pair*[of $?y$] **by** *fastforce*
 moreover have $?A \in \text{wffs}_{\alpha_y}$
proof —
 from *Suc.prem*s and $\langle (n_y, \alpha_y) = ?y \rangle$ have *var-type* $(xs ! k) = \alpha_y$
 using $\langle \text{length } ys = \text{length } xs \rangle$ and $\langle \text{map var-type } ys = \text{map var-type } xs \rangle$ and *Suc-le-lessD*
 by (*metis nth-map snd-conv*)
 with *Suc.prem*s and *assms*(2) and $\langle \text{lset } xs = \text{fmdom}' \vartheta \rangle$ and $\langle As = \text{map } ((\text{\&\&!}) \vartheta) xs \rangle$ **show**
?thesis
 using *less-eq-Suc-le* and *nth-mem* **by** *fastforce*
qed
 ultimately have *is-substitution* $\{?y \mapsto ?A\}$
 by *auto*
 have *wfs: is-substitution* $(?d \ k)$ **for** k
unfolding *is-substitution-def* **proof**
 fix v
 assume $v \in \text{fmdom}' (?d \ k)$
 with $\langle \text{fmdom}' (?d \ k) = \text{lset } xs \rangle$ obtain j where $v = xs ! j$ and $j < \text{length } xs$
 by (*fastforce simp add: in-set-conv-nth*)
 obtain α where *var-type* $v = \alpha$
 by *blast*
 show *case* v of $(x, \alpha) \Rightarrow (?d \ k) \text{ \&\&! } (x, \alpha) \in \text{wffs}_{\alpha}$
proof (*cases* $j < k$)

case *True*
with $\langle j < \text{length } xs \rangle$ **and** $\langle v = xs ! j \rangle$ **have** $(? \vartheta \ k) \ \S\!\$ \ v = As ! j$
using $\langle \text{distinct } xs \rangle$ **and** $\langle \text{length } xs = \text{length } As \rangle$ **and** $\langle \text{length } ys = \text{length } xs \rangle$ **by force**
with $\text{assms}(2) \ \langle v = xs ! j \rangle$ **and** $\langle v \in \text{fmdom}' (? \vartheta \ k) \rangle$ **and** $\langle \text{var-type } v = \alpha \rangle$ **and** $\langle j < \text{length } xs \rangle$
have $(? \vartheta \ k) \ \S\!\$ \ v \in \text{wffs}_\alpha$
using $\langle As = \text{map } ((\S\!\$) \ \vartheta) \ xs \rangle$ **and** $\langle \text{fmdom}' (? \vartheta \ k) = \text{lset } xs \rangle$ **and** $\langle \text{lset } xs = \text{fmdom}' \ \vartheta \rangle$
by auto
then show *?thesis*
using $\langle \text{var-type } v = \alpha \rangle$ **by force**
next
case *False*
with $\langle j < \text{length } xs \rangle$ **and** $\langle v = xs ! j \rangle$ **have** $(? \vartheta \ k) \ \S\!\$ \ v = FVar \ (ys ! j)$
using $\langle \text{distinct } xs \rangle$ **and** $\langle \text{length } xs = \text{length } As \rangle$ **and** $\langle \text{length } ys = \text{length } xs \rangle$ **by force**
with $\langle j < \text{length } xs \rangle$ **and** $\langle v = xs ! j \rangle$ **and** $\langle \text{var-type } v = \alpha \rangle$ **and** $\langle \text{length } ys = \text{length } xs \rangle$
have $(? \vartheta \ k) \ \S\!\$ \ v \in \text{wffs}_\alpha$
using $\langle \text{map var-type } ys = \text{map var-type } xs \rangle$ **and** $\text{surj-pair}[\text{of } ys ! j]$
by $(\text{metis } \text{nth-map } \text{snd-conv } \text{wffs-of-type-intros}(1))$
then show *?thesis*
using $\langle \text{var-type } v = \alpha \rangle$ **by force**
qed
qed
have $\vartheta' \text{-alt-def: } ? \vartheta' \ k = \text{fmap-of-list}$
 $(\text{zip } xs$
 $(\text{take } k \ (\text{map } (\lambda A'. \ \mathbf{S} \ \{ ?y \mapsto ?A \} \ A') \ As)$
 $\ @$
 $(\text{drop } k \ (\text{map } (\lambda A'. \ \mathbf{S} \ \{ ?y \mapsto ?A \} \ A') \ (\text{map } FVar \ ys))))$
proof –
have
 $\text{fmap-of-list } (\text{zip } xs \ (\text{map } (\lambda A'. \ \mathbf{S} \ \{ ?y \mapsto ?A \} \ A') \ (\text{take } k \ As \ @ \ \text{drop } k \ (\text{map } FVar \ ys))))$
 $=$
 fmap-of-list
 $(\text{zip } xs$
 $(\text{map } (\lambda A'. \ \mathbf{S} \ \{ ?y \mapsto ?A \} \ A') \ (\text{take } k \ As)$
 $\ @$
 $(\text{drop } k \ (\text{map } (\lambda A'. \ \mathbf{S} \ \{ ?y \mapsto ?A \} \ A') \ (\text{map } FVar \ ys))))$
by $(\text{simp add: drop-map})$
then show *?thesis*
by $(\text{metis take-map zip-map2})$
qed
 $-\ \mathcal{H} \vdash \S_{A_{\alpha_1}^1 \dots A_{\alpha_k}^k y_{\alpha_{k+1}}^{k+1} \dots y_{\alpha_n}^n} \ x_{\alpha_1}^1 \dots x_{\alpha_k}^k x_{\alpha_{k+1}}^{k+1} \dots x_{\alpha_n}^n \ B$
have $\mathcal{H} \vdash \mathbf{S} \ (? \vartheta \ k) \ B$
by $(\text{fact } \text{Suc.IH}[\text{OF } \text{Suc-leD}[\text{OF } \text{Suc.prem}]])$
 $-\ \mathcal{H} \vdash \S_{A_{\alpha_{k+1}}^{k+1}} \ y_{\alpha_{k+1}}^{k+1} \ \S_{A_{\alpha_1}^1 \dots A_{\alpha_k}^k y_{\alpha_{k+1}}^{k+1} \dots y_{\alpha_n}^n} \ x_{\alpha_1}^1 \dots x_{\alpha_k}^k x_{\alpha_{k+1}}^{k+1} \dots x_{\alpha_n}^n \ B$
then have $\mathcal{H} \vdash \mathbf{S} \ \{ ?y \mapsto ?A \} \ \mathbf{S} \ (? \vartheta \ k) \ B$
proof –
from $\langle (n_y, \alpha_y) = ?y \rangle$ **and** $\langle \text{length } ys = \text{length } xs \rangle$ **have** $(n_y, \alpha_y) \notin \text{free-vars } \mathcal{H}$
using $\langle \forall j < \text{length } ys. \ ys ! j \notin \text{vars } (\mathcal{H}::\text{form set}) \wedge \ ys ! j \notin \text{vars } B \rangle$

and free-vars-in-all-vars-set and Suc-le-lessD[OF Suc.prem] by fastforce
moreover have is-free-for ?A (n_y, α_y) (S (?∅ k) B)
proof –
have is-substitution (fmdrop (xs ! k) (?∅ k))
using wfs and ⟨fmdom' (?∅ k) = lset xs⟩ by force
moreover from Suc-le-lessD[OF Suc.prem] 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) (?∅ k))
by simp
moreover have
 $\forall v \in \text{fmdom}' (\text{fmdrop } (xs ! k) (?∅ k)). ?y \notin \text{vars } (\text{fmdrop } (xs ! k) (?∅ k) \text{ \&\&! } v)$
proof
fix v
assume v ∈ fmdom' (fmdrop (xs ! k) (?∅ k))
then have v ∈ fmdom' (?∅ k)
by simp
with ⟨fmdom' (?∅ k) = lset xs⟩ obtain j where v = xs ! j and j < length xs and j ≠ k
using ⟨v ∈ fmdom' (fmdrop (xs ! k) (?∅ k))⟩
and ⟨(xs ! k) ∉ fmdom' (fmdrop (xs ! k) (?∅ k))⟩ by (metis in-set-conv-nth)
then show ?y ∉ vars ((fmdrop (xs ! k) (?∅ k)) \&\&! v)
proof (cases j < k)
case True
with ⟨j < length xs⟩ and ⟨v = xs ! j⟩ have (?∅ 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) (?∅ k))⟩ by auto
next
case False
with ⟨j < length xs⟩ and ⟨v = xs ! j⟩ have (?∅ 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.prem] 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) (?∅ k))⟩
and ⟨xs ! k ∉ fmdom' (fmdrop (xs ! k) (?∅ k))⟩ and surj-pair[of ys ! j] by fastforce
qed
qed
moreover from ⟨k < length xs⟩ and ⟨length ys = length xs⟩ have ?y ∉ vars B
by (simp add: ⟨∀ j < length ys. ys ! j ∉ vars H ∧ ys ! j ∉ vars B⟩)
moreover have is-free-for ?A (xs ! k) B
proof –
from Suc-le-lessD[OF Suc.prem] and ⟨lset xs = fmdom' ∅⟩ have xs ! k ∈ fmdom' ∅
using nth-mem by blast
moreover from Suc.prem and ⟨As = map ((\&\&! ∅) xs) have ∅ \&\&! (xs ! k) = ?A

by *fastforce*
 ultimately show *?thesis*
 using *assms(3)* by *simp*
 qed
 moreover
 have $\forall v \in \text{fmdom}' (\text{fmdrop } (xs ! k) (?\vartheta k)). \text{is-free-for } (\text{fmdrop } (xs ! k) (?\vartheta k) \text{ $$$ } v) v B$
 proof
 fix v
 assume $v \in \text{fmdom}' (\text{fmdrop } (xs ! k) (?\vartheta k))$
 then have $v \in \text{fmdom}' (?\vartheta k)$
 by *simp*
 with $\langle \text{fmdom}' (?\vartheta k) = \text{lset } xs \rangle$ obtain j where $v = xs ! j$ and $j < \text{length } xs$ and $j \neq k$
 using $\langle v \in \text{fmdom}' (\text{fmdrop } (xs ! k) (?\vartheta k)) \rangle$
 and $\langle xs ! k \notin \text{fmdom}' (\text{fmdrop } (xs ! k) (?\vartheta k)) \rangle$ by *(metis in-set-conv-nth)*
 then show *is-free-for* $(\text{fmdrop } (xs ! k) (?\vartheta k) \text{ $$$ } v) v B$
 proof *(cases j < k)*
 case *True*
 with $\langle j < \text{length } xs \rangle$ and $\langle v = xs ! j \rangle$ have $(?\vartheta k) \text{ $$$ } v = As ! j$
 using $\langle \text{distinct } xs \rangle$ and $\langle \text{length } xs = \text{length } As \rangle$ and $\langle \text{length } ys = \text{length } xs \rangle$ by *force*
 moreover have *is-free-for* $(As ! j) v B$
 proof –
 from $\langle j < \text{length } xs \rangle$ and $\langle \text{lset } xs = \text{fmdom}' \vartheta \rangle$ and $\langle v = xs ! j \rangle$ have $v \in \text{fmdom}' \vartheta$
 using *nth-mem* by *blast*
 moreover have $\vartheta \text{ $$$ } v = As ! j$
 by *(simp add: $\langle As = \text{map } ((\text{$$$ } \vartheta) xs) \langle j < \text{length } xs \rangle \langle v = xs ! j \rangle$)*
 ultimately show *?thesis*
 using *assms(3)* by *simp*
 qed
 ultimately show *?thesis*
 using $\langle v \in \text{fmdom}' (\text{fmdrop } (xs ! k) (?\vartheta k)) \rangle$ by *auto*
 next
 case *False*
 with $\langle j < \text{length } xs \rangle$ and $\langle v = xs ! j \rangle$ have $(?\vartheta k) \text{ $$$ } v = \text{FVar } (ys ! j)$
 using $\langle \text{distinct } xs \rangle$ and $\langle \text{length } xs = \text{length } As \rangle$ and $\langle \text{length } ys = \text{length } xs \rangle$ by *force*
 moreover from $\langle j < \text{length } xs \rangle$ and $\langle \text{length } ys = \text{length } xs \rangle$ have $ys ! j \notin \text{vars } B$
 using $\langle \forall j < \text{length } ys. ys ! j \notin \text{vars } \mathcal{H} \wedge ys ! j \notin \text{vars } B \rangle$ by *simp*
 then have *is-free-for* $(\text{FVar } (ys ! j)) v B$
 by *(fact absent-var-is-free-for)*
 ultimately show *?thesis*
 using $\langle v \in \text{fmdom}' (\text{fmdrop } (xs ! k) (?\vartheta k)) \rangle$ by *auto*
 qed
 qed
 ultimately have *is-free-for* $?A (ys ! k) \mathbf{S} (\{xs ! k \mapsto \text{FVar } ?y\} ++_f \text{fmdrop } (xs ! k) (?\vartheta k)) B$
 using *is-free-for-with-renaming-substitution* by *presburger*
 moreover have $\mathbf{S} (\{xs ! k \mapsto \text{FVar } ?y\} ++_f \text{fmdrop } (xs ! k) (?\vartheta k)) B = \mathbf{S} (?\vartheta k) B$
 using $\langle \text{length } xs = \text{length } As \rangle$ and $\langle \text{length } ys = \text{length } xs \rangle$ and *Suc-le-eq* and *Suc.prem*
 and $\langle \text{distinct } xs \rangle$ by *simp*
 ultimately show *?thesis*
 unfolding $\langle (n_y, \alpha_y) = ?y \rangle$ by *simp*

qed
ultimately show *?thesis*
using *prop-5221-aux*[$OF \langle \mathcal{H} \vdash \mathbf{S} (?\vartheta k) B \rangle$] and $\langle ?A \in \text{wffs}_{\alpha_y} \rangle$ and $\langle (n_y, \alpha_y) = ?y \rangle$ by *metis*

qed
— $\S \frac{y_{\alpha_{k+1}}^{k+1} \quad x_{\alpha_1}^1 \dots x_{\alpha_k}^k x_{\alpha_{k+1}}^{k+1} \dots x_{\alpha_n}^n}{A_{\alpha_{k+1}}^{k+1} \quad A_{\alpha_1}^1 \dots A_{\alpha_k}^k y_{\alpha_{k+1}}^{k+1} \dots y_{\alpha_n}^n} B = \S \frac{x_{\alpha_1}^1 \dots x_{\alpha_k}^k x_{\alpha_{k+1}}^{k+1} x_{\alpha_{k+2}}^{k+2} \dots x_{\alpha_n}^n}{A_{\alpha_1}^1 \dots A_{\alpha_k}^k A_{\alpha_{k+1}}^{k+1} y_{\alpha_{k+2}}^{k+2} \dots y_{\alpha_n}^n} B$

moreover have $\mathbf{S} \{ ?y \mapsto ?A \} \mathbf{S} (?\vartheta k) B = \mathbf{S} (?\vartheta (\text{Suc } k)) B$

proof —
have $\mathbf{S} \{ ?y \mapsto ?A \} \mathbf{S} (?\vartheta k) B = \mathbf{S} \{ ?y \mapsto ?A \} ++_f (?\vartheta' k) B$

proof —
have $?y \notin \text{fmdom}' (?\vartheta k)$
using $\langle ?y \in \text{lset } ys \rangle$ and $\langle \text{fmdom}' (?\vartheta k) = \text{lset } xs \rangle$ and *ys-fresh* by *blast*
moreover have $(?\vartheta' k) = \text{fmmap} (\lambda A'. \mathbf{S} \{ ?y \mapsto ?A \} A') (?\vartheta k)$
using $\langle \text{length } xs = \text{length } As \rangle$ and $\langle \text{length } ys = \text{length } xs \rangle$ by *simp*
moreover have $\forall v' \in \text{fmdom}' (?\vartheta k). \text{is-free-for} (?\vartheta k \text{ $$$ } v') v' B$

proof
fix v'
assume $v' \in \text{fmdom}' (?\vartheta k)$
with $\langle \text{fmdom}' (?\vartheta k) = \text{lset } xs \rangle$ obtain j where $v' = xs ! j$ and $j < \text{length } xs$
by (*metis in-set-conv-nth*)
obtain α where *var-type* $v' = \alpha$
by *blast*
show *is-free-for* $(?\vartheta k \text{ $$$ } v') v' B$

proof (cases $j < k$)
case *True*
with $\langle j < \text{length } xs \rangle$ and $\langle v' = xs ! j \rangle$ have $(?\vartheta k) \text{ $$$ } v' = As ! j$
using $\langle \text{distinct } xs \rangle$ and $\langle \text{length } xs = \text{length } As \rangle$ and $\langle \text{length } ys = \text{length } xs \rangle$ by *force*
moreover from $\langle \text{lset } xs = \text{fmdom}' \vartheta \rangle$ and *assms*(\mathcal{B}) have *is-free-for* $(As ! j) (xs ! j) B$
by (*metis* $\langle As = \text{map} ((\text{$$$}) \vartheta) xs \rangle \langle j < \text{length } xs \rangle \text{nth-map nth-mem}$)
ultimately show *?thesis*
using $\langle v' = xs ! j \rangle$ by (*simp only*:)

next
case *False*
with $\langle j < \text{length } xs \rangle$ and $\langle v' = xs ! j \rangle$ have $(?\vartheta k) \text{ $$$ } v' = \text{FVar } (ys ! j)$
using $\langle \text{distinct } xs \rangle$ and $\langle \text{length } xs = \text{length } As \rangle$ and $\langle \text{length } ys = \text{length } xs \rangle$ by *force*
moreover from $\langle j < \text{length } xs \rangle$ have *is-free-for* $(\text{FVar } (ys ! j)) (xs ! j) B$
using $\langle \forall j < \text{length } ys. ys ! j \notin \text{vars } \mathcal{H} \wedge ys ! j \notin \text{vars } B \rangle$ and $\langle \text{length } ys = \text{length } xs \rangle$
and *absent-var-is-free-for* by *presburger*
ultimately show *?thesis*
using $\langle v' = xs ! j \rangle$ by (*simp only*:)

qed
qed
ultimately show *?thesis*
using *substitution-consolidation* by *simp*

qed
also have $\dots = \mathbf{S} \{ ?y \mapsto ?A \} ++_f (?\vartheta (\text{Suc } k)) B$

proof —
have $?\vartheta' k = ?\vartheta (\text{Suc } k)$
proof (*intro fsubset-antisym*[*unfolded fmsubset-alt-def*] *fmpredI*)

```

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

```

```

case  $c$ 
then have  $j > k$ 
  by simp
with  $\vartheta'$ -alt-def and  $\langle \text{distinct } xs \rangle$  and  $\langle j < \text{length } xs \rangle$  have
   $\vartheta' k \ \$\$ (xs ! j) = \text{Some } (\text{drop } k (\text{map } (\lambda A'. \mathbf{S} \{?y \mapsto ?A\} A') (\text{map } FVar \text{ } ys))) ! (j - k)$ 
  using fmap-of-list-nth-split and  $\langle \text{length } xs = \text{length } As \rangle$  and  $\langle \text{length } ys = \text{length } xs \rangle$ 
  by simp
also from Suc.prems and  $c$  have  $\dots = \text{Some } (\mathbf{S} \{?y \mapsto ?A\} (FVar (ys ! j)))$ 
  using  $\langle \text{length } ys = \text{length } xs \rangle$  by simp
also from Suc-le-lessD[OF Suc.prems] and  $\langle \text{distinct } ys \rangle$  have  $\dots = \text{Some } (FVar (ys ! j))$ 
  using  $\langle j < \text{length } xs \rangle$  and  $\langle k < j \rangle$  and  $\langle \text{length } ys = \text{length } xs \rangle$ 
  by (metis nless-le nth-eq-iff-index-eq prod.exhaust-sel singleton-substitution-simps(1))
also from  $c$  and  $\langle \text{distinct } xs \rangle$  have  $\dots = \vartheta' (Suc k) \ \$\$ v'$ 
  using  $\langle xs ! j = v' \rangle$  and  $\langle \text{length } xs = \text{length } As \rangle$  and  $\langle \text{length } ys = \text{length } xs \rangle$  by force
finally show  $?thesis$ 
  using  $\langle \vartheta' k \ \$\$ v' = \text{Some } A' \rangle$  and  $\langle xs ! j = v' \rangle$  by force
qed
}
note  $\vartheta$ -k-in-Sub-k = this
{
  fix  $v'$  and  $A'$ 
  assume  $\vartheta' (Suc k) \ \$\$ v' = \text{Some } A'$ 
  then have  $v' \in \text{fmdom}' (\vartheta' (Suc k))$ 
    by (intro fmdom'I)
  then obtain  $j$  where  $j < \text{length } xs$  and  $xs ! j = v'$ 
    using  $\langle \text{fmdom}' (\vartheta' (Suc k)) = \text{lset } xs \rangle$  by (metis in-set-conv-nth)
  then consider (a)  $j < k$  | (b)  $j = k$  | (c)  $j \in \{k..< \text{length } xs\}$ 
    by fastforce
  with  $\langle j < \text{length } xs \rangle$  and  $\langle xs ! j = v' \rangle$  and  $\vartheta$ -k-in-Sub-k show  $\vartheta' k \ \$\$ v' = \text{Some } A'$ 
    using  $\langle \bigwedge k'. \text{fmdom}' (\vartheta' k') = \text{lset } xs \rangle$  and  $\langle \vartheta' (Suc k) \ \$\$ v' = \text{Some } A' \rangle$ 
    by (metis (mono-tags, lifting) fmllookup-dom'-iff nth-mem)+
}
qed
then show  $?thesis$ 
  by presburger
qed
also have  $\dots = \mathbf{S} (\vartheta' (Suc k)) B$ 
proof –
  have  $\vartheta' (Suc k) \ \$\$ ?y = \text{None}$ 
    using  $\langle ?y \in \text{lset } ys \rangle$   $\langle \bigwedge k'. \text{fmdom}' (\vartheta' k') = \text{lset } xs \rangle$  and ys-fresh by blast
  moreover from Suc-le-lessD[OF Suc.prems] have  $?y \notin \text{vars } B$ 
    using  $\langle \forall j < \text{length } ys. ys ! j \notin \text{vars } \mathcal{H} \wedge ys ! j \notin \text{vars } B \rangle$  and  $\langle \text{length } ys = \text{length } xs \rangle$ 
    by auto
  ultimately show  $?thesis$ 
    by (rule substitution-absorption)
qed
finally show  $?thesis$  .
qed

```

$$\vdash \mathcal{H} \vdash \mathbb{S} \begin{matrix} x_{\alpha_1}^1 & \dots & x_{\alpha_k}^k & x_{\alpha_{k+1}}^{k+1} & x_{\alpha_{k+2}}^{k+2} & \dots & x_{\alpha_n}^n \\ A_{\alpha_1}^1 & \dots & A_{\alpha_k}^k & A_{\alpha_{k+1}}^{k+1} & A_{\alpha_{k+2}}^{k+2} & \dots & y_{\alpha_n}^n \end{matrix} B$$

ultimately show *?case*
by (*simp only:*)
qed
— $\mathcal{H} \vdash \S_{A_{\alpha_1}^1 \dots A_{\alpha_n}^n}^{x_{\alpha_1}^1 \dots x_{\alpha_n}^n} B$
then have $\mathcal{H} \vdash \mathbf{S} (fmap\text{-of-list } (zip\ xs\ As))\ B$
using $\langle length\ xs = length\ As \rangle$ **and** $\langle length\ ys = length\ xs \rangle$ **by force**
moreover have $fmap\text{-of-list } (zip\ xs\ As) = \vartheta$
proof (*intro fsubset-antisym[unfolded fmsubset-alt-def] fmpredI*)
fix v **and** A
assume $fmap\text{-of-list } (zip\ xs\ As)\ \mathbf{\$ \$} v = Some\ A$
with $\langle lset\ xs = fmdom'\ \vartheta \rangle$ **have** $v \in fmdom'\ \vartheta$
by (*fast dest: fmap-of-list-SomeD set-zip-leftD*)
with $\langle fmap\text{-of-list } (zip\ xs\ As)\ \mathbf{\$ \$} v = Some\ A \rangle$ **and** $\langle As = map\ ((\mathbf{\$ \$})\ \vartheta)\ xs \rangle$ **show** $\vartheta\ \mathbf{\$ \$} v = Some\ A$
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 $\vartheta\ \mathbf{\$ \$} v = Some\ A$
with $\langle As = map\ ((\mathbf{\$ \$})\ \vartheta)\ xs \rangle$ **show** $fmap\text{-of-list } (zip\ xs\ As)\ \mathbf{\$ \$} v = Some\ A$
using $\langle lset\ xs = fmdom'\ \vartheta \rangle$ **by** (*simp add: fmap-of-list.rep-eq fmdom'I map-of-zip-map*)
qed
ultimately show *?thesis*
by (*simp only:*)
qed
end
lemmas $Sub = prop\text{-}5221$

6.23 Proposition 5222 (Rule of Cases)

lemma *forall- α -conversion:*

assumes $A \in wffs_o$
and $(z, \beta) \notin free\text{-vars } A$
and *is-free-for* $(z\beta)\ (x, \beta)\ A$
shows $\vdash \forall x\beta. A =_o \forall z\beta. \mathbf{S} \{(x, \beta) \mapsto z\beta\} A$
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** $\vdash (\lambda x\beta. A) =_{\beta \rightarrow o} (\lambda z\beta. \mathbf{S} \{(x, \beta) \mapsto z\beta\} A)$
by (*rule prop-5206*)
ultimately show *?thesis*
unfolding *forall-def* **and** *PI-def* **by** (*rule rule-R [where $p = [\rangle, \rangle]$] force+*)
qed

proposition *prop-5222*:

assumes $\mathcal{H} \vdash \mathbf{S} \{(x, o) \rightsquigarrow T_o\} A$ and $\mathcal{H} \vdash \mathbf{S} \{(x, o) \rightsquigarrow F_o\} A$
and $A \in \text{wffs}_o$
shows $\mathcal{H} \vdash A$

proof –

from *assms(1)* have *is-hyps* \mathcal{H}
by (*blast elim: is-derivable-from-hyps.cases*)
have §1: $\mathcal{H} \vdash T_o =_o (\lambda x_o. A) \cdot T_o$

proof –

have $\vdash (\lambda x_o. A) \cdot T_o =_o \mathbf{S} \{(x, o) \rightsquigarrow T_o\} A$
using *prop-5207[OF true-wff assms(3) closed-is-free-for]* by *simp*
from *this* and *assms(1)* have $\mathcal{H} \vdash (\lambda x_o. A) \cdot T_o$
using *rule-RR[OF disjI2, where p = []]* by *fastforce*
moreover have $(\lambda x_o. A) \cdot T_o \in \text{wffs}_o$
by (*fact hyp-derivable-form-is-wffso[OF $\mathcal{H} \vdash (\lambda x_o. A) \cdot T_o$]*)
ultimately show *?thesis*
using *rule-T(1)* by *blast*

qed

moreover have §2: $\mathcal{H} \vdash T_o =_o (\lambda x_o. A) \cdot F_o$

proof –

have $\vdash (\lambda x_o. A) \cdot F_o =_o \mathbf{S} \{(x, o) \rightsquigarrow F_o\} A$
using *prop-5207[OF false-wff assms(3) closed-is-free-for]* by *simp*
from *this* and *assms(2)* have $\mathcal{H} \vdash (\lambda x_o. A) \cdot F_o$
using *rule-RR[OF disjI2, where p = []]* by *fastforce*
moreover have $(\lambda x_o. A) \cdot F_o \in \text{wffs}_o$
by (*fact hyp-derivable-form-is-wffso[OF $\mathcal{H} \vdash (\lambda x_o. A) \cdot F_o$]*)
ultimately show *?thesis*
using *rule-T(1)* by *blast*

qed

moreover from *prop-5212* and *<is-hyps math>* have §3: $\mathcal{H} \vdash T_o \wedge^Q T_o$

by (*rule derivability-implies-hyp-derivability*)

ultimately have $\mathcal{H} \vdash (\lambda x_o. A) \cdot T_o \wedge^Q (\lambda x_o. A) \cdot F_o$

proof –

from §3 and §1 have $\mathcal{H} \vdash (\lambda x_o. A) \cdot T_o \wedge^Q T_o$
by (*rule rule-R'[where p = [«,»]]*) (*force+*, *fastforce dest: subforms-from-app*)
from *this* and §2 show *?thesis*
by (*rule rule-R'[where p = [»]]*) (*force+*, *fastforce dest: subforms-from-app*)

qed

moreover have $\vdash (\lambda x_o. A) \cdot T_o \wedge^Q (\lambda x_o. A) \cdot F_o =_o \forall x_o. A$

proof –

have $\mathfrak{g}_{o \rightarrow o} \cdot \mathfrak{r}_o \in \text{wffs}_o$
by *blast*
have $\vdash \mathfrak{g}_{o \rightarrow o} \cdot T_o \wedge^Q \mathfrak{g}_{o \rightarrow o} \cdot F_o =_o \forall \mathfrak{r}_o. \mathfrak{g}_{o \rightarrow o} \cdot \mathfrak{r}_o$
using *axiom-1[unfolded equivalence-def]* by (*rule axiom-is-derivable-from-no-hyps*)
— By α -conversion
then have $\vdash \mathfrak{g}_{o \rightarrow o} \cdot T_o \wedge^Q \mathfrak{g}_{o \rightarrow o} \cdot F_o =_o \forall x_o. \mathfrak{g}_{o \rightarrow o} \cdot x_o$ (*is* $\vdash ?B =_o ?C$)
proof –
have $\vdash \forall \mathfrak{r}_o. \mathfrak{g}_{o \rightarrow o} \cdot \mathfrak{r}_o =_o \forall x_o. \mathfrak{g}_{o \rightarrow o} \cdot x_o$

```

proof (cases x = r)
  case True
    from ⟨go→o • ro ∈ wffso⟩ have ⊢ ∀ ro. go→o • ro =o ∀ ro. go→o • ro
      by (fact prop-5200[OF forall-wff])
    with True show ?thesis
      using identity-singleton-substitution-neutrality by simp
  next
    case False
    from ⟨go→o • ro ∈ wffso⟩
    have ⊢ ∀ ro. go→o • ro =o ∀ xo. S {(r, o) ↦ xo} (go→o • ro)
      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 ⟨⊢ go→o • To ∧Q go→o • Fo =o ∀ ro. go→o • ro⟩ show ?thesis
    using Equality-Rules(β) by blast
qed
— By Sub
then have *: ⊢ (λxo. A) • To ∧Q (λxo. A) • Fo =o ∀ xo. (λxo. A) • xo
proof —
  let ?∅ = {(g, o→o) ↦ λxo. A}
  from assms(β) have is-substitution ?∅
    by auto
  moreover have
    ∀ v ∈ fmdom' ?∅.
      var-name v ∉ free-var-names ({::form set}) ∧ is-free-for (?∅ $$$! v) v (?B =o ?C)
    by (code-simp, (unfold atomize-conj[symmetric])?, simp)+ blast
  moreover have ?∅ ≠ {$$$}
    by simp
  ultimately have ⊢ S ?∅ (?B =o ?C)
    by (rule Sub [OF ⟨⊢ ?B =o ?C⟩])
  then show ?thesis
    by simp
qed
— By λ-conversion
then show ?thesis
proof —
  have ⊢ (λxo. A) • xo =o A
    using prop-5208[where vs = [(x, o)]] and assms(β) by simp
  from * and this show ?thesis
    by (rule rule-R[where p = [»,»,«]]) force+
qed
qed
ultimately have ℋ ⊢ ∀ xo. 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 $\langle \mathcal{H} \vdash \forall x_o. A \rangle$ **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 $\vdash (T_o \supset^{\mathcal{Q}} \eta_o) =_o \eta_o$

proof –

have $\vdash (T_o \supset^{\mathcal{Q}} \eta_o) =_o (T_o =_o (T_o \wedge^{\mathcal{Q}} \eta_o))$

proof –

let $?A = (\lambda x_o. \lambda \eta_o. (x_o \equiv^{\mathcal{Q}} x_o \wedge^{\mathcal{Q}} \eta_o)) \cdot T_o \cdot \eta_o$

have $?A \in \text{wffs}_o$

by *force*

then have $\vdash ?A =_o ?A$

by (fact *prop-5200*)

then have $\vdash (T_o \supset^{\mathcal{Q}} \eta_o) =_o ?A$

unfolding *imp-fun-def* **and** *imp-op-def* .

moreover

have $\vdash (\lambda x_o. \lambda \eta_o. (x_o \equiv^{\mathcal{Q}} x_o \wedge^{\mathcal{Q}} \eta_o)) \cdot T_o =_{o \rightarrow o} \lambda \eta_o. (T_o \equiv^{\mathcal{Q}} T_o \wedge^{\mathcal{Q}} \eta_o)$

proof –

have $\lambda \eta_o. (x_o \equiv^{\mathcal{Q}} x_o \wedge^{\mathcal{Q}} \eta_o) \in \text{wffs}_{o \rightarrow o}$

by *auto*

moreover

have *is-free-for* T_o (x, o) $(\lambda \eta_o. (x_o \equiv^{\mathcal{Q}} x_o \wedge^{\mathcal{Q}} \eta_o))$

by *fastforce*

moreover

have $S \{(x, o) \mapsto T_o\} (\lambda \eta_o. (x_o \equiv^{\mathcal{Q}} x_o \wedge^{\mathcal{Q}} \eta_o)) = (\lambda \eta_o. (T_o \equiv^{\mathcal{Q}} T_o \wedge^{\mathcal{Q}} \eta_o))$

by *simp*

ultimately show *?thesis*

using *prop-5207*[*OF true-wff*] **by** *metis*

qed

ultimately have $*$: $\vdash (T_o \supset^{\mathcal{Q}} \eta_o) =_o (\lambda \eta_o. (T_o \equiv^{\mathcal{Q}} T_o \wedge^{\mathcal{Q}} \eta_o)) \cdot \eta_o$

by (rule *rule-R* [where $p = [\rangle, \langle]$]) *force+*

have $T_o \equiv^{\mathcal{Q}} T_o \wedge^{\mathcal{Q}} \eta_o \in \text{wffs}_o$

by *auto*

then have $\vdash (\lambda \eta_o. (T_o \equiv^{\mathcal{Q}} T_o \wedge^{\mathcal{Q}} \eta_o)) \cdot \eta_o =_o (T_o \equiv^{\mathcal{Q}} T_o \wedge^{\mathcal{Q}} \eta_o)$

using *prop-5208*[where $vs = [(\eta, o)]$] **by** *simp*

from $*$ **and this show** *?thesis*

by (rule *rule-R*[where $p = [\rangle]$]) *force+*

qed

with *prop-5218* **have** $\vdash (T_o \supset^{\mathcal{Q}} \eta_o) =_o (T_o \wedge^{\mathcal{Q}} \eta_o)$

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 $A \in \text{wffs}_o$

shows $\vdash (T_o \supset^Q A) =_o A$

proof –

have $T_o \supset^Q \eta_o \in \text{wffs}_o$ and *is-free-for* A (η, o) $((T_o \supset^Q \eta_o) =_o \eta_o)$

by (*blast*, *intro is-free-for-in-equality is-free-for-in-imp is-free-for-in-true is-free-for-in-var*)

from *this(2)* have $\vdash \mathbf{S} \{(\eta, o) \mapsto A\} ((T_o \supset^Q \eta_o) =_o \eta_o)$

by (*rule prop-5209[OF assms $\langle T_o \supset^Q \eta_o \in \text{wffs}_o \rangle$ 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 $\mathcal{H} \vdash A \supset^Q B$

shows $\mathcal{H} \vdash B$

proof –

have $\mathcal{H} \vdash A \supset^Q B$

by *fact*

moreover from *assms(1)* have $A \in \text{wffs}_o$

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 $\mathcal{H} \vdash T_o \supset^Q B$

by (*rule rule-R'[where $p = [\langle \langle, \rangle \rangle]$]*) (*force+*, *fastforce dest: subforms-from-app*)

have $\vdash (T_o \supset^Q B) =_o B$

proof –

let $? \vartheta = \{(\eta, o) \mapsto B\}$

have $B \in \text{wffs}_o$

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

$\forall v \in \text{fmdom}' ? \vartheta.$

$\text{var-name } v \notin \text{free-var-names } (\{ \} :: \text{form set}) \wedge$

$\text{is-free-for } (? \vartheta \ \$\$! v) v ((T_o \supset^Q \eta_o) =_o \eta_o)$

by (*code-simp*, (*unfold atomize-conj[symmetric]*)?, *simp*) + *blast*

moreover have $? \vartheta \neq \{ \ \$\$ \}$

by *simp*

ultimately have $\vdash \mathbf{S} ? \vartheta ((T_o \supset^Q \eta_o) =_o \eta_o)$

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 $\mathcal{H} \vdash hs \supset^{\mathcal{Q}_*} B$ 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 ⟨ $\mathcal{H} \vdash (hs \ @ \ [H']) \supset^{\mathcal{Q}_*} B$ ⟩ have $\mathcal{H} \vdash hs \supset^{\mathcal{Q}_*} (H' \supset^{\mathcal{Q}} B)$

by *simp*

moreover from ⟨ $\forall H \in lset\ (hs \ @ \ [H']) . \mathcal{H} \vdash H$ ⟩ have $\forall H \in lset\ hs. \mathcal{H} \vdash H$

by *simp*

ultimately show ?*thesis*

by (elim *snoc.IH*)

qed

ultimately show ?*case*

by (rule *MP*)

qed

6.26 Proposition 5225

proposition prop-5225:

shows $\vdash \prod \alpha \cdot f_{\alpha \rightarrow o} \supset^{\mathcal{Q}} f_{\alpha \rightarrow o} \cdot r_{\alpha}$

proof –

have $f_{\alpha \rightarrow o} \cdot r_{\alpha} \in wffs_o$

by *blast*

have §1:

⊢

$$\prod \alpha \cdot f_{\alpha \rightarrow o} \supset^{\mathcal{Q}} (((\lambda f_{\alpha \rightarrow o} . f_{\alpha \rightarrow o} \cdot r_{\alpha}) \cdot (\lambda r_{\alpha} . T_o))$$

$$=_o$$

$$((\lambda f_{\alpha \rightarrow o} . f_{\alpha \rightarrow o} \cdot r_{\alpha}) \cdot f_{\alpha \rightarrow o}))$$

proof –

let ? $\vartheta = \{(\mathfrak{h}, (\alpha \rightarrow o) \rightarrow o) \mapsto \lambda f_{\alpha \rightarrow o} . f_{\alpha \rightarrow o} \cdot r_{\alpha}, (\mathfrak{r}, \alpha \rightarrow o) \mapsto \lambda r_{\alpha} . T_o, (\mathfrak{h}, \alpha \rightarrow o) \mapsto f_{\alpha \rightarrow o}\}$

and ?*A* = $(r_{\alpha \rightarrow o} =_{\alpha \rightarrow o} \eta_{\alpha \rightarrow o}) \supset^{\mathcal{Q}} (\mathfrak{h}_{(\alpha \rightarrow o) \rightarrow o} \cdot r_{\alpha \rightarrow o} \equiv^{\mathcal{Q}} \mathfrak{h}_{(\alpha \rightarrow o) \rightarrow o} \cdot \eta_{\alpha \rightarrow o})$

have ⊢ ?*A*

by (fact *axiom-is-derivable-from-no-hyps*[*OF axiom-2*])

moreover have $\lambda f_{\alpha \rightarrow o} . f_{\alpha \rightarrow o} \cdot r_{\alpha} \in wffs_{(\alpha \rightarrow o) \rightarrow o}$ and $\lambda r_{\alpha} . T_o \in wffs_{\alpha \rightarrow o}$

and $f_{\alpha \rightarrow o} \in wffs_{\alpha \rightarrow o}$

by *blast+*

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

6.27 Proposition 5226

proposition *prop-5226*:

assumes $A \in \text{wffs}_{\alpha}$ **and** $B \in \text{wffs}_o$

and *is-free-for* $A \ (x, \alpha) \ B$

shows $\vdash \forall x_{\alpha}. B \supset^{\mathcal{Q}} \mathbf{S} \{(x, \alpha) \mapsto A\} \ B$

proof –

have $\vdash \prod \alpha \cdot (\lambda x_{\alpha}. B) \supset^{\mathcal{Q}} (\lambda x_{\alpha}. B) \cdot A$

proof –

let $? \vartheta = \{(\text{f}, \alpha \rightarrow o) \mapsto \lambda x_{\alpha}. B, (\text{r}, \alpha) \mapsto A\}$

have $\vdash \prod \alpha \cdot \text{f}_{\alpha \rightarrow o} \supset^{\mathcal{Q}} \text{f}_{\alpha \rightarrow o} \cdot \text{r}_{\alpha}$ (*is* $\vdash ?C$)

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

6.28 Proposition 5227

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

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

6.29 Proposition 5228

proposition *prop-5228*:

shows $\vdash (T_o \supset^{\mathcal{Q}} T_o) =_o T_o$

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 $A \in \text{wffs}_o$

shows $\vdash F_o \wedge^{\mathcal{Q}} A \equiv^{\mathcal{Q}} F_o$

proof –

have $\vdash (\lambda \mathfrak{r}_o. \lambda \eta_o. (\mathfrak{r}_o \equiv^{\mathcal{Q}} \mathfrak{r}_o \wedge^{\mathcal{Q}} \eta_o)) \cdot F_o \cdot A$

by (*intro generalized-prop-5227*[*OF assms, unfolded imp-op-def imp-fun-def*])

moreover have

\vdash

$(\lambda \mathfrak{r}_o. \lambda \eta_o. (\mathfrak{r}_o \equiv^{\mathcal{Q}} \mathfrak{r}_o \wedge^{\mathcal{Q}} \eta_o)) \cdot F_o$

$=_{o \rightarrow o}$

$\lambda \eta_o. (F_o \equiv^{\mathcal{Q}} F_o \wedge^{\mathcal{Q}} \eta_o)$

(is $\vdash (\lambda ?x ?\beta. ?A) \cdot ?B =_{? \gamma} ?C$)

proof –

have $?B \in \text{wffs}_{? \beta}$ and $?A \in \text{wffs}_{? \gamma}$ and *is-free-for* $?B$ $(?x, ?\beta)$ $?A$

by *auto*

then have $\vdash (\lambda ?x ?\beta. ?A) \cdot ?B =_{? \gamma} \mathbf{S} \{(?x, ?\beta) \mapsto ?B\} ?A$

by (*rule prop-5207*)

moreover have $\mathbf{S} \{(?x, ?\beta) \mapsto ?B\} ?A = ?C$

by *simp*

ultimately show *?thesis*

by (*simp only*:)

qed

ultimately have $\vdash (\lambda \eta_o. (F_o \equiv^{\mathcal{Q}} F_o \wedge^{\mathcal{Q}} \eta_o)) \cdot A$

by (*rule rule-R*[*where* $p = [\llbracket \cdot \rrbracket]$] *auto*)

moreover have

\vdash

$(\lambda \eta_o. (F_o \equiv^{\mathcal{Q}} F_o \wedge^{\mathcal{Q}} \eta_o)) \cdot A$

$=_o$

$(F_o \equiv^{\mathcal{Q}} F_o \wedge^{\mathcal{Q}} A)$

(is $\vdash (\lambda ?x ?\beta. ?A) \cdot ?B =_o ?C$)

proof –

have $?B \in \text{wffs}_{\beta}$ **and** $?A \in \text{wffs}_o$
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** $\vdash (\lambda ?x ?\beta. ?A) \cdot ?B =_o \mathbf{S} \{(?x, ?\beta) \mapsto ?B\} ?A$
by (*rule prop-5207*)
moreover
have $\mathbf{S} \{(?x, ?\beta) \mapsto ?B\} ?A = ?C$
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 $\S 1: \vdash (F_o \equiv^{\mathcal{Q}} T_o) \supset^{\mathcal{Q}} ((\lambda \mathbf{r}_o. (\mathbf{r}_o \equiv^{\mathcal{Q}} F_o)) \cdot F_o \equiv^{\mathcal{Q}} (\lambda \mathbf{r}_o. (\mathbf{r}_o \equiv^{\mathcal{Q}} F_o)) \cdot T_o)$

proof –

let $?\theta = \{(\mathfrak{h}, o \rightarrow o) \mapsto \lambda \mathbf{r}_o. (\mathbf{r}_o \equiv^{\mathcal{Q}} F_o), (\mathbf{x}, o) \mapsto F_o, (\mathfrak{h}, o) \mapsto T_o\}$
and $?A = (\mathbf{r}_o =_o \mathfrak{h}_o) \supset^{\mathcal{Q}} (\mathfrak{h}_{o \rightarrow o} \cdot \mathbf{r}_o \equiv^{\mathcal{Q}} \mathfrak{h}_{o \rightarrow o} \cdot \mathfrak{h}_o)$

have $\vdash ?A$
by (*fact axiom-is-derivable-from-no-hyps*[*OF axiom-2*])
moreover have *is-substitution* $?\vartheta$
by *auto*
moreover have
 $\forall v \in \text{fmdom}' \ ?\vartheta. \text{var-name } v \notin \text{free-var-names } (\{\}::\text{form set}) \wedge \text{is-free-for } (?\vartheta \ \$\$! \ v) \ v \ ?A$
by (*code-simp, unfold atomize-conj*[*symmetric*], *simp, force*)**+** *blast*
moreover have $? \vartheta \neq \{\ \$\ \$\}$
by *simp*
ultimately have $\vdash \mathbf{S} \ ?\vartheta \ ?A$
by (*rule Sub*)
then show *?thesis*
by *simp*
qed
then have $\S 2: \vdash (F_o \equiv^{\mathcal{Q}} T_o) \supset^{\mathcal{Q}} ((F_o \equiv^{\mathcal{Q}} F_o) \equiv^{\mathcal{Q}} (T_o \equiv^{\mathcal{Q}} F_o))$ (**is** $\vdash ?A2$)
proof –
have *is-free-for* A (\mathfrak{x}_o, o) $(\mathfrak{x}_o \equiv^{\mathcal{Q}} F_o)$ **for** A
by *code-simp blast*
have β -*reduction*: $\vdash (\lambda \mathfrak{x}_o. (\mathfrak{x}_o \equiv^{\mathcal{Q}} F_o)) \cdot A =_o (A \equiv^{\mathcal{Q}} F_o)$ **if** $A \in \text{wffs}_o$ **for** A
using
prop-5207
 $[$
 $\text{OF that equivalence-wff}[\text{OF wffs-of-type-intros}(1) \text{ false-wff}]$
 $\langle \text{is-free-for } A (\mathfrak{x}_o, o) (\mathfrak{x}_o \equiv^{\mathcal{Q}} F_o) \rangle$
 $]$
by *simp*
from $\S 1$ **and** β -*reduction*[*OF false-wff*] **have**
 $\vdash (F_o =_o T_o) \supset^{\mathcal{Q}} ((F_o \equiv^{\mathcal{Q}} F_o) \equiv^{\mathcal{Q}} (\lambda \mathfrak{x}_o. (\mathfrak{x}_o \equiv^{\mathcal{Q}} F_o)) \cdot T_o)$
by (*rule rule-R*[**where** $p = [\rangle, \langle, \rangle]$]) *force***+**
from this and β -*reduction*[*OF true-wff*] **show** *?thesis*
by (*rule rule-R*[**where** $p = [\rangle, \rangle]$]) *force***+**
qed
then have $\S 3: \vdash (F_o \equiv^{\mathcal{Q}} T_o) \supset^{\mathcal{Q}} F_o$
proof –
note $r1 = \text{rule-RR}[\text{OF disjI1}]$ **and** $r2 = \text{rule-RR}[\text{OF disjI2}]$
have $\S 3_1: \vdash (F_o \equiv^{\mathcal{Q}} T_o) \supset^{\mathcal{Q}} ((F_o \equiv^{\mathcal{Q}} F_o) \equiv^{\mathcal{Q}} F_o)$ (**is** $\vdash ?A3_1$)
by (*rule r1*[*OF prop-5218*[*OF false-wff*], **where** $p = [\rangle, \rangle]$ **and** $C = ?A2$]) (*use* $\S 2$ **in** $\langle \text{force} \rangle$)
have $\S 3_2: \vdash (F_o \equiv^{\mathcal{Q}} T_o) \supset^{\mathcal{Q}} (T_o \equiv^{\mathcal{Q}} F_o)$ (**is** $\vdash ?A3_2$)
by (*rule r2*[*OF prop-5210*[*OF false-wff*], **where** $p = [\rangle, \langle, \rangle]$ **and** $C = ?A3_1$]) (*use* $\S 3_1$ **in** $\langle \text{force} \rangle$)
show *?thesis*
by (*rule r1*[*OF prop-5218*[*OF false-wff*], **where** $p = [\rangle]$ **and** $C = ?A3_2$]) (*use* $\S 3_2$ **in** $\langle \text{force} \rangle$)
qed
then have $\vdash (F_o \equiv^{\mathcal{Q}} T_o) \equiv^{\mathcal{Q}} ((F_o \equiv^{\mathcal{Q}} T_o) \wedge^{\mathcal{Q}} F_o)$
proof –
have
 \vdash
 $(\lambda \mathfrak{x}_o. \lambda \eta_o. (\mathfrak{x}_o \equiv^{\mathcal{Q}} \mathfrak{x}_o \wedge^{\mathcal{Q}} \eta_o)) \cdot (F_o \equiv^{\mathcal{Q}} T_o)$
 $\stackrel{=_{o \rightarrow o}}{=} \{(\mathfrak{x}_o, o) \mapsto F_o \equiv^{\mathcal{Q}} T_o\} (\lambda \eta_o. (\mathfrak{x}_o \equiv^{\mathcal{Q}} \mathfrak{x}_o \wedge^{\mathcal{Q}} \eta_o))$

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

6.32 Proposition 5231

proposition prop-5231:
 shows $\vdash \sim^Q T_o =_o F_o$
 and $\vdash \sim^Q F_o =_o T_o$
 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 $A \in \text{wffs}_o$ and $B \in \text{wffs}_o$

shows $\vdash A \vee^o B =_o \sim^o (\sim^o A \wedge^o \sim^o B)$

proof –

let $?C = (\lambda \mathfrak{r}_o. \lambda \eta_o. \sim^o (\sim^o \mathfrak{r}_o \wedge^o \sim^o \eta_o)) \cdot A \cdot B$

from *assms* have $?C \in \text{wffs}_o$

by *blast*

have $(\sim^o (\sim^o \mathfrak{r}_o \wedge^o \sim^o \eta_o)) \in \text{wffs}_o$

by *auto*

moreover obtain z where $(z, o) \notin \{(\mathfrak{r}, o), (\eta, o)\}$ and $(z, o) \notin \text{free-vars } A$

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 $(z, o) \notin \text{free-vars } (\sim^o (\sim^o \mathfrak{r}_o \wedge^o \sim^o \eta_o))$

by *simp*

moreover have *is-free-for* $(z_o) (\eta, o) (\sim^o (\sim^o \mathfrak{r}_o \wedge^o \sim^o \eta_o))$

by (*intro is-free-for-in-conj is-free-for-in-neg is-free-for-in-var*)

ultimately have

$\vdash (\lambda \eta_o. \sim^o (\sim^o \mathfrak{r}_o \wedge^o \sim^o \eta_o)) =_{o \rightarrow o} (\lambda z_o. \mathbf{S} \{(\eta, o) \mapsto z_o\} \sim^o (\sim^o \mathfrak{r}_o \wedge^o \sim^o \eta_o))$

by (*rule α*)

then have $\vdash (\lambda \eta_o. \sim^o (\sim^o \mathfrak{r}_o \wedge^o \sim^o \eta_o)) =_{o \rightarrow o} (\lambda z_o. \sim^o (\sim^o \mathfrak{r}_o \wedge^o \sim^o z_o))$

by *simp*

from *prop-5200*[*OF* $\langle ?C \in \text{wffs}_o \rangle$] and *this* have

\vdash

$(\lambda \mathfrak{r}_o. \lambda z_o. \sim^o (\sim^o \mathfrak{r}_o \wedge^o \sim^o z_o)) \cdot A \cdot B$

$=_o$

$(\lambda \mathfrak{r}_o. \lambda \eta_o. \sim^o (\sim^o \mathfrak{r}_o \wedge^o \sim^o \eta_o)) \cdot A \cdot B$

by (*rule rule-R*[*where* $p = [\langle \langle, \rangle, \langle, \rangle, \langle, \rangle]$] *force+*)

moreover have $\lambda z_o. \sim^o (\sim^o \mathfrak{r}_o \wedge^o \sim^o z_o) \in \text{wffs}_{o \rightarrow o}$

by *blast*

have

\vdash

$(\lambda \mathfrak{r}_o. \lambda z_o. \sim^o (\sim^o \mathfrak{r}_o \wedge^o \sim^o z_o)) \cdot A$

$=_{o \rightarrow o}$

$\mathbf{S} \{(\mathfrak{r}, o) \mapsto A\} (\lambda z_o. \sim^o (\sim^o \mathfrak{r}_o \wedge^o \sim^o z_o))$

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: $\langle (z, o) \notin \text{free-vars } A \rangle$*)

)

then have $\vdash (\lambda \mathfrak{r}_o. \lambda z_o. \sim^o (\sim^o \mathfrak{r}_o \wedge^o \sim^o z_o)) \cdot A =_{o \rightarrow o} (\lambda z_o. \sim^o (\sim^o A \wedge^o \sim^o z_o))$

using $\langle (z, o) \notin \text{free-vars } (\sim^o (\sim^o \mathfrak{r}_o \wedge^o \sim^o \eta_o)) \rangle$ by *simp*

ultimately have

$\vdash (\lambda z_o. \sim^o (\sim^o A \wedge^o \sim^o z_o)) \cdot B =_o (\lambda \mathfrak{r}_o. \lambda \eta_o. \sim^o (\sim^o \mathfrak{r}_o \wedge^o \sim^o \eta_o)) \cdot A \cdot B$

by (*rule rule-R*[*where* $p = [\langle \langle, \rangle, \langle, \rangle]$] *force+*)

moreover have $\vdash (\lambda z_o. \sim^o (\sim^o A \wedge^o \sim^o z_o)) \cdot B =_o \mathbf{S} \{(z, o) \mapsto B\} (\sim^o (\sim^o A \wedge^o \sim^o z_o))$

by

(*rule prop-5207*)

(
fact, blast intro: assms(1), intro is-free-for-in-neg is-free-for-in-conj,
use $\langle(z, o) \notin \text{free-vars } A\rangle$ is-free-at-in-free-vars in $\langle\text{fastforce+}\rangle$
)
moreover have $\mathbf{S} \{(z, o) \mapsto B\} (\sim^{\mathcal{Q}} (\sim^{\mathcal{Q}} A \wedge^{\mathcal{Q}} \sim^{\mathcal{Q}} z_o)) = \sim^{\mathcal{Q}} (\sim^{\mathcal{Q}} A \wedge^{\mathcal{Q}} \sim^{\mathcal{Q}} B)$
using *free-var-singleton-substitution-neutrality[OF $\langle(z, o) \notin \text{free-vars } A\rangle$]* **by** *simp*
ultimately have $\vdash (\lambda \mathfrak{r}_o. \lambda \eta_o. \sim^{\mathcal{Q}} (\sim^{\mathcal{Q}} \mathfrak{r}_o \wedge^{\mathcal{Q}} \sim^{\mathcal{Q}} \eta_o)) \cdot A \cdot B =_o \sim^{\mathcal{Q}} (\sim^{\mathcal{Q}} A \wedge^{\mathcal{Q}} \sim^{\mathcal{Q}} B)$
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** $\langle\vdash ?A1\rangle$)
by (*rule rule-RR[OF disjI2, where $p = [\langle\langle, \rangle, \rangle, \langle, \rangle]$ and $C = ?D$]*) (*use assms(1) in $\langle\text{force+}\rangle$*)
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 $p = [\langle\langle, \rangle, \rangle, \rangle]$ and $C = ?A1$]*) (*use $\langle\vdash ?A1\rangle$ in $\langle\text{force+}\rangle$*)
moreover from *assms(2,3)* **have** $A' \in \text{wffs}_o$ **and** $B' \in \text{wffs}_o$
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** $\langle\vdash ?A\rangle$) .
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 $p = [\langle\langle, \rangle, \rangle]$ and $C = ?A$]*) (*use $\langle\vdash ?A\rangle$ in $\langle\text{force+}\rangle$*)
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 $p = [\langle\langle, \rangle, \rangle]$ and $C = ?A$]*) (*use $\langle\vdash ?A\rangle$ in $\langle\text{force+}\rangle$*)
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 $p = [\langle\langle, \rangle, \rangle]$ and $C = ?A$]*) (*use $\langle\vdash ?A\rangle$ in $\langle\text{force+}\rangle$*)
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** $\langle \vdash ?A \rangle$) .
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 $p = [\langle \langle \cdot, \cdot \rangle, \cdot \rangle]$ and $C = ?A$]*) (*use* $\langle \vdash ?A \rangle$ **in** $\langle \text{force+} \rangle$)
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 $A \in \text{pwffs}$ **and** *free-vars* $A = \{\}$

shows $(\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi A = \mathbf{T}) \longrightarrow \vdash A =_o T_o$ (**is** $?A_T \longrightarrow -$)

and $(\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi A = \mathbf{F}) \longrightarrow \vdash A =_o F_o$ (**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** $\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi T_o = \mathbf{T}$

using $\mathcal{V}_B\text{-}T$ **by** *blast*

then **have** $\neg (\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi T_o = \mathbf{F})$

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** $\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi F_o = \mathbf{F}$

using $\mathcal{V}_B\text{-}F$ **by** *blast*

then **have** $\neg (\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi F_o = \mathbf{T})$

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** $\sim^{\mathcal{Q}} B \in \text{pwffs}$ **and** *free-vars* $B = \{\}$

using *neg-pwff.premis* **by** (*force, auto elim: free-vars-form.elims*)

consider

(a) $\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi B = \mathbf{T}$

| (b) $\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi B = \mathbf{F}$

using *closed-pwff-denotation-uniqueness*[*OF* *neg-pwff.hyps* $\langle \text{free-vars } B = \{\} \rangle$]
and *neg-pwff.hyps*[*THEN* $\mathcal{V}_B\text{-graph-denotation-is-truth-value}$ [*OF* $\mathcal{V}_B\text{-graph-}\mathcal{V}_B$]]
by (*auto dest: tv-cases*) *metis*
then show *?case*
proof cases
case a
with $\langle \text{free-vars } B = \{\} \rangle$ **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* $p = [\langle \langle, \rangle, \rangle]$]) *force+*
moreover from a **have** $\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi (\sim^{\mathcal{Q}} B) = \mathbf{F}$
using $\mathcal{V}_B\text{-neg}$ [*OF* *neg-pwff.hyps*] **by** *simp*
ultimately show *?thesis*
by (*auto simp: inj-eq*)
next
case b
with $\langle \text{free-vars } B = \{\} \rangle$ **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** $\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi (\sim^{\mathcal{Q}} B) = \mathbf{T}$
using $\mathcal{V}_B\text{-neg}$ [*OF* *neg-pwff.hyps*] **by** *simp*
ultimately show *?thesis*
by (*auto simp: inj-eq*)
qed
next
case (*conj-pwff B C*)
from *conj-pwff.premis* **have** *free-vars B = {}* **and** *free-vars C = {}*
by *simp-all*
with *conj-pwff.hyps* **obtain** *b* **and** *b'*
where *B-den*: $\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi B = b$
and *C-den*: $\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi C = b'$
using *closed-pwff-denotation-uniqueness* **by** *metis*
then have $b \in \text{elts } \mathbb{B}$ **and** $b' \in \text{elts } \mathbb{B}$
using *closed-pwff-denotation-uniqueness*[*OF* *conj-pwff.hyps(1)* $\langle \text{free-vars } B = \{\} \rangle$]
and *closed-pwff-denotation-uniqueness*[*OF* *conj-pwff.hyps(2)* $\langle \text{free-vars } C = \{\} \rangle$]
and *conj-pwff.hyps*[*THEN* $\mathcal{V}_B\text{-graph-denotation-is-truth-value}$ [*OF* $\mathcal{V}_B\text{-graph-}\mathcal{V}_B$]]
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** $\vdash T_o \wedge^Q T_o =_o T_o$ (**is** $\langle \vdash ?A1 \rangle$) .
from *B-den[unfolded a(1)]* **and** $\langle \text{free-vars } B = \{\} \rangle$ **have** $\vdash B =_o T_o$
using *conj-pwff.IH(1)* **by** *simp*
then **have** $\vdash B \wedge^Q T_o =_o T_o$ (**is** $\langle \vdash ?A2 \rangle$)
by (*rule rule-RR[OF disjI2, where p = [«»,«»,«»] and C = ?A1]*) (*use* $\langle \vdash ?A1 \rangle$ **in** $\langle \text{force+} \rangle$)
from *C-den[unfolded a(2)]* **and** $\langle \text{free-vars } C = \{\} \rangle$ **have** $\vdash C =_o T_o$
using *conj-pwff.IH(2)* **by** *simp*
then **have** $\vdash B \wedge^Q C =_o T_o$
by (*rule rule-RR[OF disjI2, where p = [«»,«»,«»] and C = ?A2]*) (*use* $\langle \vdash ?A2 \rangle$ **in** $\langle \text{force+} \rangle$)
then **have** $(\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi (B \wedge^Q C) = \mathbf{T}) \longrightarrow \vdash B \wedge^Q C =_o T_o$
by *blast*
moreover **have** $\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi (B \wedge^Q C) \neq \mathbf{F}$
using *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** $\vdash T_o \wedge^Q F_o =_o F_o$ (**is** $\langle \vdash ?A1 \rangle$) .
from *B-den[unfolded b(1)]* **and** $\langle \text{free-vars } B = \{\} \rangle$ **have** $\vdash B =_o T_o$
using *conj-pwff.IH(1)* **by** *simp*
then **have** $\vdash B \wedge^Q F_o =_o F_o$ (**is** $\langle \vdash ?A2 \rangle$)
by (*rule rule-RR[OF disjI2, where p = [«»,«»,«»] and C = ?A1]*) (*use* $\langle \vdash ?A1 \rangle$ **in** $\langle \text{force+} \rangle$)
from *C-den[unfolded b(2)]* **and** $\langle \text{free-vars } C = \{\} \rangle$ **have** $\vdash C =_o F_o$
using *conj-pwff.IH(2)* **by** *simp*
then **have** $\vdash B \wedge^Q C =_o F_o$
by (*rule rule-RR[OF disjI2, where p = [«»,«»,«»] and C = ?A2]*) (*use* $\langle \vdash ?A2 \rangle$ **in** $\langle \text{force+} \rangle$)
then **have** $(\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi (B \wedge^Q C) = \mathbf{F}) \longrightarrow \vdash B \wedge^Q C =_o F_o$
by *blast*
moreover **have** $\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi (B \wedge^Q C) \neq \mathbf{T}$
using *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** $\vdash F_o \wedge^Q T_o =_o F_o$ (**is** $\langle \vdash ?A1 \rangle$) .
from *B-den[unfolded c(1)]* **and** $\langle \text{free-vars } B = \{\} \rangle$ **have** $\vdash B =_o F_o$
using *conj-pwff.IH(1)* **by** *simp*
then **have** $\vdash B \wedge^Q T_o =_o F_o$ (**is** $\langle \vdash ?A2 \rangle$)
by (*rule rule-RR[OF disjI2, where p = [«»,«»,«»] and C = ?A1]*) (*use* $\langle \vdash ?A1 \rangle$ **in** $\langle \text{force+} \rangle$)
from *C-den[unfolded c(2)]* **and** $\langle \text{free-vars } C = \{\} \rangle$ **have** $\vdash C =_o T_o$
using *conj-pwff.IH(2)* **by** *simp*
then **have** $\vdash B \wedge^Q C =_o F_o$
by (*rule rule-RR[OF disjI2, where p = [«»,«»,«»] and C = ?A2]*) (*use* $\langle \vdash ?A2 \rangle$ **in** $\langle \text{force+} \rangle$)
then **have** $(\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi (B \wedge^Q C) = \mathbf{F}) \longrightarrow \vdash B \wedge^Q C =_o F_o$
by *blast*
moreover **have** $\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi (B \wedge^Q C) \neq \mathbf{T}$
using *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 $\vdash F_o \wedge^Q F_o =_o F_o$ (is $\langle \vdash ?A1 \rangle$) .
 from B-den[unfolded d(1)] and $\langle \text{free-vars } B = \{\} \rangle$ have $\vdash B =_o F_o$
 using conj-pwff.IH(1) by simp
 then have $\vdash B \wedge^Q F_o =_o F_o$ (is $\langle \vdash ?A2 \rangle$)
 by (rule rule-RR[OF disjI2, where $p = [\langle \langle, \rangle, \langle, \rangle \rangle]$ and $C = ?A1$]) (use $\langle \vdash ?A1 \rangle$ in $\langle \text{force+} \rangle$)
 from C-den[unfolded d(2)] and $\langle \text{free-vars } C = \{\} \rangle$ have $\vdash C =_o F_o$
 using conj-pwff.IH(2) by simp
 then have $\vdash B \wedge^Q C =_o F_o$
 by (rule rule-RR[OF disjI2, where $p = [\langle \langle, \rangle, \langle, \rangle \rangle]$ and $C = ?A2$]) (use $\langle \vdash ?A2 \rangle$ in $\langle \text{force+} \rangle$)
 then have $(\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi (B \wedge^Q C) = \mathbf{F}) \longrightarrow \vdash B \wedge^Q C =_o F_o$
 by blast
 moreover have $\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi (B \wedge^Q C) \neq \mathbf{T}$
 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.premis have free-vars $B = \{\}$ and free-vars $C = \{\}$
 by simp-all
 with disj-pwff.hyps obtain b and b'
 where B-den: $\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi B = b$
 and C-den: $\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi C = b'$
 using closed-pwff-denotation-uniqueness by metis
 then have $b \in \text{elts } \mathbb{B}$ and $b' \in \text{elts } \mathbb{B}$
 using closed-pwff-denotation-uniqueness[OF disj-pwff.hyps(1) $\langle \text{free-vars } B = \{\} \rangle$]
 and closed-pwff-denotation-uniqueness[OF disj-pwff.hyps(2) $\langle \text{free-vars } C = \{\} \rangle$]
 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 $\vdash T_o \vee^Q T_o =_o T_o$ (is $\langle \vdash ?A1 \rangle$) .
 from B-den[unfolded a(1)] and $\langle \text{free-vars } B = \{\} \rangle$ have $\vdash B =_o T_o$
 using disj-pwff.IH(1) by simp
 then have $\vdash B \vee^Q T_o =_o T_o$ (is $\langle \vdash ?A2 \rangle$)
 by (rule rule-RR[OF disjI2, where $p = [\langle \langle, \rangle, \langle, \rangle \rangle]$ and $C = ?A1$]) (use $\langle \vdash ?A1 \rangle$ in $\langle \text{force+} \rangle$)

from $C\text{-den}[\text{unfolded } a(2)]$ **and** $\langle \text{free-vars } C = \{\} \rangle$ **have** $\vdash C =_o T_o$
using $\text{disj-pwff.IH}(2)$ **by** *simp*
then have $\vdash B \vee^Q C =_o T_o$
by (rule $\text{rule-RR}[\text{OF } \text{disjI2}$, **where** $p = [\langle \langle, \rangle, \rangle]$ **and** $C = ?A2]$) (use $\langle \vdash ?A2 \rangle$ **in** $\langle \text{force+} \rangle$)
then have $(\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi (B \vee^Q C) = \mathbf{T}) \longrightarrow \vdash B \vee^Q C =_o T_o$
by *blast*
moreover have $\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi (B \vee^Q C) \neq \mathbf{F}$
using $\mathcal{V}_B\text{-disj}[\text{OF } \text{disj-pwff.hyps}]$ **and** $B\text{-den}[\text{unfolded } a(1)]$ **and** $C\text{-den}[\text{unfolded } a(2)]$
by (auto *simp: inj-eq*)
ultimately show *?thesis*
by *force*

next

case b

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

next

case c

from $\text{prop-5232}(3)$ **have** $\vdash F_o \vee^Q T_o =_o T_o$ (**is** $\langle \vdash ?A1 \rangle$).
from $B\text{-den}[\text{unfolded } c(1)]$ **and** $\langle \text{free-vars } B = \{\} \rangle$ **have** $\vdash B =_o F_o$
using $\text{disj-pwff.IH}(1)$ **by** *simp*
then have $\vdash B \vee^Q T_o =_o T_o$ (**is** $\langle \vdash ?A2 \rangle$)
by (rule $\text{rule-RR}[\text{OF } \text{disjI2}$, **where** $p = [\langle \langle, \rangle, \langle, \rangle]$ **and** $C = ?A1]$) (use $\langle \vdash ?A1 \rangle$ **in** $\langle \text{force+} \rangle$)
from $C\text{-den}[\text{unfolded } c(2)]$ **and** $\langle \text{free-vars } C = \{\} \rangle$ **have** $\vdash C =_o T_o$
using $\text{disj-pwff.IH}(2)$ **by** *simp*
then have $\vdash B \vee^Q C =_o T_o$
by (rule $\text{rule-RR}[\text{OF } \text{disjI2}$, **where** $p = [\langle \langle, \rangle, \rangle]$ **and** $C = ?A2]$) (use $\langle \vdash ?A2 \rangle$ **in** $\langle \text{force+} \rangle$)
then have $(\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi (B \vee^Q C) = \mathbf{T}) \longrightarrow \vdash B \vee^Q C =_o T_o$
by *blast*
moreover have $\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi (B \vee^Q C) \neq \mathbf{F}$
using $\mathcal{V}_B\text{-disj}[\text{OF } \text{disj-pwff.hyps}]$ **and** $B\text{-den}[\text{unfolded } c(1)]$ **and** $C\text{-den}[\text{unfolded } c(2)]$
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** $\langle \vdash ?A1 \rangle$) .
from *B-den[unfolded d(1)]* **and** $\langle \text{free-vars } B = \{\} \rangle$ **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** $\langle \vdash ?A2 \rangle$)
by (*rule rule-RR[OF disjI2, where p = [«»,«»,«»] and C = ?A1]*) (*use* $\langle \vdash ?A1 \rangle$ **in** $\langle \text{force+} \rangle$)
from *C-den[unfolded d(2)]* **and** $\langle \text{free-vars } C = \{\} \rangle$ **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* $\langle \vdash ?A2 \rangle$ **in** $\langle \text{force+} \rangle$)
then have $(\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi (B \vee^{\mathcal{Q}} C) = \mathbf{T}) \longrightarrow \vdash B \vee^{\mathcal{Q}} C =_o F_o$
by *blast*
moreover have $\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi (B \vee^{\mathcal{Q}} C) \neq \mathbf{T}$
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 $\langle \vdash B \vee^{\mathcal{Q}} C =_o F_o \rangle$ **by** *auto*
qed
next
case (*imp-pwff B C*)
from *imp-pwff.prem*s **have** *free-vars B = {} and free-vars C = {}*
by *simp-all*
with *imp-pwff.hyps* **obtain** *b and b'*
where *B-den: $\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi B = b$*
and *C-den: $\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi C = b'$*
using *closed-pwff-denotation-uniqueness by metis*
then have $b \in \text{elts } \mathbb{B}$ **and** $b' \in \text{elts } \mathbb{B}$
using *closed-pwff-denotation-uniqueness[OF imp-pwff.hyps(1) $\langle \text{free-vars } B = \{\} \rangle$*
and *closed-pwff-denotation-uniqueness[OF imp-pwff.hyps(2) $\langle \text{free-vars } C = \{\} \rangle$*
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** $\langle \vdash ?A1 \rangle$) .
from *B-den[unfolded a(1)]* **and** $\langle \text{free-vars } B = \{\} \rangle$ **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** $\langle \vdash ?A2 \rangle$)
by (*rule rule-RR[OF disjI2, where p = [«»,«»,«»] and C = ?A1]*) (*use* $\langle \vdash ?A1 \rangle$ **in** $\langle \text{force+} \rangle$)
from *C-den[unfolded a(2)]* **and** $\langle \text{free-vars } C = \{\} \rangle$ **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* $\langle \vdash ?A2 \rangle$ **in** $\langle \text{force+} \rangle$)
then have $(\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi (B \supset^{\mathcal{Q}} C) = \mathbf{T}) \longrightarrow \vdash B \supset^{\mathcal{Q}} C =_o T_o$

by *blast*
moreover have $\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi (B \supset^{\mathcal{Q}} C) \neq \mathbf{F}$
 using $\mathcal{V}_B\text{-imp}[OF \text{ imp-pwff.hyps}]$ and $B\text{-den}[\text{unfolded } a(1)]$ and $C\text{-den}[\text{unfolded } a(2)]$
 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** $\langle \vdash ?A1 \rangle$) .
from $B\text{-den}[\text{unfolded } b(1)]$ and $\langle \text{free-vars } B = \{\} \rangle$ **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** $\langle \vdash ?A2 \rangle$)
 by (*rule rule-RR[OF disjI2, where p = [«»,«»,«»] and C = ?A1]*) (*use* $\langle \vdash ?A1 \rangle$ **in** $\langle \text{force+} \rangle$)
from $C\text{-den}[\text{unfolded } b(2)]$ and $\langle \text{free-vars } C = \{\} \rangle$ **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* $\langle \vdash ?A2 \rangle$ **in** $\langle \text{force+} \rangle$)
then have $(\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi (B \supset^{\mathcal{Q}} C) = \mathbf{F}) \longrightarrow \vdash B \supset^{\mathcal{Q}} C =_o F_o$
 by *blast*
moreover have $\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi (B \supset^{\mathcal{Q}} C) \neq \mathbf{T}$
 using $\mathcal{V}_B\text{-imp}[OF \text{ imp-pwff.hyps}]$ and $B\text{-den}[\text{unfolded } b(1)]$ and $C\text{-den}[\text{unfolded } b(2)]$
 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** $\langle \vdash ?A1 \rangle$) .
from $B\text{-den}[\text{unfolded } c(1)]$ and $\langle \text{free-vars } B = \{\} \rangle$ **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** $\langle \vdash ?A2 \rangle$)
 by (*rule rule-RR[OF disjI2, where p = [«»,«»,«»] and C = ?A1]*) (*use* $\langle \vdash ?A1 \rangle$ **in** $\langle \text{force+} \rangle$)
from $C\text{-den}[\text{unfolded } c(2)]$ and $\langle \text{free-vars } C = \{\} \rangle$ **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* $\langle \vdash ?A2 \rangle$ **in** $\langle \text{force+} \rangle$)
then have $(\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi (B \supset^{\mathcal{Q}} C) = \mathbf{T}) \longrightarrow \vdash B \supset^{\mathcal{Q}} C =_o T_o$
 by *blast*
moreover have $\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi (B \supset^{\mathcal{Q}} C) \neq \mathbf{F}$
 using $\mathcal{V}_B\text{-imp}[OF \text{ imp-pwff.hyps}]$ and $B\text{-den}[\text{unfolded } c(1)]$ and $C\text{-den}[\text{unfolded } c(2)]$
 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** $\langle \vdash ?A1 \rangle$) .
from $B\text{-den}[\text{unfolded } d(1)]$ and $\langle \text{free-vars } B = \{\} \rangle$ **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** $\langle \vdash ?A2 \rangle$)
 by (*rule rule-RR[OF disjI2, where p = [«»,«»,«»] and C = ?A1]*) (*use* $\langle \vdash ?A1 \rangle$ **in** $\langle \text{force+} \rangle$)

from $C\text{-den}[\text{unfolded } d(2)]$ **and** $\langle \text{free-vars } C = \{\} \rangle$ **have** $\vdash C =_o F_o$
using $\text{imp-pwff.IH}(2)$ **by** simp
then have $\vdash B \supset^{\mathcal{Q}} C =_o T_o$
by (rule $\text{rule-RR}[OF \text{ disjI2}$, **where** $p = [\langle \langle, \rangle, \rangle]$ **and** $C = ?A2]$) (use $\langle \vdash ?A2 \rangle$ **in** $\langle \text{force+} \rangle$)
then have $(\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi (B \supset^{\mathcal{Q}} C) = \mathbf{T}) \longrightarrow \vdash B \supset^{\mathcal{Q}} C =_o T_o$
by blast
moreover have $\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi (B \supset^{\mathcal{Q}} C) \neq \mathbf{F}$
using $\mathcal{V}_B\text{-imp}[OF \text{ imp-pwff.hyps}]$ **and** $B\text{-den}[\text{unfolded } d(1)]$ **and** $C\text{-den}[\text{unfolded } d(2)]$
by (auto simp : inj-eq)
ultimately show $?thesis$
by force
qed
next
case ($\text{equiv-pwff } B \ C$)
from equiv-pwff.premis **have** $\text{free-vars } B = \{\}$ **and** $\text{free-vars } C = \{\}$
by simp-all
with equiv-pwff.hyps **obtain** b **and** b'
where $B\text{-den}$: $\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi B = b$
and $C\text{-den}$: $\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi C = b'$
using $\text{closed-pwff-denotation-uniqueness}$ **by** metis
then have $b \in \text{elts } \mathbb{B}$ **and** $b' \in \text{elts } \mathbb{B}$
using $\text{closed-pwff-denotation-uniqueness}[OF \text{ equiv-pwff.hyps}(1) \langle \text{free-vars } B = \{\} \rangle]$
and $\text{closed-pwff-denotation-uniqueness}[OF \text{ equiv-pwff.hyps}(2) \langle \text{free-vars } C = \{\} \rangle]$
and $\text{equiv-pwff.hyps}[THEN \mathcal{V}_B\text{-graph-denotation-is-truth-value}[OF \mathcal{V}_B\text{-graph-}\mathcal{V}_B]]$
by force+
with equiv-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 $\text{prop-5230}(1)$ **have** $\vdash (T_o \equiv^{\mathcal{Q}} T_o) =_o T_o$ (is $\langle \vdash ?A1 \rangle$).
from $B\text{-den}[\text{unfolded } a(1)]$ **and** $\langle \text{free-vars } B = \{\} \rangle$ **have** $\vdash B =_o T_o$
using $\text{equiv-pwff.IH}(1)$ **by** simp
then have $\vdash (B \equiv^{\mathcal{Q}} T_o) =_o T_o$ (is $\langle \vdash ?A2 \rangle$)
by (rule $\text{rule-RR}[OF \text{ disjI2}$, **where** $p = [\langle \langle, \rangle, \langle, \rangle]$ **and** $C = ?A1]$) (use $\langle \vdash ?A1 \rangle$ **in** $\langle \text{force+} \rangle$)
from $C\text{-den}[\text{unfolded } a(2)]$ **and** $\langle \text{free-vars } C = \{\} \rangle$ **have** $\vdash C =_o T_o$
using $\text{equiv-pwff.IH}(2)$ **by** simp
then have $\vdash (B \equiv^{\mathcal{Q}} C) =_o T_o$
by (rule $\text{rule-RR}[OF \text{ disjI2}$, **where** $p = [\langle \langle, \rangle, \rangle]$ **and** $C = ?A2]$) (use $\langle \vdash ?A2 \rangle$ **in** $\langle \text{force+} \rangle$)
then have $(\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi (B \equiv^{\mathcal{Q}} C) = \mathbf{T}) \longrightarrow \vdash (B \equiv^{\mathcal{Q}} C) =_o T_o$
by blast
moreover have $\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi (B \equiv^{\mathcal{Q}} C) \neq \mathbf{F}$
using $\mathcal{V}_B\text{-equiv}[OF \text{ equiv-pwff.hyps}]$ **and** $B\text{-den}[\text{unfolded } a(1)]$ **and** $C\text{-den}[\text{unfolded } a(2)]$
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 $\langle \vdash ?A1 \rangle$) .
 from B-den[unfolded b(1)] and $\langle \text{free-vars } B = \{\} \rangle$ 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 $\langle \vdash ?A2 \rangle$)
 by (rule rule-RR[OF disjI2, where $p = [\langle \langle, \rangle, \langle \langle, \rangle \rangle]$ and $C = ?A1$]) (use $\langle \vdash ?A1 \rangle$ in $\langle \text{force+} \rangle$)
 from C-den[unfolded b(2)] and $\langle \text{free-vars } C = \{\} \rangle$ 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 $p = [\langle \langle, \rangle, \rangle]$ and $C = ?A2$]) (use $\langle \vdash ?A2 \rangle$ in $\langle \text{force+} \rangle$)
 then have $(\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi (B \equiv^{\mathcal{Q}} C) = \mathbf{F}) \longrightarrow \vdash (B \equiv^{\mathcal{Q}} C) =_o F_o$
 by blast
 moreover have $\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi (B \equiv^{\mathcal{Q}} C) \neq \mathbf{T}$
 using $\mathcal{V}_B\text{-eqv}[OF \text{eqv-pwff.hyps}]$ 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 $\langle \vdash ?A1 \rangle$) .
 from B-den[unfolded c(1)] and $\langle \text{free-vars } B = \{\} \rangle$ 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 $\langle \vdash ?A2 \rangle$)
 by (rule rule-RR[OF disjI2, where $p = [\langle \langle, \rangle, \langle \langle, \rangle \rangle]$ and $C = ?A1$]) (use $\langle \vdash ?A1 \rangle$ in $\langle \text{force+} \rangle$)
 from C-den[unfolded c(2)] and $\langle \text{free-vars } C = \{\} \rangle$ 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 $p = [\langle \langle, \rangle, \rangle]$ and $C = ?A2$]) (use $\langle \vdash ?A2 \rangle$ in $\langle \text{force+} \rangle$)
 then have $(\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi (B \equiv^{\mathcal{Q}} C) = \mathbf{F}) \longrightarrow \vdash (B \equiv^{\mathcal{Q}} C) =_o F_o$
 by blast
 moreover have $\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi (B \equiv^{\mathcal{Q}} C) \neq \mathbf{T}$
 using $\mathcal{V}_B\text{-eqv}[OF \text{eqv-pwff.hyps}]$ 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 $\langle \vdash ?A1 \rangle$) .
 from B-den[unfolded d(1)] and $\langle \text{free-vars } B = \{\} \rangle$ 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 $\langle \vdash ?A2 \rangle$)
 by (rule rule-RR[OF disjI2, where $p = [\langle \langle, \rangle, \langle \langle, \rangle \rangle]$ and $C = ?A1$]) (use $\langle \vdash ?A1 \rangle$ in $\langle \text{force+} \rangle$)
 from C-den[unfolded d(2)] and $\langle \text{free-vars } C = \{\} \rangle$ 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 $p = [\langle \langle, \rangle, \rangle]$ and $C = ?A2$]) (use $\langle \vdash ?A2 \rangle$ in $\langle \text{force+} \rangle$)
 then have $(\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi (B \equiv^{\mathcal{Q}} C) = \mathbf{T}) \longrightarrow \vdash (B \equiv^{\mathcal{Q}} C) =_o T_o$

by *blast*
moreover have $\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi (B \equiv^{\mathcal{Q}} C) \neq \mathbf{F}$
 using $\mathcal{V}_B\text{-eqv}[OF \text{eqv-pwff.hyps}]$ and $B\text{-den}[\text{unfolded } d(1)]$ and $C\text{-den}[\text{unfolded } d(2)]$
 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 $A \in \text{pwffs}$ and $\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi A = \mathbf{T}$

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.prem*s have $A \in \text{pwffs}$

by *blast*

with *insert.hyps(4)* obtain p where $v = (p, o)$

using *pwffs-free-vars-are-propositional* by *blast*

from $\langle v = (p, o) \rangle$ and *insert.hyps(4)* have

is-tautology ($\mathbf{S} \{(p, o) \rightsquigarrow T_o\} A$) and *is-tautology* ($\mathbf{S} \{(p, o) \rightsquigarrow F_o\} A$)

using *pwff-substitution-tautology-preservation* [*OF insert.prem*s] by *blast+*

moreover from *insert.hyps(2,4)* and $\langle v = (p, o) \rangle$ and $\langle A \in \text{pwffs} \rangle$

have *free-vars* ($\mathbf{S} \{(p, o) \rightsquigarrow T_o\} A$) = F and *free-vars* ($\mathbf{S} \{(p, o) \rightsquigarrow F_o\} A$) = 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 $\vdash \mathbf{S} \{(p, o) \rightsquigarrow T_o\} A$ and $\vdash \mathbf{S} \{(p, o) \rightsquigarrow F_o\} A$

using *insert.hyps(3)* by (*simp-all only:*)

from *this* and *tautology-is-wffo[OF insert.prem*s] **show** *?case*

by (*rule Cases*)

qed

qed

end

6.35 Proposition 5234 (Rule P)

According to the proof in [2], if $[A^1 \wedge \dots \wedge A^n] \supset B$ is tautologous, then clearly $A^1 \supset (\dots (A^n \supset B) \dots)$ 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* $(hs \supset_{\star} B)$

shows $\mathcal{H} \vdash B$

proof –

from *assms*(5) **obtain** ϑ **and** C

where *is-tautology* C

and *is-substitution* ϑ

and $\forall (x, \alpha) \in \text{fmdom}' \vartheta. \alpha = o$

and $hs \supset_{\star} B = \mathbf{S} \vartheta C$

by *blast*

then have $\vdash hs \supset_{\star} B$

proof (*cases* $\vartheta = \{\$\$\}$)

case *True*

with $\langle hs \supset_{\star} B = \mathbf{S} \vartheta C \rangle$ **have** $C = hs \supset_{\star} B$

using *empty-substitution-neutrality* **by** *simp*

with $\langle hs \supset_{\star} B = \mathbf{S} \vartheta C \rangle$ **and** $\langle \text{is-tautology } C \rangle$ **show** *?thesis*

using *prop-5233* **by** (*simp only*.)

next

case *False*

from $\langle \text{is-tautology } C \rangle$ **have** $\vdash C$ **and** $C \in \text{pwffs}$

using *prop-5233* **by** *simp-all*

moreover have

$\forall v \in \text{fmdom}' \vartheta. \text{var-name } v \notin \text{free-var-names } (\{\}\text{:form set}) \wedge \text{is-free-for } (\vartheta \ \$\$! v) v C$

proof

fix v

assume $v \in \text{fmdom}' \vartheta$

then show $\text{var-name } v \notin \text{free-var-names } (\{\}\text{:form set}) \wedge \text{is-free-for } (\vartheta \ \$\$! v) v C$

proof (*cases* $v \in \text{free-vars } C$)

case *True*

with $\langle C \in \text{pwffs} \rangle$ **show** *?thesis*

using *is-free-for-in-pwff* **by** *simp*

next

case *False*

then have $\text{is-free-for } (\vartheta \ \$\$! v) v C$

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** $\langle \text{is-substitution } \vartheta \rangle$ **and** *Sub*

by (*simp add: <math>\langle hs \supset^{\mathcal{Q}}_* B = \mathbf{S} \vartheta \ C \rangle[unfolded generalized-imp-op-def])
qed
 from *this* and *assms(1)* have $\mathcal{H} \vdash hs \supset^{\mathcal{Q}}_* B$
 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 $A \in \text{pwffs}$ and $B \in \text{pwffs}$
 and $(x, \alpha) \notin \text{free-vars } A$
 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 $? \vartheta = \{(\mathfrak{r}, o) \mapsto \forall x_\alpha. (T_o \vee^{\mathcal{Q}} B), (\mathfrak{h}, o) \mapsto \forall x_\alpha. B\}$ and $?C = \mathfrak{r}_o \supset^{\mathcal{Q}} (T_o \vee^{\mathcal{Q}} (\mathfrak{h}_o))$

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 $? \vartheta = \{(\mathfrak{r}, o) \mapsto \forall x_\alpha. B\}$ and $?C = \mathfrak{r}_o \supset^{\mathcal{Q}} (F_o \vee^{\mathcal{Q}} (\mathfrak{r}_o))$

have *is-tautology* $(\mathfrak{r}_o \supset^{\mathcal{Q}} (F_o \vee^{\mathcal{Q}} (\mathfrak{r}_o)))$ (*is <math>\langle is-tautology \ ?C \rangle)*

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 $\forall x_\alpha. B \supset^{\mathcal{Q}} (F_o \vee^{\mathcal{Q}} \forall x_\alpha. B) = \mathbf{S} \text{ ?}\vartheta \text{ ?}C$
by simp
ultimately show *?thesis*
by blast
qed
qed simp
have $\S 3: \vdash B \equiv^{\mathcal{Q}} (F_o \vee^{\mathcal{Q}} B)$
proof (*intro rule-P(2)*)
show *is-tautologous* $(B \equiv^{\mathcal{Q}} (F_o \vee^{\mathcal{Q}} B))$
proof –
let $\text{?}\vartheta = \{(x, o) \mapsto B\}$ **and** $\text{?}C = x_o \equiv^{\mathcal{Q}} (F_o \vee^{\mathcal{Q}} (x_o))$
have *is-tautology* $\text{?}C$
using \mathcal{V}_B -*simps* **by simp**
moreover from *assms(2)* **have** *is-pwff-substitution* $\text{?}\vartheta$
using *pwffs-subset-of-wffso* **by auto**
moreover have $B \equiv^{\mathcal{Q}} (F_o \vee^{\mathcal{Q}} B) = \mathbf{S} \text{ ?}\vartheta \text{ ?}C$
by simp
ultimately show *?thesis*
by blast
qed
qed simp
from $\S 2$ **and** $\S 3$ [*unfolded equivalence-def*] **have** $\S 4$:
 $\vdash \forall x_\alpha. (F_o \vee^{\mathcal{Q}} B) \supset^{\mathcal{Q}} (F_o \vee^{\mathcal{Q}} \forall x_\alpha. B)$
by (*rule rule-R[where p = [«»,»»,«]]*) *force+*
obtain p **where** $(p, o) \notin \text{vars } (\forall x_\alpha. (A \vee^{\mathcal{Q}} B) \supset^{\mathcal{Q}} (A \vee^{\mathcal{Q}} \forall x_\alpha. B))$
by (*meson fresh-var-existence vars-form-finiteness*)
then have $(p, o) \neq (x, \alpha)$ **and** $(p, o) \notin \text{vars } A$ **and** $(p, o) \notin \text{vars } B$
by simp-all
from $\langle (p, o) \notin \text{vars } B \rangle$ **have** *sub*: $\mathbf{S} \{(p, o) \mapsto C\} B = B$ **for** C
using *free-var-singleton-substitution-neutrality* **and** *free-vars-in-all-vars* **by blast**
have $\S 5: \vdash \forall x_\alpha. (p_o \vee^{\mathcal{Q}} B) \supset^{\mathcal{Q}} (p_o \vee^{\mathcal{Q}} \forall x_\alpha. B)$ (**is** $\langle \vdash \text{?}C \rangle$)
proof –
from *sub* **and** $\S 1$ **have** $\vdash \mathbf{S} \{(p, o) \mapsto T_o\} \text{?}C$
using $\langle (p, o) \neq (x, \alpha) \rangle$ **by auto**
moreover from *sub* **and** $\S 4$ **have** $\vdash \mathbf{S} \{(p, o) \mapsto F_o\} \text{?}C$
using $\langle (p, o) \neq (x, \alpha) \rangle$ **by auto**
moreover from *assms(2)* **have** $\text{?}C \in \text{wffs}_o$
using *pwffs-subset-of-wffso* **by auto**
ultimately show *?thesis*
by (*rule Cases*)
qed
then show *?thesis*
proof –
let $\text{?}\vartheta = \{(p, o) \mapsto A\}$
from *assms(1)* **have** *is-substitution* $\text{?}\vartheta$
using *pwffs-subset-of-wffso* **by auto**
moreover have
 $\forall v \in \text{fmdom}' \text{ ?}\vartheta. \text{var-name } v \notin \text{free-var-names } (\{\}::\text{form set}) \wedge \text{is-free-for } (\text{?}\vartheta \text{ $$$ } v) v \text{ ?}C$
proof

```

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

```

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

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

context begin

private lemma prop-5237-aux:

```

assumes A ∈ wffso and B ∈ wffso
and (x, α) ∉ free-vars A
shows ⊢ ∀ xα. (A ⊃Q B) ≡Q (A ⊃Q (∀ xα. B))

```

proof –

```

have is-tautology (ro ≡Q (To ⊃Q ro)) (is ⟨is-tautology ?C1⟩)
  using VB-simps by simp
have is-tautology (ro ⊃Q (ro ≡Q (Fo ⊃Q ηo))) (is ⟨is-tautology ?C2⟩)
  using VB-simps by simp
have §1: ⊢ ∀ xα. B ≡Q (To ⊃Q ∀ xα. B)

```

proof (intro rule-P(2))

show is-tautologous (∀ x_α. B ≡^Q (T_o ⊃^Q ∀ x_α. B))

proof –

```

let ?∅ = {(r, o) ↦ ∀ xα. B}
from assms(2) have is-puff-substitution ?∅
  using pwffs-subset-of-wffso by auto
moreover have ∀ xα. B ≡Q (To ⊃Q ∀ xα. B) = S ?∅ ?C1
  by simp
ultimately show ?thesis
  using ⟨is-tautology ?C1⟩ 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 ? $\vartheta$  =  $\{(x, o) \mapsto B\}$ 
    from assms(2) have is-pwff-substitution ? $\vartheta$ 
      using pwffs-subset-of-wffso by auto
    moreover have  $B \equiv^{\mathcal{Q}} T_o \supset^{\mathcal{Q}} B = \mathbf{S} \text{ ?}\vartheta \text{ ?}C_1$ 
      by simp
    ultimately show ?thesis
      using <is-tautology ? $C_1$ > by blast
  qed
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 ( $(\forall x_\alpha. T_o) \supset^{\mathcal{Q}*} (\forall x_\alpha. T_o \equiv^{\mathcal{Q}} (F_o \supset^{\mathcal{Q}} \forall x_\alpha. B))$ )
  proof -
    let ? $\vartheta$  =  $\{(x, o) \mapsto \forall x_\alpha. T_o, (y, o) \mapsto \forall x_\alpha. B\}$ 
    from assms(2) have is-pwff-substitution ? $\vartheta$ 
      using pwffs-subset-of-wffso by auto
    moreover have  $(\forall x_\alpha. T_o) \supset^{\mathcal{Q}*} (\forall x_\alpha. T_o \equiv^{\mathcal{Q}} (F_o \supset^{\mathcal{Q}} \forall x_\alpha. B)) = \mathbf{S} \text{ ?}\vartheta \text{ ?}C_2$ 
      by simp
    ultimately show ?thesis
      using <is-tautology ? $C_2$ > by blast
  qed
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 ? $\vartheta$  =  $\{(x, o) \mapsto B\}$  and ? $C$  =  $T_o \equiv^{\mathcal{Q}} (F_o \supset^{\mathcal{Q}} \text{r}_o)$ 
    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  $T_o \equiv^{\mathcal{Q}} (F_o \supset^{\mathcal{Q}} B) = \mathbf{S} \text{ ?}\vartheta \text{ ?}C$ 
      by simp
    ultimately show ?thesis
      by blast
  qed
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  $p = [\llbracket, \gg, \gg, \llbracket]$ ]) force+

```


from §1 and §2 have §7: $\vdash \forall x_\alpha. (T_o \supset^Q B) \equiv^Q (T_o \supset^Q \forall x_\alpha. B)$
unfolding equivalence-def by (rule rule-R[where $p = [\langle \langle, \rangle, \rangle, \rangle, \langle \rangle]$]) force+
obtain p where $(p, o) \notin \text{vars } B$ and $p \neq x$
using fresh-var-existence and vars-form-finiteness by (metis finite-insert insert-iff)
from $\langle (p, o) \notin \text{vars } B \rangle$ have $\text{sub}: \mathbf{S} \{(p, o) \mapsto C\} B = B$ for C
using free-var-singleton-substitution-neutrality and free-vars-in-all-vars by blast
have §8: $\vdash \forall x_\alpha. (p_o \supset^Q B) \equiv^Q (p_o \supset^Q \forall x_\alpha. B)$ (**is $\langle \vdash ?C_3 \rangle$**)
proof –
from sub and §7 have $\vdash \mathbf{S} \{(p, o) \mapsto T_o\} ?C_3$
using $\langle p \neq x \rangle$ by auto
moreover from sub and §6 have $\vdash \mathbf{S} \{(p, o) \mapsto F_o\} ?C_3$
using $\langle p \neq x \rangle$ by auto
moreover from $\text{assms}(2)$ have $?C_3 \in \text{wffs}_o$
using $\text{pwffs-subset-of-wffs}_o$ by auto
ultimately show $?thesis$
by (rule Cases)
qed
then show $?thesis$
proof –
let $?v = \{(p, o) \mapsto A\}$
from $\text{assms}(1)$ have $\text{is-substitution } ?v$
using $\text{pwffs-subset-of-wffs}_o$ by auto
moreover have
 $\forall v \in \text{fmdom}' ?v. \text{var-name } v \notin \text{free-var-names } (\{\}::\text{form set}) \wedge \text{is-free-for } (?v \text{ $$$ } v) v ?C_3$
proof
fix v
assume $v \in \text{fmdom}' ?v$
then have $v = (p, o)$
by simp
with $\text{assms}(3)$ and $\langle (p, o) \notin \text{vars } B \rangle$ have $\text{is-free-for } (?v \text{ $$$ } 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 $\text{var-name } v \notin \text{free-var-names } (\{\}::\text{form set})$
by simp
ultimately show $\text{var-name } v \notin \text{free-var-names } (\{\}::\text{form set}) \wedge \text{is-free-for } (?v \text{ $$$ } v) v ?C_3$
unfolding $\langle v = (p, o) \rangle$ by blast
qed
moreover have $?v \neq \{\text{ $$$ }\}$
by simp
ultimately have $\vdash \mathbf{S} ?v ?C_3$
by (rule Sub[OF §8])
moreover have $\mathbf{S} ?v ?C_3 = \forall x_\alpha. (A \supset^Q B) \equiv^Q (A \supset^Q \forall x_\alpha. B)$
using $\langle p \neq x \rangle$ and $\text{sub}[of A]$ 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 $(x, \alpha) \notin \text{free-vars} (\{A\} \cup \mathcal{H})$
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** $A \in \text{wffs}_o$ **and** $B \in \text{wffs}_o$
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 $\supset\forall = \text{prop-5237}$

corollary *generalized-prop-5237*:

assumes *is-hyps* \mathcal{H}
and $\mathcal{H} \vdash A \supset^{\mathcal{Q}} B$
and $\forall v \in S. v \notin \text{free-vars} (\{A\} \cup \mathcal{H})$
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.prem*s(3) **have** $\forall v' \in S. v' \notin \text{free-vars} (\{A\} \cup \mathcal{H})$
by *blast*
then show *?case*
proof (*cases v \in lset vs*)
case *True*
with *Cons.prem*s(4) **have** *lset vs = S*
by *auto*
with *assms*(1,2) **and** \ast **have** $\mathcal{H} \vdash A \supset^{\mathcal{Q}} \forall^{\mathcal{Q}}_{\star} vs B$
by (*fact Cons.IH*)
with *True* **and** $\langle \text{lset } vs = S \rangle$ **and** $\langle v = (x, \alpha) \rangle$ **and** \ast **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 $\langle v = (x, \alpha) \rangle$ **show** *?thesis*
by *simp*

```

next
  case False
  with  $\langle \text{lset } (v \# vs) = S \rangle$  have  $\text{lset } vs = S - \{v\}$ 
    by auto
  moreover from * have  $\forall v' \in S - \{v\}. v' \notin \text{free-vars } (\{A\} \cup \mathcal{H})$ 
    by blast
  ultimately have  $\mathcal{H} \vdash A \supset^{\mathcal{Q}} \forall^{\mathcal{Q}}_* vs B$ 
    using assms(1,2) by (intro Cons.IH)
  moreover from Cons.prem(4) and  $\langle v = (x, \alpha) \rangle$  and * have  $(x, \alpha) \notin \text{free-vars } (\{A\} \cup \mathcal{H})$ 
    by auto
  ultimately have  $\mathcal{H} \vdash A \supset^{\mathcal{Q}} (\forall x_{\alpha}. \forall^{\mathcal{Q}}_* vs B)$ 
    using assms(1) by (intro prop-5237)
  with  $\langle v = (x, \alpha) \rangle$  show ?thesis
    by simp
qed
qed

end

```

6.38 Proposition 5238

context begin

private lemma *prop-5238-aux*:

assumes $A \in \text{wffs}_{\alpha}$ and $B \in \text{wffs}_{\alpha}$

shows $\vdash ((\lambda x_{\beta}. A) =_{\beta \rightarrow \alpha} (\lambda x_{\beta}. B)) \equiv^{\mathcal{Q}} \forall x_{\beta}. (A =_{\alpha} B)$

proof –

have §1:

$\vdash (\text{f}_{\beta \rightarrow \alpha} =_{\beta \rightarrow \alpha} \text{g}_{\beta \rightarrow \alpha}) \equiv^{\mathcal{Q}} \forall \mathfrak{r}_{\beta}. (\text{f}_{\beta \rightarrow \alpha} \cdot \mathfrak{r}_{\beta} =_{\alpha} \text{g}_{\beta \rightarrow \alpha} \cdot \mathfrak{r}_{\beta})$ (**is** $\langle \vdash - \equiv^{\mathcal{Q}} \forall \mathfrak{r}_{\beta}. ?C_1 \rangle$)
 by (*fact axiom-is-derivable-from-no-hyps[OF axiom-3]*)

then have §2:

$\vdash (\text{f}_{\beta \rightarrow \alpha} =_{\beta \rightarrow \alpha} \text{g}_{\beta \rightarrow \alpha}) \equiv^{\mathcal{Q}} \forall x_{\beta}. (\text{f}_{\beta \rightarrow \alpha} \cdot x_{\beta} =_{\alpha} \text{g}_{\beta \rightarrow \alpha} \cdot x_{\beta})$ (**is** $\langle \vdash ?C_2 \rangle$)

proof (*cases x = r*)

case *True*

with §1 **show** *?thesis*

by (*simp only*.)

next

case *False*

have $?C_1 \in \text{wffs}_o$

by *blast*

moreover from *False* **have** $(x, \beta) \notin \text{free-vars } ?C_1$

by *simp*

moreover **have** *is-free-for* $(x_{\beta}) (\mathfrak{r}, \beta) ?C_1$

by (*intro is-free-for-in-equality is-free-for-to-app simp-all*)

ultimately **have** $\vdash \lambda \mathfrak{r}_{\beta}. ?C_1 =_{\beta \rightarrow o} \lambda x_{\beta}. (\mathbf{S} \{(\mathfrak{r}, \beta) \mapsto x_{\beta}\} ?C_1)$

by (*rule* α)

from §1 **and** *this* **show** *?thesis*

by (*rule rule-R[where p = [»,»]]*) *force+*

qed

then have §3:
 $\vdash ((\lambda x_\beta. A) =_{\beta \rightarrow \alpha} (\lambda x_\beta. B)) \equiv^Q \forall x_\beta. ((\lambda x_\beta. A) \cdot x_\beta =_\alpha (\lambda x_\beta. B) \cdot x_\beta)$
proof –
let $?\vartheta = \{(\mathbf{f}, \beta \rightarrow \alpha) \mapsto \lambda x_\beta. A, (\mathbf{g}, \beta \rightarrow \alpha) \mapsto \lambda x_\beta. B\}$
have $\lambda x_\beta. A \in \text{wffs}_{\beta \rightarrow \alpha}$ **and** $\lambda x_\beta. B \in \text{wffs}_{\beta \rightarrow \alpha}$
by (*blast intro: assms(1,2)*)
then have *is-substitution* $?\vartheta$
by *simp*
moreover have
 $\forall v \in \text{fmdom}' \ ?\vartheta. \text{var-name } v \notin \text{free-var-names } (\{\}::\text{form set}) \wedge \text{is-free-for } (?\vartheta \ \$\$! \ v) \ v \ ?C_2$
proof
fix v
assume $v \in \text{fmdom}' \ ?\vartheta$
then consider $(a) \ v = (\mathbf{f}, \beta \rightarrow \alpha) \mid (b) \ v = (\mathbf{g}, \beta \rightarrow \alpha)$
by *fastforce*
then show $\text{var-name } v \notin \text{free-var-names } (\{\}::\text{form set}) \wedge \text{is-free-for } (?\vartheta \ \$\$! \ v) \ v \ ?C_2$
proof cases
case a
have $(x, \beta) \notin \text{free-vars } (\lambda x_\beta. A)$
by *simp*
then have *is-free-for* $(\lambda x_\beta. A) (\mathbf{f}, \beta \rightarrow \alpha) \ ?C_2$
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 $(x, \beta) \notin \text{free-vars } (\lambda x_\beta. B)$
by *simp*
then have *is-free-for* $(\lambda x_\beta. B) (\mathbf{g}, \beta \rightarrow \alpha) \ ?C_2$
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 $\vdash \mathbf{S} \ ?\vartheta \ ?C_2$
by (*rule Sub[OF §2]*)
then show *?thesis*
by *simp*
qed
then have §4: $\vdash ((\lambda x_\beta. A) =_{\beta \rightarrow \alpha} (\lambda x_\beta. B)) \equiv^Q \forall x_\beta. (A =_\alpha (\lambda x_\beta. B) \cdot x_\beta)$
proof –
have $\vdash (\lambda x_\beta. A) \cdot x_\beta =_\alpha A$
using *prop-5208* [**where** $vs = [(x, \beta)]$] **and** *assms(1)* **by** *simp*
from §3 **and this show** *?thesis*
by (*rule rule-R* [**where** $p = [\rangle, \rangle, \langle, \langle, \rangle]$]) *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** $p = [\rangle, \rangle, \langle, \langle]$]) *force+*
qed
qed

proposition *prop-5238*:

assumes $vs \neq []$ **and** $A \in wffs_\alpha$ **and** $B \in wffs_\alpha$
shows $\vdash \lambda^{\mathcal{Q}_*} vs A =_{\text{foldr } (\rightarrow) \text{ (map var-type vs) } \alpha} \lambda^{\mathcal{Q}_*} vs B \equiv^{\mathcal{Q}} \forall^{\mathcal{Q}_*} vs (A =_\alpha B)$
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.prem*s **have**
 $\lambda^{\mathcal{Q}_*} vs A =_{\text{foldr } (\rightarrow) \text{ (map var-type vs) } \alpha} \lambda^{\mathcal{Q}_*} vs B \equiv^{\mathcal{Q}} \forall^{\mathcal{Q}_*} vs (A =_\alpha B) \in wffs_o$
 by *blast*
with *single.prem*s **and** $\langle v = (x, \beta) \rangle$ **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.prem*s **have** $\lambda x_\beta. A \in wffs_{\beta \rightarrow \alpha}$ **and** $\lambda x_\beta. B \in wffs_{\beta \rightarrow \alpha}$
 by *auto*
then have

\vdash
 $\lambda^{\mathcal{Q}_*} vs (\lambda x_\beta. A) =_{\text{foldr } (\rightarrow) \text{ (map var-type vs) } (\beta \rightarrow \alpha)} \lambda^{\mathcal{Q}_*} vs (\lambda x_\beta. B)$
 $\equiv^{\mathcal{Q}}$
 $\forall^{\mathcal{Q}_*} vs ((\lambda x_\beta. A) =_{\beta \rightarrow \alpha} (\lambda x_\beta. B))$
by (*fact snoc.IH*)

moreover from *snoc.prem*s **have** $\vdash \lambda x_\beta. A =_{\beta \rightarrow \alpha} \lambda x_\beta. B \equiv^{\mathcal{Q}} \forall x_\beta. (A =_\alpha B)$
by (*fact prop-5238-aux*)

ultimately have

\vdash
 $\lambda^{\mathcal{Q}_*} vs (\lambda x_\beta. A) =_{\text{foldr } (\rightarrow) \text{ (map var-type vs) } (\beta \rightarrow \alpha)} \lambda^{\mathcal{Q}_*} vs (\lambda x_\beta. B)$
 $\equiv^{\mathcal{Q}}$
 $\forall^{\mathcal{Q}_*} vs \forall x_\beta. (A =_\alpha B)$

unfolding *equivalence-def* **proof** (*induction rule: rule-R* [**where** $p = \rangle \# \text{foldr } (\lambda-. (@) [\rangle, \langle]) vs []$])

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, β)⟩ show ?case
      by simp
  qed
end

```

6.39 Proposition 5239

lemma replacement-derivability:

```

  assumes C ∈ wffsβ
  and A ≼p C
  and ⊢ A =α B
  and C⟨p ← B⟩ ▷ D
  shows ⊢ C =β D
using assms proof (induction arbitrary: D p)
  case (var-is-wff γ x)
  from var-is-wff.premis(1) have p = [] and A = xγ
  by (auto elim: is-subform-at.elims(2))
  with var-is-wff.premis(2) have α = γ
  using hyp-derivable-form-is-wffso and wff-has-unique-type and wffs-from-equality by blast
  moreover from ⟨p = []⟩ and var-is-wff.premis(3) have D = B
  using is-replacement-at-minimal-change(1) and is-subform-at.simps(1) by iprover
  ultimately show ?case
  using ⟨A = xγ⟩ and var-is-wff.premis(2) by (simp only:)
next
  case (con-is-wff γ c)
  from con-is-wff.premis(1) have p = [] and A = {c}γ
  by (auto elim: is-subform-at.elims(2))
  with con-is-wff.premis(2) have α = γ
  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.premis(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}γ⟩ and con-is-wff.premis(2) by (simp only:)
next
  case (app-is-wff γ δ C1 C2)
  from app-is-wff.premis(1) consider
  (a) p = []
  | (b) ∃ p'. p = « # p' ∧ A ≼p' C1
  | (c) ∃ p'. p = » # p' ∧ A ≼p' C2
  using subforms-from-app by blast
  then show ?case
  proof cases
    case a

```

with *app-is-wff.premis(1)* **have** $A = C_1 \cdot C_2$
by *simp*
moreover from *a* **and** *app-is-wff.premis(3)* **have** $D = B$
using *is-replacement-at-minimal-change(1)* **and** *at-top-is-self-subform* **by** *blast*
moreover from $\langle A = C_1 \cdot C_2 \rangle$ **and** $\langle D = B \rangle$ **and** *app-is-wff.hyps(1,2)* **and** *assms(3)* **have** $\alpha =$
 δ
using *hyp-derivable-form-is-wffso* **and** *wff-has-unique-type*
by (*blast dest: wffs-from-equality*)
ultimately show *?thesis*
using *assms(3)* **by** (*simp only:*)
next
case *b*
then obtain p' **where** $p = \ll \# p'$ **and** $A \preceq_{p'} C_1$
by *blast*
moreover obtain D_1 **where** $D = D_1 \cdot C_2$ **and** $C_1 \langle p' \leftarrow B \rangle \triangleright D_1$
using *app-is-wff.premis(3)* **and** $\langle p = \ll \# p' \rangle$ **by** (*force dest: is-replacement-at.cases*)
ultimately have $\vdash C_1 =_{\gamma \rightarrow \delta} D_1$
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 $\langle D = D_1 \cdot C_2 \rangle$ **show** *?thesis*
by (*simp only:*)
next
case *c*
then obtain p' **where** $p = \gg \# p'$ **and** $A \preceq_{p'} C_2$
by *blast*
moreover obtain D_2 **where** $D = C_1 \cdot D_2$ **and** $C_2 \langle p' \leftarrow B \rangle \triangleright D_2$
using *app-is-wff.premis(3)* **and** $\langle p = \gg \# p' \rangle$ **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 $\vdash C_1 =_{\gamma \rightarrow \delta} C_1$
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 $\langle D = C_1 \cdot D_2 \rangle$ **show** *?thesis*
by (*simp only:*)
qed
next
case (*abs-is-wff* δ $C' \gamma x$)
from *abs-is-wff.premis(1)* **consider** (a) $p = []$ | (b) $\exists p'. p = \ll \# p' \wedge A \preceq_{p'} C'$
using *subforms-from-abs* **by** *blast*
then show *?case*
proof cases
case *a*
with *abs-is-wff.premis(1)* **have** $A = \lambda x \gamma. C'$
by *simp*
moreover from *a* **and** *abs-is-wff.premis(3)* **have** $D = B$

using *is-replacement-at-minimal-change(1)* **and** *at-top-is-self-subform* **by** *blast*
moreover from $\langle A = \lambda x_\gamma. C' \rangle$ **and** $\langle D = B \rangle$ **and** *abs-is-wff.hyps(1)* **and** *assms(3)* **have** $\alpha =$
 $\gamma \rightarrow \delta$
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** $p = \ll \# p' \text{ and } A \preceq_{p'} C' \gg$
by *blast*
moreover obtain D' **where** $D = \lambda x_\gamma. D'$ **and** $C' \langle p' \leftarrow B \rangle \triangleright D'$
using *abs-is-wff.prem(3)* **and** $\langle p = \ll \# p' \gg$ **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 $\vdash \lambda x_\gamma. C' =_{\gamma \rightarrow \delta} \lambda x_\gamma. D'$
proof –
from $\langle \vdash C' =_\delta D' \rangle$ **have** $\vdash \forall x_\gamma. (C' =_\delta D')$
using *Gen* **by** *simp*
moreover from $\langle \vdash C' =_\delta D' \rangle$ **and** *abs-is-wff.hyps* **have** $D' \in \text{wffs}_\delta$
using *hyp-derivable-form-is-wffso* **by** (*blast dest: wffs-from-equality*)
with *abs-is-wff.hyps* **have** $\vdash (\lambda x_\gamma. C' =_{\gamma \rightarrow \delta} \lambda x_\gamma. D') \equiv^{\mathcal{Q}} \forall x_\gamma. (C' =_\delta D')$
using *prop-5238[where vs = [(x, \gamma)]]* **by** *simp*
ultimately show *?thesis*
using *Equality-Rules(1,2)* **unfolding** *equivalence-def* **by** *blast*
qed
with $\langle D = \lambda x_\gamma. D' \rangle$ **show** *?thesis*
by (*simp only:*)
qed
qed

context
begin

private lemma *prop-5239-aux-1:*
assumes $p \in \text{positions } (\cdot^{\mathcal{Q}}_* (FVar v) (\text{map } FVar vs))$
and $p \neq \text{replicate } (\text{length } vs) \ll$
shows
 $(\exists A B. A \cdot B \preceq_p (\cdot^{\mathcal{Q}}_* (FVar v) (\text{map } FVar vs)))$
 \vee
 $(\exists v \in \text{lset } vs. \text{occurs-at } v p (\cdot^{\mathcal{Q}}_* (FVar v) (\text{map } FVar vs)))$
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.prem(1)* **consider**
 $(a) p = []$


```

| (b)  $p = [\gg]$ 
| (c)  $\exists p' \in \text{positions } (\cdot^{\mathcal{Q}}_{\star} (FVar v) (\text{map } FVar vs))$ .  $p = \ll \# p'$ 
  using surj-pair[of  $v'$ ] by fastforce
then show ?case
proof cases
  case c
  then obtain  $p'$  where  $p' \in \text{positions } (\cdot^{\mathcal{Q}}_{\star} (FVar v) (\text{map } FVar vs))$  and  $p = \ll \# p'$ 
  by blast
  from  $\langle p = \ll \# p' \rangle$  and snoc.prems(2) have  $p' \neq \text{replicate } (\text{length } vs) \ll$ 
  by force
  then have
     $(\exists A B. A \cdot B \preceq_{p'} \cdot^{\mathcal{Q}}_{\star} (FVar v) (\text{map } FVar vs))$ 
     $\vee$ 
     $(\exists v \in \text{lset } vs. \text{occurs-at } v p' (\cdot^{\mathcal{Q}}_{\star} (FVar v) (\text{map } FVar vs)))$ 
  using  $\langle p' \in \text{positions } (\cdot^{\mathcal{Q}}_{\star} (FVar v) (\text{map } FVar vs)) \rangle$  and snoc.IH by simp
  with  $\langle p = \ll \# p' \rangle$  show ?thesis
  by auto
qed simp-all
qed

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

```

```

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

```

```

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

private lemma prop-5239-aux-3:
  assumes  $t \notin \text{lset } vs \cup \text{vars } \{A, C\}$ 
  and  $C \langle p \leftarrow (\cdot^{\mathcal{Q}}_{\star} (FVar\ t) (map\ FVar\ vs)) \rangle \triangleright G$ 
  and occurs-at  $t\ p'\ G$ 
  shows  $p' = p @ replicate\ (length\ vs)\ \ll (\text{is } \langle p' = ?p_t \rangle)$ 
proof (cases vs = [])
  case True
  then have  $t \notin \text{vars } C$ 
    using assms(1) by auto
  moreover from True and assms(2) have  $C \langle p \leftarrow FVar\ t \rangle \triangleright G$ 
    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' ∈ positions G
  using is-subform-implies-in-positions by fastforce
from assms(2) have ?pt ∈ 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 ?pt 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' ∈ 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 ∈ 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 ?pt G⟩ and ⟨p' ≠ ?pt⟩ 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' ?pt
    proof (rule ccontr)
      assume ¬ strict-prefix p' ?pt
      then consider
        (b1) p' = ?pt
        | (b2) strict-prefix ?pt p'
        | (b3) ¬ prefix p' ?pt and ¬ prefix ?pt p'
      by fastforce
    then show False
  proof cases
    case b1
    with ⟨p' ≠ ?pt⟩ show ?thesis
      by contradiction
  next
  next

```

```

case  $b_2$ 
with  $\langle \text{occurs-at } t \ ?p_t \ G \rangle$  and  $\text{assms}(3)$  show  $?thesis$ 
  using  $\text{loop-subform-impossibility}$  by  $\text{blast}$ 
next
case  $b_3$ 
from  $b$  obtain  $p''$  where  $p' = p @ p''$  and  $p'' \in \text{positions } (\cdot^{\mathcal{Q}}_{\star} (FVar\ t) (\text{map } FVar\ vs))$ 
  using  $\text{is-replacement-at-new-positions}$  and  $\langle p' \in \text{positions } G \rangle$  and  $\text{assms}(2)$  by  $\text{blast}$ 
moreover have  $p'' \neq \text{replicate } (\text{length } vs) \ll$ 
  using  $\langle p' = p @ p'' \rangle$  and  $\langle p' \neq ?p_t \rangle$  by  $\text{blast}$ 
ultimately consider
   $(b_{3-1}) \exists F_1\ F_2. F_1 \cdot F_2 \preceq_{p''} (\cdot^{\mathcal{Q}}_{\star} (FVar\ t) (\text{map } FVar\ vs))$ 
|  $(b_{3-2}) \exists v \in \text{lset } vs. \text{occurs-at } v\ p'' (\cdot^{\mathcal{Q}}_{\star} (FVar\ t) (\text{map } FVar\ vs))$ 
  using  $\text{prop-5239-aux-1}$  and  $b_3(1,2)$  and  $\text{is-replacement-at-occurs}$ 
  and  $\text{leftmost-subform-in-generalized-app-replacement}$ 
  by  $(\text{metis } (\text{no-types, opaque-lifting}) \text{length-map prefix-append})$ 
then show  $?thesis$ 
proof cases
  case  $b_{3-1}$ 
  with  $\text{assms}(2)$  and  $\langle p' = p @ p'' \rangle$  have  $\exists F_1\ F_2. F_1 \cdot F_2 \preceq_{p'} G$ 
  using  $\text{is-replacement-at-minimal-change}(1)$  and  $\text{is-subform-at-transitivity}$  by  $\text{meson}$ 
  with  $\langle \text{occurs-at } t\ p' \ G \rangle$  show  $?thesis$ 
  using  $\text{is-subform-at-uniqueness}$  by  $\text{fastforce}$ 
next
  case  $b_{3-2}$ 
  with  $\text{assms}(2)$  and  $\langle p' = p @ p'' \rangle$  have  $\exists v \in \text{lset } vs. \text{occurs-at } v\ p' \ G$ 
  unfolding  $\text{occurs-at-def}$ 
  using  $\text{is-replacement-at-minimal-change}(1)$  and  $\text{is-subform-at-transitivity}$  by  $\text{meson}$ 
  with  $\text{assms}(1,3)$  show  $?thesis$ 
  using  $\text{is-subform-at-uniqueness}$  by  $\text{fastforce}$ 
qed
qed
qed
with  $\langle \text{occurs-at } t \ ?p_t \ G \rangle$  and  $\text{assms}(3)$  show  $?thesis$ 
  using  $\text{loop-subform-impossibility}$  by  $\text{blast}$ 
qed
qed
qed
qed

```

```

private lemma  $\text{prop-5239-aux-4}$ :
  assumes  $t \notin \text{lset } vs \cup \text{vars } \{A, C\}$ 
  and  $A \preceq_p C$ 
  and  $\text{lset } vs \supseteq \text{capture-exposed-vars-at } p\ C\ A$ 
  and  $C \langle p \leftarrow (\cdot^{\mathcal{Q}}_{\star} (FVar\ t) (\text{map } FVar\ vs)) \rangle \triangleright G$ 
  shows  $\text{is-free-for } (\lambda^{\mathcal{Q}}_{\star} vs\ A)\ t\ G$ 
unfolding  $\text{is-free-for-def}$  proof  $(\text{intro ballI impI})$ 
  let  $?p_t = p @ \text{replicate } (\text{length } vs) \ll$ 
  from  $\text{assms}(4)$  have  $FVar\ t \preceq_{?p_t} G$ 
  using  $\text{is-replacement-at-minimal-change}(1)$  and  $\text{is-subform-at-transitivity}$ 

```

and *leftmost-subform-in-generalized-app* **by** (*metis length-map*)
fix v' **and** p'
assume $v' \in \text{free-vars } (\lambda^{\mathcal{Q}_*} vs A)$ **and** $p' \in \text{positions } G$ **and** *is-free-at* $t p' G$
have $v' \notin \text{binders-at } G \text{ ?}p_t$
proof –
have $\text{free-vars } (\lambda^{\mathcal{Q}_*} vs A) = \text{free-vars } A - \text{lset } vs$
by (*fact free-vars-of-generalized-abs*)
also from *assms(2,3)* **have** $\dots \subseteq \text{free-vars } A - (\text{binders-at } C p \cap \text{free-vars } A)$
using *capture-exposed-vars-at-alt-def* **and** *is-subform-implies-in-positions* **by** *fastforce*
also have $\dots = \text{free-vars } A - (\text{binders-at } G p \cap \text{free-vars } A)$
using *assms(2,4)* *is-replacement-at-binders* *is-subform-implies-in-positions* **by** *blast*
finally have $\text{free-vars } (\lambda^{\mathcal{Q}_*} vs A) \subseteq \text{free-vars } A - (\text{binders-at } G p \cap \text{free-vars } A)$.
moreover have $\text{binders-at } (\cdot^{\mathcal{Q}_*} (FVar t) (\text{map } FVar vs)) (\text{replicate } (\text{length } vs) \llbracket \rrbracket) = \{\}$
by (*induction vs rule: rev-induct*) *simp-all*
with *assms(4)* **have** $\text{binders-at } G \text{ ?}p_t = \text{binders-at } G p$
using *binders-at-concat* **and** *is-replacement-at-minimal-change(1)* **by** *blast*
ultimately show *?thesis*
using $\langle v' \in \text{free-vars } (\lambda^{\mathcal{Q}_*} vs A) \rangle$ **by** *blast*
qed
moreover have $p' = \text{?}p_t$
by
(
fact prop-5239-aux-3
[*OF assms(1,4)* $\langle \text{is-free-at } t p' G \rangle$ [*unfolded is-free-at-def, THEN conjunct1*]]
)
ultimately show $\neg \text{in-scope-of-abs } v' p' G$
using *binders-at-alt-def* [*OF* $\langle p' \in \text{positions } G \rangle$] **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 $\text{lset } vs =$

$\{(x, \beta) \mid x \beta p' E. \text{strict-prefix } p' p \wedge \lambda x \beta. E \preceq_{p'} C \wedge (x, \beta) \in \text{free-vars } (A =_{\alpha} B)\}$

shows $\vdash \forall \mathcal{Q}_* vs (A =_{\alpha} B) \supset^{\mathcal{Q}} (C \equiv^{\mathcal{Q}} D)$

proof –

let $\text{?}\gamma = \text{foldr } (\rightarrow) (\text{map } \text{var-type } vs) \alpha$

obtain t **where** $(t, \text{?}\gamma) \notin \text{lset } vs \cup \text{vars } \{A, B, C, D\}$

using *fresh-var-existence* **and** *vars-form-set-finiteness*

by (*metis List.finite-set finite.simps finite-UnI*)

from *assms(1)* **have** $A \in \text{wffs}_{\alpha}$ **and** $B \in \text{wffs}_{\alpha}$ **and** $A \preceq_p C$

using *wffs-from-equality* [*OF equality-wff*] **by** *simp-all*

from *assms(1)* **have** $C \in \text{wffs}_o$ **and** $D \in \text{wffs}_o$

using *replacement-preserves-typing* **by** *fastforce+*

have $\cdot^{\mathcal{Q}_*} t_{\text{?}\gamma} (\text{map } FVar vs) \in \text{wffs}_{\alpha}$

using *generalized-app-wff* [**where** $As = \text{map } FVar vs$ **and** $ts = \text{map } \text{var-type } vs$]

by (*metis eq-snd-iff length-map nth-map wffs-of-type-intros(1)*)

from *assms(1)* **have** $p \in \text{positions } C$

using *is-subform-implies-in-positions* **by** *fastforce*

then obtain G **where** $C \llbracket p \leftarrow (\cdot^{\mathcal{Q}_*} t_{\text{?}\gamma} (\text{map } FVar vs)) \rrbracket \triangleright G$

using *is-replacement-at-existence* **by** *blast*
with $\langle A \preceq_p C \rangle$ **and** $\langle \cdot^{\mathcal{Q}}_* t_{\gamma} (\text{map } FVar \text{ vs}) \in \text{wffs}_\alpha \rangle$ **have** $G \in \text{wffs}_o$
using $\langle A \in \text{wffs}_\alpha \rangle$ **and** $\langle C \in \text{wffs}_o \rangle$ **and** *replacement-preserves-typing* **by** *blast*
let $? \vartheta = \{ (\mathfrak{h}, ?\gamma \rightarrow o) \mapsto \lambda t_{\gamma}. G, (\mathfrak{r}, ?\gamma) \mapsto \lambda^{\mathcal{Q}}_* \text{vs } A, (\mathfrak{h}, ?\gamma) \mapsto \lambda^{\mathcal{Q}}_* \text{vs } B \}$
and $?A = (\mathfrak{r}_{?\gamma} =_{?\gamma} \mathfrak{h}_{?\gamma}) \supset^{\mathcal{Q}} (\mathfrak{h}_{?\gamma \rightarrow o} \cdot \mathfrak{r}_{?\gamma} \equiv^{\mathcal{Q}} \mathfrak{h}_{?\gamma \rightarrow o} \cdot \mathfrak{h}_{?\gamma})$
have $\vdash ?A$
by (*fact axiom-is-derivable-from-no-hyps*[*OF axiom-2*])
moreover **have** $\lambda t_{\gamma}. G \in \text{wffs}_{?\gamma \rightarrow o}$ **and** $\lambda^{\mathcal{Q}}_* \text{vs } A \in \text{wffs}_{?\gamma}$ **and** $\lambda^{\mathcal{Q}}_* \text{vs } B \in \text{wffs}_{?\gamma}$
by (*blast intro*: $\langle G \in \text{wffs}_o \rangle \langle A \in \text{wffs}_\alpha \rangle \langle B \in \text{wffs}_\alpha \rangle$)
then **have** *is-substitution* $? \vartheta$
by *simp*
moreover **have**
 $\forall v \in \text{fmdom}' ? \vartheta. \text{var-name } v \notin \text{free-var-names} (\{ \cdot \} :: \text{form set}) \wedge \text{is-free-for} (? \vartheta \ \$\$! v) v ?A$
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** $\vdash \mathbf{S} ? \vartheta ?A$
by (*rule Sub*)
moreover **have**
 $\mathbf{S} ? \vartheta ?A = (\lambda^{\mathcal{Q}}_* \text{vs } A =_{?\gamma} \lambda^{\mathcal{Q}}_* \text{vs } B) \supset^{\mathcal{Q}} ((\lambda t_{\gamma}. G) \cdot (\lambda^{\mathcal{Q}}_* \text{vs } A) \equiv^{\mathcal{Q}} (\lambda t_{\gamma}. G) \cdot (\lambda^{\mathcal{Q}}_* \text{vs } B))$
by *simp*
ultimately **have** §1:
 $\vdash (\lambda^{\mathcal{Q}}_* \text{vs } A =_{?\gamma} \lambda^{\mathcal{Q}}_* \text{vs } B) \supset^{\mathcal{Q}} ((\lambda t_{\gamma}. G) \cdot (\lambda^{\mathcal{Q}}_* \text{vs } A) \equiv^{\mathcal{Q}} (\lambda t_{\gamma}. G) \cdot (\lambda^{\mathcal{Q}}_* \text{vs } B))$
by (*simp only*:)
then **have** §2: $\vdash (\forall^{\mathcal{Q}}_* \text{vs } (A =_\alpha B)) \supset^{\mathcal{Q}} ((\lambda t_{\gamma}. G) \cdot (\lambda^{\mathcal{Q}}_* \text{vs } A) \equiv^{\mathcal{Q}} (\lambda t_{\gamma}. G) \cdot (\lambda^{\mathcal{Q}}_* \text{vs } B))$
proof (*cases vs = []*)
case *True*
with §1 **show** *?thesis*
by *simp*
next
case *False*
from §1 **and** *prop-5238*[*OF False* $\langle A \in \text{wffs}_\alpha \rangle \langle B \in \text{wffs}_\alpha \rangle$] **show** *?thesis*
unfolding *equivalence-def* **by** (*rule rule-R*[**where** $p = [\langle \langle, \rangle \rangle]$]) *force* +
qed
moreover **have** $\vdash (\lambda t_{\gamma}. G) \cdot (\lambda^{\mathcal{Q}}_* \text{vs } A) =_o C$ **and** $\vdash (\lambda t_{\gamma}. G) \cdot (\lambda^{\mathcal{Q}}_* \text{vs } B) =_o D$
proof –
from *assms*(1) **have** $B \preceq_p D$
using *is-replacement-at-minimal-change*(1) **by** *force*
from *assms*(1) **have** $D \langle p \leftarrow (\cdot^{\mathcal{Q}}_* t_{\gamma} (\text{map } FVar \text{ vs})) \rangle \triangleright G$
using $\langle C \langle p \leftarrow (\cdot^{\mathcal{Q}}_* t_{\gamma} (\text{map } FVar \text{ vs})) \rangle \triangleright G \rangle$ **and** *replacement-override*
by (*meson is-rule-R-app-def*)

from $\langle B \preceq_p D \rangle$ **have** $p \in \text{positions } D$
using *is-subform-implies-in-positions* **by** *auto*
from *assms(1)* **have** $\text{binders-at } D \ p = \text{binders-at } C \ p$
using *is-replacement-at-binders* **by** *fastforce*
then have $\text{binders-at } D \ p \cap \text{free-vars } B = \text{binders-at } C \ p \cap \text{free-vars } B$
by *simp*
with *assms(2)*
[
folded capture-exposed-vars-at-def,
unfolded capture-exposed-vars-at-alt-def[*OF* $\langle p \in \text{positions } C \rangle$]
] **have** $\text{lset } vs \supseteq \text{capture-exposed-vars-at } p \ D \ B$
unfolding *capture-exposed-vars-at-alt-def*[*OF* $\langle p \in \text{positions } D \rangle$] **by** *auto*
have *is-free-for* $(\lambda^{\mathcal{Q}_*} vs \ A) \ (t, \ ?\gamma) \ G$ **and** *is-free-for* $(\lambda^{\mathcal{Q}_*} vs \ B) \ (t, \ ?\gamma) \ G$
proof –
have $(t, \ ?\gamma) \notin \text{lset } vs \cup \text{vars } \{A, C\}$ **and** $(t, \ ?\gamma) \notin \text{lset } vs \cup \text{vars } \{B, D\}$
using $\langle (t, \ ?\gamma) \notin \text{lset } vs \cup \text{vars } \{A, B, C, D\} \rangle$ **by** *simp-all*
moreover from *assms(2)* **have**
 $\text{lset } vs \supseteq \text{capture-exposed-vars-at } p \ C \ A$ **and** $\text{lset } vs \supseteq \text{capture-exposed-vars-at } p \ D \ B$
by *fastforce fact*
ultimately show *is-free-for* $(\lambda^{\mathcal{Q}_*} vs \ A) \ (t, \ ?\gamma) \ G$ **and** *is-free-for* $(\lambda^{\mathcal{Q}_*} vs \ B) \ (t, \ ?\gamma) \ G$
using *prop-5239-aux-4* **and** $\langle B \preceq_p D \rangle$ **and** $\langle A \preceq_p C \rangle$ **and** $\langle C \langle p \leftarrow (\cdot^{\mathcal{Q}_*} t_{?\gamma} (\text{map } FVar \ vs)) \rangle \rangle$
 $\triangleright G \rangle$
and $\langle D \langle p \leftarrow (\cdot^{\mathcal{Q}_*} t_{?\gamma} (\text{map } FVar \ vs)) \rangle \rangle \triangleright G \rangle$ **by** *meson+*
qed
then have $\vdash (\lambda t_{?\gamma}. G) \cdot (\lambda^{\mathcal{Q}_*} vs \ A) =_o \mathbf{S} \{(t, \ ?\gamma) \mapsto \lambda^{\mathcal{Q}_*} vs \ A\} \ G$
and $\vdash (\lambda t_{?\gamma}. G) \cdot (\lambda^{\mathcal{Q}_*} vs \ B) =_o \mathbf{S} \{(t, \ ?\gamma) \mapsto \lambda^{\mathcal{Q}_*} vs \ B\} \ G$
using *prop-5207*[*OF* $\langle \lambda^{\mathcal{Q}_*} vs \ A \in \text{wffs}_{?\gamma} \rangle \langle G \in \text{wffs}_o \rangle$]
and *prop-5207*[*OF* $\langle \lambda^{\mathcal{Q}_*} vs \ B \in \text{wffs}_{?\gamma} \rangle \langle G \in \text{wffs}_o \rangle$] **by** *auto*
moreover obtain G'_1 **and** G'_2
where $C \langle p \leftarrow (\cdot^{\mathcal{Q}_*} (\lambda^{\mathcal{Q}_*} vs \ A) (\text{map } FVar \ vs)) \rangle \triangleright G'_1$
and $D \langle p \leftarrow (\cdot^{\mathcal{Q}_*} (\lambda^{\mathcal{Q}_*} vs \ B) (\text{map } FVar \ vs)) \rangle \triangleright G'_2$
using *is-replacement-at-existence*[*OF* $\langle p \in \text{positions } C \rangle$]
and *is-replacement-at-existence*[*OF* $\langle p \in \text{positions } D \rangle$] **by** *metis*
then have $\mathbf{S} \{(t, \ ?\gamma) \mapsto \lambda^{\mathcal{Q}_*} vs \ A\} \ G = G'_1$ **and** $\mathbf{S} \{(t, \ ?\gamma) \mapsto \lambda^{\mathcal{Q}_*} vs \ B\} \ G = G'_2$
proof –
have $(t, \ ?\gamma) \notin \text{lset } vs \cup \text{vars } C$ **and** $(t, \ ?\gamma) \notin \text{lset } vs \cup \text{vars } D$
using $\langle (t, \ ?\gamma) \notin \text{lset } vs \cup \text{vars } \{A, B, C, D\} \rangle$ **by** *simp-all*
then show $\mathbf{S} \{(t, \ ?\gamma) \mapsto \lambda^{\mathcal{Q}_*} vs \ A\} \ G = G'_1$ **and** $\mathbf{S} \{(t, \ ?\gamma) \mapsto \lambda^{\mathcal{Q}_*} vs \ B\} \ G = G'_2$
using $\langle C \langle p \leftarrow (\cdot^{\mathcal{Q}_*} t_{?\gamma} (\text{map } FVar \ vs)) \rangle \triangleright G \rangle$ **and** $\langle D \langle p \leftarrow (\cdot^{\mathcal{Q}_*} t_{?\gamma} \text{map } FVar \ vs) \rangle \triangleright G \rangle$
and $\langle C \langle p \leftarrow (\cdot^{\mathcal{Q}_*} (\lambda^{\mathcal{Q}_*} vs \ A) (\text{map } FVar \ vs)) \rangle \triangleright G'_1 \rangle$
and $\langle D \langle p \leftarrow (\cdot^{\mathcal{Q}_*} (\lambda^{\mathcal{Q}_*} vs \ B) (\text{map } FVar \ vs)) \rangle \triangleright G'_2 \rangle$ **and** *prop-5239-aux-2* **by** *blast+*
qed
ultimately have $\vdash (\lambda t_{?\gamma}. G) \cdot (\lambda^{\mathcal{Q}_*} vs \ A) =_o G'_1$ **and** $\vdash (\lambda t_{?\gamma}. G) \cdot (\lambda^{\mathcal{Q}_*} vs \ B) =_o G'_2$
by (*simp-all only*:)
moreover
have $\vdash A =_\alpha (\cdot^{\mathcal{Q}_*} (\lambda^{\mathcal{Q}_*} vs \ A) (\text{map } FVar \ vs))$ **and** $\vdash B =_\alpha (\cdot^{\mathcal{Q}_*} (\lambda^{\mathcal{Q}_*} vs \ B) (\text{map } FVar \ vs))$
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 $\langle vs = [] \rangle$ **using** *prop-5200* **and** $\langle A \in wffs_{\alpha} \rangle$ **and** $\langle B \in wffs_{\alpha} \rangle$ **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 $\langle vs \neq [] \rangle$]]* **and** $\langle A \in wffs_{\alpha} \rangle$ **and** $\langle B \in wffs_{\alpha} \rangle$
by *blast+*
qed
with
 $\langle C \langle p \leftarrow (\cdot^{\mathcal{Q}}_{\star} (\lambda^{\mathcal{Q}}_{\star} vs A) (map FVar vs)) \rangle \triangleright G'_1 \rangle$
and
 $\langle D \langle p \leftarrow (\cdot^{\mathcal{Q}}_{\star} (\lambda^{\mathcal{Q}}_{\star} vs B) (map FVar vs)) \rangle \triangleright G'_2 \rangle$
have $\vdash G'_1 =_o C$ **and** $\vdash G'_2 =_o D$
using *Equality-Rules(2)[OF replacement-derivability]* **and** $\langle C \in wffs_o \rangle$ **and** $\langle D \in wffs_o \rangle$
and $\langle A \preceq_p C \rangle$ **and** $\langle B \preceq_p D \rangle$ **by** *blast+*
ultimately show $\vdash (\lambda t_{\varrho\gamma}. G) \cdot (\lambda^{\mathcal{Q}}_{\star} vs A) =_o C$ **and** $\vdash (\lambda t_{\varrho\gamma}. G) \cdot (\lambda^{\mathcal{Q}}_{\star} vs B) =_o D$
using *Equality-Rules(3)* **by** *blast+*
qed
ultimately show *?thesis*
proof –
from §2 **and** $\vdash (\lambda t_{\varrho\gamma}. G) \cdot (\lambda^{\mathcal{Q}}_{\star} vs A) =_o C$ **have**
 $\vdash (\forall^{\mathcal{Q}}_{\star} vs (A =_{\alpha} B)) \supset^{\mathcal{Q}} (C \equiv^{\mathcal{Q}} (\lambda t_{\varrho\gamma}. G) \cdot (\lambda^{\mathcal{Q}}_{\star} vs B))$
by (*rule rule-R[where $p = [\rangle, \langle, \rangle]$]*) *force+*
from this and $\vdash (\lambda t_{\varrho\gamma}. G) \cdot (\lambda^{\mathcal{Q}}_{\star} vs B) =_o D$ **show** *?thesis*
by (*rule rule-R[where $p = [\rangle, \rangle]$]*) *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 $?\vartheta = \{(\mathfrak{r}, o) \mapsto C, (\mathfrak{h}, o) \mapsto D, (\mathfrak{z}, o) \mapsto E, (\mathfrak{h}, o) \mapsto H\}$

have

is-tautology

$((\mathfrak{h}_o \supset^{\mathcal{Q}} \mathfrak{r}_o) \supset^{\mathcal{Q}} ((\mathfrak{h}_o \supset^{\mathcal{Q}} \mathfrak{z}_o) \supset^{\mathcal{Q}} ((\mathfrak{z}_o \supset^{\mathcal{Q}} (\mathfrak{r}_o \equiv^{\mathcal{Q}} \mathfrak{h}_o)) \supset^{\mathcal{Q}} (\mathfrak{h}_o \supset^{\mathcal{Q}} \mathfrak{h}_o))))$

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 $?\vartheta$ $((\mathfrak{h}_o \supset^{\mathcal{Q}} \mathfrak{r}_o) \supset^{\mathcal{Q}} ((\mathfrak{h}_o \supset^{\mathcal{Q}} \mathfrak{z}_o) \supset^{\mathcal{Q}} ((\mathfrak{z}_o \supset^{\mathcal{Q}} (\mathfrak{r}_o \equiv^{\mathcal{Q}} \mathfrak{h}_o)) \supset^{\mathcal{Q}} (\mathfrak{h}_o \supset^{\mathcal{Q}} \mathfrak{h}_o))))$

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

syntax

-HypDer :: *form* \Rightarrow *form set* \Rightarrow *form* \Rightarrow *bool* (*-*, *-* \vdash - [50, 50, 50] 50)

translations

$\mathcal{H}, H \vdash P \mapsto \mathcal{H} \cup \{H\} \vdash P$

theorem *thm-5240*:

assumes *finite* \mathcal{H}

and $\mathcal{H}, H \vdash P$

shows $\mathcal{H} \vdash H \supset^{\mathcal{Q}} P$

proof –

from $\langle \mathcal{H}, H \vdash P \rangle$ 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 $i' < \text{length } \mathcal{S}_2$ for i'

using that **proof** (*induction* i' rule: *less-induct*)

case (*less* i')

let $?R = \mathcal{S}_2 ! i'$

from *less.prem*s(1) and * have *is-hyps* \mathcal{H}

by *fastforce*

from *less.prem*s and * have $?R \in \text{wffs}_o$

using *elem-of-proof-is-wffo*[*simplified*] by *auto*

from *less.prem*s 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*) $?R \in \text{lset } \mathcal{S}_1$

| (*rule-R'*) $\exists j k p. \{j, k\} \subseteq \{0..<i'\} \wedge \text{is-rule-R'-app } (\mathcal{H} \cup \{H\}) p ?R (\mathcal{S}_2 ! j) (\mathcal{S}_2 ! k)$

by *force*

then show *?case*

proof *cases*

case *hyp*

then show *?thesis*

proof (*cases* $?R = H$)

case *True*

with $\langle ?R \in \text{wffs}_o \rangle$ have *is-tautologous* $(H \supset^{\mathcal{Q}} ?R)$

using *implication-reflexivity-is-tautologous* by (*simp only*:)

with $\langle \text{is-hyps } \mathcal{H} \rangle$ show *?thesis*

by (*rule rule-P(2)*)

next

case *False*

with *hyp* have $?R \in \mathcal{H}$

by *blast*

with $\langle \text{is-hyps } \mathcal{H} \rangle$ have $\mathcal{H} \vdash ?R$

by (*intro dv-hyp*)

moreover from *less.prem*s(1) and * have *is-tautologous* $(?R \supset^{\mathcal{Q}} (H \supset^{\mathcal{Q}} ?R))$

using *principle-of-simplification-is-tautologous*[*OF* $\langle ?R \in \text{wffs}_o \rangle$] **by force**
moreover from $\langle ?R \in \text{wffs}_o \rangle$ **have** *is-hyps* $\{?R\}$
by simp
ultimately show *?thesis*
using *rule-P(1)*[**where** $\mathcal{G} = \{?R\}$ **and** $hs = [?R]$, *OF* $\langle \text{is-hyps } \mathcal{H} \rangle$] **by simp**
qed
next
case *seq*
then have $S_1 \neq []$
by force
moreover from *less.premis(1)* **and** $*$ **have** *is-proof* S_1
by fastforce
moreover from *seq* **obtain** i'' **where** $i'' < \text{length } S_1$ **and** $?R = S_1 ! i''$
by (*metis in-set-conv-nth*)
ultimately have *is-theorem* $?R$
using *proof-form-is-theorem* **by fastforce**
with $\langle \text{is-hyps } \mathcal{H} \rangle$ **have** $\mathcal{H} \vdash ?R$
by (*intro dv-thm*)
moreover from $\langle ?R \in \text{wffs}_o \rangle$ **and** *less.premis(1)* **and** $*$ **have** *is-tautologous* $(?R \supset^Q (H \supset^Q$
 $?R))$
using *principle-of-simplification-is-tautologous* **by force**
moreover from $\langle ?R \in \text{wffs}_o \rangle$ **have** *is-hyps* $\{?R\}$
by simp
ultimately show *?thesis*
using *rule-P(1)*[**where** $\mathcal{G} = \{?R\}$ **and** $hs = [?R]$, *OF* $\langle \text{is-hyps } \mathcal{H} \rangle$] **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* $(\mathcal{H} \cup \{H\}) p ?R (S_2 ! j) (S_2 ! k)$
by auto
then obtain A **and** B **and** C **and** α **where** $C = S_2 ! j$ **and** $S_2 ! k = A =_\alpha B$
by fastforce
with $\langle \{j, k\} \subseteq \{0..<i'\} \rangle$ **have** $\mathcal{H} \vdash H \supset^Q C$ **and** $\mathcal{H} \vdash H \supset^Q (A =_\alpha B)$
using *less.IH* **and** *less.premis(1)* **by** (*simp, force*)
define S **where** $S \equiv$
 $\{(x, \beta) \mid x \beta p' E. \text{strict-prefix } p' p \wedge \lambda x \beta. E \preceq_{p'} C \wedge (x, \beta) \in \text{free-vars } (A =_\alpha B)\}$
with $\langle C = S_2 ! j \rangle$ **and** $\langle S_2 ! k = A =_\alpha B \rangle$ **have** $\forall v \in S. v \notin \text{free-vars } (\mathcal{H} \cup \{H\})$
using *rule-R'-app* **by fastforce**
moreover have $S \subseteq \text{free-vars } (A =_\alpha B)$
unfolding *S-def* **by blast**
then have *finite* S
by (*fact rev-finite-subset[OF free-vars-form-finiteness]*)
then obtain vs **where** $\text{lset } vs = S$
using *finite-list* **by blast**
ultimately have $\mathcal{H} \vdash H \supset^Q \forall \star vs (A =_\alpha B)$
using *generalized-prop-5237[OF \langle \text{is-hyps } \mathcal{H} \rangle \langle \mathcal{H} \vdash H \supset^Q (A =_\alpha B) \rangle]* **by simp**
moreover have *rule-R-app: is-rule-R-app* $p ?R (S_2 ! j) (S_2 ! k)$
using *rule-R'-app* **by fastforce**
with *S-def* **and** $\langle \text{lset } vs = S \rangle$ **have** $\vdash \forall \star vs (A =_\alpha B) \supset^Q (C \equiv^Q ?R)$

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

lemmas *Deduction-Theorem = thm-5240*

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

corollary *generalized-deduction-theorem*:

assumes *finite* \mathcal{H} **and** *finite* \mathcal{H}'
and $\mathcal{H} \cup \mathcal{H}' \vdash P$
and $\text{lset } hs = \mathcal{H}'$
shows $\mathcal{H} \vdash hs \supset^{\mathcal{Q}_*} P$
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 $\langle \text{lset } (hs @ [H]) = \mathcal{H}' \rangle$ **have** $H \in \mathcal{H}'$
by *fastforce*

from $\langle \text{lset } (hs @ [H]) = \mathcal{H}' \rangle$ **obtain** \mathcal{H}'' **where** $\mathcal{H}'' \cup \{H\} = \mathcal{H}'$ **and** $\mathcal{H}'' = \text{lset } hs$
by *simp*
from $\langle \mathcal{H}'' \cup \{H\} = \mathcal{H}' \rangle$ **and** $\langle \mathcal{H} \cup \mathcal{H}' \vdash P \rangle$ **have** $\mathcal{H} \cup \mathcal{H}'' \cup \{H\} \vdash P$
by *fastforce*
with $\langle \text{finite } \mathcal{H} \rangle$ **and** $\langle \text{finite } \mathcal{H}' \rangle$ **and** $\langle \mathcal{H}'' = \text{lset } hs \rangle$ **have** $\mathcal{H} \cup \mathcal{H}'' \vdash H \supset^{\mathcal{Q}} P$
using *Deduction-Theorem* **by** *simp*
with $\langle \mathcal{H}'' = \text{lset } hs \rangle$ **and** $\langle \text{finite } \mathcal{H} \rangle$ **have** $\mathcal{H} \vdash \text{foldr } (\supset^{\mathcal{Q}}) \text{ } hs \text{ } (H \supset^{\mathcal{Q}} P)$
using *snoc.IH* **by** *fastforce*
moreover **have** $(hs @ [H]) \supset^{\mathcal{Q}_*} P = hs \supset^{\mathcal{Q}_*} (H \supset^{\mathcal{Q}} P)$
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** $\text{lset } hs = \mathcal{H}$ **and** $hs \neq []$
using *hyp-derivability-implies-hyp-proof-existence*[*OF* *assms*(2)] **unfolding** *is-hyp-proof-of-def*
by (*metis empty-set finite-list*)
with *assms*(2) **have** $\vdash hs \supset^{\mathcal{Q}_*} A$
using *generalized-deduction-theorem* **by** *force*
moreover from $\langle \text{lset } hs = \mathcal{H} \rangle$ **and** *assms*(1,3) **have** $\mathcal{G} \vdash H$ **if** $H \in \text{lset } hs$ **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 $A \in \text{wffs}_\alpha$ **and** $B \in \text{wffs}_o$
and $\mathcal{H} \vdash \mathbf{S} \{(x, \alpha) \mapsto A\} B$
and *is-free-for* $A \ (x, \alpha) \ B$
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 $\mathcal{H} \vdash \forall x_\alpha. \sim^{\mathcal{Q}} B \supset^{\mathcal{Q}} \sim^{\mathcal{Q}} \mathbf{S} \{(x, \alpha) \mapsto A\} B$ (**is** $\langle \mathcal{H} \vdash ?C \supset^{\mathcal{Q}} \sim^{\mathcal{Q}} ?D \rangle$)
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* $((?C \supset^Q \sim^Q ?D) \supset^Q (?D \supset^Q \sim^Q ?C))$
proof –
have $?C \in \text{wffs}_o$ **and** $?D \in \text{wffs}_o$
using *assms(2)* **and** *hyp-derivable-form-is-wffso*[*OF assms(3)*] **by** *auto*
then show *?thesis*
by (*fact pseudo-modus-tollens-is-tautologous*)
qed
moreover from *assms(3)* **and** $\langle \mathcal{H} \vdash ?C \supset^Q \sim^Q ?D \rangle$ **have** *is-hyps* $\{?C \supset^Q \sim^Q ?D, ?D\}$
using *hyp-derivable-form-is-wffso* **by** *force*
ultimately show *?thesis*
unfolding *exists-def* **using** *assms(3)*
and *rule-P(1)*
[
where $\mathcal{G} = \{?C \supset^Q \sim^Q ?D, ?D\}$ **and** $hs = [?C \supset^Q \sim^Q ?D, ?D]$ **and** $B = \sim^Q ?C$,
OF $\langle \text{is-hyps } \mathcal{H} \rangle$
]
by *simp*
qed

lemmas $\exists Gen = \text{prop-5242}$

6.43 Proposition 5243 (Comprehension Theorem)

context

begin

private lemma *prop-5243-aux*:

assumes $\bullet^Q_* B (\text{map } FVar \text{ } vs) \in \text{wffs}_\gamma$

and $B \in \text{wffs}_\beta$

and $k < \text{length } vs$

shows $\beta \neq \text{var-type } (vs ! k)$

proof –

from *assms(1)* **obtain** ts

where $\text{length } ts = \text{length } (\text{map } FVar \text{ } vs)$

and *: $\forall k < \text{length } (\text{map } FVar \text{ } vs). (\text{map } FVar \text{ } vs) ! k \in \text{wffs}_{ts ! k}$

and $B \in \text{wffs}_{\text{foldr } (\rightarrow) \text{ } ts \ \gamma}$

using *wffs-from-generalized-app* **by** *force*

have $\beta = \text{foldr } (\rightarrow) \text{ } ts \ \gamma$

by (*fact wff-has-unique-type*[*OF assms(2)*] $\langle B \in \text{wffs}_{\text{foldr } (\rightarrow) \text{ } ts \ \gamma} \rangle$)

have $ts = \text{map } \text{var-type } vs$

proof –

have $\text{length } ts = \text{length } (\text{map } \text{var-type } vs)$

by (*simp add*: $\langle \text{length } ts = \text{length } (\text{map } FVar \text{ } vs) \rangle$)

moreover have $\forall k < \text{length } ts. ts ! k = (\text{map } \text{var-type } vs) ! k$

proof (*intro allI impI*)

fix k

assume $k < \text{length } ts$

with * **have** $(\text{map } FVar \text{ } vs) ! k \in \text{wffs}_{ts ! k}$

by (*simp add*: $\langle \text{length } ts = \text{length } (\text{map } FVar \text{ } vs) \rangle$)

with $\langle k < \text{length } ts \rangle$ **and** $\langle \text{length } ts = \text{length } (\text{map } \text{var-type } vs) \rangle$
show $ts ! k = (\text{map } \text{var-type } vs) ! k$
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 $\langle \beta = \text{foldr } (\rightarrow) ts \ \gamma \rangle$ **and** *assms(3)* **show** *?thesis*
using *fun-type-atoms-neq-fun-type* **by** (*metis* *length-map* *nth-map*)
qed

proposition *prop-5243*:

assumes $B \in \text{wffs}_\beta$
and $\gamma = \text{foldr } (\rightarrow) (\text{map } \text{var-type } vs) \ \beta$
and $(u, \gamma) \notin \text{free-vars } B$
shows $\vdash \exists u_\gamma. \forall \cdot^{\mathcal{Q}_*} vs ((\cdot^{\mathcal{Q}_*} u_\gamma (\text{map } FVar \ vs)) =_\beta B)$
proof (*cases* $vs = []$)
case *True*
with *assms(2)* **have** $\gamma = \beta$
by *simp*
from *assms(1)* **have** $u_\beta =_\beta B \in \text{wffs}_o$
by *blast*
moreover **have** $\vdash B =_\beta B$
by (*fact* *prop-5200*[*OF* *assms(1)*])
then **have** $\vdash \mathbf{S} \{(u, \beta) \mapsto B\} (u_\beta =_\beta B)$
using *free-var-singleton-substitution-neutrality*[*OF* *assms(3)*] **unfolding** $\langle \gamma = \beta \rangle$ **by** *simp*
moreover from *assms(3)*[*unfolded* $\langle \gamma = \beta \rangle$] **have** *is-free-for* $B (u, \beta) (u_\beta =_\beta B)$
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* $\exists \text{Gen}$ [*OF* *assms(1)*])
with $\langle \gamma = \beta \rangle$ **and** *True* **show** *?thesis*
by *simp*

next

case *False*
let $?v = \{(u, \gamma) \mapsto \lambda \cdot^{\mathcal{Q}_*} vs \ B\}$
from *assms(2)* **have** $*$: $(u, \gamma) \neq v$ **if** $v \in \text{lset } vs$ **for** v
using *that* **and** *fun-type-atoms-neq-fun-type* **by** (*metis* *in-set-conv-nth* *length-map* *nth-map* *snd-conv*)
from *False* **and** *assms(1)* **have** $\vdash \cdot^{\mathcal{Q}_*} (\lambda \cdot^{\mathcal{Q}_*} vs \ B) (\text{map } FVar \ vs) =_\beta B$
by (*fact* *prop-5208*)
then **have** $\vdash \forall \cdot^{\mathcal{Q}_*} vs (\cdot^{\mathcal{Q}_*} (\lambda \cdot^{\mathcal{Q}_*} vs \ B) (\text{map } FVar \ vs) =_\beta B)$
using *generalized-Gen* **by** *simp*
moreover
have $\mathbf{S} ?v (\forall \cdot^{\mathcal{Q}_*} vs ((\cdot^{\mathcal{Q}_*} u_\gamma (\text{map } FVar \ vs)) =_\beta B)) = \forall \cdot^{\mathcal{Q}_*} vs (\cdot^{\mathcal{Q}_*} (\lambda \cdot^{\mathcal{Q}_*} vs \ B) (\text{map } FVar \ vs) =_\beta B)$
 $B)$

proof –

from $*$ **have** $**$: $\text{map } (\lambda A. \mathbf{S} \{(u, \gamma) \mapsto B\} \ A) (\text{map } FVar \ vs) = \text{map } FVar \ vs$ **for** B
by (*induction* vs) *fastforce+*

from $*$ **have**

$\mathbf{S} ?v (\forall \cdot^{\mathcal{Q}_*} vs ((\cdot^{\mathcal{Q}_*} u_\gamma (\text{map } FVar \ vs)) =_\beta B)) = \forall \cdot^{\mathcal{Q}_*} vs (\mathbf{S} ?v ((\cdot^{\mathcal{Q}_*} u_\gamma (\text{map } FVar \ vs)) =_\beta B))$

B)
using *generalized-forall-substitution by force*
also have $\dots = \forall^{\mathcal{Q}_*} vs ((\mathbf{S} \ ?\vartheta (\bullet^{\mathcal{Q}_*} u_\gamma (\text{map } FVar \text{ vs}))) =_\beta \mathbf{S} \{(u, \gamma) \mapsto \lambda^{\mathcal{Q}_*} vs B\} B)$
by *simp*
also from *assms(3)* **have** $\dots = \forall^{\mathcal{Q}_*} vs ((\mathbf{S} \ ?\vartheta (\bullet^{\mathcal{Q}_*} u_\gamma (\text{map } FVar \text{ vs}))) =_\beta B)$
using *free-var-singleton-substitution-neutrality by simp*
also have $\dots = \forall^{\mathcal{Q}_*} vs (\bullet^{\mathcal{Q}_*} \mathbf{S} \ ?\vartheta (u_\gamma) (\text{map } (\lambda A. \mathbf{S} \ ?\vartheta A) (\text{map } FVar \text{ vs}))) =_\beta B)$
using *generalized-app-substitution by simp*
also have $\dots = \forall^{\mathcal{Q}_*} vs (\bullet^{\mathcal{Q}_*} (\lambda^{\mathcal{Q}_*} vs B) (\text{map } (\lambda A. \mathbf{S} \ ?\vartheta A) (\text{map } FVar \text{ vs}))) =_\beta B)$
by *simp*
also from **** have** $\dots = \forall^{\mathcal{Q}_*} vs (\bullet^{\mathcal{Q}_*} (\lambda^{\mathcal{Q}_*} vs B) (\text{map } FVar \text{ vs})) =_\beta B)$
by *presburger*
finally show *?thesis* .
qed
ultimately have $\vdash \mathbf{S} \ ?\vartheta (\forall^{\mathcal{Q}_*} vs (\bullet^{\mathcal{Q}_*} u_\gamma (\text{map } FVar \text{ vs})) =_\beta B)$
by *simp*
moreover from *assms(3)* **have** *is-free-for* $(\lambda^{\mathcal{Q}_*} vs B) (u, \gamma) (\forall^{\mathcal{Q}_*} vs (\bullet^{\mathcal{Q}_*} u_\gamma (\text{map } FVar \text{ vs})) =_\beta B)$
 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 $\lambda^{\mathcal{Q}_*} vs B \in \text{wffs}_\gamma$ **and** $\forall^{\mathcal{Q}_*} vs (\bullet^{\mathcal{Q}_*} u_\gamma (\text{map } FVar \text{ vs})) =_\beta B \in \text{wffs}_o$
proof –
have $FVar (vs ! k) \in \text{wffs}_{\text{var-type}} (vs ! k)$ **if** $k < \text{length } vs$ **for** k
using *that and surj-pair[of vs ! k] by fastforce*
with *assms(2)* **have** $\bullet^{\mathcal{Q}_*} u_\gamma (\text{map } FVar \text{ vs}) \in \text{wffs}_\beta$
using *generalized-app-wff[where ts = map var-type vs] by force*
with *assms(1)* **show** $\forall^{\mathcal{Q}_*} vs (\bullet^{\mathcal{Q}_*} u_\gamma (\text{map } FVar \text{ vs})) =_\beta B \in \text{wffs}_o$
by *(auto simp only:)*
qed *(use assms(1,2) in blast)*
ultimately show *?thesis*
using $\exists \text{ Gen}$ **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 $A \in \text{wffs}_o$ **and** $B \in \text{wffs}_o$
and $(x, \alpha) \notin \text{free-vars } A$
shows $\vdash \forall x_\alpha. (B \supset^{\mathcal{Q}} A) \supset^{\mathcal{Q}} (\exists x_\alpha. B \supset^{\mathcal{Q}} A)$
proof –
have $B \supset^{\mathcal{Q}} A \in \text{wffs}_o$
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 $\langle B \supset^\mathcal{Q} A \in \text{wffs}_o \rangle$ by *blast*
 ultimately have $\S 1: \{\forall x_\alpha. (B \supset^\mathcal{Q} A)\} \vdash \forall x_\alpha. (B \supset^\mathcal{Q} A) \supset^\mathcal{Q} (B \supset^\mathcal{Q} A)$
 by (*fact derivability-implies-hyp-derivability*)
 have $\S 2: \{\forall x_\alpha. (B \supset^\mathcal{Q} A)\} \vdash \forall x_\alpha. (B \supset^\mathcal{Q} A)$
 using $\langle B \supset^\mathcal{Q} A \in \text{wffs}_o \rangle$ by (*blast intro: dv-hyp*)
 have $\S 3: \{\forall x_\alpha. (B \supset^\mathcal{Q} A)\} \vdash \sim^\mathcal{Q} A \supset^\mathcal{Q} \sim^\mathcal{Q} B$
 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* $([C \supset^\mathcal{Q} (B \supset^\mathcal{Q} A), C] \supset^\mathcal{Q}_* (\sim^\mathcal{Q} A \supset^\mathcal{Q} \sim^\mathcal{Q} B))$ if $C \in \text{wffs}_o$ for C
 proof –
 let $\text{?}\vartheta = \{(\mathfrak{x}, o) \mapsto A, (\mathfrak{y}, o) \mapsto B, (\mathfrak{z}, o) \mapsto C\}$
 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* $\text{?}\vartheta$
 using *assms(1,2)* and that by *auto*
 moreover have $[C \supset^\mathcal{Q} (B \supset^\mathcal{Q} A), C] \supset^\mathcal{Q}_* (\sim^\mathcal{Q} A \supset^\mathcal{Q} \sim^\mathcal{Q} B) = \mathbf{S} \text{?}\vartheta \text{?}A$
 by *simp*
 ultimately show *?thesis*
 by *blast*
 qed
 then show *is-tautologous* $([\forall x_\alpha. (B \supset^\mathcal{Q} A) \supset^\mathcal{Q} (B \supset^\mathcal{Q} A), \forall x_\alpha. (B \supset^\mathcal{Q} A)] \supset^\mathcal{Q}_* (\sim^\mathcal{Q} A \supset^\mathcal{Q} \sim^\mathcal{Q} B))$
 using $\langle B \supset^\mathcal{Q} A \in \text{wffs}_o \rangle$ and *forall-wff* by *simp*
 qed (use $\S 1 \ \S 2$ *is-hyps* $\{\forall x_\alpha. (B \supset^\mathcal{Q} A)\}$ *hyp-derivable-form-is-wffso*[*OF* $\S 1$] *in force*)+
 have $\S 4: \{\forall x_\alpha. (B \supset^\mathcal{Q} A)\} \vdash \sim^\mathcal{Q} A \supset^\mathcal{Q} \forall x_\alpha. \sim^\mathcal{Q} B$
 using *prop-5237*[*OF* *is-hyps* $\{\forall x_\alpha. (B \supset^\mathcal{Q} A)\}$ $\S 3$] and *assms(3)* by *auto*
 have $\S 5: \{\forall x_\alpha. (B \supset^\mathcal{Q} A)\} \vdash \exists x_\alpha. B \supset^\mathcal{Q} A$
 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* $([\sim^\mathcal{Q} A \supset^\mathcal{Q} C] \supset^\mathcal{Q}_* (\sim^\mathcal{Q} C \supset^\mathcal{Q} A))$ if $C \in \text{wffs}_o$ for C
 proof –
 let $\text{?}\vartheta = \{(\mathfrak{x}, o) \mapsto A, (\mathfrak{y}, o) \mapsto C\}$
 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* $\text{?}\vartheta$
 using *assms(1)* and that by *auto*
 moreover have $[\sim^\mathcal{Q} A \supset^\mathcal{Q} C] \supset^\mathcal{Q}_* (\sim^\mathcal{Q} C \supset^\mathcal{Q} A) = \mathbf{S} \text{?}\vartheta \text{?}A$
 by *simp*
 ultimately show *?thesis*
 by *blast*
 qed
 then show *is-tautologous* $([\sim^\mathcal{Q} A \supset^\mathcal{Q} \forall x_\alpha. \sim^\mathcal{Q} B] \supset^\mathcal{Q}_* (\sim^\mathcal{Q} \forall x_\alpha. \sim^\mathcal{Q} B \supset^\mathcal{Q} A))$
 using *forall-wff*[*OF neg-wff*[*OF assms(2)*]] by (*simp only*):

qed (use §4 $\langle is-hyps \{\forall x_\alpha. (B \supset^\mathcal{Q} A)\} \rangle$ 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 $\langle \mathcal{H} \vdash B \supset^\mathcal{Q} A \rangle$, THEN *wffs-from-imp-op(1)*]
)

with *assms(2)* and $\langle is-hyps \mathcal{H} \rangle$ 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 $\langle B \in wffs_o \rangle$] and $\langle is-hyps \mathcal{H} \rangle$

by (*meson Un-subset-iff empty-subsetI finite.simps finite-Un inf-sup-ord(3) insert-subsetI*)

moreover from $\langle is-hyps \mathcal{H} \rangle$ and $\langle B \in wffs_o \rangle$ 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* $(y_\alpha) (x, \alpha) B$

shows *is-free-for* $(x_\alpha) (y, \alpha) \mathbf{S} \{(x, \alpha) \mapsto y_\alpha\} B$

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

```

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

proposition *prop-5245*:

assumes $\mathcal{H} \vdash \exists x_\alpha. B$
 and $\mathcal{H}, \mathbf{S} \{(x, \alpha) \mapsto y_\alpha\} B \vdash A$
 and *is-free-for* $(y_\alpha) (x, \alpha) B$
 and $(y, \alpha) \notin \text{free-vars } (\mathcal{H} \cup \{\exists x_\alpha. B, A\})$
 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** $\mathcal{H}, \exists y_\alpha. \mathbf{S} \{(x, \alpha) \mapsto y_\alpha\} B \vdash A$
 using \exists -*Rule* **by** *simp*
then have $\ast: \mathcal{H} \vdash (\exists y_\alpha. \mathbf{S} \{(x, \alpha) \mapsto y_\alpha\} B) \supset^Q A$ (**is** $\langle \vdash ?F \rangle$)
 using *Deduction-Theorem* **and** $\langle \text{is-hyps } \mathcal{H} \rangle$ **by** *blast*
then have $\mathcal{H} \vdash \exists x_\alpha. B \supset^Q A$
proof (*cases* $x = y$)
case *True*
with \ast **show** *?thesis*
 using *identity-singleton-substitution-neutrality* **by** *force*

next

case *False*
from *assms*(4) **have** $(y, \alpha) \notin \text{free-vars } (\exists x_\alpha. B)$
 using *free-vars-in-all-vars* **by** *auto*
have $\sim^Q \mathbf{S} \{(x, \alpha) \mapsto y_\alpha\} B \in \text{wffs}_0$

```

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, α) ∉ free-vars (∼Q S {(x, α) ↦ yα} B)
  using free-var-in-renaming-substitution by simp
moreover have is-free-for (xα) (y, α) (∼Q S {(x, α) ↦ yα} B)
  by (intro is-free-for-in-neg prop-5245-aux[OF False ⟨(y, α) ∉ free-vars (∃ xα. B)⟩ assms(3)])
moreover from assms(3,4) have S {(y, α) ↦ xα} S {(x, α) ↦ yα} B = B
  using identity-singleton-substitution-neutrality and renaming-substitution-composability
  by force
ultimately have ⊢ (λyα. ∼Q S {(x, α) ↦ yα} B) =α→o (λxα. ∼Q B)
  using α[where A = ∼Q S {(x, α) ↦ yα} 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 ⇒ V ⇒ V (infixr ↦ 60) where

A ↦ B ≡ VPi A (λ. B)

notation app (infixl · 300)

syntax

-vlambda :: ptrn ⇒ V ⇒ (V ⇒ V) ⇒ V ((3λ:- ./ -) [0, 0, 3] 3)

translations

λx : A. f ⇒ CONST VLambda A (λx. f)

lemma vlamba-extensionality:

assumes $\bigwedge x. x \in \text{elts } A \implies f x = g x$

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

7.1 Frames

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

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

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

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

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

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

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

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

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

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

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

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

lemma *identity-relation-is-domain-respecting*:

shows $q_\alpha \in \text{elts } (\mathcal{D} \alpha \mapsto \mathcal{D} \alpha \mapsto \mathcal{D} o)$

using *VPi-I* **and** *one-element-function-is-domain-respecting* **by** *simp*

lemma *q-is-equality*:

assumes $\{x, y\} \subseteq \text{elts } (\mathcal{D} \alpha)$

shows $(q_\alpha) \cdot x \cdot y = \mathbf{T} \longleftrightarrow x = y$

unfolding *identity-relation-def*

using *assms* **and** *injD[OF bool-to-V-injectivity]* **by** *fastforce*

Unique member selector:

definition *is-unique-member-selector* :: $V \Rightarrow \text{bool}$ **where**

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

Assignment:

definition *is-assignment* :: $(\text{var} \Rightarrow V) \Rightarrow \text{bool}$ **where**

[iff]: *is-assignment* $\varphi \longleftrightarrow (\forall x \alpha. \varphi(x, \alpha) \in \text{elts } (\mathcal{D} \alpha))$

end

abbreviation *one-element-function-in* ($\{-\}_\cdot$ [901, 0, 0] 900) **where**

$\{x\}_\alpha^{\mathcal{D}} \equiv \text{frame.one-element-function } \mathcal{D} x \alpha$

abbreviation *identity-relation-in* (q_\cdot [0, 0] 100) **where**

$q_\alpha^{\mathcal{D}} \equiv \text{frame.identity-relation } \mathcal{D} \alpha$

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

definition *is-variant-of* :: $(\text{var} \Rightarrow V) \Rightarrow \text{var} \Rightarrow (\text{var} \Rightarrow V) \Rightarrow \text{bool}$ ($-\sim_v-$ [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 $\mathcal{J} :: \text{con} \Rightarrow V$

assumes *Q-denotation*: $\forall \alpha. \mathcal{J} (Q\text{-constant-of-type } \alpha) = q_\alpha$

and *i-denotation*: *is-unique-member-selector* (\mathcal{J} *iota-constant*)

and *non-logical-constant-denotation*: $\forall c \alpha. \neg \text{is-logical-constant } (c, \alpha) \longrightarrow \mathcal{J}(c, \alpha) \in \text{elts } (\mathcal{D} \alpha)$

begin

Wff denotation function:

definition *is-wff-denotation-function* :: ((*var* \Rightarrow *V*) \Rightarrow *form* \Rightarrow *V*) \Rightarrow *bool* **where**
[iff]: *is-wff-denotation-function* $\mathcal{V} \longleftrightarrow$
(
$$\begin{aligned} & \forall \varphi. \textit{is-assignment } \varphi \longrightarrow \\ & (\forall A \alpha. A \in \textit{wffs}_\alpha \longrightarrow \mathcal{V} \varphi A \in \textit{elts } (\mathcal{D} \alpha)) \wedge \text{--- closure condition, see note in page 186} \\ & (\forall x \alpha. \mathcal{V} \varphi (x_\alpha) = \varphi (x, \alpha)) \wedge \\ & (\forall c \alpha. \mathcal{V} \varphi (\{c\}_\alpha) = \mathcal{J} (c, \alpha)) \wedge \\ & (\forall A B \alpha \beta. A \in \textit{wffs}_{\beta \rightarrow \alpha} \wedge B \in \textit{wffs}_\beta \longrightarrow \mathcal{V} \varphi (A \cdot B) = (\mathcal{V} \varphi A) \cdot (\mathcal{V} \varphi B)) \wedge \\ & (\forall x B \alpha \beta. B \in \textit{wffs}_\beta \longrightarrow \mathcal{V} \varphi (\lambda x_\alpha. B) = (\lambda z : \mathcal{D} \alpha. \mathcal{V} (\varphi((x, \alpha) := z)) B)) \end{aligned}$$
)

lemma *wff-denotation-function-is-domain-respecting*:

assumes *is-wff-denotation-function* \mathcal{V}
and $A \in \textit{wffs}_\alpha$
and *is-assignment* φ
shows $\mathcal{V} \varphi A \in \textit{elts } (\mathcal{D} \alpha)$
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 $A \in \textit{wffs}_{\beta \rightarrow \alpha}$
and $B \in \textit{wffs}_\beta$
shows $\mathcal{V} \varphi (A \cdot B) = \mathcal{V} \varphi A \cdot \mathcal{V} \varphi B$


```

using assms by blast

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

lemma wff-denotation-function-is-uniquely-determined:
  assumes is-wff-denotation-function  $\mathcal{V}$ 
  and is-wff-denotation-function  $\mathcal{V}'$ 
  and is-assignment  $\varphi$ 
  and  $A \in \text{wffs}$ 
  shows  $\mathcal{V} \varphi A = \mathcal{V}' \varphi A$ 
proof –
  obtain  $\alpha$  where  $A \in \text{wffs}_\alpha$ 
    using assms(4) by blast
  then show ?thesis
  using assms(3) proof (induction  $A$  arbitrary:  $\varphi$ )
    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  $\beta A \alpha x$ )
      have is-assignment  $(\varphi((x, \alpha) := z))$  if  $z \in \text{elts} (\mathcal{D} \alpha)$  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  $z \in \text{elts} (\mathcal{D} \alpha)$  for  $z$ 
        using abs-is-wff.IH and that by blast
      have  $\mathcal{V} \varphi (\lambda x_\alpha. A) = (\lambda z : \mathcal{D} \alpha. \mathcal{V} (\varphi((x, \alpha) := z)) A)$ 
        by (fact wff-abs-denotation[OF assms(1) abs-is-wff.prems abs-is-wff.hyps])
      also have  $\dots = (\lambda z : \mathcal{D} \alpha. \mathcal{V}' (\varphi((x, \alpha) := z)) A)$ 
        using  $*$  and vlambda-extensionality by fastforce
      also have  $\dots = \mathcal{V}' \varphi (\lambda x_\alpha. A)$ 
        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* \Rightarrow *V*) \times (*con* \Rightarrow *V*) \times ((*var* \Rightarrow *V*) \Rightarrow *form* \Rightarrow *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 $\mathcal{V} :: (\text{var} \Rightarrow V) \Rightarrow \text{form} \Rightarrow V$
assumes \mathcal{V} -*is-wff-denotation-function*: *is-wff-denotation-function* \mathcal{V}
begin

lemma *mixed-beta-conversion*:
assumes *is-assignment* φ
and $y \in \text{elts } (\mathcal{D} \alpha)$
and $B \in \text{wffs}_\beta$
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 $\mathcal{V} \varphi (\wedge_{o \rightarrow o \rightarrow o}) \in \text{elts } (\mathcal{D} (o \rightarrow o \rightarrow o))$
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 $\{x, y\} \subseteq \text{elts } (\mathcal{D} o)$
shows $\mathcal{V} \varphi (\wedge_{o \rightarrow o \rightarrow o}) \cdot x \cdot y \in \text{elts } (\mathcal{D} o)$
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 $\mathcal{V} \varphi (\supset_{o \rightarrow o \rightarrow o}) \in \text{elts } (\mathcal{D} (o \rightarrow o \rightarrow o))$
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 $\{x, y\} \subseteq \text{elts } (\mathcal{D} o)$
shows $\mathcal{V} \varphi (\supset_{o \rightarrow o \rightarrow o}) \cdot x \cdot y \in \text{elts } (\mathcal{D} o)$
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* \Rightarrow *bool* **where**
is-general-model $\mathcal{M} \equiv \text{case } \mathcal{M} \text{ of } (\mathcal{D}, \mathcal{J}, \mathcal{V}) \Rightarrow \text{general-model } \mathcal{D} \mathcal{J} \mathcal{V}$

7.4 Standard models

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

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 $A \mathcal{M} \equiv \exists \varphi. \varphi \rightsquigarrow_M \mathcal{M} \wedge \mathcal{M} \models_{\varphi} A$

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

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

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

abbreviation *is-true-sentence-in* **where**
is-true-sentence-in $A \mathcal{M} \equiv \textit{is-sentence } A \wedge \mathcal{M} \models_{\textit{undefined}} A$ — assignments are not meaningful

abbreviation *is-false-sentence-in* **where**
is-false-sentence-in $A \mathcal{M} \equiv \textit{is-sentence } A \wedge \neg \mathcal{M} \models_{\textit{undefined}} A$ — 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** \leftrightarrow 60)

8.1 Proposition 5400

proposition (in *general-model*) *prop-5400*:

assumes $A \in \text{wffs}_\alpha$

and $\varphi \rightsquigarrow \mathcal{D}$ and $\psi \rightsquigarrow \mathcal{D}$

and $\forall v \in \text{free-vars } A. \varphi v = \psi v$

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* αx)

have $(x, \alpha) \in \text{free-vars } (x_\alpha)$

by *simp*

from *assms(1)* and *var-is-wff.prem(1)* have $\mathcal{V} \varphi (x_\alpha) = \varphi (x, \alpha)$

using *\mathcal{V} -is-wff-denotation-function* by *fastforce*

also from $\langle (x, \alpha) \in \text{free-vars } (x_\alpha) \rangle$ and *var-is-wff.prem(3)* have $\dots = \psi (x, \alpha)$

by (*simp only*.)

also from *assms(1)* and *var-is-wff.prem(2)* have $\dots = \mathcal{V} \psi (x_\alpha)$

using *\mathcal{V} -is-wff-denotation-function* by *fastforce*

finally show *?case* .

next

case (*con-is-wff* αc)

from *assms(1)* and *con-is-wff.prem(1)* have $\mathcal{V} \varphi (\llbracket c \rrbracket_\alpha) = \mathcal{J} (c, \alpha)$

using *\mathcal{V} -is-wff-denotation-function* by *fastforce*

also from *assms(1)* and *con-is-wff.prem(2)* have $\dots = \mathcal{V} \psi (\llbracket c \rrbracket_\alpha)$

using *\mathcal{V} -is-wff-denotation-function* by *fastforce*

finally show *?case* .

next

case (*app-is-wff* $\alpha \beta A B$)

have $\text{free-vars } (A \cdot B) = \text{free-vars } A \cup \text{free-vars } B$

by *simp*

with *app-is-wff.prem(3)*

have $\forall v \in \text{free-vars } A. \varphi v = \psi v$ and $\forall v \in \text{free-vars } B. \varphi v = \psi v$

by *blast+*

with *app-is-wff.IH* and *app-is-wff.prem(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.prem(1)* **and** *app-is-wff.hyps* **have** $\mathcal{V} \varphi (A \cdot B) = \mathcal{V} \varphi A \cdot \mathcal{V} \varphi B$
using *\mathcal{V} -is-wff-denotation-function* **by** *fastforce*
also from $\langle \mathcal{V} \varphi A = \mathcal{V} \psi A \rangle$ **and** $\langle \mathcal{V} \varphi B = \mathcal{V} \psi B \rangle$ **have** $\dots = \mathcal{V} \psi A \cdot \mathcal{V} \psi B$
by (*simp only*:)
also from *assms(1)* **and** *app-is-wff.prem(2)* **and** *app-is-wff.hyps* **have** $\dots = \mathcal{V} \psi (A \cdot B)$
using *\mathcal{V} -is-wff-denotation-function* **by** *fastforce*
finally show *?case* .
next
case (*abs-is-wff* $\beta A \alpha x$)
have *free-vars* $(\lambda x_{\alpha}. A) = \text{free-vars } A - \{(x, \alpha)\}$
by *simp*
with *abs-is-wff.prem(3)* **have** $\forall v \in \text{free-vars } A. v \neq (x, \alpha) \longrightarrow \varphi v = \psi v$
by *blast*
then have $\forall v \in \text{free-vars } A. (\varphi((x, \alpha) := z)) v = (\psi((x, \alpha) := z)) v$ **if** $z \in \text{elts } (\mathcal{D} \alpha)$ **for** z
by *simp*
moreover from *abs-is-wff.prem(1,2)*
have $\forall x' \alpha'. (\varphi((x, \alpha) := z)) (x', \alpha') \in \text{elts } (\mathcal{D} \alpha')$
and $\forall x' \alpha'. (\psi((x, \alpha) := z)) (x', \alpha') \in \text{elts } (\mathcal{D} \alpha')$
if $z \in \text{elts } (\mathcal{D} \alpha)$ **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** $z \in \text{elts } (\mathcal{D} \alpha)$ **for** z
using *abs-is-wff.IH* **and** *that by simp*
from *assms(1)* **and** *abs-is-wff.prem(1)* **and** *abs-is-wff.hyps*
have $\mathcal{V} \varphi (\lambda x_{\alpha}. A) = (\lambda z : \mathcal{D} \alpha. \mathcal{V} (\varphi((x, \alpha) := z)) A)$
using *wff-abs-denotation[OF \mathcal{V} -is-wff-denotation-function]* **by** *simp*
also from *\mathcal{V} - φ - ψ -eq* **have** $\dots = (\lambda z : \mathcal{D} \alpha. \mathcal{V} (\psi((x, \alpha) := z)) A)$
by (*fact vlambd-extensionality*)
also from *assms(1)* **and** *abs-is-wff.hyps* **have** $\dots = \mathcal{V} \psi (\lambda x_{\alpha}. A)$
using *wff-abs-denotation[OF \mathcal{V} -is-wff-denotation-function abs-is-wff.prem(2)]* **by** *simp*
finally show *?case* .
qed
qed

corollary (*in general-model*) *closed-wff-is-meaningful-regardless-of-assignment*:
assumes *is-closed-wff-of-type* $A \alpha$
and $\varphi \rightsquigarrow \mathcal{D}$ **and** $\psi \rightsquigarrow \mathcal{D}$
shows $\mathcal{V} \varphi A = \mathcal{V} \psi A$
using *assms* **and** *prop-5400* **by** *blast*

8.2 Proposition 5401

lemma (*in general-model*) *prop-5401-a*:
assumes $\varphi \rightsquigarrow \mathcal{D}$
and $A \in \text{wffs}_{\alpha}$
and $B \in \text{wffs}_{\beta}$
shows $\mathcal{V} \varphi ((\lambda x_{\alpha}. B) \cdot A) = \mathcal{V} (\varphi((x, \alpha) := \mathcal{V} \varphi A)) B$
proof –
from *assms(2,3)* **have** $\lambda x_{\alpha}. B \in \text{wffs}_{\alpha \rightarrow \beta}$
by *blast*

with *assms*(1,2) **have** $\mathcal{V} \varphi ((\lambda x_\alpha. B) \cdot A) = \mathcal{V} \varphi (\lambda x_\alpha. B) \cdot \mathcal{V} \varphi A$
using *\mathcal{V} -is-wff-denotation-function* **by** *blast*
also from *assms*(1,3) **have** $\dots = \text{app } (\lambda z : \mathcal{D} \alpha. \mathcal{V} (\varphi((x, \alpha) := z)) B) (\mathcal{V} \varphi A)$
using *wff-abs-denotation*[*OF* *\mathcal{V} -is-wff-denotation-function*] **by** *simp*
also from *assms*(1,2) **have** $\dots = \mathcal{V} (\varphi((x, \alpha) := \mathcal{V} \varphi A)) B$
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 $A \in \text{wffs}_\alpha$
and $B \in \text{wffs}_\alpha$
shows $\mathcal{V} \varphi (A =_\alpha B) = \mathbf{T} \longleftrightarrow \mathcal{V} \varphi A = \mathcal{V} \varphi B$

proof –

from *assms* **have** $\{\mathcal{V} \varphi A, \mathcal{V} \varphi B\} \subseteq \text{elts } (\mathcal{D} \alpha)$
using *\mathcal{V} -is-wff-denotation-function* **by** *auto*
have $\mathcal{V} \varphi (A =_\alpha B) = \mathcal{V} \varphi (Q_\alpha \cdot A \cdot B)$
by *simp*
also from *assms* **have** $\dots = \mathcal{V} \varphi (Q_\alpha \cdot A) \cdot \mathcal{V} \varphi B$
using *\mathcal{V} -is-wff-denotation-function* **by** *blast*
also from *assms* **have** $\dots = \mathcal{V} \varphi (Q_\alpha) \cdot \mathcal{V} \varphi A \cdot \mathcal{V} \varphi B$
using *Q-wff* **and** *wff-app-denotation*[*OF* *\mathcal{V} -is-wff-denotation-function*] **by** *fastforce*
also from *assms*(1) **have** $\dots = (q_\alpha) \cdot \mathcal{V} \varphi A \cdot \mathcal{V} \varphi B$
using *Q-denotation* **and** *\mathcal{V} -is-wff-denotation-function* **by** *fastforce*
also from $\langle \{\mathcal{V} \varphi A, \mathcal{V} \varphi B\} \subseteq \text{elts } (\mathcal{D} \alpha) \rangle$ **have** $\dots = \mathbf{T} \longleftrightarrow \mathcal{V} \varphi A = \mathcal{V} \varphi B$
using *q-is-equality* **by** *simp*
finally show *?thesis* .
qed

corollary (in *general-model*) *prop-5401-b'*:

assumes $\varphi \rightsquigarrow \mathcal{D}$
and $A \in \text{wffs}_o$
and $B \in \text{wffs}_o$
shows $\mathcal{V} \varphi (A \equiv^Q B) = \mathbf{T} \longleftrightarrow \mathcal{V} \varphi A = \mathcal{V} \varphi B$
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 $Q_o \in \text{wffs}_{o \rightarrow o \rightarrow o}$
by *blast*
moreover have $\mathcal{V} \varphi T_o = \mathcal{V} \varphi (Q_o =_{o \rightarrow o \rightarrow o} Q_o)$
unfolding *true-def* ..
ultimately have $\dots = \mathbf{T} \longleftrightarrow \mathcal{V} \varphi (Q_o) = \mathcal{V} \varphi (Q_o)$
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 $\lambda \mathfrak{x}_o. T_o \in \text{wffs}_{o \rightarrow o}$ and $\lambda \mathfrak{x}_o. \mathfrak{x}_o \in \text{wffs}_{o \rightarrow o}$

by *blast+*

moreover have $\mathcal{V} \varphi F_o = \mathcal{V} \varphi (\lambda \mathfrak{x}_o. T_o =_{o \rightarrow o} \lambda \mathfrak{x}_o. \mathfrak{x}_o)$

unfolding *false-def* ..

ultimately have $\mathcal{V} \varphi F_o = \mathbf{T} \iff \mathcal{V} \varphi (\lambda \mathfrak{x}_o. T_o) = \mathcal{V} \varphi (\lambda \mathfrak{x}_o. \mathfrak{x}_o)$

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 $\mathcal{V} \varphi (\lambda \mathfrak{x}_o. T_o) = (\lambda z : \mathcal{D} o. \mathbf{T})$

proof –

from *assms* have *T-denotation*: $\mathcal{V} (\varphi((\mathfrak{x}, o) := z)) T_o = \mathbf{T}$ if $z \in \text{elts}(\mathcal{D} o)$ for z

using *prop-5401-c* and that by *simp*

from *assms* have $\mathcal{V} \varphi (\lambda \mathfrak{x}_o. T_o) = (\lambda z : \mathcal{D} o. \mathcal{V} (\varphi((\mathfrak{x}, o) := z)) T_o)$

using *wff-abs-denotation[OF V-is-wff-denotation-function]* by *blast*

also from *assms* and *T-denotation* have $\dots = (\lambda z : \mathcal{D} o. \mathbf{T})$

using *vlambda-extensionality* by *fastforce*

finally show *?thesis* .

qed

moreover have $\mathcal{V} \varphi (\lambda \mathfrak{x}_o. \mathfrak{x}_o) = (\lambda z : \mathcal{D} o. z)$

proof –

from *assms* have *x-denotation*: $\mathcal{V} (\varphi((\mathfrak{x}, o) := z)) (\mathfrak{x}_o) = z$ if $z \in \text{elts}(\mathcal{D} o)$ for z

using that and *V-is-wff-denotation-function* by *auto*

from *assms* have $\mathcal{V} \varphi (\lambda \mathfrak{x}_o. \mathfrak{x}_o) = (\lambda z : \mathcal{D} o. \mathcal{V} (\varphi((\mathfrak{x}, o) := z)) (\mathfrak{x}_o))$

using *wff-abs-denotation[OF V-is-wff-denotation-function]* by *blast*

also from *x-denotation* have $\dots = (\lambda z : (\mathcal{D} o). z)$

using *vlambda-extensionality* by *fastforce*

finally show *?thesis* .

qed

moreover have $(\lambda z : \mathcal{D} o. \mathbf{T}) \neq (\lambda z : \mathcal{D} o. z)$

proof –

from *assms(1)* have $(\lambda z : \mathcal{D} o. \mathbf{T}) \cdot \mathbf{F} = \mathbf{T}$

by (*simp add: truth-values-domain-def*)

moreover from *assms(1)* have $(\lambda z : \mathcal{D} o. z) \cdot \mathbf{F} = \mathbf{F}$

by (*simp add: truth-values-domain-def*)

ultimately show *?thesis*

by (*auto simp add: inj-eq*)

qed

ultimately show *?thesis*

by *simp*

qed

moreover from *assms* have $\mathcal{V} \varphi F_o \in \text{elts}(\mathcal{D} o)$

using *false-wff* and *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 $\{x, y\} \subseteq \text{elts}(\mathcal{D} \ o)$

shows $\mathcal{V} \ \varphi \ (\wedge_{o \rightarrow o \rightarrow o}) \cdot x \cdot y = (\text{if } x = \mathbf{T} \wedge y = \mathbf{T} \text{ then } \mathbf{T} \text{ else } \mathbf{F})$

proof –

let $?B_{leq} = \lambda \mathbf{g}_{o \rightarrow o \rightarrow o} \cdot \mathbf{g}_{o \rightarrow o \rightarrow o} \cdot T_o \cdot T_o$

let $?B_{req} = \lambda \mathbf{g}_{o \rightarrow o \rightarrow o} \cdot \mathbf{g}_{o \rightarrow o \rightarrow o} \cdot \mathbf{r}_o \cdot \eta_o$

let $?B_{eq} = ?B_{leq} =_{(o \rightarrow o \rightarrow o) \rightarrow o} ?B_{req}$

let $?B_{\eta} = \lambda \eta_o \cdot ?B_{eq}$

let $?B_{\mathbf{r}} = \lambda \mathbf{r}_o \cdot ?B_{\eta}$

let $? \varphi' = \varphi((\mathbf{r}, o) := x, (\eta, o) := y)$

let $? \varphi'' = \lambda g \cdot ? \varphi'((\mathbf{g}, o \rightarrow o \rightarrow o) := g)$

have $\mathbf{g}_{o \rightarrow o \rightarrow o} \cdot T_o \in \text{wffs}_{o \rightarrow o}$

by *blast*

have $\mathbf{g}_{o \rightarrow o \rightarrow o} \cdot T_o \cdot T_o \in \text{wffs}_o$ **and** $\mathbf{g}_{o \rightarrow o \rightarrow o} \cdot \mathbf{r}_o \cdot \eta_o \in \text{wffs}_o$

by *blast+*

have $?B_{leq} \in \text{wffs}_{(o \rightarrow o \rightarrow o) \rightarrow o}$ **and** $?B_{req} \in \text{wffs}_{(o \rightarrow o \rightarrow o) \rightarrow o}$

by *blast+*

then have $?B_{eq} \in \text{wffs}_o$ **and** $?B_{\eta} \in \text{wffs}_{o \rightarrow o}$ **and** $?B_{\mathbf{r}} \in \text{wffs}_{o \rightarrow o \rightarrow o}$

by *blast+*

have $\mathcal{V} \ \varphi \ (\wedge_{o \rightarrow o \rightarrow o}) \cdot x \cdot y = \mathcal{V} \ \varphi \ ?B_{\mathbf{r}} \cdot x \cdot y$

by *simp*

also from *assms* **and** $\langle ?B_{\eta} \in \text{wffs}_{o \rightarrow o} \rangle$ **have** $\dots = \mathcal{V} \ (\varphi((\mathbf{r}, o) := x)) \ ?B_{\eta} \cdot y$

using *mixed-beta-conversion* by *simp*

also from *assms* **and** $\langle ?B_{eq} \in \text{wffs}_o \rangle$ **have** $\dots = \mathcal{V} \ ? \varphi' \ ?B_{eq}$

using *mixed-beta-conversion* by *simp*

finally have $\mathcal{V} \ \varphi \ (\wedge_{o \rightarrow o \rightarrow o}) \cdot x \cdot y = \mathbf{T} \iff \mathcal{V} \ ? \varphi' \ ?B_{leq} = \mathcal{V} \ ? \varphi' \ ?B_{req}$

using *assms* **and** $\langle ?B_{leq} \in \text{wffs}_{(o \rightarrow o \rightarrow o) \rightarrow o} \rangle$ **and** $\langle ?B_{req} \in \text{wffs}_{(o \rightarrow o \rightarrow o) \rightarrow o} \rangle$ **and** *prop-5401-b*

by *simp*

also have $\dots \iff (\lambda g : \mathcal{D} \ (o \rightarrow o \rightarrow o) \cdot g \cdot \mathbf{T} \cdot \mathbf{T}) = (\lambda g : \mathcal{D} \ (o \rightarrow o \rightarrow o) \cdot g \cdot x \cdot y)$

proof –

have *leq*: $\mathcal{V} \ ? \varphi' \ ?B_{leq} = (\lambda g : \mathcal{D} \ (o \rightarrow o \rightarrow o) \cdot g \cdot \mathbf{T} \cdot \mathbf{T})$

and *req*: $\mathcal{V} \ ? \varphi' \ ?B_{req} = (\lambda g : \mathcal{D} \ (o \rightarrow o \rightarrow o) \cdot g \cdot x \cdot y)$

proof –

from *assms(1,2)* **have** *is-assg- φ''* : $? \varphi'' \ g \rightsquigarrow \mathcal{D}$ **if** $g \in \text{elts}(\mathcal{D} \ (o \rightarrow o \rightarrow o))$ **for** g

using *that* by *auto*

have *side-eq-denotation*:

$\mathcal{V} \ ? \varphi' \ (\lambda \mathbf{g}_{o \rightarrow o \rightarrow o} \cdot \mathbf{g}_{o \rightarrow o \rightarrow o} \cdot A \cdot B) = (\lambda g : \mathcal{D} \ (o \rightarrow o \rightarrow o) \cdot g \cdot \mathcal{V} \ (? \varphi'' \ g) \ A \cdot \mathcal{V} \ (? \varphi'' \ g) \ B)$

if $A \in \text{wffs}_o$ **and** $B \in \text{wffs}_o$ **for** A **and** B

proof –

from *that* **have** $\mathbf{g}_{o \rightarrow o \rightarrow o} \cdot A \cdot B \in \text{wffs}_o$

by *blast*

have $\mathcal{V} \ (? \varphi'' \ g) \ (\mathbf{g}_{o \rightarrow o \rightarrow o} \cdot A \cdot B) = g \cdot \mathcal{V} \ (? \varphi'' \ g) \ A \cdot \mathcal{V} \ (? \varphi'' \ g) \ B$

if $g \in \text{elts}(\mathcal{D} \ (o \rightarrow o \rightarrow o))$ **for** g

proof –

from $\langle A \in \text{wffs}_o \rangle$ **have** $\mathbf{g}_{o \rightarrow o \rightarrow o} \cdot A \in \text{wffs}_{o \rightarrow o}$

by *blast*
 with *that* and *is-assg-φ''* and $\langle B \in \text{wffs}_o \rangle$ have
 $\mathcal{V} (?φ'' g) (\mathfrak{g}_{o \rightarrow o \rightarrow o} \cdot A \cdot B) = \mathcal{V} (?φ'' g) (\mathfrak{g}_{o \rightarrow o \rightarrow o} \cdot A) \cdot \mathcal{V} (?φ'' g) B$
 using *wff-app-denotation*[*OF* *V-is-wff-denotation-function*] by *simp*
 also from *that* and $\langle A \in \text{wffs}_o \rangle$ and *is-assg-φ''* have
 $\dots = \mathcal{V} (?φ'' g) (\mathfrak{g}_{o \rightarrow o \rightarrow o}) \cdot \mathcal{V} (?φ'' g) A \cdot \mathcal{V} (?φ'' g) B$
 by (*metis* *V-is-wff-denotation-function* *wff-app-denotation* *wffs-of-type-intros(1)*)
 finally show *?thesis*
 using *that* and *is-assg-φ''* and *V-is-wff-denotation-function* by *auto*
 qed
 moreover from *assms* have *is-assignment* *?φ'*
 by *auto*
 with $\langle \mathfrak{g}_{o \rightarrow o \rightarrow o} \cdot A \cdot B \in \text{wffs}_o \rangle$ have
 $\mathcal{V} ?φ' (\lambda \mathfrak{g}_{o \rightarrow o \rightarrow o}. \mathfrak{g}_{o \rightarrow o \rightarrow o} \cdot A \cdot B) = (\lambda g : \mathcal{D} (o \rightarrow o \rightarrow o). \mathcal{V} (?φ'' g) (\mathfrak{g}_{o \rightarrow o \rightarrow o} \cdot A \cdot B))$
 using *wff-abs-denotation*[*OF* *V-is-wff-denotation-function*] by *simp*
 ultimately show *?thesis*
 using *vlambda-extensionality* by *fastforce*
 qed
 — Proof of *leg*:
 show $\mathcal{V} ?φ' ?B_{leg} = (\lambda g : \mathcal{D} (o \rightarrow o \rightarrow o). g \cdot \mathbf{T} \cdot \mathbf{T})$
 proof –
 have $\mathcal{V} (?φ'' g) T_o = \mathbf{T}$ if $g \in \text{elts} (\mathcal{D} (o \rightarrow o \rightarrow o))$ 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 $\mathcal{V} ?φ' ?B_{req} = (\lambda g : \mathcal{D} (o \rightarrow o \rightarrow o). g \cdot x \cdot y)$
 proof –
 from *is-assg-φ''* have $\mathcal{V} (?φ'' g) (\mathfrak{x}_o) = x$ and $\mathcal{V} (?φ'' g) (\mathfrak{y}_o) = y$
 if $g \in \text{elts} (\mathcal{D} (o \rightarrow o \rightarrow o))$ for g
 using *that* and *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 $\dots \longleftrightarrow (\forall g \in \text{elts} (\mathcal{D} (o \rightarrow o \rightarrow o)). g \cdot \mathbf{T} \cdot \mathbf{T} = g \cdot x \cdot y)$
 using *vlambda-extensionality* and *VLambda-eq-D2* by *fastforce*
 finally have *and-eqv*:
 $\mathcal{V} \varphi (\wedge_{o \rightarrow o \rightarrow o}) \cdot x \cdot y = \mathbf{T} \longleftrightarrow (\forall g \in \text{elts} (\mathcal{D} (o \rightarrow o \rightarrow o)). g \cdot \mathbf{T} \cdot \mathbf{T} = g \cdot x \cdot y)$
 by *blast*
 then show *?thesis*
 proof –
 from *assms(1,2)* have *is-assg-1*: $\varphi(\mathfrak{x}, o) := \mathbf{T} \rightsquigarrow \mathcal{D}$
 by (*simp add: truth-values-domain-def*)
 then have *is-assg-2*: $\varphi(\mathfrak{x}, o) := \mathbf{T}, (\mathfrak{y}, o) := \mathbf{T} \rightsquigarrow \mathcal{D}$

unfolding *is-assignment-def* **by** (*metis fun-upd-apply prod.sel(2)*)
from *assms* **consider** (a) $x = \mathbf{T} \wedge y = \mathbf{T} \mid$ (b) $x \neq \mathbf{T} \mid$ (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** $g \in \text{elts } (\mathcal{D} (o \rightarrow o \rightarrow o))$ **for** g
by *simp*
with *a* **and** *and-eqv* **show** *?thesis*
by *simp*
next
case *b*
let $?g\text{-witness} = \lambda \mathfrak{r}_o. \lambda \eta_o. \mathfrak{r}_o$
have $\lambda \eta_o. \mathfrak{r}_o \in \text{wffs}_{o \rightarrow o}$
by *blast*
then have *is-closed-wff-of-type ?g-witness (o → o → o)*
by *force*
moreover from *assms* **have** *is-assg-φ': ?φ' ∼ D*
by *simp*
ultimately have $\mathcal{V} \varphi ?g\text{-witness} \cdot \mathbf{T} \cdot \mathbf{T} = \mathcal{V} ?\varphi' ?g\text{-witness} \cdot \mathbf{T} \cdot \mathbf{T}$
using *assms(1)* **and** *closed-wff-is-meaningful-regardless-of-assignment* **by** *metis*
also from *assms* **and** $\langle \lambda \eta_o. \mathfrak{r}_o \in \text{wffs}_{o \rightarrow o} \rangle$ **have**
 $\mathcal{V} ?\varphi' ?g\text{-witness} \cdot \mathbf{T} \cdot \mathbf{T} = \mathcal{V} (?\varphi'((\mathfrak{r}, o) := \mathbf{T})) (\lambda \eta_o. \mathfrak{r}_o) \cdot \mathbf{T}$
using *mixed-beta-conversion* **and** *truth-values-domain-def* **by** *auto*
also from *assms(1)* **and** $\langle \lambda \eta_o. \mathfrak{r}_o \in \text{wffs}_{o \rightarrow o} \rangle$ **and** *is-assg-1* **and** *calculation* **have**
 $\dots = \mathcal{V} (?\varphi'((\mathfrak{r}, o) := \mathbf{T}, (\eta, o) := \mathbf{T})) (\mathfrak{r}_o)$
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 $\dots = \mathbf{T}$
using *is-assg-2* **and** *V-is-wff-denotation-function* **by** *fastforce*
finally have $\mathcal{V} \varphi ?g\text{-witness} \cdot \mathbf{T} \cdot \mathbf{T} = \mathbf{T}$.
with *b* **have** $\mathcal{V} \varphi ?g\text{-witness} \cdot \mathbf{T} \cdot \mathbf{T} \neq x$
by *blast*
moreover have $x = \mathcal{V} \varphi ?g\text{-witness} \cdot x \cdot y$
proof –
from *is-assg-φ'* **have** $x = \mathcal{V} ?\varphi' (\mathfrak{r}_o)$
using *V-is-wff-denotation-function* **by** *auto*
also from *assms(2)* **and** *is-assg-φ'* **have** $\dots = \mathcal{V} ?\varphi' (\lambda \eta_o. \mathfrak{r}_o) \cdot y$
using *wffs-of-type-intros(1)* [**where** $x = \mathfrak{r}$ **and** $\alpha = o$]
by (*simp add: mixed-beta-conversion V-is-wff-denotation-function*)
also from *assms(2)* **have** $\dots = \mathcal{V} ?\varphi' ?g\text{-witness} \cdot x \cdot y$
using *is-assg-φ'* **and** $\langle \lambda \eta_o. \mathfrak{r}_o \in \text{wffs}_{o \rightarrow o} \rangle$
by (*simp add: mixed-beta-conversion fun-upd-twist*)
also from *assms(1,2)* **have** $\dots = \mathcal{V} \varphi ?g\text{-witness} \cdot x \cdot y$
using *is-assg-φ'* **and** $\langle \text{is-closed-wff-of-type ?g-witness (o → o → o)} \rangle$
and *closed-wff-is-meaningful-regardless-of-assignment* **by** *metis*
finally show *?thesis* .
qed
moreover from *assms(1,2)* **have** $\mathcal{V} \varphi ?g\text{-witness} \in \text{elts } (\mathcal{D} (o \rightarrow o \rightarrow o))$

using $\langle \text{is-closed-wff-of-type } ?g\text{-witness } (o \rightarrow o \rightarrow o) \rangle$ and \mathcal{V} -is-wff-denotation-function by simp
 ultimately have $\exists g \in \text{elts } (\mathcal{D} (o \rightarrow o \rightarrow o)). g \cdot \mathbf{T} \cdot \mathbf{T} \neq g \cdot x \cdot y$
 by auto
 moreover from *assms* have $\mathcal{V} \varphi (\wedge_{o \rightarrow o \rightarrow o}) \cdot x \cdot y \in \text{elts } (\mathcal{D} o)$
 by (rule *fully-applied-conj-fun-is-domain-respecting*)
 ultimately have $\mathcal{V} \varphi (\wedge_{o \rightarrow o \rightarrow o}) \cdot x \cdot y = \mathbf{F}$
 using *and-eqv* and *truth-values-domain-def* by *fastforce*
 with *b* show *?thesis*
 by *simp*
 next
 case *c*
 let $?g\text{-witness} = \lambda \mathbf{x}_o. \lambda \eta_o. \eta_o$
 have $\lambda \eta_o. \eta_o \in \text{wffs}_{o \rightarrow o}$
 by *blast*
 then have *is-closed-wff-of-type ?g-witness (o → o → o)*
 by *force*
 moreover from *assms(1,2)* have *is-assg-φ'*: $?φ' \rightsquigarrow \mathcal{D}$
 by *simp*
 ultimately have $\mathcal{V} \varphi ?g\text{-witness} \cdot \mathbf{T} \cdot \mathbf{T} = \mathcal{V} ?φ' ?g\text{-witness} \cdot \mathbf{T} \cdot \mathbf{T}$
 using *assms(1)* and *closed-wff-is-meaningful-regardless-of-assignment* by *metis*
 also from *is-assg-1* and *is-assg-φ'* have $\dots = \mathcal{V} (?φ'((\mathbf{x}, o) := \mathbf{T})) (\lambda \eta_o. \eta_o) \cdot \mathbf{T}$
 using $\langle \lambda \eta_o. \eta_o \in \text{wffs}_{o \rightarrow o} \rangle$ and *mixed-beta-conversion* and *truth-values-domain-def* by *auto*
 also from *assms(1)* and $\langle \lambda \eta_o. \eta_o \in \text{wffs}_{o \rightarrow o} \rangle$ and *is-assg-1* and *calculation* have
 $\dots = \mathcal{V} (?φ'((\mathbf{x}, o) := \mathbf{T}, (\eta, o) := \mathbf{T})) (\eta_o)$
 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 $\dots = \mathbf{T}$
 using *is-assg-2* and \mathcal{V} -is-wff-denotation-function by *force*
 finally have $\mathcal{V} \varphi ?g\text{-witness} \cdot \mathbf{T} \cdot \mathbf{T} = \mathbf{T}$.
 with *c* have $\mathcal{V} \varphi ?g\text{-witness} \cdot \mathbf{T} \cdot \mathbf{T} \neq y$
 by *blast*
 moreover have $y = \mathcal{V} \varphi ?g\text{-witness} \cdot x \cdot y$
 proof –
 from *assms(2)* and *is-assg-φ'* have $y = \mathcal{V} ?φ' (\lambda \eta_o. \eta_o) \cdot y$
 using *wffs-of-type-intros(1)* [where $x = \eta$ and $\alpha = o$]
 and \mathcal{V} -is-wff-denotation-function and *mixed-beta-conversion* by *auto*
 also from *assms(2)* and $\langle \lambda \eta_o. \eta_o \in \text{wffs}_{o \rightarrow o} \rangle$ have $\dots = \mathcal{V} ?φ' ?g\text{-witness} \cdot x \cdot y$
 using *is-assg-φ'* by (*simp add: mixed-beta-conversion fun-upd-twist*)
 also from *assms(1,2)* have $\dots = \mathcal{V} \varphi ?g\text{-witness} \cdot x \cdot y$
 using *is-assg-φ'* and $\langle \text{is-closed-wff-of-type } ?g\text{-witness } (o \rightarrow o \rightarrow o) \rangle$
 and *closed-wff-is-meaningful-regardless-of-assignment* by *metis*
 finally show *?thesis* .
 qed
 moreover from *assms(1)* have $\mathcal{V} \varphi ?g\text{-witness} \in \text{elts } (\mathcal{D} (o \rightarrow o \rightarrow o))$
 using $\langle \text{is-closed-wff-of-type } ?g\text{-witness } (o \rightarrow o \rightarrow o) \rangle$ and \mathcal{V} -is-wff-denotation-function by *auto*
 ultimately have $\exists g \in \text{elts } (\mathcal{D} (o \rightarrow o \rightarrow o)). g \cdot \mathbf{T} \cdot \mathbf{T} \neq g \cdot x \cdot y$
 by *auto*
 moreover from *assms* have $\mathcal{V} \varphi (\wedge_{o \rightarrow o \rightarrow o}) \cdot x \cdot y \in \text{elts } (\mathcal{D} o)$
 by (rule *fully-applied-conj-fun-is-domain-respecting*)

ultimately have $\mathcal{V} \varphi (\wedge_{o \rightarrow o \rightarrow o}) \cdot x \cdot y = \mathbf{F}$
 using *and-eqv* and *truth-values-domain-def* by *fastforce*
 with *c* show *?thesis*
 by *simp*
 qed
 qed
 qed

corollary (in *general-model*) *prop-5401-e'*:

assumes $\varphi \rightsquigarrow \mathcal{D}$
 and $A \in \text{wffs}_o$ and $B \in \text{wffs}_o$
 shows $\mathcal{V} \varphi (A \wedge^{\mathcal{Q}} B) = \mathcal{V} \varphi A \wedge \mathcal{V} \varphi B$
 proof –
 from *assms* have $\{\mathcal{V} \varphi A, \mathcal{V} \varphi B\} \subseteq \text{elts } (\mathcal{D} \ o)$
 using *\mathcal{V} -is-wff-denotation-function* by *simp*
 from *assms*(2) have $\wedge_{o \rightarrow o \rightarrow o} \cdot A \in \text{wffs}_{o \rightarrow o}$
 by *blast*
 have $\mathcal{V} \varphi (A \wedge^{\mathcal{Q}} B) = \mathcal{V} \varphi (\wedge_{o \rightarrow o \rightarrow o} \cdot A \cdot B)$
 by *simp*
 also from *assms* have $\dots = \mathcal{V} \varphi (\wedge_{o \rightarrow o \rightarrow o} \cdot A) \cdot \mathcal{V} \varphi B$
 using *\mathcal{V} -is-wff-denotation-function* and $\langle \wedge_{o \rightarrow o \rightarrow o} \cdot A \in \text{wffs}_{o \rightarrow o} \rangle$ by *blast*
 also from *assms* have $\dots = \mathcal{V} \varphi (\wedge_{o \rightarrow o \rightarrow o}) \cdot \mathcal{V} \varphi A \cdot \mathcal{V} \varphi B$
 using *\mathcal{V} -is-wff-denotation-function* and *conj-fun-wff* by *fastforce*
 also from *assms*(1,2) have $\dots = (\text{if } \mathcal{V} \varphi A = \mathbf{T} \wedge \mathcal{V} \varphi B = \mathbf{T} \text{ then } \mathbf{T} \text{ else } \mathbf{F})$
 using $\langle \{\mathcal{V} \varphi A, \mathcal{V} \varphi B\} \subseteq \text{elts } (\mathcal{D} \ o) \rangle$ and *prop-5401-e* by *simp*
 also have $\dots = \mathcal{V} \varphi A \wedge \mathcal{V} \varphi B$
 using *truth-values-domain-def* and $\langle \{\mathcal{V} \varphi A, \mathcal{V} \varphi B\} \subseteq \text{elts } (\mathcal{D} \ o) \rangle$ by *fastforce*
 finally show *?thesis* .
 qed

lemma (in *general-model*) *prop-5401-f*:

assumes $\varphi \rightsquigarrow \mathcal{D}$
 and $\{x, y\} \subseteq \text{elts } (\mathcal{D} \ o)$
 shows $\mathcal{V} \varphi (\supset_{o \rightarrow o \rightarrow o}) \cdot x \cdot y = (\text{if } x = \mathbf{T} \wedge y = \mathbf{F} \text{ then } \mathbf{F} \text{ else } \mathbf{T})$
 proof –
 let $?\varphi' = \varphi((\mathfrak{r}, o) := x, (\mathfrak{h}, o) := y)$
 from *assms*(2) have $\{x, y\} \subseteq \text{elts } \mathbb{B}$
 unfolding *truth-values-domain-def* .
 have $(\mathfrak{r}_o \equiv^{\mathcal{Q}} \mathfrak{r}_o \wedge^{\mathcal{Q}} \mathfrak{h}_o) \in \text{wffs}_o$
 by *blast*
 then have $\lambda \mathfrak{h}_o. (\mathfrak{r}_o \equiv^{\mathcal{Q}} \mathfrak{r}_o \wedge^{\mathcal{Q}} \mathfrak{h}_o) \in \text{wffs}_{o \rightarrow o}$
 by *blast*
 from *assms* have *is-assg- φ'* : $?\varphi' \rightsquigarrow \mathcal{D}$
 by *simp*
 from *assms*(1) have $\mathcal{V} ?\varphi' (\mathfrak{r}_o) = x$ and $\mathcal{V} ?\varphi' (\mathfrak{h}_o) = y$
 using *is-assg- φ'* and *\mathcal{V} -is-wff-denotation-function* by *force+*
 have $\mathcal{V} \varphi (\supset_{o \rightarrow o \rightarrow o}) \cdot x \cdot y = \mathcal{V} \varphi (\lambda \mathfrak{r}_o. \lambda \mathfrak{h}_o. (\mathfrak{r}_o \equiv^{\mathcal{Q}} \mathfrak{r}_o \wedge^{\mathcal{Q}} \mathfrak{h}_o)) \cdot x \cdot y$
 by *simp*
 also from *assms* have $\dots = \mathcal{V} (\varphi((\mathfrak{r}, o) := x)) (\lambda \mathfrak{h}_o. (\mathfrak{r}_o \equiv^{\mathcal{Q}} \mathfrak{r}_o \wedge^{\mathcal{Q}} \mathfrak{h}_o)) \cdot y$

using $\langle \lambda \eta_o. (r_o \equiv^Q r_o \wedge^Q \eta_o) \in \text{wffs}_{o \rightarrow o} \rangle$ and *mixed-beta-conversion* by *simp*
 also from *assms* have $\dots = \mathcal{V} \ ?\varphi' (r_o \equiv^Q r_o \wedge^Q \eta_o)$
 using *mixed-beta-conversion* and $\langle r_o \equiv^Q r_o \wedge^Q \eta_o \rangle \in \text{wffs}_o$ by *simp*
 finally have $\mathcal{V} \varphi (\supset_{o \rightarrow o \rightarrow o}) \cdot x \cdot y = \mathbf{T} \longleftrightarrow \mathcal{V} \ ?\varphi' (r_o) = \mathcal{V} \ ?\varphi' (r_o \wedge^Q \eta_o)$
 using *prop-5401-b'* [*OF is-assg-φ*] and *conj-op-wff* and *wffs-of-type-intros(1)* by *simp*
 also have $\dots \longleftrightarrow x = x \wedge y$
 unfolding *prop-5401-e'* [*OF is-assg-φ'* *wffs-of-type-intros(1)* *wffs-of-type-intros(1)*]
 and $\langle \mathcal{V} \ ?\varphi' (r_o) = x \rangle$ and $\langle \mathcal{V} \ ?\varphi' (\eta_o) = y \rangle$..
 also have $\dots \longleftrightarrow x = (\text{if } x = \mathbf{T} \wedge y = \mathbf{T} \text{ then } \mathbf{T} \text{ else } \mathbf{F})$
 using $\langle \{x, y\} \subseteq \text{elts } \mathbb{B} \rangle$ by *auto*
 also have $\dots \longleftrightarrow \mathbf{T} = (\text{if } x = \mathbf{T} \wedge y = \mathbf{F} \text{ then } \mathbf{F} \text{ else } \mathbf{T})$
 using $\langle \{x, y\} \subseteq \text{elts } \mathbb{B} \rangle$ 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 $\varphi \rightsquigarrow \mathcal{D}$
 and $A \in \text{wffs}_o$ and $B \in \text{wffs}_o$
 shows $\mathcal{V} \varphi (A \supset^Q B) = \mathcal{V} \varphi A \supset \mathcal{V} \varphi B$

proof –

from *assms* have $\{\mathcal{V} \varphi A, \mathcal{V} \varphi B\} \subseteq \text{elts } (\mathcal{D} \ o)$
 using *ℳ-is-wff-denotation-function* by *simp*
 from *assms(2)* have $\supset_{o \rightarrow o \rightarrow o} \cdot A \in \text{wffs}_{o \rightarrow o}$
 by *blast*
 have $\mathcal{V} \varphi (A \supset^Q B) = \mathcal{V} \varphi (\supset_{o \rightarrow o \rightarrow o} \cdot A \cdot B)$
 by *simp*
 also from *assms(1,3)* have $\dots = \mathcal{V} \varphi (\supset_{o \rightarrow o \rightarrow o} \cdot A) \cdot \mathcal{V} \varphi B$
 using *ℳ-is-wff-denotation-function* and $\langle \supset_{o \rightarrow o \rightarrow o} \cdot A \in \text{wffs}_{o \rightarrow o} \rangle$ by *blast*
 also from *assms* have $\dots = \mathcal{V} \varphi (\supset_{o \rightarrow o \rightarrow o}) \cdot \mathcal{V} \varphi A \cdot \mathcal{V} \varphi B$
 using *ℳ-is-wff-denotation-function* and *imp-fun-wff* by *fastforce*
 also from *assms* have $\dots = (\text{if } \mathcal{V} \varphi A = \mathbf{T} \wedge \mathcal{V} \varphi B = \mathbf{F} \text{ then } \mathbf{F} \text{ else } \mathbf{T})$
 using $\langle \{\mathcal{V} \varphi A, \mathcal{V} \varphi B\} \subseteq \text{elts } (\mathcal{D} \ o) \rangle$ and *prop-5401-f* by *simp*
 also have $\dots = \mathcal{V} \varphi A \supset \mathcal{V} \varphi B$
 using *truth-values-domain-def* and $\langle \{\mathcal{V} \varphi A, \mathcal{V} \varphi B\} \subseteq \text{elts } (\mathcal{D} \ o) \rangle$ by *auto*
 finally show *?thesis* .

qed

lemma (in *general-model*) *forall-denotation*:

assumes $\varphi \rightsquigarrow \mathcal{D}$
 and $A \in \text{wffs}_o$
 shows $\mathcal{V} \varphi (\forall x_\alpha. A) = \mathbf{T} \longleftrightarrow (\forall z \in \text{elts } (\mathcal{D} \ \alpha). \mathcal{V} (\varphi((x, \alpha) := z)) A = \mathbf{T})$

proof –

from *assms(1)* have *lhs*: $\mathcal{V} \varphi (\lambda r_\alpha. T_o) \cdot z = \mathbf{T}$ if $z \in \text{elts } (\mathcal{D} \ \alpha)$ 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 $z \in \text{elts } (\mathcal{D} \ \alpha)$ for z
 using *that* by (*simp add: mixed-beta-conversion*)
 from *assms(2)* have $\lambda r_\alpha. T_o \in \text{wffs}_{\alpha \rightarrow o}$ and $\lambda x_\alpha. A \in \text{wffs}_{\alpha \rightarrow o}$

by *auto*
 have $\mathcal{V} \varphi (\forall x_\alpha. A) = \mathcal{V} \varphi (\prod \alpha \cdot (\lambda x_\alpha. A))$
 unfolding *forall-def* ..
 also have $\dots = \mathcal{V} \varphi (Q_{\alpha \rightarrow o} \cdot (\lambda \mathfrak{x}_\alpha. T_o) \cdot (\lambda x_\alpha. A))$
 unfolding *PI-def* ..
 also have $\dots = \mathcal{V} \varphi ((\lambda \mathfrak{x}_\alpha. T_o) =_{\alpha \rightarrow o} (\lambda x_\alpha. A))$
 unfolding *equality-of-type-def* ..
 finally have $\mathcal{V} \varphi (\forall x_\alpha. A) = \mathcal{V} \varphi ((\lambda \mathfrak{x}_\alpha. T_o) =_{\alpha \rightarrow o} (\lambda x_\alpha. A))$.
 moreover from *assms(1,2)* have
 $\mathcal{V} \varphi ((\lambda \mathfrak{x}_\alpha. T_o) =_{\alpha \rightarrow o} (\lambda x_\alpha. A)) = \mathbf{T} \longleftrightarrow \mathcal{V} \varphi (\lambda \mathfrak{x}_\alpha. T_o) = \mathcal{V} \varphi (\lambda x_\alpha. A)$
 using $\langle \lambda \mathfrak{x}_\alpha. T_o \in \text{wffs}_{\alpha \rightarrow o} \rangle$ and $\langle \lambda x_\alpha. A \in \text{wffs}_{\alpha \rightarrow o} \rangle$ and *prop-5401-b* by *blast*
 moreover
 have $(\mathcal{V} \varphi (\lambda \mathfrak{x}_\alpha. T_o) = \mathcal{V} \varphi (\lambda x_\alpha. A)) \longleftrightarrow (\forall z \in \text{elts} (\mathcal{D} \alpha). \mathcal{V} (\varphi((x, \alpha) := z)) A = \mathbf{T})$
 proof
 assume $\mathcal{V} \varphi (\lambda \mathfrak{x}_\alpha. T_o) = \mathcal{V} \varphi (\lambda x_\alpha. A)$
 with *lhs* and *rhs* show $\forall z \in \text{elts} (\mathcal{D} \alpha). \mathcal{V} (\varphi((x, \alpha) := z)) A = \mathbf{T}$
 by *auto*
 next
 assume $\forall z \in \text{elts} (\mathcal{D} \alpha). \mathcal{V} (\varphi((x, \alpha) := z)) A = \mathbf{T}$
 moreover from *assms* have $\mathcal{V} \varphi (\lambda \mathfrak{x}_\alpha. T_o) = (\lambda z : \mathcal{D} \alpha. \mathcal{V} (\varphi((\mathfrak{x}, \alpha) := z)) T_o)$
 using *wff-abs-denotation[OF V-is-wff-denotation-function]* by *blast*
 moreover from *assms* have $\mathcal{V} \varphi (\lambda x_\alpha. A) = (\lambda z : \mathcal{D} \alpha. \mathcal{V} (\varphi((x, \alpha) := z)) A)$
 using *wff-abs-denotation[OF V-is-wff-denotation-function]* by *blast*
 ultimately show $\mathcal{V} \varphi (\lambda \mathfrak{x}_\alpha. T_o) = \mathcal{V} \varphi (\lambda x_\alpha. A)$
 using *lhs* and *vlambda-extensionality* by *fastforce*
 qed
 ultimately show *?thesis*
 by (*simp only:*)
 qed

lemma *prop-5401-g*:

assumes *is-general-model* \mathcal{M}
 and $\varphi \rightsquigarrow_{\mathcal{M}} \mathcal{M}$
 and $A \in \text{wffs}_o$
 shows $\mathcal{M} \models_{\varphi} \forall x_\alpha. A \longleftrightarrow (\forall \psi. \psi \rightsquigarrow_{\mathcal{M}} \mathcal{M} \wedge \psi \sim_{(x, \alpha)} \varphi \longrightarrow \mathcal{M} \models_{\psi} A)$
 proof –
 obtain \mathcal{D} and \mathcal{J} and \mathcal{V} where $\mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V})$
 using *prod-cases3* by *blast*
 with *assms* have
 $\mathcal{M} \models_{\varphi} \forall x_\alpha. A$
 \longleftrightarrow
 $\forall x_\alpha. A \in \text{wffs}_o \wedge \text{is-general-model} (\mathcal{D}, \mathcal{J}, \mathcal{V}) \wedge \varphi \rightsquigarrow \mathcal{D} \wedge \mathcal{V} \varphi (\forall x_\alpha. A) = \mathbf{T}$
 by *fastforce*
 also from *assms* and $\langle \mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V}) \rangle$ have $\dots \longleftrightarrow (\forall z \in \text{elts} (\mathcal{D} \alpha). \mathcal{V} (\varphi((x, \alpha) := z)) A = \mathbf{T})$
 using *general-model.forall-denotation* by *fastforce*
 also have $\dots \longleftrightarrow (\forall \psi. \psi \rightsquigarrow \mathcal{D} \wedge \psi \sim_{(x, \alpha)} \varphi \longrightarrow \mathcal{M} \models_{\psi} A)$
 proof
 assume $*$: $\forall z \in \text{elts} (\mathcal{D} \alpha). \mathcal{V} (\varphi((x, \alpha) := z)) A = \mathbf{T}$

```

{
  fix  $\psi$ 
  assume  $\psi \rightsquigarrow \mathcal{D}$  and  $\psi \sim_{(x, \alpha)} \varphi$ 
  have  $\mathcal{V} \psi A = \mathbf{T}$ 
  proof -
    have  $\exists z \in \text{elts } (\mathcal{D} \ \alpha). \psi = \varphi((x, \alpha) := z)$ 
    proof (rule ccontr)
      assume  $\neg (\exists z \in \text{elts } (\mathcal{D} \ \alpha). \psi = \varphi((x, \alpha) := z))$ 
      with  $\langle \psi \sim_{(x, \alpha)} \varphi \rangle$  have  $\forall z \in \text{elts } (\mathcal{D} \ \alpha). \psi(x, \alpha) \neq z$ 
      by fastforce
      then have  $\psi(x, \alpha) \notin \text{elts } (\mathcal{D} \ \alpha)$ 
      by blast
      moreover from assms(1) and  $\langle \mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V}) \rangle$  and  $\langle \psi \rightsquigarrow \mathcal{D} \rangle$  have  $\psi(x, \alpha) \in \text{elts } (\mathcal{D} \ \alpha)$ 
      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  $\langle \mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V}) \rangle$  have  $\mathcal{M} \models_{\psi} A$ 
  by simp
}
then show  $\forall \psi. \psi \rightsquigarrow \mathcal{D} \wedge \psi \sim_{(x, \alpha)} \varphi \longrightarrow \mathcal{M} \models_{\psi} A$ 
by blast
next
assume *:  $\forall \psi. \psi \rightsquigarrow \mathcal{D} \wedge \psi \sim_{(x, \alpha)} \varphi \longrightarrow \mathcal{M} \models_{\psi} A$ 
show  $\forall z \in \text{elts } (\mathcal{D} \ \alpha). \mathcal{V} (\varphi((x, \alpha) := z)) A = \mathbf{T}$ 
proof
  fix  $z$ 
  assume  $z \in \text{elts } (\mathcal{D} \ \alpha)$ 
  with assms(1,2) and  $\langle \mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V}) \rangle$  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  $\langle \mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V}) \rangle$  and  $\langle \varphi((x, \alpha) := z) \rightsquigarrow \mathcal{D} \rangle$  show  $\mathcal{V} (\varphi((x, \alpha) := z)) A =$ 
T
  by simp
qed
qed
finally show ?thesis
  using  $\langle \mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V}) \rangle$ 
  by simp
qed

```

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

assumes $\varphi \rightsquigarrow \mathcal{D}$
shows $\mathcal{V} \varphi (\mathfrak{g}_{o \rightarrow o} \cdot T_o \wedge^{\mathcal{Q}} \mathfrak{g}_{o \rightarrow o} \cdot F_o \equiv^{\mathcal{Q}} \forall \mathfrak{r}_o. \mathfrak{g}_{o \rightarrow o} \cdot \mathfrak{r}_o) = \mathbf{T}$ (is $\mathcal{V} \varphi (?A \equiv^{\mathcal{Q}} ?B) = \mathbf{T}$)
proof –
let $?M = (\mathcal{D}, \mathcal{J}, \mathcal{V})$
from *assms* **have** $*$: *is-general-model* $?M \varphi \rightsquigarrow_M ?M$
using *general-model-axioms* **by** *blast+*
have $?A \equiv^{\mathcal{Q}} ?B \in \text{wffs}_o$
using *axioms.axiom-1* **and** *axioms-are-wffs-of-type-o* **by** *blast*
have *lhs*: $\mathcal{V} \varphi ?A = \varphi (\mathfrak{g}, o \rightarrow o) \cdot \mathbf{T} \wedge \varphi (\mathfrak{g}, o \rightarrow o) \cdot \mathbf{F}$
proof –
have $\mathfrak{g}_{o \rightarrow o} \cdot T_o \in \text{wffs}_o$ **and** $\mathfrak{g}_{o \rightarrow o} \cdot F_o \in \text{wffs}_o$
by *blast+*
with *assms* **have** $\mathcal{V} \varphi ?A = \mathcal{V} \varphi (\mathfrak{g}_{o \rightarrow o} \cdot T_o) \wedge \mathcal{V} \varphi (\mathfrak{g}_{o \rightarrow o} \cdot F_o)$
using *prop-5401-e'* **by** *simp*
also from *assms* **have** $\dots = \varphi (\mathfrak{g}, o \rightarrow o) \cdot \mathcal{V} \varphi (T_o) \wedge \varphi (\mathfrak{g}, o \rightarrow o) \cdot \mathcal{V} \varphi (F_o)$
using *wff-app-denotation*[*OF* *V-is-wff-denotation-function*]
and *wff-var-denotation*[*OF* *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* $\forall z \in \text{elts}(\mathcal{D} \ o). \varphi (\mathfrak{g}, o \rightarrow o) \cdot z = \mathbf{T}$)
case *True*
with *assms* **have** $\varphi (\mathfrak{g}, o \rightarrow o) \cdot \mathbf{T} = \mathbf{T}$ **and** $\varphi (\mathfrak{g}, o \rightarrow o) \cdot \mathbf{F} = \mathbf{T}$
using *truth-values-domain-def* **by** *auto*
with *lhs* **have** $\mathcal{V} \varphi ?A = \mathbf{T} \wedge \mathbf{T}$
by (*simp only:*)
also have $\dots = \mathbf{T}$
by *simp*
finally have $\mathcal{V} \varphi ?A = \mathbf{T}$.
moreover have $\mathcal{V} \varphi ?B = \mathbf{T}$
proof –
have $\mathfrak{g}_{o \rightarrow o} \cdot \mathfrak{r}_o \in \text{wffs}_o$
by *blast*
moreover
{
fix ψ
assume $\psi \rightsquigarrow \mathcal{D}$ **and** $\psi \sim_{(\mathfrak{r}, o)} \varphi$
with *assms* **have** $\mathcal{V} \psi (\mathfrak{g}_{o \rightarrow o} \cdot \mathfrak{r}_o) = \mathcal{V} \psi (\mathfrak{g}_{o \rightarrow o}) \cdot \mathcal{V} \psi (\mathfrak{r}_o)$
using *V-is-wff-denotation-function* **by** *blast*
also from $\langle \psi \rightsquigarrow \mathcal{D} \rangle$ **have** $\dots = \psi (\mathfrak{g}, o \rightarrow o) \cdot \psi (\mathfrak{r}, o)$
using *V-is-wff-denotation-function* **by** *auto*
also from $\langle \psi \sim_{(\mathfrak{r}, o)} \varphi \rangle$ **have** $\dots = \varphi (\mathfrak{g}, o \rightarrow o) \cdot \psi (\mathfrak{r}, o)$
by *simp*
also from *True* **and** $\langle \psi \rightsquigarrow \mathcal{D} \rangle$ **have** $\dots = \mathbf{T}$
by *blast*
finally have $\mathcal{V} \psi (\mathfrak{g}_{o \rightarrow o} \cdot \mathfrak{r}_o) = \mathbf{T}$.
with *assms* **and** $\langle \mathfrak{g}_{o \rightarrow o} \cdot \mathfrak{r}_o \in \text{wffs}_o \rangle$ **have** $?M \models_{\psi} \mathfrak{g}_{o \rightarrow o} \cdot \mathfrak{r}_o$

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

```

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

```

lemma axiom-1-validity:

```

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

```

using *wffs-from-equivalence* and *prop-5401-b'* by *metis*
 moreover have $\mathcal{V} \varphi (\mathfrak{h}_{\alpha \rightarrow o} \cdot \mathfrak{r}_\alpha) = \mathcal{V} \varphi (\mathfrak{h}_{\alpha \rightarrow o} \cdot \mathfrak{r}_\alpha)$
 proof –
 from *assms* and $\langle ?B \in \text{wffs}_o \rangle$ have $\mathcal{V} \varphi (\mathfrak{h}_{\alpha \rightarrow o} \cdot \mathfrak{r}_\alpha) = \mathcal{V} \varphi (\mathfrak{h}_{\alpha \rightarrow o}) \cdot \mathcal{V} \varphi (\mathfrak{r}_\alpha)$
 using *\mathcal{V} -is-wff-denotation-function* by *blast*
 also from *assms* have $\dots = \varphi (\mathfrak{h}, \alpha \rightarrow o) \cdot \varphi (\mathfrak{r}, \alpha)$
 using *\mathcal{V} -is-wff-denotation-function* by *auto*
 also from *True* have $\dots = \varphi (\mathfrak{h}, \alpha \rightarrow o) \cdot \varphi (\mathfrak{r}, \alpha)$
 by (*simp only*):
 also from *assms* have $\dots = \mathcal{V} \varphi (\mathfrak{h}_{\alpha \rightarrow o}) \cdot \mathcal{V} \varphi (\mathfrak{r}_\alpha)$
 using *\mathcal{V} -is-wff-denotation-function* by *auto*
 also from *assms* and $\langle ?B \in \text{wffs}_o \rangle$ have $\dots = \mathcal{V} \varphi (\mathfrak{h}_{\alpha \rightarrow o} \cdot \mathfrak{r}_\alpha)$
 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 $\mathcal{V} \varphi ?A = \mathbf{T} \longleftrightarrow \mathcal{V} \varphi (\mathfrak{r}_\alpha) = \mathcal{V} \varphi (\mathfrak{r}_\alpha)$
 using *prop-5401-b* by *blast*
 moreover from *False* and *assms* have $\mathcal{V} \varphi (\mathfrak{r}_\alpha) \neq \mathcal{V} \varphi (\mathfrak{r}_\alpha)$
 using *\mathcal{V} -is-wff-denotation-function* by *auto*
 ultimately have $\mathcal{V} \varphi ?A = \mathbf{F}$
 using *assms* and $\langle \{\mathcal{V} \varphi ?A, \mathcal{V} \varphi ?B\} \subseteq \text{elts } \mathbb{B} \rangle$ by *simp*
 then show *?thesis*
 using *\mathcal{V} -imp-T* by *simp*
 qed
 qed
 lemma *axiom-2-validity*:
 shows $\models (\mathfrak{r}_\alpha =_\alpha \mathfrak{r}_\alpha) \supset^\mathcal{Q} (\mathfrak{h}_{\alpha \rightarrow o} \cdot \mathfrak{r}_\alpha \equiv^\mathcal{Q} \mathfrak{h}_{\alpha \rightarrow o} \cdot \mathfrak{r}_\alpha)$ (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 $\langle \mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V}) \rangle$ 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 $\mathcal{V} \varphi ((\mathfrak{f}_{\alpha \rightarrow \beta} =_{\alpha \rightarrow \beta} \mathfrak{g}_{\alpha \rightarrow \beta}) \equiv^\mathcal{Q} \forall \mathfrak{r}_\alpha. (\mathfrak{f}_{\alpha \rightarrow \beta} \cdot \mathfrak{r}_\alpha =_\beta \mathfrak{g}_{\alpha \rightarrow \beta} \cdot \mathfrak{r}_\alpha)) = \mathbf{T}$

```

(is  $\mathcal{V} \varphi (?A \equiv^{\mathcal{Q}} ?B) = \mathbf{T}$ )
proof -
  let  $?M = (\mathcal{D}, \mathcal{J}, \mathcal{V})$ 
  from assms have *: is-general-model  $?M \varphi \rightsquigarrow_M ?M$ 
    using general-model-axioms by blast+
  have  $B^l\text{-wffo}$ :  $f_{\alpha \rightarrow \beta} \cdot \mathfrak{r}\alpha = \beta \ \mathfrak{g}_{\alpha \rightarrow \beta} \cdot \mathfrak{r}\alpha \in \text{wffs}_o$ 
    by blast
  have  $?A \equiv^{\mathcal{Q}} ?B \in \text{wffs}_o$  and  $?A \in \text{wffs}_o$  and  $?B \in \text{wffs}_o$ 
  proof -
    show  $?A \equiv^{\mathcal{Q}} ?B \in \text{wffs}_o$ 
      using axioms.axiom-3 and axioms-are-wffs-of-type-o
      by blast
    then show  $?A \in \text{wffs}_o$  and  $?B \in \text{wffs}_o$ 
      by (blast dest: wffs-from-equivalence)+
  qed
  have  $\mathcal{V} \varphi ?A = \mathcal{V} \varphi ?B$ 
  proof (cases  $\varphi (f, \alpha \rightarrow \beta) = \varphi (g, \alpha \rightarrow \beta)$ )
    case True
    have  $\mathcal{V} \varphi ?A = \mathbf{T}$ 
    proof -
      from assms have  $\mathcal{V} \varphi (f_{\alpha \rightarrow \beta}) = \varphi (f, \alpha \rightarrow \beta)$ 
        using  $\mathcal{V}$ -is-wff-denotation-function by auto
      also from True have  $\dots = \varphi (g, \alpha \rightarrow \beta)$ 
        by (simp only:)
      also from assms have  $\dots = \mathcal{V} \varphi (g_{\alpha \rightarrow \beta})$ 
        using  $\mathcal{V}$ -is-wff-denotation-function by auto
      finally have  $\mathcal{V} \varphi (f_{\alpha \rightarrow \beta}) = \mathcal{V} \varphi (g_{\alpha \rightarrow \beta}) \cdot$ 
        with assms show ?thesis
        using prop-5401-b by blast
    qed
    moreover have  $\mathcal{V} \varphi ?B = \mathbf{T}$ 
  proof -
    {
      fix  $\psi$ 
      assume  $\psi \rightsquigarrow \mathcal{D}$  and  $\psi \sim_{(\mathfrak{r}, \alpha)} \varphi$ 
      from assms and  $\langle \psi \rightsquigarrow \mathcal{D} \rangle$  have  $\mathcal{V} \psi (f_{\alpha \rightarrow \beta} \cdot \mathfrak{r}\alpha) = \mathcal{V} \psi (f_{\alpha \rightarrow \beta}) \cdot \mathcal{V} \psi (\mathfrak{r}\alpha)$ 
        using  $\mathcal{V}$ -is-wff-denotation-function by blast
      also from assms and  $\langle \psi \rightsquigarrow \mathcal{D} \rangle$  have  $\dots = \psi (f, \alpha \rightarrow \beta) \cdot \psi (\mathfrak{r}, \alpha)$ 
        using  $\mathcal{V}$ -is-wff-denotation-function by auto
      also from  $\langle \psi \sim_{(\mathfrak{r}, \alpha)} \varphi \rangle$  have  $\dots = \varphi (f, \alpha \rightarrow \beta) \cdot \psi (\mathfrak{r}, \alpha)$ 
        by simp
      also from True have  $\dots = \varphi (g, \alpha \rightarrow \beta) \cdot \psi (\mathfrak{r}, \alpha)$ 
        by (simp only:)
      also from  $\langle \psi \sim_{(\mathfrak{r}, \alpha)} \varphi \rangle$  have  $\dots = \psi (g, \alpha \rightarrow \beta) \cdot \psi (\mathfrak{r}, \alpha)$ 
        by simp
      also from assms and  $\langle \psi \rightsquigarrow \mathcal{D} \rangle$  have  $\dots = \mathcal{V} \psi (g_{\alpha \rightarrow \beta}) \cdot \mathcal{V} \psi (\mathfrak{r}\alpha)$ 
        using  $\mathcal{V}$ -is-wff-denotation-function by auto
      also from assms and  $\langle \psi \rightsquigarrow \mathcal{D} \rangle$  have  $\dots = \mathcal{V} \psi (g_{\alpha \rightarrow \beta} \cdot \mathfrak{r}\alpha)$ 
        using wff-app-denotation[OF  $\mathcal{V}$ -is-wff-denotation-function] by (metis wffs-of-type-intros(1))
    }
  }

```

finally have $\mathcal{V} \psi (f_{\alpha \rightarrow \beta} \cdot r_{\alpha}) = \mathcal{V} \psi (g_{\alpha \rightarrow \beta} \cdot r_{\alpha}) \cdot$
with B' -wffo **and** *assms* **and** $\langle \psi \rightsquigarrow \mathcal{D} \rangle$ **have** $\mathcal{V} \psi (f_{\alpha \rightarrow \beta} \cdot r_{\alpha} =_{\beta} g_{\alpha \rightarrow \beta} \cdot r_{\alpha}) = \mathbf{T}$
using *prop-5401-b* **and** *wffs-from-equality* **by** *blast*
with $*(2)$ **have** $?M \models_{\psi} f_{\alpha \rightarrow \beta} \cdot r_{\alpha} =_{\beta} g_{\alpha \rightarrow \beta} \cdot r_{\alpha}$
by *simp*
}
with $*$ **and** B' -wffo **have** $?M \models_{\varphi} ?B$
using *prop-5401-g* **by** *force*
with $*(2)$ **show** *?thesis*
by *auto*
qed
ultimately show *?thesis ..*
next
case *False*
from $*$ **have** $\varphi (f, \alpha \rightarrow \beta) \in \text{elts} (\mathcal{D} \alpha \mapsto \mathcal{D} \beta)$ **and** $\varphi (g, \alpha \rightarrow \beta) \in \text{elts} (\mathcal{D} \alpha \mapsto \mathcal{D} \beta)$
by (*simp-all add: function-domainD*)
with *False* **obtain** z **where** $z \in \text{elts} (\mathcal{D} \alpha)$ **and** $\varphi (f, \alpha \rightarrow \beta) \cdot z \neq \varphi (g, \alpha \rightarrow \beta) \cdot z$
by (*blast dest: fun-ext*)
define ψ **where** $\psi = \varphi((r, \alpha) := z)$
from $*$ **and** $\langle z \in \text{elts} (\mathcal{D} \alpha) \rangle$ **have** $\psi \rightsquigarrow \mathcal{D}$ **and** $\psi \sim_{(r, \alpha)} \varphi$
unfolding ψ -def **by** *fastforce+*
have $\mathcal{V} \psi (f_{\alpha \rightarrow \beta} \cdot r_{\alpha}) = \varphi (f, \alpha \rightarrow \beta) \cdot z$ **for** f
proof –
from $\langle \psi \rightsquigarrow \mathcal{D} \rangle$ **have** $\mathcal{V} \psi (f_{\alpha \rightarrow \beta} \cdot r_{\alpha}) = \mathcal{V} \psi (f_{\alpha \rightarrow \beta}) \cdot \mathcal{V} \psi (r_{\alpha})$
using \mathcal{V} -is-wff-denotation-function **by** *blast*
also from $\langle \psi \rightsquigarrow \mathcal{D} \rangle$ **have** $\dots = \psi (f, \alpha \rightarrow \beta) \cdot \psi (r, \alpha)$
using \mathcal{V} -is-wff-denotation-function **by** *auto*
finally show *?thesis*
unfolding ψ -def **by** *simp*
qed
then have $\mathcal{V} \psi (f_{\alpha \rightarrow \beta} \cdot r_{\alpha}) = \varphi (f, \alpha \rightarrow \beta) \cdot z$ **and** $\mathcal{V} \psi (g_{\alpha \rightarrow \beta} \cdot r_{\alpha}) = \varphi (g, \alpha \rightarrow \beta) \cdot z$
by (*simp-all only:*)
with $\langle \varphi (f, \alpha \rightarrow \beta) \cdot z \neq \varphi (g, \alpha \rightarrow \beta) \cdot z \rangle$ **have** $\mathcal{V} \psi (f_{\alpha \rightarrow \beta} \cdot r_{\alpha}) \neq \mathcal{V} \psi (g_{\alpha \rightarrow \beta} \cdot r_{\alpha})$
by *simp*
then have $\mathcal{V} \psi (f_{\alpha \rightarrow \beta} \cdot r_{\alpha} =_{\beta} g_{\alpha \rightarrow \beta} \cdot r_{\alpha}) = \mathbf{F}$
proof –
from B' -wffo **and** $\langle \psi \rightsquigarrow \mathcal{D} \rangle$ **and** $*$ **have** $\mathcal{V} \psi (f_{\alpha \rightarrow \beta} \cdot r_{\alpha} =_{\beta} g_{\alpha \rightarrow \beta} \cdot r_{\alpha}) \in \text{elts} (\mathcal{D} o)$
using \mathcal{V} -is-wff-denotation-function **by** *auto*
moreover from B' -wffo **have** $\{f_{\alpha \rightarrow \beta} \cdot r_{\alpha}, g_{\alpha \rightarrow \beta} \cdot r_{\alpha}\} \subseteq \text{wffs}_{\beta}$
by *blast*
with $\langle \psi \rightsquigarrow \mathcal{D} \rangle$ **and** $\langle \mathcal{V} \psi (f_{\alpha \rightarrow \beta} \cdot r_{\alpha}) \neq \mathcal{V} \psi (g_{\alpha \rightarrow \beta} \cdot r_{\alpha}) \rangle$ **and** B' -wffo
have $\mathcal{V} \psi (f_{\alpha \rightarrow \beta} \cdot r_{\alpha} =_{\beta} g_{\alpha \rightarrow \beta} \cdot r_{\alpha}) \neq \mathbf{T}$
using *prop-5401-b* **by** *simp*
ultimately show *?thesis*
by (*simp add: truth-values-domain-def*)
qed
with $\langle \psi \rightsquigarrow \mathcal{D} \rangle$ **have** $\neg ?M \models_{\psi} f_{\alpha \rightarrow \beta} \cdot r_{\alpha} =_{\beta} g_{\alpha \rightarrow \beta} \cdot r_{\alpha}$
by (*auto simp add: inj-eq*)
with $\langle \psi \rightsquigarrow \mathcal{D} \rangle$ **and** $\langle \psi \sim_{(r, \alpha)} \varphi \rangle$

have $\exists \psi. \psi \rightsquigarrow \mathcal{D} \wedge \psi \sim_{(\mathfrak{r}, \alpha)} \varphi \wedge \neg ?\mathcal{M} \models_{\psi} \mathfrak{f}_{\alpha \rightarrow \beta} \cdot \mathfrak{r}_{\alpha} =_{\beta} \mathfrak{g}_{\alpha \rightarrow \beta} \cdot \mathfrak{r}_{\alpha}$
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 $\langle ?B \in \text{wffs}_o \rangle$ **and** $*$ **have** $\mathcal{V} \varphi ?B \in \text{elts}(\mathcal{D} \ o)$
using *V-is-wff-denotation-function* **by** *auto*
with $\langle \neg ?\mathcal{M} \models_{\varphi} ?B \rangle$ **and** $\langle ?B \in \text{wffs}_o \rangle$ **show** *?thesis*
using *truth-values-domain-def* **by** *fastforce*
qed
moreover have $\mathcal{V} \varphi (\mathfrak{f}_{\alpha \rightarrow \beta} =_{\alpha \rightarrow \beta} \mathfrak{g}_{\alpha \rightarrow \beta}) = \mathbf{F}$
proof –
from $*$ **have** $\mathcal{V} \varphi (\mathfrak{f}_{\alpha \rightarrow \beta}) = \varphi(\mathfrak{f}, \alpha \rightarrow \beta)$ **and** $\mathcal{V} \varphi (\mathfrak{g}_{\alpha \rightarrow \beta}) = \varphi(\mathfrak{g}, \alpha \rightarrow \beta)$
using *V-is-wff-denotation-function* **by** *auto*
with *False* **have** $\mathcal{V} \varphi (\mathfrak{f}_{\alpha \rightarrow \beta}) \neq \mathcal{V} \varphi (\mathfrak{g}_{\alpha \rightarrow \beta})$
by *simp*
with $*$ **have** $\mathcal{V} \varphi (\mathfrak{f}_{\alpha \rightarrow \beta} =_{\alpha \rightarrow \beta} \mathfrak{g}_{\alpha \rightarrow \beta}) \neq \mathbf{T}$
using *prop-5401-b* **by** *blast*
moreover from $*$ **and** $\langle ?A \in \text{wffs}_o \rangle$ **have** $\mathcal{V} \varphi (\mathfrak{f}_{\alpha \rightarrow \beta} =_{\alpha \rightarrow \beta} \mathfrak{g}_{\alpha \rightarrow \beta}) \in \text{elts}(\mathcal{D} \ o)$
using *V-is-wff-denotation-function* **by** *auto*
ultimately show *?thesis*
by (*simp add: truth-values-domain-def*)
qed
ultimately show *?thesis*
by (*simp only:*)
qed
with $*$ **and** $\langle ?A \in \text{wffs}_o \rangle$ **and** $\langle ?B \in \text{wffs}_o \rangle$ **show** *?thesis*
using *prop-5401-b'* **by** *simp*
qed

lemma *axiom-3-validity*:
shows $\models (\mathfrak{f}_{\alpha \rightarrow \beta} =_{\alpha \rightarrow \beta} \mathfrak{g}_{\alpha \rightarrow \beta}) \equiv^{\mathcal{Q}} \forall \mathfrak{r}_{\alpha}. (\mathfrak{f}_{\alpha \rightarrow \beta} \cdot \mathfrak{r}_{\alpha} =_{\beta} \mathfrak{g}_{\alpha \rightarrow \beta} \cdot \mathfrak{r}_{\alpha})$ (**is** $\models ?A \equiv^{\mathcal{Q}} ?B$)
proof (*intro all 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** $\langle \mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V}) \rangle$ **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 $\varphi \rightsquigarrow \mathcal{D}$

and $A \in \text{wffs}_\alpha$
shows $\mathcal{V} \varphi ((\lambda x_\alpha. \llbracket c \rrbracket_\beta) \cdot A =_\beta \llbracket c \rrbracket_\beta) = \mathbf{T}$
proof –
from *assms(2)* **have** $(\lambda x_\alpha. \llbracket c \rrbracket_\beta) \cdot A =_\beta \llbracket c \rrbracket_\beta \in \text{wffs}_o$
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** $\mathcal{V} \varphi ((\lambda x_\alpha. \llbracket c \rrbracket_\beta) \cdot A) = \mathcal{V} (\varphi((x, \alpha) := \mathcal{V} \varphi A)) (\llbracket c \rrbracket_\beta)$
using *prop-5401-a* **by** *blast*
also have $\dots = \mathcal{V} \psi (\llbracket c \rrbracket_\beta)$
unfolding ψ -*def* ..
also from *assms* **and** ψ -*def* **have** $\dots = \mathcal{V} \varphi (\llbracket c \rrbracket_\beta)$
using \mathcal{V} -*is-wff-denotation-function* **by** *auto*
finally have $\mathcal{V} \varphi ((\lambda x_\alpha. \llbracket c \rrbracket_\beta) \cdot A) = \mathcal{V} \varphi (\llbracket c \rrbracket_\beta)$.
with *assms(1)* **and** $\langle (\lambda x_\alpha. \llbracket c \rrbracket_\beta) \cdot A =_\beta \llbracket c \rrbracket_\beta \in \text{wffs}_o \rangle$ **show** *?thesis*
using *wffs-from-equality(1)* **and** *prop-5401-b* **by** *blast*
qed

lemma *axiom-4-1-con-validity*:

assumes $A \in \text{wffs}_\alpha$
shows $\models (\lambda x_\alpha. \llbracket c \rrbracket_\beta) \cdot A =_\beta \llbracket c \rrbracket_\beta$

proof (*intro allI impI*)

fix \mathcal{M} **and** φ

assume *: *is-general-model* $\mathcal{M} \varphi \rightsquigarrow_M \mathcal{M}$

show $\mathcal{M} \models_\varphi (\lambda x_\alpha. \llbracket c \rrbracket_\beta) \cdot A =_\beta \llbracket c \rrbracket_\beta$

proof –

obtain \mathcal{D} **and** \mathcal{J} **and** \mathcal{V} **where** $\mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V})$

using *prod-cases3* **by** *blast*

moreover from *assms* **and** * **and** $\langle \mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V}) \rangle$ **have** $\mathcal{V} \varphi ((\lambda x_\alpha. \llbracket c \rrbracket_\beta) \cdot A =_\beta \llbracket c \rrbracket_\beta) = \mathbf{T}$

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 $A \in \text{wffs}_\alpha$

and $(y, \beta) \neq (x, \alpha)$

shows $\mathcal{V} \varphi ((\lambda x_\alpha. y_\beta) \cdot A =_\beta y_\beta) = \mathbf{T}$

proof –

from *assms(2)* **have** $(\lambda x_\alpha. y_\beta) \cdot A =_\beta y_\beta \in \text{wffs}_o$

using *axioms.axiom-4-1-var* **and** *axioms-are-wffs-of-type-o* **by** *blast*

define ψ **where** $\psi = \varphi((x, \alpha) := \mathcal{V} \varphi A)$

with *assms(1,2)* **have** $\mathcal{V} \varphi ((\lambda x_\alpha. y_\beta) \cdot A) = \mathcal{V} (\varphi((x, \alpha) := \mathcal{V} \varphi A)) (y_\beta)$

using *prop-5401-a* **by** *blast*

also have $\dots = \mathcal{V} \psi (y_\beta)$

unfolding ψ -*def* ..

also have $\dots = \mathcal{V} \varphi (y_\beta)$

proof –

from *assms*(1,2) **have** $\mathcal{V} \varphi A \in \text{elts } (\mathcal{D} \alpha)$
using *V-is-wff-denotation-function* **by** *auto*
with *ψ-def* **and** *assms*(1) **have** $\psi \rightsquigarrow \mathcal{D}$
by *simp*
moreover **have** $\text{free-vars } (y_\beta) = \{(y, \beta)\}$
by *simp*
with *ψ-def* **and** *assms*(3) **have** $\forall v \in \text{free-vars } (y_\beta). \varphi v = \psi v$
by *auto*
ultimately show *?thesis*
using *prop-5400[OF wffs-of-type-intros(1) assms(1)]* **by** *simp*
qed
finally **have** $\mathcal{V} \varphi ((\lambda x_\alpha. y_\beta) \cdot A) = \mathcal{V} \varphi (y_\beta)$.
with $\langle (\lambda x_\alpha. y_\beta) \cdot A =_\beta y_\beta \in \text{wffs}_o \rangle$ **show** *?thesis*
using *wffs-from-equality(1)* **and** *prop-5401-b[OF assms(1)]* **by** *blast*
qed

lemma *axiom-4-1-var-validity*:

assumes $A \in \text{wffs}_\alpha$
and $(y, \beta) \neq (x, \alpha)$
shows $\models (\lambda x_\alpha. y_\beta) \cdot A =_\beta y_\beta$
proof (*intro allI impI*)
fix \mathcal{M} **and** φ
assume *: *is-general-model* $\mathcal{M} \varphi \rightsquigarrow_M \mathcal{M}$
show $\mathcal{M} \models_\varphi (\lambda x_\alpha. y_\beta) \cdot A =_\beta y_\beta$
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** $\langle \mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V}) \rangle$ **have** $\mathcal{V} \varphi ((\lambda x_\alpha. y_\beta) \cdot A =_\beta y_\beta) = \mathbf{T}$
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 $A \in \text{wffs}_\alpha$
shows $\mathcal{V} \varphi ((\lambda x_\alpha. x_\alpha) \cdot A =_\alpha A) = \mathbf{T}$
proof –
from *assms*(2) **have** $(\lambda x_\alpha. x_\alpha) \cdot A =_\alpha A \in \text{wffs}_o$
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** $\mathcal{V} \varphi ((\lambda x_\alpha. x_\alpha) \cdot A) = \mathcal{V} \psi (x_\alpha)$
using *prop-5401-a* **by** *blast*
also **from** *assms* **and** *ψ-def* **have** $\dots = \psi (x, \alpha)$
using *V-is-wff-denotation-function* **by** *force*
also **from** *ψ-def* **have** $\dots = \mathcal{V} \varphi A$
by *simp*
finally **have** $\mathcal{V} \varphi ((\lambda x_\alpha. x_\alpha) \cdot A) = \mathcal{V} \varphi A$.

with *assms*(1) and $\langle \lambda x_\alpha. x_\alpha \cdot A =_\alpha A \in \text{wffs}_o \rangle$ show *?thesis*
 using *wffs-from-equality* and *prop-5401-b* by *meson*
 qed

lemma *axiom-4-2-validity*:

assumes $A \in \text{wffs}_\alpha$
 shows $\models (\lambda x_\alpha. x_\alpha) \cdot A =_\alpha A$
 proof (intro allI impI)
 fix \mathcal{M} and φ
 assume *: *is-general-model* $\mathcal{M} \varphi \rightsquigarrow_M \mathcal{M}$
 show $\mathcal{M} \models_\varphi (\lambda x_\alpha. x_\alpha) \cdot A =_\alpha A$
 proof –
 obtain \mathcal{D} and \mathcal{J} and \mathcal{V} where $\mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V})$
 using *prod-cases3* by *blast*
 moreover from *assms* and * and $\langle \mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V}) \rangle$ have $\mathcal{V} \varphi ((\lambda x_\alpha. x_\alpha) \cdot A =_\alpha A) = \mathbf{T}$
 using *general-model.axiom-4-2-validity-aux* by *simp*
 ultimately show *?thesis*
 by *simp*
 qed
 qed

lemma (in *general-model*) *axiom-4-3-validity-aux*:

assumes $\varphi \rightsquigarrow \mathcal{D}$
 and $A \in \text{wffs}_\alpha$ and $B \in \text{wffs}_{\gamma \rightarrow \beta}$ and $C \in \text{wffs}_\gamma$
 shows $\mathcal{V} \varphi ((\lambda x_\alpha. B \cdot C) \cdot A =_\beta ((\lambda x_\alpha. B) \cdot A) \cdot ((\lambda x_\alpha. C) \cdot A)) = \mathbf{T}$
 (is $\mathcal{V} \varphi (?A =_\beta ?B) = \mathbf{T}$)
 proof –
 from *assms*(2–4) have $?A =_\beta ?B \in \text{wffs}_o$
 using *axioms.axiom-4-3* and *axioms-are-wffs-of-type-o* by *blast*
 define ψ where $\psi = \varphi((x, \alpha) := \mathcal{V} \varphi A)$
 with *assms*(1,2) have $\psi \rightsquigarrow \mathcal{D}$
 using *\mathcal{V}*-*is-wff-denotation-function* by *auto*
 from *assms* and ψ -*def* have $\mathcal{V} \varphi ?A = \mathcal{V} \psi (B \cdot C)$
 using *prop-5401-a* by *blast*
 also from *assms*(3,4) and ψ -*def* and $\langle \psi \rightsquigarrow \mathcal{D} \rangle$ have $\dots = \mathcal{V} \psi B \cdot \mathcal{V} \psi C$
 using *\mathcal{V}*-*is-wff-denotation-function* by *blast*
 also from *assms*(1–3) and ψ -*def* have $\dots = \mathcal{V} \varphi ((\lambda x_\alpha. B) \cdot A) \cdot \mathcal{V} \psi C$
 using *prop-5401-a* by *simp*
 also from *assms*(1,2,4) and ψ -*def* have $\dots = \mathcal{V} \varphi ((\lambda x_\alpha. B) \cdot A) \cdot \mathcal{V} \varphi ((\lambda x_\alpha. C) \cdot A)$
 using *prop-5401-a* by *simp*
 also have $\dots = \mathcal{V} \varphi ?B$
 proof –
 have $(\lambda x_\alpha. B) \cdot A \in \text{wffs}_{\gamma \rightarrow \beta}$ and $(\lambda x_\alpha. C) \cdot A \in \text{wffs}_\gamma$
 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 $\langle ?A =_\beta ?B \in \text{wffs}_o \rangle$ show *?thesis*

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

lemma *axiom-4-3-validity*:

assumes $A \in \text{wffs}_\alpha$ and $B \in \text{wffs}_{\gamma \rightarrow \beta}$ and $C \in \text{wffs}_\gamma$

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 $\langle \mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V}) \rangle$ 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 $A \in \text{wffs}_\alpha$ and $B \in \text{wffs}_\delta$

and $(y, \gamma) \notin \{(x, \alpha)\} \cup \text{vars } A$

shows $\mathcal{V} \varphi ((\lambda x_\alpha. \lambda y_\gamma. B) \cdot A =_{\gamma \rightarrow \delta} (\lambda y_\gamma. (\lambda x_\alpha. B) \cdot A)) = \mathbf{T}$

(is $\mathcal{V} \varphi (?A =_{\gamma \rightarrow \delta} ?B) = \mathbf{T}$)

proof –

from *assms(2,3)* have $?A =_{\gamma \rightarrow \delta} ?B \in \text{wffs}_o$

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 *ℳ-is-wff-denotation-function* by *simp*

{

fix z

assume $z \in \text{elts } (\mathcal{D} \ \gamma)$

define φ' where $\varphi' = \varphi((y, \gamma) := z)$

from *assms(1)* and $\langle z \in \text{elts } (\mathcal{D} \ \gamma) \rangle$ and *φ'-def* have $\varphi' \rightsquigarrow \mathcal{D}$

by *simp*

moreover from *φ'-def* and *assms(4)* have $\forall v \in \text{free-vars } A. \varphi v = \varphi' v$

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 $\langle \psi \rightsquigarrow \mathcal{D} \rangle$ and $\langle z \in \text{elts } (\mathcal{D} \ \gamma) \rangle$ 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 $\langle \varphi' \rightsquigarrow \mathcal{D} \rangle$ and *assms(2,3)* have $\dots = \mathcal{V} \varphi' ((\lambda x_\alpha. B) \cdot A)$

using *prop-5401-a* and $\langle \varphi'((x, \alpha) := \mathcal{V} \varphi' A) = \psi((y, \gamma) := z) \rangle$ by *simp*

**also from φ' -def and $assms(1)$ and $\langle z \in elts(\mathcal{D} \ \gamma) \rangle$ and $\langle ?A =_{\gamma \rightarrow \delta} ?B \in wffs_o \rangle$
have $\dots = \mathcal{V} \ \varphi \ ?B \cdot z$
by (*metis mixed-beta-conversion wffs-from-abs wffs-from-equality(2)*)
finally have $\mathcal{V} \ \psi \ ?D \cdot z = \mathcal{V} \ \varphi \ ?B \cdot z$.
}
note $*$ = *this*
then have $\mathcal{V} \ \psi \ ?D = \mathcal{V} \ \varphi \ ?B$
proof –
from $\langle \psi \rightsquigarrow \mathcal{D} \rangle$ **and** $assms(3)$ **have** $\mathcal{V} \ \psi \ ?D = (\lambda z : \mathcal{D} \ \gamma. \ \mathcal{V} \ (\psi((y, \gamma) := z)) \ B)$
using *wff-abs-denotation[OF \mathcal{V} -is-wff-denotation-function]* **by** *simp*
moreover from $assms(1)$ **have** $\mathcal{V} \ \varphi \ ?B = (\lambda z : \mathcal{D} \ \gamma. \ \mathcal{V} \ (\varphi((y, \gamma) := z)) \ ((\lambda x_\alpha. \ B) \cdot A))$
using *wffs-from-abs[OF wffs-from-equality(2)[OF $\langle ?A =_{\gamma \rightarrow \delta} ?B \in wffs_o \rangle]$*
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** $\mathcal{V} \ \varphi \ ?A = \mathcal{V} \ \varphi \ ?B$
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 $\langle ?A =_{\gamma \rightarrow \delta} ?B \in wffs_o \rangle]$* **by** *blast*
qed**

lemma *axiom-4-4-validity*:

assumes $A \in wffs_\alpha$ **and** $B \in wffs_\delta$
and $(y, \gamma) \notin \{(x, \alpha)\} \cup vars \ A$
shows $\models (\lambda x_\alpha. \ \lambda y_\gamma. \ B) \cdot A =_{\gamma \rightarrow \delta} (\lambda y_\gamma. \ (\lambda x_\alpha. \ B) \cdot A)$ (**is** $\models ?A =_{\gamma \rightarrow \delta} ?B$)

proof (*intro allI impI*)

fix \mathcal{M} **and** φ
assume $*$: *is-general-model* $\mathcal{M} \ \varphi \rightsquigarrow_M \ \mathcal{M}$
show $\mathcal{M} \models_\varphi ?A =_{\gamma \rightarrow \delta} ?B$
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** $\langle \mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V}) \rangle$ **have** $\mathcal{V} \ \varphi \ (?A =_{\gamma \rightarrow \delta} ?B) = \mathbf{T}$
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 $\varphi \rightsquigarrow \mathcal{D}$
and $A \in wffs_\alpha$ **and** $B \in wffs_\delta$
shows $\mathcal{V} \ \varphi \ ((\lambda x_\alpha. \ \lambda x_\alpha. \ B) \cdot A =_{\alpha \rightarrow \delta} (\lambda x_\alpha. \ B)) = \mathbf{T}$

proof –

define ψ **where** $\psi = \varphi((x, \alpha) := \mathcal{V} \ \varphi \ A)$
from $assms$ **have** *wff*: $(\lambda x_\alpha. \ \lambda x_\alpha. \ B) \cdot A =_{\alpha \rightarrow \delta} (\lambda x_\alpha. \ B) \in wffs_o$
using *axioms.axiom-4-5* **and** *axioms-are-wffs-of-type-o* **by** *blast*
with $assms(1,2)$ **and** ψ -def **have** $\mathcal{V} \ \varphi \ ((\lambda x_\alpha. \ \lambda x_\alpha. \ B) \cdot A) = \mathcal{V} \ \psi \ (\lambda x_\alpha. \ B)$

using *prop-5401-a* and *wffs-from-equality(2)* by *blast*
 also have $\dots = \mathcal{V} \varphi (\lambda x_\alpha. B)$
proof –
 have $(x, \alpha) \notin \text{free-vars} (\lambda x_\alpha. B)$
 by *simp*
 with $\psi\text{-def}$ have $\forall v \in \text{free-vars} (\lambda x_\alpha. B). \varphi v = \psi v$
 by *simp*
 moreover from $\psi\text{-def}$ and *assms(1,2)* have $\psi \rightsquigarrow \mathcal{D}$
 using *\mathcal{V} -is-wff-denotation-function* by *simp*
 moreover from *assms(2,3)* have $(\lambda x_\alpha. B) \in \text{wffs}_{\alpha \rightarrow \delta}$
 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 $A \in \text{wffs}_\alpha$ and $B \in \text{wffs}_\delta$
 shows $\models (\lambda x_\alpha. \lambda x_\alpha. B) \cdot A =_{\alpha \rightarrow \delta} (\lambda x_\alpha. B)$
proof (*intro allI impI*)
 fix \mathcal{M} and φ
 assume *: *is-general-model* $\mathcal{M} \varphi \rightsquigarrow_M \mathcal{M}$
 show $\mathcal{M} \models_\varphi (\lambda x_\alpha. \lambda x_\alpha. B) \cdot A =_{\alpha \rightarrow \delta} (\lambda x_\alpha. B)$
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 $\mathcal{V} \varphi ((\lambda x_\alpha. \lambda x_\alpha. B) \cdot A =_{\alpha \rightarrow \delta} (\lambda x_\alpha. B)) = \mathbf{T}$
 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 $\mathcal{V} \varphi (\iota \cdot (Q_i \cdot \eta_i) =_i \eta_i) = \mathbf{T}$
proof –
 have $\iota \cdot (Q_i \cdot \eta_i) =_i \eta_i \in \text{wffs}_o$
 using *axioms.axiom-5* and *axioms-are-wffs-of-type-o* by *blast*
 have $Q_i \cdot \eta_i \in \text{wffs}_{i \rightarrow o}$
 by *blast*
 with *assms* have $\mathcal{V} \varphi (\iota \cdot (Q_i \cdot \eta_i)) = \mathcal{V} \varphi \iota \cdot \mathcal{V} \varphi (Q_i \cdot \eta_i)$
 using *\mathcal{V} -is-wff-denotation-function* by *blast*
 also from *assms* have $\dots = \mathcal{V} \varphi \iota \cdot (\mathcal{V} \varphi (Q_i) \cdot \mathcal{V} \varphi (\eta_i))$
 using *wff-app-denotation[OF \mathcal{V} -is-wff-denotation-function]* by (*metis Q-wff wffs-of-type-intros(1)*)

also from *assms* have $\dots = \mathcal{J}(\mathbf{c}_\iota, (i \rightarrow o) \rightarrow i) \cdot (\mathcal{J}(\mathbf{c}_Q, i \rightarrow i \rightarrow o) \cdot \mathcal{V} \varphi(\eta_i))$
using *V-is-wff-denotation-function* by *auto*
also from *assms* have $\dots = \mathcal{J}(\mathbf{c}_\iota, (i \rightarrow o) \rightarrow i) \cdot ((q_i^{\mathcal{D}}) \cdot \mathcal{V} \varphi(\eta_i))$
using *Q-constant-of-type-def* and *Q-denotation* by *simp*
also from *assms* have $\dots = \mathcal{J}(\mathbf{c}_\iota, (i \rightarrow o) \rightarrow i) \cdot \{\mathcal{V} \varphi(\eta_i)\}_i^{\mathcal{D}}$
using *V-is-wff-denotation-function* by *auto*
finally have $\mathcal{V} \varphi(\iota \cdot (Q_i \cdot \eta_i)) = \mathcal{J}(\mathbf{c}_\iota, (i \rightarrow o) \rightarrow i) \cdot \{\mathcal{V} \varphi(\eta_i)\}_i^{\mathcal{D}}$.
moreover from *assms* have $\mathcal{J}(\mathbf{c}_\iota, (i \rightarrow o) \rightarrow i) \cdot \{\mathcal{V} \varphi(\eta_i)\}_i^{\mathcal{D}} = \mathcal{V} \varphi(\eta_i)$
using *V-is-wff-denotation-function* and ι -denotation by *force*
ultimately have $\mathcal{V} \varphi(\iota \cdot (Q_i \cdot \eta_i)) = \mathcal{V} \varphi(\eta_i)$
by (*simp only*):
with *assms* and $\langle Q_i \cdot \eta_i \in \text{wffs}_{i \rightarrow o} \rangle$ show *?thesis*
using *prop-5401-b* by *blast*
qed

lemma *axiom-5-validity*:

shows $\models \iota \cdot (Q_i \cdot \eta_i) =_i \eta_i$
proof (*intro all impI*)
fix \mathcal{M} and φ
assume $*$: *is-general-model* $\mathcal{M} \varphi \rightsquigarrow_M \mathcal{M}$
show $\mathcal{M} \models_\varphi \iota \cdot (Q_i \cdot \eta_i) =_i \eta_i$
proof –
obtain \mathcal{D} and \mathcal{J} and \mathcal{V} where $\mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V})$
using *prod-cases3* by *blast*
moreover from $*$ and $\langle \mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V}) \rangle$ have $\mathcal{V} \varphi(\iota \cdot (Q_i \cdot \eta_i) =_i \eta_i) = \mathbf{T}$
using *general-model.axiom-5-validity-aux* by *simp*
ultimately show *?thesis*
by *simp*
qed
qed

lemma *axioms-validity*:

assumes $A \in \text{axioms}$
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 $A \in \text{wffs}_\alpha$ and $B \in \text{wffs}_\alpha$

and $\forall \varphi. \varphi \rightsquigarrow \mathcal{D} \longrightarrow \mathcal{V} \varphi A = \mathcal{V} \varphi B$
and $C \in \text{wffs}_\beta$ **and** $C' \in \text{wffs}_\beta$
and $p \in \text{positions } C$ **and** $A \preceq_p C$ **and** $C \langle p \leftarrow B \rangle \triangleright C'$
shows $\forall \varphi. \varphi \rightsquigarrow \mathcal{D} \longrightarrow \mathcal{V} \varphi C = \mathcal{V} \varphi C'$
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 $\varphi \rightsquigarrow \mathcal{D}$
 from $\langle G \cdot H \in \text{wffs}_\beta \rangle$ **obtain** γ **where** $G \in \text{wffs}_{\gamma \rightarrow \beta}$ **and** $H \in \text{wffs}_\gamma$
 by (*rule wffs-from-app*)
 with $\langle G' \cdot H \in \text{wffs}_\beta \rangle$ **have** $G' \in \text{wffs}_{\gamma \rightarrow \beta}$
 by (*metis wff-has-unique-type wffs-from-app*)
 from *assms(1)* **and** $\langle \varphi \rightsquigarrow \mathcal{D} \rangle$ **and** $\langle G \in \text{wffs}_{\gamma \rightarrow \beta} \rangle$ **and** $\langle H \in \text{wffs}_\gamma \rangle$
 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 $\langle \varphi \rightsquigarrow \mathcal{D} \rangle$ **and** $\langle G \in \text{wffs}_{\gamma \rightarrow \beta} \rangle$ **and** $\langle G' \in \text{wffs}_{\gamma \rightarrow \beta} \rangle$ **have** $\dots = \mathcal{V} \varphi G' \cdot \mathcal{V} \varphi H$
 using *replace-left-app.IH* **and** *replace-left-app.prem(1,4)* **by** *simp*
 also from *assms(1)* **and** $\langle \varphi \rightsquigarrow \mathcal{D} \rangle$ **and** $\langle G' \in \text{wffs}_{\gamma \rightarrow \beta} \rangle$ **and** $\langle H \in \text{wffs}_\gamma \rangle$
 have $\dots = \mathcal{V} \varphi (G' \cdot H)$
 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 $\varphi \rightsquigarrow \mathcal{D}$
 from $\langle G \cdot H \in \text{wffs}_\beta \rangle$ **obtain** γ **where** $G \in \text{wffs}_{\gamma \rightarrow \beta}$ **and** $H \in \text{wffs}_\gamma$
 by (*rule wffs-from-app*)
 with $\langle G \cdot H' \in \text{wffs}_\beta \rangle$ **have** $H' \in \text{wffs}_\gamma$
 using *wff-has-unique-type* **and** *wffs-from-app* **by** (*metis type.inject*)
 from *assms(1)* **and** $\langle \varphi \rightsquigarrow \mathcal{D} \rangle$ **and** $\langle G \in \text{wffs}_{\gamma \rightarrow \beta} \rangle$ **and** $\langle H \in \text{wffs}_\gamma \rangle$
 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 $\langle \varphi \rightsquigarrow \mathcal{D} \rangle$ **and** $\langle H \in \text{wffs}_\gamma \rangle$ **and** $\langle H' \in \text{wffs}_\gamma \rangle$ **have** $\dots = \mathcal{V} \varphi G \cdot \mathcal{V} \varphi H'$
 using *replace-right-app.IH* **and** *replace-right-app.prem(1,4)* **by** *force*
 also from *assms(1)* **and** $\langle \varphi \rightsquigarrow \mathcal{D} \rangle$ **and** $\langle G \in \text{wffs}_{\gamma \rightarrow \beta} \rangle$ **and** $\langle H' \in \text{wffs}_\gamma \rangle$
 have $\dots = \mathcal{V} \varphi (G \cdot H')$
 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-abs p E B' E' x  $\gamma$ )
show ?case
proof (intro allI impI)
  fix  $\varphi$ 
  assume  $\varphi \rightsquigarrow \mathcal{D}$ 
  define  $\psi$  where  $\psi z = \varphi((x, \gamma) := z)$  for  $z$ 
  with  $\langle \varphi \rightsquigarrow \mathcal{D} \rangle$  have  $\psi$ -assg:  $\psi z \rightsquigarrow \mathcal{D}$  if  $z \in \text{elts}(\mathcal{D} \ \gamma)$  for  $z$ 
  by (simp add: that)
  from  $\langle \lambda x \gamma. E \in \text{wffs}_\beta \rangle$  obtain  $\delta$  where  $\beta = \gamma \rightarrow \delta$  and  $E \in \text{wffs}_\delta$ 
  by (rule wffs-from-abs)
  with  $\langle \lambda x \gamma. E' \in \text{wffs}_\beta \rangle$  have  $E' \in \text{wffs}_\delta$ 
  using wffs-from-abs by blast
  from assms(1) and  $\langle \varphi \rightsquigarrow \mathcal{D} \rangle$  and  $\langle E \in \text{wffs}_\delta \rangle$  and  $\psi$ -def
  have  $\mathcal{V} \varphi (\lambda x \gamma. E) = (\lambda z : \mathcal{D} \ \gamma. \mathcal{V} (\psi z) E)$ 
  using wff-abs-denotation[OF V-is-wff-denotation-function] by simp
  also have  $\dots = (\lambda z : \mathcal{D} \ \gamma. \mathcal{V} (\psi z) E')$ 
  proof (intro vlambda-extensionality)
    fix  $z$ 
    assume  $z \in \text{elts}(\mathcal{D} \ \gamma)$ 
    from  $\langle E \in \text{wffs}_\delta \rangle$  and  $\langle E' \in \text{wffs}_\delta \rangle$  have  $\forall \varphi. \varphi \rightsquigarrow \mathcal{D} \longrightarrow \mathcal{V} \varphi E = \mathcal{V} \varphi E'$ 
    using replace-abs.premis(1,4) and replace-abs.IH by simp
    with  $\psi$ -assg and  $\langle z \in \text{elts}(\mathcal{D} \ \gamma) \rangle$  show  $\mathcal{V} (\psi z) E = \mathcal{V} (\psi z) E'$ 
    by simp
  qed
  also from assms(1) and  $\langle \varphi \rightsquigarrow \mathcal{D} \rangle$  and  $\langle E' \in \text{wffs}_\delta \rangle$  and  $\psi$ -def
  have  $\dots = \mathcal{V} \varphi (\lambda x \gamma. E')$ 
  using wff-abs-denotation[OF 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  $C \in \text{wffs}_o$  and  $C' \in \text{wffs}_o$  and  $E \in \text{wffs}_o$ 
  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  $\varphi$ 
  assume is-general-model  $\mathcal{M}$  and  $\varphi \rightsquigarrow_{\mathcal{M}} \mathcal{M}$ 
  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
  qed

```

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

lemma *individual-proof-step-validity*:

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

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

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

proof (*intro allI impI*)
fix φ
assume $\varphi \rightsquigarrow_M \mathcal{M}$
moreover obtain \mathcal{D} and \mathcal{J} and \mathcal{V} where $\mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V})$
using *prod-cases3* **by** *blast*
ultimately have $\varphi \rightsquigarrow \mathcal{D}$
by *simp*
show $\mathcal{M} \models_{\varphi} B$
proof –
from *assms(4)* **have** $\mathcal{V} \varphi (A \supset^{\mathcal{Q}} B) = \mathbf{T}$
using $\langle \mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V}) \rangle$ **and** $\langle \varphi \rightsquigarrow_M \mathcal{M} \rangle$ **by** *auto*
with *assms(1-3)* **have** $\mathcal{V} \varphi A \supset \mathcal{V} \varphi B = \mathbf{T}$
using $\langle \mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V}) \rangle$ **and** $\langle \varphi \rightsquigarrow_M \mathcal{M} \rangle$ **and** *general-model.prop-5401-f'* **by** *simp*
moreover from *assms(5)* **have** $\mathcal{V} \varphi A = \mathbf{T}$
using $\langle \mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V}) \rangle$ **and** $\langle \varphi \rightsquigarrow \mathcal{D} \rangle$ **by** *auto*
moreover from $\langle \mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V}) \rangle$ **and** *assms(1)* **have** $\text{elts } (\mathcal{D} \ o) = \text{elts } \mathbb{B}$
using *frame.truth-values-domain-def* **and** *general-model-def* **and** *premodel-def* **by** *fastforce*
with *assms* **and** $\langle \mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V}) \rangle$ **and** $\langle \varphi \rightsquigarrow \mathcal{D} \rangle$ **and** $\langle \mathcal{V} \varphi A, \mathcal{V} \varphi B \rangle \subseteq \text{elts}$
 \mathbb{B}
using *general-model.V-is-wff-denotation-function*
and *premodel.wff-denotation-function-is-domain-respecting* **and** *general-model.axioms(1)* **by** *blast*
ultimately have $\mathcal{V} \varphi B = \mathbf{T}$
by *fastforce*
with $\langle \mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V}) \rangle$ **and** *assms(1)* **and** $\langle \varphi \rightsquigarrow \mathcal{D} \rangle$ **show** *?thesis*
by *simp*
qed
qed

lemma *generalized-semantic-modus-ponens*:
assumes *is-general-model* \mathcal{M}
and $\text{lset } hs \subseteq \text{wffs}_o$
and $\forall H \in \text{lset } hs. \mathcal{M} \models H$
and $P \in \text{wffs}_o$
and $\mathcal{M} \models hs \supset^{\mathcal{Q},*} P$
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 $\langle \mathcal{M} \models (hs @ [H']) \supset^{\mathcal{Q},*} P \rangle$ **have** $\mathcal{M} \models hs \supset^{\mathcal{Q},*} (H' \supset^{\mathcal{Q}} P)$
by *simp*
moreover from $\langle \forall H \in \text{lset } (hs @ [H']). \mathcal{M} \models H \rangle$ **and** $\langle \text{lset } (hs @ [H']) \subseteq \text{wffs}_o \rangle$
have $\forall H \in \text{lset } hs. \mathcal{M} \models H$ **and** $\text{lset } hs \subseteq \text{wffs}_o$
by *simp-all*
moreover from $\langle \text{lset } (hs @ [H']) \subseteq \text{wffs}_o \rangle$ **and** $\langle P \in \text{wffs}_o \rangle$ **have** $H' \supset^{\mathcal{Q}} P \in \text{wffs}_o$
by *auto*
ultimately have $\mathcal{M} \models H' \supset^{\mathcal{Q}} P$
by (*elim snoc.IH*)

moreover from $\langle \forall H \in \text{lset } (hs @ [H']). \mathcal{M} \models H \rangle$ **have** $\mathcal{M} \models H'$
by *simp*
moreover from $\langle H' \supset^Q P \in \text{wffs}_o \rangle$ **have** $H' \in \text{wffs}_o$
using *wffs-from-imp-op(1)* **by** *blast*
ultimately show *?case*
using *assms(1)* **and** $\langle P \in \text{wffs}_o \rangle$ **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** $A \in \text{wffs}_o$
by (*fact hyp-derivable-form-is-wffso*)
from $\langle \mathcal{G} \vdash A \rangle$ **and** $\langle \text{is-hyps } \mathcal{G} \rangle$ **obtain** \mathcal{H} **where** *finite* \mathcal{H} **and** $\mathcal{H} \subseteq \mathcal{G}$ **and** $\mathcal{H} \vdash A$
by *blast*
moreover from $\langle \text{finite } \mathcal{H} \rangle$ **obtain** hs **where** $\text{lset } hs = \mathcal{H}$
using *finite-list* **by** *blast*
ultimately have $\vdash hs \supset^Q_* A$
using *generalized-deduction-theorem* **by** *simp*
with *assms(4)* **have** $\mathcal{M} \models hs \supset^Q_* A$
using *derivability-from-no-hyps-theoremhood-equivalence* **and** *theoremhood-implies-validity*
by *blast*
moreover from $\langle \mathcal{H} \subseteq \mathcal{G} \rangle$ **and** *assms(2)* **have** $\mathcal{M} \models H$ **if** $H \in \mathcal{H}$ **for** H
using *that* **by** *blast*
moreover from $\langle \mathcal{H} \subseteq \mathcal{G} \rangle$ **and** $\langle \text{lset } hs = \mathcal{H} \rangle$ **and** *assms(1)* **have** $\text{lset } hs \subseteq \text{wffs}_o$
by *blast*
ultimately show *?thesis*
using *assms(1,4)* **and** $\langle A \in \text{wffs}_o \rangle$ **and** $\langle \text{lset } hs = \mathcal{H} \rangle$ **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* \Rightarrow *bool* **where**

[*iff*]: *is-inconsistent-set* $\mathcal{G} \longleftrightarrow \mathcal{G} \vdash F_o$

definition *Q₀-is-inconsistent* :: *bool* **where**

[*iff*]: *Q₀-is-inconsistent* $\longleftrightarrow \vdash F_o$

definition *is-wffo-consistent-with* :: *form* \Rightarrow *form set* \Rightarrow *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., $\mathcal{D}^S i = \text{ZFC-in-HOL.set } \{0\}$

| $\mathcal{D}^S o = \mathbb{B}$

| $\mathcal{D}^S (\alpha \rightarrow \beta) = \mathcal{D}^S \alpha \mapsto \mathcal{D}^S \beta$

interpretation *singleton-standard-frame*: *frame* \mathcal{D}^S

proof *unfold-locales*

```
{
  fix  $\alpha$ 
  have  $\mathcal{D}^S \alpha \neq 0$ 
  proof (induction  $\alpha$ )
    case (TFun  $\beta \gamma$ )
    from  $\langle \mathcal{D}^S \gamma \neq 0 \rangle$  obtain  $y$  where  $y \in \text{elts } (\mathcal{D}^S \gamma)$ 
    by fastforce
    then have  $(\lambda z : \mathcal{D}^S \beta. y) \in \text{elts } (\mathcal{D}^S \beta \mapsto \mathcal{D}^S \gamma)$ 
    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
     $\exists \beta. \text{is-Q-constant-of-type } k \beta$ 
  then
```

```

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

```

interpretation *singleton-standard-premodel*: premodel $\mathcal{D}^S \mathcal{J}^S$

proof (*unfold-locales*)

```

  show  $\forall \alpha. \mathcal{J}^S (Q\text{-constant-of-type } \alpha) = q_\alpha^{\mathcal{D}^S}$ 
  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  $x \in \text{elts } (\mathcal{D}^S i)$ 

```

```

  then have  $x = 0$ 

```

```

  by simp

```

```

  moreover have  $(\lambda z : \mathcal{D}^S (i \rightarrow o). 0) \cdot \{0\}_i^{\mathcal{D}^S} = 0$ 

```

```

  using beta[OF singleton-standard-frame.one-element-function-is-domain-respecting]

```

```

  unfolding singleton-standard-domain-family.simps(3) by blast

```

```

  ultimately show  $(\mathcal{J}^S \text{iota-constant}) \cdot \{x\}_i^{\mathcal{D}^S} = x$ 

```

```

  by fastforce

```

```

qed

```

next

```

  show  $\forall c \alpha. \neg \text{is-logical-constant } (c, \alpha) \longrightarrow \mathcal{J}^S (c, \alpha) \in \text{elts } (\mathcal{D}^S \alpha)$ 

```

```

  proof (intro allI impI)

```

```

  fix  $c$  and  $\alpha$ 

```

```

  assume  $\neg \text{is-logical-constant } (c, \alpha)$ 

```

```

  then have  $\mathcal{J}^S (c, \alpha) = (\text{SOME } z. z \in \text{elts } (\mathcal{D}^S \alpha))$ 

```

```

  by auto

```

```

  moreover have  $\exists z. z \in \text{elts } (\mathcal{D}^S \alpha)$ 

```

```

  using eq0-iff and singleton-standard-frame.domain-nonemptiness by presburger

```

```

  then have  $(\text{SOME } z. z \in \text{elts } (\mathcal{D}^S \alpha)) \in \text{elts } (\mathcal{D}^S \alpha)$ 

```

```

  using some-in-eq by auto

```

```

  ultimately show  $\mathcal{J}^S (c, \alpha) \in \text{elts } (\mathcal{D}^S \alpha)$ 

```

```

  by auto

```

```

qed

```

qed

fun *singleton-standard-wff-denotation-function* (\mathcal{V}^S) **where**

```

   $\mathcal{V}^S \varphi (x_\alpha) = \varphi (x, \alpha)$ 
|  $\mathcal{V}^S \varphi (\{c\}_\alpha) = \mathcal{J}^S (c, \alpha)$ 
|  $\mathcal{V}^S \varphi (A \cdot B) = (\mathcal{V}^S \varphi A) \cdot (\mathcal{V}^S \varphi B)$ 
|  $\mathcal{V}^S \varphi (\lambda x_\alpha. A) = (\lambda z : \mathcal{D}^S \alpha. \mathcal{V}^S \varphi ((x, \alpha) := z)) A$ 

```

lemma *singleton-standard-wff-denotation-function-closure*:
assumes *frame.is-assignment* $\mathcal{D}^S \varphi$
and $A \in \text{wffs}_\alpha$
shows $\mathcal{V}^S \varphi A \in \text{elts}(\mathcal{D}^S \alpha)$
using *assms(2,1)* **proof** (*induction A arbitrary: φ*)
case (*var-is-wff* αx)
then show *?case*
by *simp*
next
case (*con-is-wff* αc)
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 $\mathcal{V}^S \varphi (\llbracket c \rrbracket_\alpha) = q_\beta^{\mathcal{D}^S}$
by *simp*
moreover have $q_\beta^{\mathcal{D}^S} \in \text{elts}(\mathcal{D}^S (\beta \rightarrow \beta \rightarrow o))$
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 $\mathcal{V}^S \varphi (\llbracket c \rrbracket_\alpha) = (\lambda z : \mathcal{D}^S (i \rightarrow o). 0)$
by *simp*
moreover have $(\lambda z : \mathcal{D}^S (i \rightarrow o). 0) \in \text{elts}(\mathcal{D}^S ((i \rightarrow o) \rightarrow i))$
by (*simp add: VPi-I*)
ultimately show *?thesis*
using *ι -constant* **by** *simp*
qed
next
case (*app-is-wff* $\alpha \beta A B$)
have $\mathcal{V}^S \varphi (A \cdot B) = (\mathcal{V}^S \varphi A) \cdot (\mathcal{V}^S \varphi B)$
using *singleton-standard-wff-denotation-function.simps(3)* .
moreover have $\mathcal{V}^S \varphi A \in \text{elts}(\mathcal{D}^S (\alpha \rightarrow \beta))$ **and** $\mathcal{V}^S \varphi B \in \text{elts}(\mathcal{D}^S \alpha)$
using *app-is-wff.IH* **and** *app-is-wff.prem* **by** *simp-all*
ultimately show *?case*
by (*simp only: singleton-standard-frame.app-is-domain-respecting*)
next
case (*abs-is-wff* $\beta A \alpha x$)
have $\mathcal{V}^S \varphi (\lambda x_\alpha. A) = (\lambda z : \mathcal{D}^S \alpha. \mathcal{V}^S (\varphi((x, \alpha) := z)) A)$
using *singleton-standard-wff-denotation-function.simps(4)* .
moreover have $\mathcal{V}^S (\varphi((x, \alpha) := z)) A \in \text{elts}(\mathcal{D}^S \beta)$ **if** $z \in \text{elts}(\mathcal{D}^S \alpha)$ **for** z
using *that* **and** *abs-is-wff.IH* **and** *abs-is-wff.prem* **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 $\forall \alpha \beta. \mathcal{D}^S (\alpha \rightarrow \beta) = \mathcal{D}^S \alpha \mapsto \mathcal{D}^S \beta$
using *singleton-standard-domain-family.simps(3)* **by** (*intro allI*)
qed

proposition *standard-model-existence:*
shows $\exists \mathcal{M}. \text{is-standard-model } \mathcal{M}$
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 $\exists \mathcal{M}. \text{is-general-model } \mathcal{M} \wedge \text{is-model-for } \mathcal{M} \mathcal{G}$
shows $\neg \text{is-inconsistent-set } \mathcal{G}$
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 $\neg \neg \text{is-inconsistent-set } \mathcal{G}$
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 M G]* **by** *simp*
then have $\mathcal{V} \varphi F_o = \mathbf{T}$ **if** $\varphi \rightsquigarrow \mathcal{D}$ **for** φ
using *that* **and** $\langle \mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V}) \rangle$ **by** *force*
moreover have $\mathcal{V} \varphi F_o = \mathbf{F}$ **if** $\varphi \rightsquigarrow \mathcal{D}$ **for** φ
using $\langle \mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V}) \rangle$ **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 $?\psi = \lambda v. \text{case } v \text{ of } (-, \alpha) \Rightarrow \text{SOME } z. z \in \text{elts } (\mathcal{D} \alpha)$
from $\langle \mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V}) \rangle$ **and** *is-general-model* \mathcal{M} **have** $\forall \alpha. \text{elts } (\mathcal{D} \alpha) \neq \{\}$
using *frame.domain-nonemptiness* **and** *premodel-def* **and** *general-model.axioms(1)* **by** *auto*
with $\langle \mathcal{M} = (\mathcal{D}, \mathcal{J}, \mathcal{V}) \rangle$ **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  $\exists \mathcal{M}. \text{is-general-model } \mathcal{M} \wedge \text{is-model-for } \mathcal{M} \{ \}$ 
    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  $\mathcal{G} \longleftrightarrow (\forall A \in (\text{wffs}_o). \mathcal{G} \vdash A)$ 
proof
  assume is-inconsistent-set  $\mathcal{G}$ 
  show  $\forall A \in (\text{wffs}_o). \mathcal{G} \vdash A$ 
    proof
      fix A
      assume  $A \in \text{wffs}_o$ 
      from  $\langle \text{is-inconsistent-set } \mathcal{G} \rangle$  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  $\langle A \in \text{wffs}_o \rangle$  have  $\mathcal{G} \vdash \mathbf{S} \{ (\mathfrak{x}, o) \mapsto A \} (\mathfrak{x}_o)$ 
        using  $\forall I$  by fastforce
      then show  $\mathcal{G} \vdash A$ 
        by simp
    qed
  next
  assume  $\forall A \in (\text{wffs}_o). \mathcal{G} \vdash A$ 
  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.