

Metatheory of \mathcal{Q}_0

Javier Díaz

<javier.diaz.manzi@gmail.com>

April 18, 2024

Abstract

This entry is a formalization of the metatheory of \mathcal{Q}_0 in Isabelle/HOL. \mathcal{Q}_0 [2] is a classical higher-order logic equivalent to Church's Simple Theory of Types. In this entry we formalize Chapter 5 of [2], up to and including the proofs of soundness and consistency of \mathcal{Q}_0 . These 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	7
2.1	Type symbols	7
2.2	Variables	7
2.3	Constants	8
2.4	Formulas	8
2.5	Generalized operators	9
2.6	Subformulas	9
2.7	Free and bound variables	13
2.8	Free and bound occurrences	15
2.9	Free variables for a formula in another formula	20
2.10	Replacement of subformulas	21
2.11	Logical constants	23
2.12	Definitions and abbreviations	23
2.13	Well-formed formulas	25
2.14	Substitutions	31
2.15	Renaming of bound variables	35
3	Boolean Algebra	37
4	Propositional Well-Formed Formulas	39
4.1	Syntax	39
4.2	Semantics	41
5	Proof System	46
5.1	Axioms	46
5.2	Inference rule R	47
5.3	Proof and derivability	47
5.4	Hypothetical proof and derivability	50
6	Elementary Logic	55
6.1	Proposition 5200	55
6.2	Proposition 5201 (Equality Rules)	56
6.3	Proposition 5202 (Rule RR)	56
6.4	Proposition 5203	56
6.5	Proposition 5204	57
6.6	Proposition 5205 (η -conversion)	57
6.7	Proposition 5206 (α -conversion)	57
6.8	Proposition 5207 (β -conversion)	57
6.9	Proposition 5208	58

6.10	Proposition 5209	58
6.11	Proposition 5210	58
6.12	Proposition 5211	58
6.13	Proposition 5212	58
6.14	Proposition 5213	58
6.15	Proposition 5214	58
6.16	Proposition 5215 (Universal Instantiation)	59
6.17	Proposition 5216	59
6.18	Proposition 5217	59
6.19	Proposition 5218	59
6.20	Proposition 5219 (Rule T)	59
6.21	Proposition 5220 (Universal Generalization)	59
6.22	Proposition 5221 (Substitution)	60
6.23	Proposition 5222 (Rule of Cases)	60
6.24	Proposition 5223	61
6.25	Proposition 5224 (Modus Ponens)	61
6.26	Proposition 5225	61
6.27	Proposition 5226	61
6.28	Proposition 5227	62
6.29	Proposition 5228	62
6.30	Proposition 5229	62
6.31	Proposition 5230	62
6.32	Proposition 5231	62
6.33	Proposition 5232	63
6.34	Proposition 5233	63
6.35	Proposition 5234 (Rule P)	63
6.36	Proposition 5235	64
6.37	Proposition 5237 ($\supset \forall$ Rule)	64
6.38	Proposition 5238	65
6.39	Proposition 5239	65
6.40	Theorem 5240 (Deduction Theorem)	66
6.41	Proposition 5241	67
6.42	Proposition 5242 (Rule of Existential Generalization)	67
6.43	Proposition 5243 (Comprehension Theorem)	67
6.44	Proposition 5244 (Existential Rule)	67
6.45	Proposition 5245 (Rule C)	68
7	Semantics	68
7.1	Frames	69
7.2	Pre-models (interpretations)	70
7.3	General models	72
7.4	Standard models	73
7.5	Validity	73

8	Soundness	74
8.1	Proposition 5400	74
8.2	Proposition 5401	75
8.3	Proposition 5402(a)	79
8.4	Proposition 5402(b)	79
8.5	Theorem 5402 (Soundness Theorem)	79
9	Consistency	80
9.1	Existence of a standard model	80
9.2	Theorem 5403 (Consistency Theorem)	81

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
```

```
<proof>
```

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)
```

```
<proof>
```

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

```
  shows fmran' {k ↦ v} = {v}
```

```
<proof>
```

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

```
  assumes m $$ x = None
```

```
  shows fmran' (m(x ↦ y)) = {y} ∪ fmran' m
```

```
<proof>
```

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

```
  assumes fmdom' m ∩ fmdom' m' = {}
```

```
  shows fmran' (m ++f m') = fmran' m ∪ fmran' m'
```

```
<proof>
```

lemma *finite-fmran'*:

shows *finite* (*fmran' m*)
<proof>

lemma *fmap-of-zipped-list-range*:

assumes *length ks = length vs*
and *m = fmap-of-list (zip ks vs)*
and *k ∈ fmdom' m*
shows *m \$\$\$ k ∈ lset vs*
<proof>

lemma *fmap-of-zip-nth [simp]*:

assumes *length ks = length vs*
and *distinct ks*
and *i < length ks*
shows *fmap-of-list (zip ks vs) \$\$\$ (ks ! i) = vs ! i*
<proof>

lemma *fmap-of-zipped-list-fmran' [simp]*:

assumes *distinct (map fst ps)*
shows *fmran' (fmap-of-list ps) = lset (map snd ps)*
<proof>

lemma *fmap-of-list-nth [simp]*:

assumes *distinct (map fst ps)*
and *j < length ps*
shows *fmap-of-list ps \$\$\$ ((map fst ps) ! j) = Some (map snd ps ! j)*
<proof>

lemma *fmap-of-list-nth-split [simp]*:

assumes *distinct xs*
and *j < length xs*
and *length ys = length xs and length zs = length xs*
shows *fmap-of-list (zip xs (take k ys @ drop k zs)) \$\$\$ (xs ! j) =*
(if j < k then Some (take k ys ! j) else Some (drop k zs ! (j - k)))
<proof>

lemma *fmadd-drop-cancellation [simp]*:

assumes *m \$\$\$ k = Some v*
shows *{k ↦ v} ++_f fmdrop k m = m*
<proof>

lemma *fmap-of-list-fmmap [simp]*:

shows *fmap-of-list (map2 (λv' A'. (v', f A')) xs ys) = fmmap f (fmap-of-list (zip xs ys))*
<proof>

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 ( $\alpha \rightarrow \beta$ ) =  $\{\alpha, \beta\} \cup$  subtypes  $\alpha \cup$  subtypes  $\beta$ 
```

```
lemma subtype-size-decrease:
  assumes  $\alpha \in$  subtypes  $\beta$ 
  shows type-size  $\alpha <$  type-size  $\beta$ 
   $\langle$ proof $\rangle$ 
```

```
lemma subtype-is-not-type:
  assumes  $\alpha \in$  subtypes  $\beta$ 
  shows  $\alpha \neq \beta$ 
   $\langle$ proof $\rangle$ 
```

```
lemma fun-type-atoms-in-subtypes:
  assumes  $k <$  length ts
  shows  $ts ! k \in$  subtypes (foldr ( $\rightarrow$ ) ts  $\gamma$ )
   $\langle$ proof $\rangle$ 
```

```
lemma fun-type-atoms-neq-fun-type:
  assumes  $k <$  length ts
  shows  $ts ! k \neq$  foldr ( $\rightarrow$ ) ts  $\gamma$ 
   $\langle$ proof $\rangle$ 
```

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 $var\text{-}name :: var \Rightarrow nat$ **where**
 $var\text{-}name \equiv fst$

abbreviation $var\text{-}type :: var \Rightarrow type$ **where**
 $var\text{-}type \equiv snd$

lemma *fresh-var-existence*:
assumes $finite (vs :: var\ set)$
obtains x **where** $(x, \alpha) \notin vs$
 $\langle proof \rangle$

lemma *fresh-var-name-list-existence*:
assumes $finite (ns :: nat\ set)$
obtains ns' **where** $length\ ns' = n$ **and** $distinct\ ns'$ **and** $lset\ ns' \cap ns = \{\}$
 $\langle proof \rangle$

lemma *fresh-var-list-existence*:
fixes $xs :: var\ list$
and $ns :: nat\ set$
assumes $finite\ ns$
obtains $vs' :: var\ list$
where $length\ vs' = length\ xs$
and $distinct\ vs'$
and $var\text{-}name\ 'lset\ vs' \cap (ns \cup var\text{-}name\ 'lset\ xs) = \{\}$
and $map\ var\text{-}type\ vs' = map\ var\text{-}type\ xs$
 $\langle proof \rangle$

2.3 Constants

type-synonym $con = nat \times type$

2.4 Formulas

datatype $form =$
 $FVar\ var$
 $| FCon\ con$
 $| FApp\ form\ form\ (\mathbf{infixl}\ 200)$
 $| FAbs\ var\ form$

syntax

$-FVar :: nat \Rightarrow type \Rightarrow form\ (-\ [899, 0]\ 900)$
 $-FCon :: nat \Rightarrow type \Rightarrow form\ (\{-\}\ [899, 0]\ 900)$
 $-FAbs :: nat \Rightarrow type \Rightarrow form \Rightarrow form\ ((\lambda\ -.\ / -)\ [0, 0, 104]\ 104)$

translations

$x_\alpha \equiv CONST\ FVar\ (x, \alpha)$
 $\{-c\}_\alpha \equiv CONST\ FCon\ (c, \alpha)$
 $\lambda x_\alpha. A \equiv CONST\ FAbs\ (x, \alpha)\ A$

2.5 Generalized operators

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

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

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

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

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

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

2.6 Subformulas

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

datatype *direction* = *Left* (\ll) | *Right* (\gg)

type-synonym *position* = *direction list*

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

lemma *empty-is-position* [*simp*]:
shows $\!\{\}\!\} \in \text{positions } A$
\langle proof \rangle

fun *subform-at* :: *form* \Rightarrow *position* \rightarrow *form* **where**
subform-at $A \!\{\}\!\} = \text{Some } A$
| *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* [] *A'* = (*A* = *A'*)
| *is-subform-at* *C* (« # *p*) (*A* • *B*) = *is-subform-at* *C* *p* *A*
| *is-subform-at* *C* (» # *p*) (*A* • *B*) = *is-subform-at* *C* *p* *B*
| *is-subform-at* *C* (« # *p*) ($\lambda x_\alpha. A$) = *is-subform-at* *C* *p* *A*
| *is-subform-at* - - - = *False*

lemma *is-subform-at-alt-def*:

shows $A' \preceq_p A = (\text{case } \text{subform-at } A \text{ } p \text{ of } \text{Some } B \Rightarrow B = A' \mid \text{None} \Rightarrow \text{False})$
<proof>

lemma *superform-existence*:

assumes $B \preceq_p @ [d] C$
obtains *A* **where** $B \preceq_{[d]} A$ **and** $A \preceq_p C$
<proof>

lemma *subform-at-subforms-con*:

assumes $\{c\}_\alpha \preceq_p C$
shows $\# A. A \preceq_p @ [d] C$
<proof>

lemma *subform-at-subforms-var*:

assumes $x_\alpha \preceq_p C$
shows $\# A. A \preceq_p @ [d] C$
<proof>

lemma *subform-at-subforms-app*:

assumes $A \cdot B \preceq_p C$
shows $A \preceq_p @ [\ll] C$ **and** $B \preceq_p @ [\gg] C$
<proof>

lemma *subform-at-subforms-abs*:

assumes $\lambda x_\alpha. A \preceq_p C$
shows $A \preceq_p @ [\ll] C$
<proof>

lemma *is-subform-implies-in-positions*:

assumes $B \preceq_p A$
shows $p \in \text{positions } A$
<proof>

lemma *subform-size-decrease*:

assumes $A \preceq_p B$ **and** $p \neq []$
shows $\text{form-size } A < \text{form-size } B$
<proof>

lemma *strict-subform-is-not-form*:

assumes $p \neq []$ **and** $A' \preceq_p A$
shows $A' \neq A$
 ⟨proof⟩

lemma *no-right-subform-of-abs*:
shows $\nexists B. B \preceq \#_p \lambda x_\alpha. A$
 ⟨proof⟩

lemma *subforms-from-var*:
assumes $A \preceq_p x_\alpha$
shows $A = x_\alpha$ **and** $p = []$
 ⟨proof⟩

lemma *subforms-from-con*:
assumes $A \preceq_p \{c\}_\alpha$
shows $A = \{c\}_\alpha$ **and** $p = []$
 ⟨proof⟩

lemma *subforms-from-app*:
assumes $A \preceq_p B \cdot C$
shows
 $(A = B \cdot C \wedge p = []) \vee$
 $(A \neq B \cdot C \wedge$
 $(\exists p' \in \text{positions } B. p = \ll \# p' \wedge A \preceq_{p'} B) \vee (\exists p' \in \text{positions } C. p = \gg \# p' \wedge A \preceq_{p'} C))$
 ⟨proof⟩

lemma *subforms-from-abs*:
assumes $A \preceq_p \lambda x_\alpha. B$
shows $(A = \lambda x_\alpha. B \wedge p = []) \vee (A \neq \lambda x_\alpha. B \wedge (\exists p' \in \text{positions } B. p = \ll \# p' \wedge A \preceq_{p'} B))$
 ⟨proof⟩

lemma *leftmost-subform-in-generalized-app*:
shows $B \preceq_{\text{replicate}} (\text{length } As) \ll \cdot^{\mathcal{Q}} \star B As$
 ⟨proof⟩

lemma *self-subform-is-at-top*:
assumes $A \preceq_p A$
shows $p = []$
 ⟨proof⟩

lemma *at-top-is-self-subform*:
assumes $A \preceq [] B$
shows $A = B$
 ⟨proof⟩

lemma *is-subform-at-uniqueness*:
assumes $B \preceq_p A$ **and** $C \preceq_p A$
shows $B = C$
 ⟨proof⟩

lemma *is-subform-at-existence*:

assumes $p \in \text{positions } A$
obtains B **where** $B \preceq_p A$
<proof>

lemma *is-subform-at-transitivity*:

assumes $A \preceq_{p_1} B$ **and** $B \preceq_{p_2} C$
shows $A \preceq_{p_2 @ p_1} C$
<proof>

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>

lemma *loop-subform-impossibility*:

assumes $B \preceq_p A$
and *strict-prefix* $p' p$
shows $\neg B \preceq_{p'} A$
<proof>

lemma *nested-subform-size-decreases*:

assumes *strict-prefix* $p' p$
and $B \preceq_{p'} A$
and $C \preceq_p A$
shows *form-size* $C < \text{form-size } B$
<proof>

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 *<proof>*

end

instance *form* :: *preorder*
<proof>

lemma *position-subform-existence-equivalence*:
shows $p \in \text{positions } A \longleftrightarrow (\exists A'. A' \preceq_p A)$
<proof>

lemma *position-prefix-is-position*:
assumes $p \in \text{positions } A$ **and** *prefix* $p' p$
shows $p' \in \text{positions } A$
<proof>

2.7 Free and bound variables

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

overloading

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

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

begin

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

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

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

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

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

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

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

end

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

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

lemma *vars-form-finiteness*:

fixes $A :: \text{form}$

shows *finite* $(\text{vars } A)$

<proof>

lemma *vars-form-set-finiteness*:

fixes $S :: \text{form set}$

assumes *finite* S

shows *finite* $(\text{vars } S)$

<proof>

lemma *form-var-names-finiteness*:

fixes $A :: \text{form}$

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

<proof>

lemma *form-set-var-names-finiteness*:

fixes $S :: \text{form set}$

assumes *finite S*

shows *finite (var-names S)*

<proof>

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

overloading

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

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

begin

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

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

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

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

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

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

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

end

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

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

lemma *free-vars-form-finiteness*:

fixes $A :: \text{form}$

shows *finite (free-vars A)*

<proof>

lemma *free-vars-of-generalized-app*:

shows $\text{free-vars } (\cdot^{\mathcal{Q}}_* A Bs) = \text{free-vars } A \cup \text{free-vars } (\text{lset } Bs)$

<proof>

lemma *free-vars-of-generalized-abs*:

shows $\text{free-vars } (\lambda^{\mathcal{Q}}_* vs A) = \text{free-vars } A - \text{lset } vs$

<proof>

lemma *free-vars-in-all-vars*:

fixes $A :: \text{form}$

shows $\text{free-vars } A \subseteq \text{vars } A$

<proof>

lemma *free-vars-in-all-vars-set*:

fixes $S :: \text{form set}$

shows $\text{free-vars } S \subseteq \text{vars } S$

<proof>

lemma *singleton-form-set-vars*:

shows $\text{vars } \{FVar\ y\} = \{y\}$
 ⟨*proof*⟩

fun *bound-vars where*

$\text{bound-vars } (x_\alpha) = \{\}$
 | $\text{bound-vars } (\{c\}_\alpha) = \{\}$
 | $\text{bound-vars } (B \cdot C) = \text{bound-vars } B \cup \text{bound-vars } C$
 | $\text{bound-vars } (\lambda x_\alpha. B) = \{(x, \alpha)\} \cup \text{bound-vars } B$

lemma *vars-is-free-and-bound-vars*:

shows $\text{vars } A = \text{free-vars } A \cup \text{bound-vars } A$
 ⟨*proof*⟩

fun *binders-at :: form \Rightarrow position \Rightarrow var set where*

$\text{binders-at } (A \cdot B) (\ll \# p) = \text{binders-at } A\ p$
 | $\text{binders-at } (A \cdot B) (\gg \# p) = \text{binders-at } B\ p$
 | $\text{binders-at } (\lambda x_\alpha. A) (\ll \# p) = \{(x, \alpha)\} \cup \text{binders-at } A\ p$
 | $\text{binders-at } A\ [] = \{\}$
 | $\text{binders-at } A\ p = \{\}$

lemma *binders-at-concat*:

assumes $A' \preceq_p A$
shows $\text{binders-at } A\ (p @ p') = \text{binders-at } A\ p \cup \text{binders-at } A'\ p'$
 ⟨*proof*⟩

2.8 Free and bound occurrences

definition *occurs-at :: var \Rightarrow position \Rightarrow form \Rightarrow bool where*

[*iff*]: $\text{occurs-at } v\ p\ B \longleftrightarrow (FVar\ v \preceq_p B)$

lemma *occurs-at-alt-def*:

shows $\text{occurs-at } v\ []\ (FVar\ v') \longleftrightarrow (v = v')$
and $\text{occurs-at } v\ p\ (\{c\}_\alpha) \longleftrightarrow \text{False}$
and $\text{occurs-at } v\ (\ll \# p)\ (A \cdot B) \longleftrightarrow \text{occurs-at } v\ p\ A$
and $\text{occurs-at } v\ (\gg \# p)\ (A \cdot B) \longleftrightarrow \text{occurs-at } v\ p\ B$
and $\text{occurs-at } v\ (\ll \# p)\ (\lambda x_\alpha. A) \longleftrightarrow \text{occurs-at } v\ p\ A$
and $\text{occurs-at } v\ (d \# p)\ (FVar\ v') \longleftrightarrow \text{False}$
and $\text{occurs-at } v\ (\gg \# p)\ (\lambda x_\alpha. A) \longleftrightarrow \text{False}$
and $\text{occurs-at } v\ []\ (A \cdot B) \longleftrightarrow \text{False}$
and $\text{occurs-at } v\ []\ (\lambda x_\alpha. A) \longleftrightarrow \text{False}$
 ⟨*proof*⟩

definition *occurs :: var \Rightarrow form \Rightarrow bool where*

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

lemma *occurs-in-vars*:

assumes $\text{occurs } v\ A$

shows $v \in \text{vars } A$
 ⟨proof⟩

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

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

lemma *in-scope-of-abs-alt-def*:
shows
 $\text{in-scope-of-abs } v \ p \ B$
 \longleftrightarrow
 $p \neq [] \wedge (\exists p' \in \text{positions } B. \exists C. \text{strict-prefix } p' \ p \wedge FAbs \ v \ C \preceq_{p'} B)$
 ⟨proof⟩

lemma *in-scope-of-abs-in-left-app*:
shows $\text{in-scope-of-abs } v \ (\ll \# \ p) \ (A \bullet B) \longleftrightarrow \text{in-scope-of-abs } v \ p \ A$
 ⟨proof⟩

lemma *in-scope-of-abs-in-right-app*:
shows $\text{in-scope-of-abs } v \ (\gg \# \ p) \ (A \bullet B) \longleftrightarrow \text{in-scope-of-abs } v \ p \ B$
 ⟨proof⟩

lemma *in-scope-of-abs-in-app*:
assumes $\text{in-scope-of-abs } v \ p \ (A \bullet B)$
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⟩

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

lemma *in-scope-of-abs-in-abs*:
shows $\text{in-scope-of-abs } v \ (\ll \# \ p) \ (FAbs \ v' \ B) \longleftrightarrow v = v' \vee \text{in-scope-of-abs } v \ p \ B$
 ⟨proof⟩

lemma *not-in-scope-of-abs-in-var*:
shows \neg *in-scope-of-abs* v p ($FVar$ v')
 \langle *proof* \rangle

lemma *in-scope-of-abs-in-vars*:
assumes *in-scope-of-abs* v p A
shows $v \in vars$ A
 \langle *proof* \rangle

lemma *binders-at-alt-def*:
assumes $p \in positions$ A
shows *binders-at* A $p = \{v \mid v. in-scope-of-abs$ v p $A\}$
 \langle *proof* \rangle

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

lemma *not-is-bound-at-in-var*:
shows \neg *is-bound-at* v p ($FVar$ v')
 \langle *proof* \rangle

lemma *not-is-bound-at-in-con*:
shows \neg *is-bound-at* v p ($FCon$ k)
 \langle *proof* \rangle

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

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

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 $p' C)$
 \langle *proof* \rangle

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
 \langle *proof* \rangle

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 $p' B$
 \langle *proof* \rangle

lemma *is-bound-at-to-abs*:

assumes $(v = v' \wedge \text{occurs-at } v \text{ } p \text{ } B) \vee \text{is-bound-at } v \text{ } p \text{ } B$

shows $\text{is-bound-at } v \text{ } (\ll \# p) (FAbs \ v' \ B)$

<proof>

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

assumes $p \in \text{positions } A$

and $\text{is-bound-at } v \text{ } p \ A \vee v \in \text{binders-at } A \ p$

shows $v \in \text{bound-vars } A$

<proof>

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

assumes $v \in \text{bound-vars } A$

obtains p **where** $p \in \text{positions } A$ **and** $\text{is-bound-at } v \text{ } p \ A \vee v \in \text{binders-at } A \ p$

<proof>

lemma *bound-vars-alt-def*:

shows $\text{bound-vars } A = \{v \mid v \text{ } p. \ p \in \text{positions } A \wedge (\text{is-bound-at } v \text{ } p \ A \vee v \in \text{binders-at } A \ p)\}$

<proof>

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

[iff]: $\text{is-free-at } v \text{ } p \ B \longleftrightarrow \text{occurs-at } v \text{ } p \ B \wedge \neg \text{in-scope-of-abs } v \text{ } p \ B$

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

shows $\text{is-free-at } v \ \square \ (FVar \ v') \longleftrightarrow v = v'$

<proof>

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

shows $\neg \text{is-free-at } v \ \square \ (\{\!|c|\!\}_\alpha)$

<proof>

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

shows $\text{is-free-at } v \ (\ll \# p) (B \cdot C) \longleftrightarrow \text{is-free-at } v \text{ } p \ B$

<proof>

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

shows $\text{is-free-at } v \ (\gg \# p) (B \cdot C) \longleftrightarrow \text{is-free-at } v \text{ } p \ C$

<proof>

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

assumes $\text{is-free-at } v \text{ } p \ (B \cdot C)$

obtains p' **where** $(p = \ll \# p' \wedge \text{is-free-at } v \text{ } p' \ B) \vee (p = \gg \# p' \wedge \text{is-free-at } v \text{ } p' \ C)$

<proof>

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

assumes $\text{is-free-at } v \ (\ll \# p) (FAbs \ v' \ B)$

shows $\text{is-free-at } v \text{ } p \ B$

<proof>

lemma *is-free-at-from-absE*:
assumes *is-free-at v p (FAbs v' B)*
obtains *p' where p = « # p' and is-free-at v p' B*
⟨*proof*⟩

lemma *is-free-at-to-abs*:
assumes *is-free-at v p B and v ≠ v'*
shows *is-free-at v (« # p) (FAbs v' B)*
⟨*proof*⟩

lemma *is-free-at-in-free-vars*:
assumes *p ∈ positions A and is-free-at v p A*
shows *v ∈ free-vars A*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *free-vars-alt-def*:
shows *free-vars A = {v | v p. p ∈ positions A ∧ is-free-at v p A}*
⟨*proof*⟩

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

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

lemma *is-bound-in-app-homomorphism*:
shows *is-bound v (A • B) ⟷ is-bound v A ∨ is-bound v B*
⟨*proof*⟩

lemma *is-bound-in-abs-body*:
assumes *is-bound v A*
shows *is-bound v (λx_α. A)*
⟨*proof*⟩

lemma *absent-var-is-not-bound*:
assumes *v ∉ vars A*
shows \neg *is-bound v A*
⟨*proof*⟩

lemma *bound-vars-alt-def2*:
shows *bound-vars A = {v ∈ vars A. is-bound v A}*
⟨*proof*⟩

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

2.9 Free variables for a formula in another formula

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

[*iff*]: *is-free-for* A v $B \iff$
 (
 $\forall v' \in \text{free-vars } A.$
 $\forall p \in \text{positions } B.$
 $\text{is-free-at } v$ 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
 ⟨*proof*⟩

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

shows *is-free-for* A v (x_α)
 ⟨*proof*⟩

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

shows *is-free-for* A v $(\{c\}_\alpha)$
 ⟨*proof*⟩

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

assumes *is-free-for* A v $(B \cdot C)$
shows *is-free-for* A v B **and** *is-free-for* A v C
 ⟨*proof*⟩

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 $(B \cdot C)$
 ⟨*proof*⟩

lemma *is-free-for-in-app*:

shows *is-free-for* A v $(B \cdot C) \iff \text{is-free-for } A$ v $B \wedge \text{is-free-for } A$ v C
 ⟨*proof*⟩

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

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

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

assumes *is-free-for* A v $(\lambda x_\alpha. B)$ **and** $v \neq (x, \alpha)$
shows *is-free-for* A v B
 ⟨*proof*⟩

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

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

$\langle proof \rangle$

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

assumes *free-vars* $B = \{\}$

shows *is-free-for* $A \ v \ B$

$\langle proof \rangle$

lemma *is-free-for-alt-def*:

shows

is-free-for $A \ v \ B$

\longleftrightarrow

(

$\nexists p.$

(

$p \in \text{positions } B \wedge \text{is-free-at } v \ p \ B \wedge p \neq [] \wedge$

$(\exists v' \in \text{free-vars } A. \exists p' \ C. \text{strict-prefix } p' \ p \wedge \text{FAbs } v' \ C \preceq_{p'} B)$

)

)

$\langle proof \rangle$

lemma *binding-var-not-free-for-in-abs*:

assumes *is-free* $x \ B$ **and** $x \neq w$

shows $\neg \text{is-free-for } (\text{FVar } w) \ x \ (\text{FAbs } w \ B)$

$\langle proof \rangle$

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

assumes $x \notin \text{vars } A$

shows *is-free-for* $(\text{FVar } x) \ y \ A$

$\langle proof \rangle$

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

assumes $x \notin \text{vars } A$

shows *is-free-for* $B \ x \ A$

$\langle proof \rangle$

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 $\neg \text{is-free-for } A \ v \ (\text{FAbs } v' \ B)$

$\langle proof \rangle$

2.10 Replacement of subformulas

inductive

is-replacement-at :: *form* \Rightarrow *position* \Rightarrow *form* \Rightarrow *form* \Rightarrow *bool*

((λ - λ - \leftarrow - \rightarrow) \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*: $(G \cdot H) \langle \# \ p \leftarrow C \rangle \triangleright (G' \cdot H)$ **if** $p \in \text{positions } G$ **and** $G \langle p \leftarrow C \rangle \triangleright G'$

| *replace-right-app*: $(G \cdot H) \langle \# \ p \leftarrow C \rangle \triangleright (G \cdot H')$ **if** $p \in \text{positions } H$ **and** $H \langle p \leftarrow C \rangle \triangleright H'$

| *replace-abs*: $(\lambda x \gamma. E) \langle \langle \# p \leftarrow C \rangle \rangle \triangleright (\lambda x \gamma. E')$ **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 $p \in \text{positions } C$

$\langle \text{proof} \rangle$

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

lemma *is-replacement-at-existence*:

assumes $p \in \text{positions } C$

obtains D **where** $C \langle p \leftarrow A \rangle \triangleright D$

$\langle \text{proof} \rangle$

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

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

shows $A \preceq_p D$

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

$\langle \text{proof} \rangle$

lemma *is-replacement-at-binders*:

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

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

$\langle \text{proof} \rangle$

lemma *is-replacement-at-occurs*:

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

and $\neg \text{prefix } p' p$ **and** $\neg \text{prefix } p p'$

shows $\text{occurs-at } v p' C \longleftrightarrow \text{occurs-at } v p' D$

$\langle \text{proof} \rangle$

lemma *fresh-var-replacement-position-uniqueness*:

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

and $C \langle p \leftarrow FVar v \rangle \triangleright G$

and $\text{occurs-at } v p' G$

shows $p' = p$

$\langle \text{proof} \rangle$

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

assumes $C \langle p \leftarrow A \rangle \triangleright D$ **and** $\text{prefix } p p'$ **and** $p' \in \text{positions } D$

obtains p'' **where** $p' = p @ p''$ **and** $p'' \in \text{positions } A$

$\langle \text{proof} \rangle$

lemma *replacement-override*:

assumes $C \langle p \leftarrow B \rangle \triangleright D$ **and** $C \langle p \leftarrow A \rangle \triangleright F$

shows $D \langle p \leftarrow A \rangle \triangleright F$

$\langle \text{proof} \rangle$

lemma *leftmost-subform-in-generalized-app-replacement*:

shows $(\cdot^Q_* C As) \langle \text{replicate } (\text{length } As) \ll \leftarrow D \rangle \triangleright (\cdot^Q_* D As)$
<proof>

2.11 Logical constants

abbreviation *(input)* \mathfrak{x} **where** $\mathfrak{x} \equiv 0$

abbreviation *(input)* \mathfrak{y} **where** $\mathfrak{y} \equiv \text{Suc } \mathfrak{x}$

abbreviation *(input)* \mathfrak{z} **where** $\mathfrak{z} \equiv \text{Suc } \mathfrak{y}$

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}_ι **where** $\mathfrak{c}_\iota \equiv \text{Suc } \mathfrak{c}_Q$

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* $\equiv (\mathfrak{c}_\iota, (i \rightarrow o) \rightarrow i)$

definition *Q* $:: \text{type} \Rightarrow \text{form } (Q\cdot)$ **where**

[simp]: $Q_\alpha = \text{FCon } (Q\text{-constant-of-type } \alpha)$

definition *iota* $:: \text{form } (\iota)$ **where**

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

definition *is-Q-constant-of-type* $:: \text{con} \Rightarrow \text{type} \Rightarrow \text{bool}$ **where**

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

definition *is-iota-constant* $:: \text{con} \Rightarrow \text{bool}$ **where**

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

definition *is-logical-constant* $:: \text{con} \Rightarrow \text{bool}$ **where**

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

definition *type-of-Q-constant* $:: \text{con} \Rightarrow \text{type}$ **where**

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

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

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

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

and $\text{is-iota-constant } p \Longrightarrow P$

shows P

<proof>

2.12 Definitions and abbreviations

definition *equality-of-type* $:: \text{form} \Rightarrow \text{type} \Rightarrow \text{form} \Rightarrow \text{form} ((- =_/ -) [103, 0, 103] 102)$ **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 ($(\lambda \forall _ _ / _)$ [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 $A \preceq_{foldr} (\lambda _ p. [»,«] @ p) vs [] \forall^{\mathcal{Q}}_\star vs A$

<proof>

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

assumes $vs \neq []$

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

<proof>

lemma *false-is-forall*:

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

<proof>

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

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

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

(

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

)

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 $\wedge^{\mathcal{Q}}_\star [A_1, \dots, A_n]$ 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 ($\wedge^{\mathcal{Q}}_\star _ _$ [0] 131) **where**

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

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* $(\exists x_{\alpha}. A) = \text{free-vars } A - \{(x, \alpha)\}$
 ⟨proof⟩

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

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* $\alpha = (\lambda f. f \in wffs_{\alpha})$
 ⟨proof⟩

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 $\text{length } As = \text{length } ts$

and $\forall k < \text{length } As. As ! k \in \text{wffs}_{ts ! k}$

and $B \in \text{wffs}_{\text{foldr } (\rightarrow) ts \beta}$

shows $\bullet^{\mathcal{Q}}_{\star} B As \in \text{wffs}_{\beta}$

<proof>

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

assumes $B \in \text{wffs}_{\beta}$

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

<proof>

lemma *Q-wff* [*intro*]:

shows $Q_{\alpha} \in \text{wffs}_{\alpha \rightarrow \alpha \rightarrow o}$

<proof>

lemma *iota-wff* [*intro*]:

shows $\iota \in \text{wffs}_{(i \rightarrow o) \rightarrow i}$

<proof>

lemma *equality-wff* [*intro*]:

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

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

<proof>

lemma *equivalence-wff* [*intro*]:

assumes $A \in \text{wffs}_o$ **and** $B \in \text{wffs}_o$

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

<proof>

lemma *true-wff* [*intro*]:

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

<proof>

lemma *false-wff* [*intro*]:

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

<proof>

lemma *pi-wff* [*intro*]:

shows $\prod \alpha \in \text{wffs}_{(\alpha \rightarrow o) \rightarrow o}$

<proof>

lemma *forall-wff* [*intro*]:

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

shows $\forall x_\alpha. A \in wffs_o$
<proof>

lemma *generalized-forall-wff* [intro]:
assumes $B \in wffs_o$
shows $\forall^Q_* vs B \in wffs_o$
<proof>

lemma *conj-fun-wff* [intro]:
shows $\wedge_{o \rightarrow o \rightarrow o} \in wffs_{o \rightarrow o \rightarrow o}$
<proof>

lemma *conj-op-wff* [intro]:
assumes $A \in wffs_o$ **and** $B \in wffs_o$
shows $A \wedge^Q B \in wffs_o$
<proof>

lemma *imp-fun-wff* [intro]:
shows $\supset_{o \rightarrow o \rightarrow o} \in wffs_{o \rightarrow o \rightarrow o}$
<proof>

lemma *imp-op-wff* [intro]:
assumes $A \in wffs_o$ **and** $B \in wffs_o$
shows $A \supset^Q B \in wffs_o$
<proof>

lemma *neg-wff* [intro]:
assumes $A \in wffs_o$
shows $\sim^Q A \in wffs_o$
<proof>

lemma *disj-fun-wff* [intro]:
shows $\vee_{o \rightarrow o \rightarrow o} \in wffs_{o \rightarrow o \rightarrow o}$
<proof>

lemma *disj-op-wff* [intro]:
assumes $A \in wffs_o$ **and** $B \in wffs_o$
shows $A \vee^Q B \in wffs_o$
<proof>

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

lemma *inequality-wff* [intro]:
assumes $A \in wffs_\alpha$ **and** $B \in wffs_\alpha$
shows $A \neq_\alpha B \in wffs_o$
<proof>

lemma *wffs-from-app*:
assumes $A \cdot B \in \text{wffs}_\beta$
obtains α **where** $A \in \text{wffs}_{\alpha \rightarrow \beta}$ **and** $B \in \text{wffs}_\alpha$
<proof>

lemma *wffs-from-generalized-app*:
assumes $\cdot^{\mathcal{Q}}_* B \text{ As} \in \text{wffs}_\beta$
obtains ts
where $\text{length } ts = \text{length } \text{As}$
and $\forall k < \text{length } \text{As}. \text{As } ! k \in \text{wffs}_{ts ! k}$
and $B \in \text{wffs}_{\text{foldr } (\rightarrow) ts \beta}$
<proof>

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

lemma *wffs-from-equality*:
assumes $A =_\alpha B \in \text{wffs}_\sigma$
shows $A \in \text{wffs}_\alpha$ **and** $B \in \text{wffs}_\alpha$
<proof>

lemma *wffs-from-equivalence*:
assumes $A \equiv^{\mathcal{Q}} B \in \text{wffs}_\sigma$
shows $A \in \text{wffs}_\sigma$ **and** $B \in \text{wffs}_\sigma$
<proof>

lemma *wffs-from-forall*:
assumes $\forall x_\alpha. A \in \text{wffs}_\sigma$
shows $A \in \text{wffs}_\sigma$
<proof>

lemma *wffs-from-conj-fun*:
assumes $\wedge_{\sigma \rightarrow \sigma \rightarrow \sigma} \cdot A \cdot B \in \text{wffs}_\sigma$
shows $A \in \text{wffs}_\sigma$ **and** $B \in \text{wffs}_\sigma$
<proof>

lemma *wffs-from-conj-op*:
assumes $A \wedge^{\mathcal{Q}} B \in \text{wffs}_\sigma$
shows $A \in \text{wffs}_\sigma$ **and** $B \in \text{wffs}_\sigma$
<proof>

lemma *wffs-from-imp-fun*:
assumes $\supset_{\sigma \rightarrow \sigma \rightarrow \sigma} \cdot A \cdot B \in \text{wffs}_\sigma$
shows $A \in \text{wffs}_\sigma$ **and** $B \in \text{wffs}_\sigma$
<proof>

lemma *wffs-from-imp-op*:
assumes $A \supset^{\mathcal{Q}} B \in \text{wffs}_o$
shows $A \in \text{wffs}_o$ **and** $B \in \text{wffs}_o$
<proof>

lemma *wffs-from-neg*:
assumes $\sim^{\mathcal{Q}} A \in \text{wffs}_o$
shows $A \in \text{wffs}_o$
<proof>

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** $B \in \text{wffs}_o$
<proof>

lemma *wffs-from-disj-op*:
assumes $A \vee^{\mathcal{Q}} B \in \text{wffs}_o$
shows $A \in \text{wffs}_o$ **and** $B \in \text{wffs}_o$
<proof>

lemma *wffs-from-exists*:
assumes $\exists x_{\alpha}. A \in \text{wffs}_o$
shows $A \in \text{wffs}_o$
<proof>

lemma *wffs-from-inequality*:
assumes $A \neq_{\alpha} B \in \text{wffs}_o$
shows $A \in \text{wffs}_{\alpha}$ **and** $B \in \text{wffs}_{\alpha}$
<proof>

lemma *wff-has-unique-type*:
assumes $A \in \text{wffs}_{\alpha}$ **and** $A \in \text{wffs}_{\beta}$
shows $\alpha = \beta$
<proof>

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 $\mathcal{P} A$
<proof>

lemma *diff-types-implies-diff-wffs*:
assumes $A \in \text{wffs}_{\alpha}$ **and** $B \in \text{wffs}_{\beta}$
and $\alpha \neq \beta$
shows $A \neq B$
<proof>

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 $(\bullet^{\mathcal{Q}}_{\star} B Cs)$
⟨proof⟩

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 $(B =_{\alpha} C)$
⟨proof⟩

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 $(B \equiv^{\mathcal{Q}} C)$
⟨proof⟩

lemma *is-free-for-in-true* [intro]:
shows *is-free-for* A v (T_o)
⟨proof⟩

lemma *is-free-for-in-false* [intro]:
shows *is-free-for* A v (F_o)
⟨proof⟩

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 $(\forall x_{\alpha}. B)$
⟨proof⟩

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 $(\forall^{\mathcal{Q}}_{\star} vs B)$
⟨proof⟩

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⟩

lemma *is-free-for-in-imp* [intro]:
assumes *is-free-for* A v B **and** *is-free-for* A v C
shows *is-free-for* A v $(B \supset^{\mathcal{Q}} C)$
⟨proof⟩

lemma *is-free-for-in-neg* [intro]:
assumes *is-free-for* A v B
shows *is-free-for* A v $(\sim^{\mathcal{Q}} B)$
⟨proof⟩

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

assumes *is-free-for* $A \vee B$ **and** *is-free-for* $A \vee C$
shows *is-free-for* $A \vee (B \vee^Q C)$
 \langle *proof* \rangle

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 $C \in \text{wffs}_\beta \longleftrightarrow D \in \text{wffs}_\beta$
 \langle *proof* \rangle

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 $\beta = \gamma$
 \langle *proof* \rangle

Closed formulas and sentences:

definition *is-closed-wff-of-type* :: *form* \Rightarrow *type* \Rightarrow *bool* **where**
 $[\text{iff}]$: *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**
 $[\text{iff}]$: *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**
 $[\text{iff}]$: *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** $\{\$\$ \} A = A$
 \langle *proof* \rangle

lemma *substitution-preserves-typing*:

assumes *is-substitution* ϑ
and $A \in \text{wffs}_\alpha$
shows **S** $\vartheta A \in \text{wffs}_\alpha$
 \langle *proof* \rangle

lemma *derived-substitution-simps:*

shows $\mathbf{S} \vartheta T_o = T_o$
and $\mathbf{S} \vartheta F_o = F_o$
and $\mathbf{S} \vartheta (\prod_{\alpha} B) = \prod_{\alpha} (\mathbf{S} \vartheta B)$
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 $\mathbf{S} \vartheta (\exists x_{\alpha}. B) = (\text{if } (x, \alpha) \notin \text{fmdom}' \vartheta \text{ then } \exists x_{\alpha}. \mathbf{S} \vartheta B \text{ else } \exists x_{\alpha}. \mathbf{S} (\text{fmdrop } (x, \alpha) \vartheta) B)$
 $\langle \text{proof} \rangle$

lemma *generalized-app-substitution:*

shows $\mathbf{S} \vartheta (\cdot^{\mathcal{Q}}_{*} A Bs) = \cdot^{\mathcal{Q}}_{*} (\mathbf{S} \vartheta A) (\text{map } (\lambda B. \mathbf{S} \vartheta B) Bs)$
 $\langle \text{proof} \rangle$

lemma *generalized-abs-substitution:*

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

lemma *generalized-forall-substitution:*

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

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 $\mathbf{S} \{(x, \alpha) \mapsto A\} (\lambda y_{\beta}. B) = \lambda y_{\beta}. (\text{if } (x, \alpha) = (y, \beta) \text{ then } B \text{ else } \mathbf{S} \{(x, \alpha) \mapsto A\} B)$
 $\langle \text{proof} \rangle$

lemma *substitution-preserves-freeness:*

assumes $y \notin \text{free-vars } A$ **and** $y \neq z$
shows $y \notin \text{free-vars } \mathbf{S} \{x \mapsto FVar z\} A$
 $\langle \text{proof} \rangle$

lemma *renaming-substitution-minimal-change:*

assumes $y \notin \text{vars } A$ **and** $y \neq z$
shows $y \notin \text{vars } (\mathbf{S} \{x \mapsto FVar z\} A)$
 $\langle \text{proof} \rangle$

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

assumes $v \notin \text{free-vars } A$
shows $\mathbf{S} \{v \mapsto B\} A = A$
 $\langle \text{proof} \rangle$

lemma *identity-singleton-substitution-neutrality:*

shows $\mathbf{S} \{v \mapsto FVar\ v\} A = A$

$\langle proof \rangle$

lemma *free-var-in-renaming-substitution:*

assumes $x \neq y$

shows $(x, \alpha) \notin \text{free-vars} (\mathbf{S} \{(x, \alpha) \mapsto y_\alpha\} B)$

$\langle proof \rangle$

lemma *renaming-substitution-preserves-form-size:*

shows $\text{form-size} (\mathbf{S} \{v \mapsto FVar\ v'\} A) = \text{form-size } A$

$\langle proof \rangle$

The following lemma corresponds to X5100 in [2]:

lemma *substitution-composability:*

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

shows $\mathbf{S} \{v' \mapsto A\} \mathbf{S} \{v \mapsto FVar\ v'\} B = \mathbf{S} \{v \mapsto A\} B$

$\langle proof \rangle$

The following lemma corresponds to X5101 in [2]:

lemma *renaming-substitution-composability:*

assumes $z \notin \text{free-vars } A$ **and** *is-free-for* $(FVar\ z)\ x\ A$

shows $\mathbf{S} \{z \mapsto FVar\ y\} \mathbf{S} \{x \mapsto FVar\ z\} A = \mathbf{S} \{x \mapsto FVar\ y\} A$

$\langle proof \rangle$

lemma *absent-vars-substitution-preservation:*

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

and $\forall v' \in \text{fndom}'\ \vartheta. v \notin \text{vars} (\vartheta\ \$\$!\ v')$

shows $v \notin \text{vars} (\mathbf{S}\ \vartheta\ A)$

$\langle proof \rangle$

lemma *substitution-free-absorption:*

assumes $\vartheta\ \$\$ v = \text{None}$ **and** $v \notin \text{free-vars } B$

shows $\mathbf{S} (\{v \mapsto A\} ++_f\ \vartheta) B = \mathbf{S}\ \vartheta\ B$

$\langle proof \rangle$

lemma *substitution-absorption:*

assumes $\vartheta\ \$\$ v = \text{None}$ **and** $v \notin \text{vars } B$

shows $\mathbf{S} (\{v \mapsto A\} ++_f\ \vartheta) B = \mathbf{S}\ \vartheta\ B$

$\langle proof \rangle$

lemma *is-free-for-with-renaming-substitution:*

assumes *is-free-for* $A\ x\ B$

and $y \notin \text{vars } B$

and $x \notin \text{fndom}'\ \vartheta$

and $\forall v \in \text{fndom}'\ \vartheta. y \notin \text{vars} (\vartheta\ \$\$!\ v)$

and $\forall v \in \text{fndom}'\ \vartheta. \text{is-free-for} (\vartheta\ \$\$!\ v)\ v\ B$

shows *is-free-for* $A\ y\ (\mathbf{S} (\{x \mapsto FVar\ y\} ++_f\ \vartheta) B)$

$\langle proof \rangle$

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

lemma *substitution-fusion*:

assumes *is-substitution* ϑ **and** *is-substitution* $\{v \mapsto A\}$
and $\vartheta \ \$\$ v = \text{None}$ **and** $\forall v' \in \text{fmdom}' \vartheta. v \notin \text{vars} (\vartheta \ \$\$! v')$
shows $\mathbf{S} \{v \mapsto A\} \mathbf{S} \vartheta B = \mathbf{S} (\{v \mapsto A\} ++_f \vartheta) B$

<proof>

lemma *updated-substitution-is-substitution*:

assumes $v \notin \text{fmdom}' \vartheta$ **and** *is-substitution* $(\vartheta(v \mapsto A))$
shows *is-substitution* ϑ

<proof>

definition *is-renaming-substitution where*

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

The following lemma proves that $\S_{y_{\alpha_1}^1 \dots y_{\alpha_n}^n}^{x_{\alpha_1}^1 \dots x_{\alpha_n}^n} B = \S_{y_{\alpha_1}^1}^{x_{\alpha_1}^1} \dots \S_{y_{\alpha_n}^n}^{x_{\alpha_n}^n} B$ 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* $(\text{map } \text{fst } ps)$ **and** *distinct* $(\text{map } \text{snd } ps)$
and $\text{vars} (\text{fmrans}' \vartheta) \cap (\text{fmdom}' \vartheta \cup \text{vars } B) = \{\}$
shows $\mathbf{S} \vartheta B = \text{foldr} (\lambda(x, y) C. \mathbf{S} \{x \mapsto y\} C) ps B$

<proof>

lemma *free-vars-agreement-substitution-equality*:

assumes $\text{fmdom}' \vartheta = \text{fmdom}' \vartheta'$
and $\forall v \in \text{free-vars } A \cap \text{fmdom}' \vartheta. \vartheta \ \$\$! v = \vartheta' \ \$\$! v$
shows $\mathbf{S} \vartheta A = \mathbf{S} \vartheta' A$

<proof>

The following lemma proves that $\S_{A_\alpha}^{x_\alpha} \S_{A_{\alpha_1}^1 \dots A_{\alpha_n}^n}^{x_{\alpha_1}^1 \dots x_{\alpha_n}^n} B = \S_{A_\alpha}^{x_\alpha} \S_{A_\alpha}^{x_{\alpha_1}^1} \dots \S_{A_\alpha}^{x_{\alpha_n}^n} B$ 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 $\forall v' \in \text{fmdom}' \vartheta. \text{is-free-for} (\vartheta \ \$\$! v') v' B$
shows $\mathbf{S} \{v \mapsto A\} \mathbf{S} \vartheta B = \mathbf{S} (\{v \mapsto A\} ++_f \text{fmmmap} (\lambda A'. \mathbf{S} \{v \mapsto A\} A') \vartheta) B$

<proof>

lemma *vars-range-substitution*:
assumes *is-substitution* ϑ
and $v \notin \text{vars} (\text{fmran}' \vartheta)$
shows $v \notin \text{vars} (\text{fmran}' (\text{fmdrop } w \vartheta))$
 $\langle \text{proof} \rangle$

lemma *excluded-var-from-substitution*:
assumes *is-substitution* ϑ
and $v \notin \text{fmdom}' \vartheta$
and $v \notin \text{vars} (\text{fmran}' \vartheta)$
and $v \notin \text{vars } A$
shows $v \notin \text{vars} (\mathbf{S} \vartheta A)$
 $\langle \text{proof} \rangle$

2.15 Renaming of bound variables

fun *rename-bound-var* :: *var* \Rightarrow *nat* \Rightarrow *form* \Rightarrow *form* **where**
rename-bound-var $v \ y \ (x_\alpha) = x_\alpha$
| *rename-bound-var* $v \ y \ (\llbracket c \rrbracket_\alpha) = \llbracket c \rrbracket_\alpha$
| *rename-bound-var* $v \ y \ (B \cdot C) = \text{rename-bound-var } v \ y \ B \cdot \text{rename-bound-var } v \ y \ C$
| *rename-bound-var* $v \ y \ (\lambda x_\alpha. B) =$
(
 if $(x, \alpha) = v$ then
 $\lambda y_\alpha. \mathbf{S} \{(x, \alpha) \mapsto y_\alpha\} (\text{rename-bound-var } v \ y \ B)$
 else
 $\lambda x_\alpha. (\text{rename-bound-var } v \ y \ B)$
))

lemma *rename-bound-var-preserves-typing*:
assumes $A \in \text{wffs}_\alpha$
shows *rename-bound-var* $(y, \gamma) \ z \ A \in \text{wffs}_\alpha$
 $\langle \text{proof} \rangle$

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 $(y, \gamma) \notin \text{free-vars} (\text{rename-bound-var } (y, \gamma) \ z \ (\lambda y_\gamma. A))$
 $\langle \text{proof} \rangle$

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 $(z, \gamma) \notin \text{free-vars} (\text{rename-bound-var } (y, \gamma) \ z \ A)$
 $\langle \text{proof} \rangle$

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 $(y, \gamma) \notin \text{free-vars } (\text{rename-bound-var } (y, \gamma) z A)$
 <proof>

lemma *old-bound-var-not-occurring-after-renaming:*

assumes $A \in \text{wffs}_\alpha$
and $z_\gamma \neq y_\gamma$
shows $\neg \text{occurs-at } (y, \gamma) p (\mathbf{S} \{(y, \gamma) \mapsto z_\gamma\} (\text{rename-bound-var } (y, \gamma) z A))$
 <proof>

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 $\neg \text{in-scope-of-abs } (z, \gamma) p (\text{rename-bound-var } (y, \gamma) z A)$
 <proof>

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 $\text{is-free-for } (z_\gamma) (y, \gamma) (\text{rename-bound-var } (y, \gamma) z A)$
 <proof>

lemma *renaming-substitution-preserves-bound-vars:*

shows $\text{bound-vars } (\mathbf{S} \{(y, \gamma) \mapsto z_\gamma\} A) = \text{bound-vars } A$
 <proof>

lemma *rename-bound-var-bound-vars:*

assumes $A \in \text{wffs}_\alpha$
and $z_\gamma \neq y_\gamma$
shows $(y, \gamma) \notin \text{bound-vars } (\text{rename-bound-var } (y, \gamma) z A)$
 <proof>

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 $(y, \gamma) \notin \text{vars } (\text{rename-bound-var } (y, \gamma) z A)$
 <proof>

end

3 Boolean Algebra

```

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

```

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

```

hide-const (open) List.set

```

```

definition bool-to-V :: bool  $\Rightarrow$  V where
  bool-to-V = (SOME f. inj f)

```

```

lemma bool-to-V-injectivity [simp]:
  shows inj bool-to-V
  <proof>

```

```

definition bool-from-V :: V  $\Rightarrow$  bool where
  [simp]: bool-from-V = inv bool-to-V

```

```

definition top :: V (T) where
  [simp]: T = bool-to-V True

```

```

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

```

```

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

```

```

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

```

```

definition conjunction :: V  $\Rightarrow$  V  $\Rightarrow$  V (infixr  $\wedge$  136) where
  [simp]: p  $\wedge$  q = bool-to-V (bool-from-V p  $\wedge$  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  $\in$  elts B and q  $\in$  elts B and r  $\in$  elts 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 $p = \mathbf{T} \vee p = \mathbf{F}$
<proof>

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
<proof>

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*: $T_o \in \text{pwffs}$
 F -*pwff*: $F_o \in \text{pwffs}$
 var -*pwff*: $p_o \in \text{pwffs}$
 neg -*pwff*: $\sim^Q A \in \text{pwffs}$ **if** $A \in \text{pwffs}$
 $conj$ -*pwff*: $A \wedge^Q B \in \text{pwffs}$ **if** $A \in \text{pwffs}$ **and** $B \in \text{pwffs}$
 $disj$ -*pwff*: $A \vee^Q B \in \text{pwffs}$ **if** $A \in \text{pwffs}$ **and** $B \in \text{pwffs}$
 imp -*pwff*: $A \supset^Q B \in \text{pwffs}$ **if** $A \in \text{pwffs}$ **and** $B \in \text{pwffs}$
 eqv -*pwff*: $A \equiv^Q B \in \text{pwffs}$ **if** $A \in \text{pwffs}$ **and** $B \in \text{pwffs}$

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 $F_o \neq A \wedge^{\mathcal{Q}} B$
and $F_o \neq A \vee^{\mathcal{Q}} B$
and $F_o \neq A \supset^{\mathcal{Q}} B$
and $F_o \neq A \equiv^{\mathcal{Q}} B$
and $p_o \neq \sim^{\mathcal{Q}} A$
and $p_o \neq A \wedge^{\mathcal{Q}} B$
and $p_o \neq A \vee^{\mathcal{Q}} B$
and $p_o \neq A \supset^{\mathcal{Q}} B$
and $p_o \neq A \equiv^{\mathcal{Q}} B$
and $\sim^{\mathcal{Q}} A \neq B \wedge^{\mathcal{Q}} C$
and $\sim^{\mathcal{Q}} A \neq B \vee^{\mathcal{Q}} C$
and $\sim^{\mathcal{Q}} A \neq B \supset^{\mathcal{Q}} C$
and $\neg (B = F_o \wedge A = C) \implies \sim^{\mathcal{Q}} A \neq B \equiv^{\mathcal{Q}} C$ — $\sim^{\mathcal{Q}} A$ is the same as $F_o \equiv^{\mathcal{Q}} A$
and $A \wedge^{\mathcal{Q}} B \neq C \vee^{\mathcal{Q}} D$
and $A \wedge^{\mathcal{Q}} B \neq C \supset^{\mathcal{Q}} D$
and $A \wedge^{\mathcal{Q}} B \neq C \equiv^{\mathcal{Q}} D$
and $A \vee^{\mathcal{Q}} B \neq C \supset^{\mathcal{Q}} D$
and $A \vee^{\mathcal{Q}} B \neq C \equiv^{\mathcal{Q}} D$
and $A \supset^{\mathcal{Q}} B \neq C \equiv^{\mathcal{Q}} D$
<proof>

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 $A \equiv^{\mathcal{Q}} B = A' \equiv^{\mathcal{Q}} B' \implies A = A' \wedge B = B'$
<proof>

lemma *pwff-from-neg-pwff* [*elim!*]:

assumes $\sim^{\mathcal{Q}} A \in \text{pwffs}$
shows $A \in \text{pwffs}$
<proof>

lemma *pwffs-from-conj-pwff* [*elim!*]:

assumes $A \wedge^{\mathcal{Q}} B \in \text{pwffs}$
shows $\{A, B\} \subseteq \text{pwffs}$
<proof>

lemma *pwffs-from-disj-pwff* [*elim!*]:

assumes $A \vee^{\mathcal{Q}} B \in \text{pwffs}$
shows $\{A, B\} \subseteq \text{pwffs}$
<proof>

lemma *pwffs-from-imp-pwff* [*elim!*]:

assumes $A \supset^{\mathcal{Q}} B \in \text{pwffs}$
shows $\{A, B\} \subseteq \text{pwffs}$
<proof>

lemma *pwffs-from-quiv-pwff* [*elim!*]:
assumes $A \equiv^{\mathcal{Q}} B \in \text{pwffs}$
shows $\{A, B\} \subseteq \text{pwffs}$
 $\langle \text{proof} \rangle$

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

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

lemma *pwffs-free-vars-are-propositional*:
assumes $A \in \text{pwffs}$
and $v \in \text{free-vars } A$
obtains p **where** $v = (p, o)$
 $\langle \text{proof} \rangle$

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
 $\langle \text{proof} \rangle$

4.2 Semantics

Assignment of truth values to propositional variables:

definition *is-tv-assignment* $:: (\text{nat} \Rightarrow V) \Rightarrow \text{bool}$ **where**
 $[\text{iff}]$: *is-tv-assignment* $\varphi \longleftrightarrow (\forall p. \varphi p \in \text{elts } \mathbb{B})$

Denotation of a pwff:

definition *is-pwff-denotation-function* **where**

$[\text{iff}]$: *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* φ
and *is-pwff-denotation-function* \mathcal{V}
shows $\mathcal{V} \varphi A \in \text{elts } \mathbb{B}$

<proof>

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

assumes $A \in \text{pwffs}$
and *free-vars* $A = \{\}$
and *is-tv-assignment* φ
and *is-tv-assignment* ψ
and *is-pwff-denotation-function* \mathcal{V}
shows $\mathcal{V} \varphi A = \mathcal{V} \psi A$

<proof>

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* $\varphi (\sim^{\mathcal{Q}} A) (\sim b_A)$ **if** \mathcal{V}_B -*graph* $\varphi A b_A$
 \mathcal{V}_B -*graph-conj*: \mathcal{V}_B -*graph* $\varphi (A \wedge^{\mathcal{Q}} B) (b_A \wedge b_B)$ **if** \mathcal{V}_B -*graph* $\varphi A b_A$ **and** \mathcal{V}_B -*graph* $\varphi B b_B$
 \mathcal{V}_B -*graph-disj*: \mathcal{V}_B -*graph* $\varphi (A \vee^{\mathcal{Q}} B) (b_A \vee b_B)$ **if** \mathcal{V}_B -*graph* $\varphi A b_A$ **and** \mathcal{V}_B -*graph* $\varphi B b_B$
 \mathcal{V}_B -*graph-imp*: \mathcal{V}_B -*graph* $\varphi (A \supset^{\mathcal{Q}} B) (b_A \supset b_B)$ **if** \mathcal{V}_B -*graph* $\varphi A b_A$ **and** \mathcal{V}_B -*graph* $\varphi B b_B$
 \mathcal{V}_B -*graph-eqv*: \mathcal{V}_B -*graph* $\varphi (A \equiv^{\mathcal{Q}} B) (b_A \equiv b_B)$ **if** \mathcal{V}_B -*graph* $\varphi A b_A$ **and** \mathcal{V}_B -*graph* $\varphi B b_B$ **and** $A \neq F_o$

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

lemma \mathcal{V}_B -*graph-denotation-is-truth-value* [*elim!*]:

assumes \mathcal{V}_B -*graph* $\varphi A b$
and *is-tv-assignment* φ
shows $b \in \text{elts } \mathbb{B}$

<proof>

lemma \mathcal{V}_B -*graph-denotation-uniqueness*:

assumes $A \in \text{pwffs}$
and *is-tv-assignment* φ
and \mathcal{V}_B -*graph* $\varphi A b$ **and** \mathcal{V}_B -*graph* $\varphi A b'$
shows $b = b'$

<proof>

lemma \mathcal{V}_B -*graph-denotation-existence*:

assumes $A \in \text{pwffs}$
and *is-tv-assignment* φ
shows $\exists b. \mathcal{V}_B\text{-graph } \varphi A b$
 $\langle \text{proof} \rangle$

lemma $\mathcal{V}_B\text{-graph-is-functional}$:
assumes $A \in \text{pwffs}$
and *is-tv-assignment* φ
shows $\exists! b. \mathcal{V}_B\text{-graph } \varphi A b$
 $\langle \text{proof} \rangle$

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

lemma $\mathcal{V}_B\text{-equality}$:
assumes $A \in \text{pwffs}$
and *is-tv-assignment* φ
and $\mathcal{V}_B\text{-graph } \varphi A b$
shows $\mathcal{V}_B \varphi A = b$
 $\langle \text{proof} \rangle$

lemma $\mathcal{V}_B\text{-graph-}\mathcal{V}_B$:
assumes $A \in \text{pwffs}$
and *is-tv-assignment* φ
shows $\mathcal{V}_B\text{-graph } \varphi A (\mathcal{V}_B \varphi A)$
 $\langle \text{proof} \rangle$

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

lemma $\mathcal{V}_B\text{-T}$ [$\mathcal{V}_B\text{-simps}$]:
assumes *is-tv-assignment* φ
shows $\mathcal{V}_B \varphi T_o = \mathbf{T}$
 $\langle \text{proof} \rangle$

lemma $\mathcal{V}_B\text{-F}$ [$\mathcal{V}_B\text{-simps}$]:
assumes *is-tv-assignment* φ
shows $\mathcal{V}_B \varphi F_o = \mathbf{F}$
 $\langle \text{proof} \rangle$

lemma $\mathcal{V}_B\text{-var}$ [$\mathcal{V}_B\text{-simps}$]:
assumes *is-tv-assignment* φ
shows $\mathcal{V}_B \varphi (p_o) = \varphi p$
 $\langle \text{proof} \rangle$

lemma $\mathcal{V}_B\text{-neg}$ [$\mathcal{V}_B\text{-simps}$]:
assumes $A \in \text{pwffs}$
and *is-tv-assignment* φ
shows $\mathcal{V}_B \varphi (\sim^{\mathcal{Q}} A) = \sim \mathcal{V}_B \varphi A$
 $\langle \text{proof} \rangle$

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 $\mathcal{V}_B \varphi (A \vee^{\mathcal{Q}} B) = \mathcal{V}_B \varphi A \vee \mathcal{V}_B \varphi B$
 $\langle \text{proof} \rangle$

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 $\mathcal{V}_B \varphi (A \wedge^{\mathcal{Q}} B) = \mathcal{V}_B \varphi A \wedge \mathcal{V}_B \varphi B$
 $\langle \text{proof} \rangle$

lemma \mathcal{V}_B -imp [\mathcal{V}_B -simps]:
assumes $A \in \text{pwffs}$ **and** $B \in \text{pwffs}$
and *is-tv-assignment* φ
shows $\mathcal{V}_B \varphi (A \supset^{\mathcal{Q}} B) = \mathcal{V}_B \varphi A \supset \mathcal{V}_B \varphi B$
 $\langle \text{proof} \rangle$

lemma \mathcal{V}_B -equiv [\mathcal{V}_B -simps]:
assumes $A \in \text{pwffs}$ **and** $B \in \text{pwffs}$
and *is-tv-assignment* φ
shows $\mathcal{V}_B \varphi (A \equiv^{\mathcal{Q}} B) = \mathcal{V}_B \varphi A \equiv \mathcal{V}_B \varphi B$
 $\langle \text{proof} \rangle$

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

lemma *pwff-denotation-function-existence*:
shows *is-pwff-denotation-function* \mathcal{V}_B
 $\langle \text{proof} \rangle$

Tautologies:

definition *is-tautology* :: *form* \Rightarrow *bool* **where**
 $[\text{iff}]$: *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 $A \in \text{wffs}_o$
 $\langle \text{proof} \rangle$

lemma *propositional-implication-reflexivity-is-tautology*:
shows *is-tautology* $(p_o \supset^{\mathcal{Q}} p_o)$
 $\langle \text{proof} \rangle$

lemma *propositional-principle-of-simplification-is-tautology*:
shows *is-tautology* $(p_o \supset^{\mathcal{Q}} (r_o \supset^{\mathcal{Q}} p_o))$
 $\langle \text{proof} \rangle$

lemma *closed-pwff-denotation-uniqueness*:

assumes $A \in \text{pwffs}$ **and** $\text{free-vars } A = \{\}$
obtains b **where** $\forall \varphi. \text{is-tv-assignment } \varphi \longrightarrow \mathcal{V}_B \varphi A = b$
 $\langle \text{proof} \rangle$

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 $\mathbf{S} \{(p, o) \mapsto A\} (B \equiv^{\mathcal{Q}} C) = (\mathbf{S} \{(p, o) \mapsto A\} B) \equiv^{\mathcal{Q}} (\mathbf{S} \{(p, o) \mapsto A\} C)$
 $\langle \text{proof} \rangle$

lemma *pwff-substitution-in-pwffs:*

assumes $A \in \text{pwffs}$ **and** $B \in \text{pwffs}$
shows $\mathbf{S} \{(p, o) \mapsto A\} B \in \text{pwffs}$
 $\langle \text{proof} \rangle$

lemma *pwff-substitution-denotation:*

assumes $A \in \text{pwffs}$ **and** $B \in \text{pwffs}$
and *is-tv-assignment* φ
shows $\mathcal{V}_B \varphi (\mathbf{S} \{(p, o) \mapsto A\} B) = \mathcal{V}_B (\varphi(p := \mathcal{V}_B \varphi A)) B$
 $\langle \text{proof} \rangle$

lemma *pwff-substitution-tautology-preservation:*

assumes *is-tautology* B **and** $A \in \text{pwffs}$
and $(p, o) \in \text{free-vars } B$
shows *is-tautology* $(\mathbf{S} \{(p, o) \mapsto A\} B)$
 $\langle \text{proof} \rangle$

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

assumes $A \in \text{pwffs}$ **and** $B \in \text{pwffs}$
and $\text{free-vars } A = \{\}$
and $(p, o) \in \text{free-vars } B$
shows $\text{free-vars } (\mathbf{S} \{(p, o) \mapsto A\} B) = \text{free-vars } B - \{(p, o)\}$ (**is** $\langle \text{free-vars } (\mathbf{S} \vartheta B) = \rightarrow \rangle$)
 $\langle \text{proof} \rangle$

Substitution in a pwff:

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

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

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 $A \in wffs_o$
 ⟨proof⟩

lemma *implication-reflexivity-is-tautologous:*

assumes $A \in wffs_o$
shows *is-tautologous* $(A \supset^Q A)$
 ⟨proof⟩

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

assumes $A \in wffs_o$ **and** $B \in wffs_o$
shows *is-tautologous* $(A \supset^Q (B \supset^Q A))$
 ⟨proof⟩

lemma *pseudo-modus-tollens-is-tautologous:*

assumes $A \in wffs_o$ **and** $B \in wffs_o$
shows *is-tautologous* $((A \supset^Q \sim^Q B) \supset^Q (B \supset^Q \sim^Q A))$
 ⟨proof⟩

end

5 Proof System

theory *Proof-System*

imports

Syntax

begin

5.1 Axioms

inductive-set

axioms :: form set

where

axiom-1:

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

| *axiom-2:*

$(\mathfrak{r}_\alpha =_\alpha \mathfrak{r}_\alpha) \supset^Q (\mathfrak{h}_{\alpha \rightarrow o} \cdot \mathfrak{r}_\alpha \equiv^Q \mathfrak{h}_{\alpha \rightarrow o} \cdot \mathfrak{r}_\alpha) \in \text{axioms}$

| *axiom-3:*

$(\mathfrak{f}_{\alpha \rightarrow \beta} =_{\alpha \rightarrow \beta} \mathfrak{g}_{\alpha \rightarrow \beta}) \equiv^Q \forall \mathfrak{r}_\alpha. (\mathfrak{f}_{\alpha \rightarrow \beta} \cdot \mathfrak{r}_\alpha =_\beta \mathfrak{g}_{\alpha \rightarrow \beta} \cdot \mathfrak{r}_\alpha) \in \text{axioms}$

| *axiom-4-1-con:*

$(\lambda x_\alpha. \mathfrak{f}c\mathfrak{f}_\beta) \cdot A =_\beta \mathfrak{f}c\mathfrak{f}_\beta \in \text{axioms}$ **if** $A \in wffs_\alpha$

| *axiom-4-1-var:*

$(\lambda x_\alpha. y_\beta) \cdot A =_\beta y_\beta \in \text{axioms}$ **if** $A \in wffs_\alpha$ **and** $y_\beta \neq x_\alpha$

| *axiom-4-2:*

$(\lambda x_\alpha. x_\alpha) \cdot A =_\alpha A \in \text{axioms}$ **if** $A \in wffs_\alpha$

| *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 $A \in wffs_\alpha$ **and** $B \in wffs_{\gamma \rightarrow \beta}$ **and** $C \in wffs_\gamma$

| *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*:
 $\iota \cdot (Q_i \cdot \eta_i) =_i \eta_i \in \text{axioms}$

lemma *axioms-are-wffs-of-type-o*:
shows $\text{axioms} \subseteq \text{wffs}_o$
<proof>

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

lemma *rule-R-original-form-is-wffo*:
assumes *is-rule-R-app* p D C E
shows $C \in \text{wffs}_o$
<proof>

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 $A \in \text{wffs}_o$
<proof>

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* $(\mathcal{S} @ \mathcal{S}_2) i'$

$\langle proof \rangle$

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* $(\mathcal{S} @ \mathcal{S}') i'$

$\langle proof \rangle$

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* $(\mathcal{S} @ [A])$

$\langle proof \rangle$

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* $(\mathcal{S} @ \mathcal{S}') i'$

$\langle proof \rangle$

lemma *proof-but-last-is-proof:*

assumes *is-proof* $(\mathcal{S} @ [A])$

shows *is-proof* \mathcal{S}

$\langle proof \rangle$

lemma *proof-prefix-is-proof:*

assumes *is-proof* $(\mathcal{S}_1 @ \mathcal{S}_2)$

shows *is-proof* \mathcal{S}_1

$\langle proof \rangle$

lemma *single-axiom-is-proof:*

assumes $A \in \text{axioms}$

shows *is-proof* $[A]$

$\langle proof \rangle$

lemma *proofs-concatenation-is-proof:*

assumes *is-proof* \mathcal{S}_1 **and** *is-proof* \mathcal{S}_2

shows *is-proof* $(\mathcal{S}_1 @ \mathcal{S}_2)$

$\langle proof \rangle$

lemma *elem-of-proof-is-wffo:*

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

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

$\langle proof \rangle$

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

assumes *is-proof* \mathcal{S}

and $A \in \text{axioms}$

shows *is-proof* $([A] @ \mathcal{S})$

$\langle proof \rangle$

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

assumes *is-proof* \mathcal{S}

and $A \in \text{axioms}$

shows *is-proof* $(\mathcal{S} @ [A])$

$\langle proof \rangle$

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* $(\mathcal{S} @ [D])$

$\langle proof \rangle$

definition *is-proof-of* $:: \text{form list} \Rightarrow \text{form} \Rightarrow \text{bool}$ **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} $(\text{last } \mathcal{S})$

$\langle proof \rangle$

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

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

lemma *proof-form-is-wffo*:

assumes *is-proof-of* $\mathcal{S} A$

and $B \in \text{lset } \mathcal{S}$

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

$\langle proof \rangle$

lemma *proof-form-is-theorem*:

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

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

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

$\langle proof \rangle$

theorem *derivable-form-is-theorem*:

assumes *is-derivable* A

shows *is-theorem* A

$\langle proof \rangle$

theorem *theorem-is-derivable-form*:

assumes *is-theorem* A

shows *is-derivable* A

$\langle proof \rangle$

theorem *theoremhood-derivability-equivalence*:
shows *is-theorem* $A \longleftrightarrow$ *is-derivable* A
 ⟨*proof*⟩

lemma *theorem-is-wffo*:
assumes *is-theorem* A
shows $A \in \text{wffs}_o$
 ⟨*proof*⟩

lemma *equality-reflexivity*:
assumes $A \in \text{wffs}_\alpha$
shows *is-theorem* $(A =_\alpha A)$ (**is** *is-theorem* $?A_2$)
 ⟨*proof*⟩

lemma *equality-reflexivity'*:
assumes $A \in \text{wffs}_\alpha$
shows *is-theorem* $(A =_\alpha A)$ (**is** *is-theorem* $?A_2$)
 ⟨*proof*⟩

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* $p A \mathcal{X} = \text{binders-at } A p \cap \text{free-vars } \mathcal{X}$
 ⟨*proof*⟩

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* $\mathcal{H} p D C E \longleftrightarrow$
 $\text{capture-exposed-vars-at } p C E \cap \text{capture-exposed-vars-at } p C \mathcal{H} = \{\}$

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

)
 ⟨proof⟩

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

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

shows

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

\longleftrightarrow

(

$\exists \alpha$ A B .

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

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

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

(

$\nexists x$ β E p' .

strict-prefix p' $p \wedge$

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

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

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

)

)

⟨proof⟩

lemma *rule-R'-preserves-typing*:

assumes *is-rule-R'-app* \mathcal{H} p D C E

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

⟨proof⟩

abbreviation *is-hyps* :: form set ⇒ 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 ⇒ form ⇒ 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 $A \in \text{wffs}_o$

⟨proof⟩

definition *is-hyp-proof-step* :: form set ⇒ form list ⇒ form list ⇒ nat ⇒ bool **where**

[iff]: *is-hyp-proof-step* \mathcal{H} \mathcal{S}_1 \mathcal{S}_2 i' \longleftrightarrow

$\mathcal{S}_2 ! i' \in \mathcal{H} \vee$

$\mathcal{S}_2 ! i' \in \text{lset } \mathcal{S}_1 \vee$

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

type-synonym $\text{hyp-proof} = \text{form list} \times \text{form list}$

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

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

lemma *added-suffix-thms-hyp-proof-preservation*:
assumes $\text{is-hyp-proof } \mathcal{H} \mathcal{S}_1 \mathcal{S}_2$
shows $\text{is-hyp-proof } \mathcal{H} (\mathcal{S}_1 @ \mathcal{S}_1') \mathcal{S}_2$
<proof>

lemma *added-suffix-hyp-proof-preservation*:
assumes $\text{is-hyp-proof } \mathcal{H} \mathcal{S}_1 \mathcal{S}_2$
and $i' < \text{length } (\mathcal{S}_2 @ \mathcal{S}_2') - \text{length } \mathcal{S}_2'$
shows $\text{is-hyp-proof-step } \mathcal{H} \mathcal{S}_1 (\mathcal{S}_2 @ \mathcal{S}_2') i'$
<proof>

lemma *appended-hyp-proof-step-is-hyp-proof*:
assumes $\text{is-hyp-proof } \mathcal{H} \mathcal{S}_1 \mathcal{S}_2$
and $\text{is-hyp-proof-step } \mathcal{H} \mathcal{S}_1 (\mathcal{S}_2 @ [A]) (\text{length } (\mathcal{S}_2 @ [A]) - 1)$
shows $\text{is-hyp-proof } \mathcal{H} \mathcal{S}_1 (\mathcal{S}_2 @ [A])$
<proof>

lemma *added-prefix-hyp-proof-preservation*:
assumes $\text{is-hyp-proof } \mathcal{H} \mathcal{S}_1 \mathcal{S}_2'$
and $i' \in \{\text{length } \mathcal{S}_2..<\text{length } (\mathcal{S}_2 @ \mathcal{S}_2')\}$
shows $\text{is-hyp-proof-step } \mathcal{H} \mathcal{S}_1 (\mathcal{S}_2 @ \mathcal{S}_2') i'$
<proof>

lemma *hyp-proof-but-last-is-hyp-proof*:
assumes $\text{is-hyp-proof } \mathcal{H} \mathcal{S}_1 (\mathcal{S}_2 @ [A])$
shows $\text{is-hyp-proof } \mathcal{H} \mathcal{S}_1 \mathcal{S}_2$
<proof>

lemma *hyp-proof-prefix-is-hyp-proof*:
assumes $\text{is-hyp-proof } \mathcal{H} \mathcal{S}_1 (\mathcal{S}_2 @ \mathcal{S}_2')$
shows $\text{is-hyp-proof } \mathcal{H} \mathcal{S}_1 \mathcal{S}_2$
<proof>

lemma *single-hyp-is-hyp-proof*:
assumes $A \in \mathcal{H}$
shows $\text{is-hyp-proof } \mathcal{H} \mathcal{S}_1 [A]$
<proof>

lemma *single-thm-is-hyp-proof*:

assumes $A \in \text{lset } \mathcal{S}_1$
shows $\text{is-hyp-proof } \mathcal{H} \mathcal{S}_1 [A]$
 $\langle \text{proof} \rangle$

lemma *hyp-proofs-from-concatenation-is-hyp-proof*:

assumes $\text{is-hyp-proof } \mathcal{H} \mathcal{S}_1 \mathcal{S}_1'$ **and** $\text{is-hyp-proof } \mathcal{H} \mathcal{S}_2 \mathcal{S}_2'$
shows $\text{is-hyp-proof } \mathcal{H} (\mathcal{S}_1 @ \mathcal{S}_2) (\mathcal{S}_1' @ \mathcal{S}_2')$
 $\langle \text{proof} \rangle$

lemma *elem-of-hyp-proof-is-woff*:

assumes $\text{is-hyps } \mathcal{H}$
and $\text{lset } \mathcal{S}_1 \subseteq \text{woffs}_o$
and $\text{is-hyp-proof } \mathcal{H} \mathcal{S}_1 \mathcal{S}_2$
and $A \in \text{lset } \mathcal{S}_2$
shows $A \in \text{woffs}_o$
 $\langle \text{proof} \rangle$

lemma *hyp-prepended-to-hyp-proof-is-hyp-proof*:

assumes $\text{is-hyp-proof } \mathcal{H} \mathcal{S}_1 \mathcal{S}_2$
and $A \in \mathcal{H}$
shows $\text{is-hyp-proof } \mathcal{H} \mathcal{S}_1 ([A] @ \mathcal{S}_2)$
 $\langle \text{proof} \rangle$

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

assumes $\text{is-hyp-proof } \mathcal{H} \mathcal{S}_1 \mathcal{S}_2$
and $A \in \mathcal{H}$
shows $\text{is-hyp-proof } \mathcal{H} \mathcal{S}_1 (\mathcal{S}_2 @ [A])$
 $\langle \text{proof} \rangle$

lemma *dropped-duplicated-thm-in-hyp-proof-is-hyp-proof*:

assumes $\text{is-hyp-proof } \mathcal{H} (A \# \mathcal{S}_1) \mathcal{S}_2$
and $A \in \text{lset } \mathcal{S}_1$
shows $\text{is-hyp-proof } \mathcal{H} \mathcal{S}_1 \mathcal{S}_2$
 $\langle \text{proof} \rangle$

lemma *thm-prepended-to-hyp-proof-is-hyp-proof*:

assumes $\text{is-hyp-proof } \mathcal{H} \mathcal{S}_1 \mathcal{S}_2$
and $A \in \text{lset } \mathcal{S}_1$
shows $\text{is-hyp-proof } \mathcal{H} \mathcal{S}_1 ([A] @ \mathcal{S}_2)$
 $\langle \text{proof} \rangle$

lemma *thm-appended-to-hyp-proof-is-hyp-proof*:

assumes $\text{is-hyp-proof } \mathcal{H} \mathcal{S}_1 \mathcal{S}_2$
and $A \in \text{lset } \mathcal{S}_1$
shows $\text{is-hyp-proof } \mathcal{H} \mathcal{S}_1 (\mathcal{S}_2 @ [A])$
 $\langle \text{proof} \rangle$

lemma *rule-R'-app-appended-to-hyp-proof-is-hyp-proof*:

assumes *is-hyp-proof* $\mathcal{H} \mathcal{S}' S$
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* $\mathcal{H} \mathcal{S}' (S @ [D])$
 <proof>

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

theorem *hyp-derivability-implies-hyp-proof-existence*:
assumes $\mathcal{H} \vdash A$
shows $\exists \mathcal{S}_1 \mathcal{S}_2. \text{is-hyp-proof-of } \mathcal{H} \mathcal{S}_1 \mathcal{S}_2 A$
 <proof>

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>

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

proposition *derivability-from-no-hyps-theoremhood-equivalence*:
shows $\{\} \vdash A \longleftrightarrow \text{is-theorem } A$
 <proof>

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 $\mathcal{H} \vdash A$
 <proof>

lemma *axiom-is-derivable-from-no-hyps*:

assumes $A \in \text{axioms}$
shows $\vdash A$
 $\langle \text{proof} \rangle$

lemma *axiom-is-derivable-from-hyps*:
assumes $A \in \text{axioms}$ **and** *is-hyps* \mathcal{H}
shows $\mathcal{H} \vdash A$
 $\langle \text{proof} \rangle$

lemma *rule-R* [*consumes* \mathcal{Q} , *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 $\vdash D$
 $\langle \text{proof} \rangle$

lemma *rule-R'* [*consumes* \mathcal{Q} , *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 $\mathcal{H} \vdash D$
 $\langle \text{proof} \rangle$

end

6 Elementary Logic

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

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

6.1 Proposition 5200

proposition *prop-5200*:
assumes $A \in \text{wffs}_{\alpha}$
shows $\vdash A =_{\alpha} A$
 $\langle \text{proof} \rangle$

corollary *hyp-prop-5200*:
assumes *is-hyps* \mathcal{H} **and** $A \in \text{wffs}_{\alpha}$
shows $\mathcal{H} \vdash A =_{\alpha} A$
 $\langle \text{proof} \rangle$

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>

proposition *prop-5201-2*:

assumes $\mathcal{H} \vdash A =_{\alpha} B$

shows $\mathcal{H} \vdash B =_{\alpha} A$

<proof>

proposition *prop-5201-3*:

assumes $\mathcal{H} \vdash A =_{\alpha} B$ **and** $\mathcal{H} \vdash B =_{\alpha} C$

shows $\mathcal{H} \vdash A =_{\alpha} C$

<proof>

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>

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>

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>

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>

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 $\vdash (\lambda x_\alpha. B) \cdot A =_\beta \mathbf{S} \{(x, \alpha) \mapsto A\} B$
 $\langle \text{proof} \rangle$

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 $\vdash \mathbf{S} \{(x, \alpha) \mapsto A\} (B =_\beta C)$
 $\langle \text{proof} \rangle$

6.6 Proposition 5205 (η -conversion)

proposition *prop-5205*:
shows $\vdash \mathbf{f}_{\alpha \rightarrow \beta} =_{\alpha \rightarrow \beta} (\lambda y_\alpha. \mathbf{f}_{\alpha \rightarrow \beta} \cdot y_\alpha)$
 $\langle \text{proof} \rangle$

6.7 Proposition 5206 (α -conversion)

proposition *prop-5206*:
assumes $A \in \text{wffs}_\alpha$
and $(z, \beta) \notin \text{free-vars } A$
and $\text{is-free-for } (z_\beta) (x, \beta) A$
shows $\vdash (\lambda x_\beta. A) =_{\beta \rightarrow \alpha} (\lambda z_\beta. \mathbf{S} \{(x, \beta) \mapsto z_\beta\} A)$
 $\langle \text{proof} \rangle$

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 $\vdash A =_\alpha \text{rename-bound-var } (y, \gamma) z A$
 $\langle \text{proof} \rangle$

proposition *prop-5207*:
assumes $A \in \text{wffs}_\alpha$ **and** $B \in \text{wffs}_\beta$
and $\text{is-free-for } A (x, \alpha) B$
shows $\vdash (\lambda x_\alpha. B) \cdot A =_\beta \mathbf{S} \{(x, \alpha) \mapsto A\} B$
 $\langle \text{proof} \rangle$

end

6.9 Proposition 5208

proposition *prop-5208*:

assumes $vs \neq []$ **and** $B \in wffs_\beta$

shows $\vdash \cdot^\mathcal{Q}_* (\lambda^\mathcal{Q}_* vs B) (map FVar vs) =_\beta B$

<proof>

6.10 Proposition 5209

proposition *prop-5209*:

assumes $A \in wffs_\alpha$ **and** $B \in wffs_\beta$ **and** $C \in wffs_\beta$

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>

6.11 Proposition 5210

proposition *prop-5210*:

assumes $B \in wffs_\beta$

shows $\vdash T_o =_o (B =_\beta B)$

<proof>

6.12 Proposition 5211

proposition *prop-5211*:

shows $\vdash (T_o \wedge^\mathcal{Q} T_o) =_o T_o$

<proof>

lemma *true-is-derivable*:

shows $\vdash T_o$

<proof>

6.13 Proposition 5212

proposition *prop-5212*:

shows $\vdash T_o \wedge^\mathcal{Q} T_o$

<proof>

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>

6.15 Proposition 5214

proposition *prop-5214*:

shows $\vdash T_o \wedge^\mathcal{Q} F_o =_o F_o$

<proof>

6.16 Proposition 5215 (Universal Instantiation)

proposition *prop-5215*:

assumes $\mathcal{H} \vdash \forall x_\alpha. B$ and $A \in wffs_\alpha$

and *is-free-for* $A (x, \alpha) B$

shows $\mathcal{H} \vdash \mathbf{S} \{(x, \alpha) \mapsto A\} B$

<proof>

lemmas $\forall I = \text{prop-5215}$

6.17 Proposition 5216

proposition *prop-5216*:

assumes $A \in wffs_o$

shows $\vdash (T_o \wedge^{\mathcal{Q}} A) =_o A$

<proof>

6.18 Proposition 5217

proposition *prop-5217*:

shows $\vdash (T_o =_o F_o) =_o F_o$

<proof>

6.19 Proposition 5218

proposition *prop-5218*:

assumes $A \in wffs_o$

shows $\vdash (T_o =_o A) =_o A$

<proof>

6.20 Proposition 5219 (Rule T)

proposition *prop-5219-1*:

assumes $A \in wffs_o$

shows $\mathcal{H} \vdash A \longleftrightarrow \mathcal{H} \vdash T_o =_o A$

<proof>

proposition *prop-5219-2*:

assumes $A \in wffs_o$

shows $\mathcal{H} \vdash A \longleftrightarrow \mathcal{H} \vdash A =_o T_o$

<proof>

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

6.21 Proposition 5220 (Universal Generalization)

context

begin

private lemma *const-true- α -conversion*:

shows $\vdash (\lambda x_\alpha. T_o) =_{\alpha \rightarrow o} (\lambda z_\alpha. T_o)$

$\langle proof \rangle$

proposition *prop-5220*:

assumes $\mathcal{H} \vdash A$

and $(x, \alpha) \notin \text{free-vars } \mathcal{H}$

shows $\mathcal{H} \vdash \forall x_\alpha. A$

$\langle proof \rangle$

end

lemmas *Gen* = *prop-5220*

proposition *generalized-Gen*:

assumes $\mathcal{H} \vdash A$

and $\text{lset } vs \cap \text{free-vars } \mathcal{H} = \{\}$

shows $\mathcal{H} \vdash \forall^{\mathcal{Q}_*} vs A$

$\langle proof \rangle$

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 $\mathcal{H} \vdash \mathbf{S} \{(x, \alpha) \mapsto A\} B$

$\langle proof \rangle$

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 $\mathcal{H} \vdash \mathbf{S} \vartheta B$

$\langle proof \rangle$

end

lemmas *Sub* = *prop-5221*

6.23 Proposition 5222 (Rule of Cases)

lemma *forall- α -conversion*:

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

and $(z, \beta) \notin \text{free-vars } A$
and *is-free-for* $(z_\beta) (x, \beta) A$
shows $\vdash \forall x_\beta. A =_o \forall z_\beta. \mathbf{S} \{(x, \beta) \mapsto z_\beta\} A$
 $\langle \text{proof} \rangle$

proposition *prop-5222*:
assumes $\mathcal{H} \vdash \mathbf{S} \{(x, o) \mapsto T_o\} A$ **and** $\mathcal{H} \vdash \mathbf{S} \{(x, o) \mapsto F_o\} A$
and $A \in \text{wffs}_o$
shows $\mathcal{H} \vdash A$
 $\langle \text{proof} \rangle$

lemmas *Cases* = *prop-5222*

6.24 Proposition 5223

proposition *prop-5223*:
shows $\vdash (T_o \supset^{\mathcal{Q}} \eta_o) =_o \eta_o$
 $\langle \text{proof} \rangle$

corollary *generalized-prop-5223*:
assumes $A \in \text{wffs}_o$
shows $\vdash (T_o \supset^{\mathcal{Q}} A) =_o A$
 $\langle \text{proof} \rangle$

6.25 Proposition 5224 (Modus Ponens)

proposition *prop-5224*:
assumes $\mathcal{H} \vdash A$ **and** $\mathcal{H} \vdash A \supset^{\mathcal{Q}} B$
shows $\mathcal{H} \vdash B$
 $\langle \text{proof} \rangle$

lemmas *MP* = *prop-5224*

corollary *generalized-modus-ponens*:
assumes $\mathcal{H} \vdash \text{hs} \supset^{\mathcal{Q}}_* B$ **and** $\forall H \in \text{lset hs. } \mathcal{H} \vdash H$
shows $\mathcal{H} \vdash B$
 $\langle \text{proof} \rangle$

6.26 Proposition 5225

proposition *prop-5225*:
shows $\vdash \prod \alpha \cdot \text{f}_{\alpha \rightarrow o} \supset^{\mathcal{Q}} \text{f}_{\alpha \rightarrow o} \cdot \text{r}_\alpha$
 $\langle \text{proof} \rangle$

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 $\vdash \forall x_\alpha. B \supset^{\mathcal{Q}} \mathbf{S} \{(x, \alpha) \mapsto A\} B$

$\langle proof \rangle$

6.28 Proposition 5227

corollary *prop-5227*:

shows $\vdash F_o \supset^Q \mathfrak{r}_o$

$\langle proof \rangle$

corollary *generalized-prop-5227*:

assumes $A \in wffs_o$

shows $\vdash F_o \supset^Q A$

$\langle proof \rangle$

6.29 Proposition 5228

proposition *prop-5228*:

shows $\vdash (T_o \supset^Q T_o) =_o T_o$

and $\vdash (T_o \supset^Q F_o) =_o F_o$

and $\vdash (F_o \supset^Q T_o) =_o T_o$

and $\vdash (F_o \supset^Q F_o) =_o T_o$

$\langle proof \rangle$

6.30 Proposition 5229

lemma *false-in-conj-provability*:

assumes $A \in wffs_o$

shows $\vdash F_o \wedge^Q A \equiv^Q F_o$

$\langle proof \rangle$

proposition *prop-5229*:

shows $\vdash (T_o \wedge^Q T_o) =_o T_o$

and $\vdash (T_o \wedge^Q F_o) =_o F_o$

and $\vdash (F_o \wedge^Q T_o) =_o F_o$

and $\vdash (F_o \wedge^Q F_o) =_o F_o$

$\langle proof \rangle$

6.31 Proposition 5230

proposition *prop-5230*:

shows $\vdash (T_o \equiv^Q T_o) =_o T_o$

and $\vdash (T_o \equiv^Q F_o) =_o F_o$

and $\vdash (F_o \equiv^Q T_o) =_o F_o$

and $\vdash (F_o \equiv^Q F_o) =_o T_o$

$\langle proof \rangle$

6.32 Proposition 5231

proposition *prop-5231*:

shows $\vdash \sim^Q T_o =_o F_o$

and $\vdash \sim^Q F_o =_o T_o$

<proof>

6.33 Proposition 5232

lemma *disj-op-alt-def-provability*:

assumes $A \in \text{wffs}_o$ **and** $B \in \text{wffs}_o$

shows $\vdash A \vee^{\mathcal{Q}} B =_o \sim^{\mathcal{Q}} (\sim^{\mathcal{Q}} A \wedge^{\mathcal{Q}} \sim^{\mathcal{Q}} B)$

<proof>

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>

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>

end

6.34 Proposition 5233

context begin

private lemma *lem-prop-5233-no-free-vars*:

assumes $A \in \text{pwffs}$ **and** $\text{free-vars } A = \{\}$

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>

proposition *prop-5233*:

assumes *is-tautology* A

shows $\vdash A$

<proof>

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* ($hs \supset^{\mathcal{Q}}_* B$)
shows $\mathcal{H} \vdash B$
 $\langle proof \rangle$

corollary *tautologous-is-hyp-derivable*:
assumes *is-hyps* \mathcal{H}
and *is-tautologous* B
shows $\mathcal{H} \vdash B$
 $\langle proof \rangle$

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 $A \in pffs$ **and** $B \in pffs$
and $(x, \alpha) \notin \text{free-vars } A$
shows $\vdash \forall x_\alpha. (A \vee^{\mathcal{Q}} B) \supset^{\mathcal{Q}} (A \vee^{\mathcal{Q}} \forall x_\alpha. B)$
 $\langle proof \rangle$

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

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

context begin

private lemma *prop-5237-aux*:
assumes $A \in wffs_o$ **and** $B \in wffs_o$
and $(x, \alpha) \notin \text{free-vars } A$
shows $\vdash \forall x_\alpha. (A \supset^{\mathcal{Q}} B) \equiv^{\mathcal{Q}} (A \supset^{\mathcal{Q}} (\forall x_\alpha. B))$
 $\langle proof \rangle$

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 $\mathcal{H} \vdash A \supset^{\mathcal{Q}} (\forall x_\alpha. B)$
 $\langle proof \rangle$

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

corollary *generalized-prop-5237*:
assumes *is-hyps* \mathcal{H}
and $\mathcal{H} \vdash A \supset^{\mathcal{Q}} B$

and $\forall v \in S. v \notin \text{free-vars} (\{A\} \cup \mathcal{H})$
and $\text{lset } vs = S$
shows $\mathcal{H} \vdash A \supset^{\mathcal{Q}} (\forall^{\mathcal{Q}}_* vs B)$
 $\langle \text{proof} \rangle$

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 $\vdash ((\lambda x_\beta. A) =_{\beta \rightarrow \alpha} (\lambda x_\beta. B)) \equiv^{\mathcal{Q}} \forall x_\beta. (A =_\alpha B)$
 $\langle \text{proof} \rangle$

proposition *prop-5238*:
assumes $vs \neq []$ **and** $A \in \text{wffs}_\alpha$ **and** $B \in \text{wffs}_\alpha$
shows $\vdash \lambda^{\mathcal{Q}}_* vs A =_{\text{foldr } (\rightarrow) \text{ (map var-type vs)}} \alpha \lambda^{\mathcal{Q}}_* vs B \equiv^{\mathcal{Q}} \forall^{\mathcal{Q}}_* vs (A =_\alpha B)$
 $\langle \text{proof} \rangle$

end

6.39 Proposition 5239

lemma *replacement-derivability*:

assumes $C \in \text{wffs}_\beta$
and $A \preceq_p C$
and $\vdash A =_\alpha B$
and $C \langle p \leftarrow B \rangle \triangleright D$
shows $\vdash C =_\beta D$
 $\langle \text{proof} \rangle$

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
 $(\exists v \in \text{lset } vs. \text{occurs-at } v p (\cdot^{\mathcal{Q}}_* (FVar v) \text{ (map FVar vs))))$

$\langle \text{proof} \rangle$ **lemma** *prop-5239-aux-2*:
assumes $t \notin \text{lset } vs \cup \text{vars } C$
and $C \langle p \leftarrow (\cdot^{\mathcal{Q}}_* (FVar t) \text{ (map FVar vs)}) \rangle \triangleright G$
and $C \langle p \leftarrow (\cdot^{\mathcal{Q}}_* (\lambda^{\mathcal{Q}}_* vs A) \text{ (map FVar vs)}) \rangle \triangleright G'$
shows $\mathbf{S} \{t \mapsto \lambda^{\mathcal{Q}}_* vs A\} G = G' \text{ (is } \langle \mathbf{S} \ ?\vartheta G = G' \rangle)$

$\langle \text{proof} \rangle$ **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 $\text{occurs-at } t \ p' \ G$
shows $p' = p \ @ \text{ replicate } (\text{length } vs) \ \ll \ (\text{is } \langle p' = ?p_t \rangle)$
 $\langle \text{proof} \rangle$ **lemma** *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 $C \langle p \leftarrow (\cdot^{\mathcal{Q}}_{\star} (FVar t) (map FVar vs)) \rangle \triangleright G$
shows $\text{is-free-for } (\lambda^{\mathcal{Q}}_{\star} vs \ A) \ t \ G$
 $\langle \text{proof} \rangle$

proposition *prop-5239*:

assumes $\text{is-rule-R-app } p \ D \ C \ (A =_{\alpha} B)$
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 $\vdash \forall^{\mathcal{Q}}_{\star} vs \ (A =_{\alpha} B) \supset^{\mathcal{Q}} (C \equiv^{\mathcal{Q}} D)$
 $\langle \text{proof} \rangle$

end

6.40 Theorem 5240 (Deduction Theorem)

lemma *pseudo-rule-R-is-tautologous*:

assumes $C \in \text{wffs}_o$ **and** $D \in \text{wffs}_o$ **and** $E \in \text{wffs}_o$ **and** $H \in \text{wffs}_o$
shows $\text{is-tautologous } (((H \supset^{\mathcal{Q}} C) \supset^{\mathcal{Q}} ((H \supset^{\mathcal{Q}} E) \supset^{\mathcal{Q}} ((E \supset^{\mathcal{Q}} (C \equiv^{\mathcal{Q}} D)) \supset^{\mathcal{Q}} (H \supset^{\mathcal{Q}} D))))))$
 $\langle \text{proof} \rangle$

syntax

-HypDer :: $\text{form} \Rightarrow \text{form set} \Rightarrow \text{form} \Rightarrow \text{bool} \ (-, - \vdash - [50, 50, 50] \ 50)$

translations

$\mathcal{H}, H \vdash P \mapsto \mathcal{H} \cup \{H\} \vdash P$

theorem *thm-5240*:

assumes $\text{finite } \mathcal{H}$
and $\mathcal{H}, H \vdash P$
shows $\mathcal{H} \vdash H \supset^{\mathcal{Q}} P$
 $\langle \text{proof} \rangle$

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 $\text{finite } \mathcal{H}$ **and** $\text{finite } \mathcal{H}'$
and $\mathcal{H} \cup \mathcal{H}' \vdash P$
and $\text{lset } hs = \mathcal{H}'$
shows $\mathcal{H} \vdash hs \supset^{\mathcal{Q}}_{\star} P$
 $\langle \text{proof} \rangle$

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>

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>

lemmas $\exists \text{Gen} = \text{prop-5242}$

6.43 Proposition 5243 (Comprehension Theorem)

context
begin

private lemma *prop-5243-aux*:
assumes $\bullet^{\mathcal{Q}}_* B (\text{map } F\text{Var } vs) \in \text{wffs}_\gamma$
and $B \in \text{wffs}_\beta$
and $k < \text{length } vs$
shows $\beta \neq \text{var-type } (vs ! k)$
<proof>

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 $\vdash \exists u_\gamma. \forall \bullet^{\mathcal{Q}}_* vs ((\bullet^{\mathcal{Q}}_* u_\gamma (\text{map } F\text{Var } vs)) =_\beta B)$
<proof>

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>

proposition *prop-5244*:
assumes $\mathcal{H}, B \vdash A$
and $(x, \alpha) \notin \text{free-vars } (\mathcal{H} \cup \{A\})$
shows $\mathcal{H}, \exists x_\alpha. B \vdash A$
 $\langle \text{proof} \rangle$

lemmas $\exists\text{-Rule} = \text{prop-5244}$

6.45 Proposition 5245 (Rule C)

lemma *prop-5245-aux*:
assumes $x \neq y$
and $(y, \alpha) \notin \text{free-vars } (\exists x_\alpha. B)$
and *is-free-for* $(y_\alpha) (x, \alpha) B$
shows *is-free-for* $(x_\alpha) (y, \alpha) \mathbf{S} \{(x, \alpha) \mapsto y_\alpha\} B$
 $\langle \text{proof} \rangle$

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 $\mathcal{H} \vdash A$
 $\langle \text{proof} \rangle$

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

end

7 Semantics

theory *Semantics*

imports

ZFC-in-HOL.ZFC-Typeclasses

Syntax

Boolean-Algebra

begin

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

notation *funcset* (**infixr** \rightarrow 60)

abbreviation *vfuncset* $:: V \Rightarrow V \Rightarrow V$ (**infixr** \mapsto 60) **where**
 $A \mapsto B \equiv \text{VPi } A (\lambda-. B)$

notation *app* (**infixl** \cdot 300)

syntax

-vlambda $:: \text{pttrn} \Rightarrow V \Rightarrow (V \Rightarrow V) \Rightarrow V ((\exists \lambda \text{-} \cdot / \cdot) [0, 0, 3] 3)$

translations

$\lambda x : A. f \equiv \text{CONST } \text{VLambda } A (\lambda x. f)$

lemma *vlambda-extensionality*:

assumes $\bigwedge x. x \in \text{elts } A \implies f x = g x$

shows $(\lambda x : A. f x) = (\lambda x : A. g x)$

<proof>

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 $f \in \text{elts } (\mathcal{D } \alpha \mapsto \mathcal{D } \beta)$

<proof>

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** $\forall x \in \text{elts } (\mathcal{D } \alpha). b x \in \text{elts } (\mathcal{D } \beta)$

<proof>

lemma *app-is-domain-respecting*:

assumes $f \in \text{elts } (\mathcal{D } (\alpha \rightarrow \beta))$ **and** $x \in \text{elts } (\mathcal{D } \alpha)$

shows $f \cdot x \in \text{elts } (\mathcal{D } \beta)$

<proof>

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

definition *one-element-function* :: $V \Rightarrow \text{type} \Rightarrow V (\{-\}. [901, 0] 900)$ **where**

[simp]: $\{x\}_\alpha = (\lambda y : \mathcal{D } \alpha. \text{bool-to-}V (y = x))$

lemma *one-element-function-is-domain-respecting*:

shows $\{x\}_\alpha \in \text{elts } (\mathcal{D } \alpha \mapsto \mathcal{D } o)$

<proof>

lemma *one-element-function-simps*:

shows $x \in \text{elts } (\mathcal{D } \alpha) \implies \{x\}_\alpha \cdot x = \mathbf{T}$

and $\llbracket \{x, y\} \subseteq \text{elts } (\mathcal{D } \alpha); y \neq x \rrbracket \implies \{x\}_\alpha \cdot y = \mathbf{F}$

<proof>

lemma *one-element-function-injectivity*:

assumes $\{x, x'\} \subseteq \text{elts } (\mathcal{D } i)$ **and** $\{x\}_i = \{x'\}_i$

shows $x = x'$

<proof>

lemma *one-element-function-uniqueness*:

assumes $x \in \text{elts } (\mathcal{D} \ i)$
shows $(\text{SOME } x'. x' \in \text{elts } (\mathcal{D} \ i) \wedge \{x\}_i = \{x'\}_i) = x$
<proof>

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

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

lemma *identity-relation-is-domain-respecting*:

shows $q_\alpha \in \text{elts } (\mathcal{D} \ \alpha \mapsto \mathcal{D} \ \alpha \mapsto \mathcal{D} \ o)$
<proof>

lemma *q-is-equality*:

assumes $\{x, y\} \subseteq \text{elts } (\mathcal{D} \ \alpha)$
shows $(q_\alpha) \cdot x \cdot y = \mathbf{T} \iff x = y$
<proof>

Unique member selector:

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

Assignment:

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

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]: $\psi \sim_v \varphi \iff (\forall v'. v' \neq v \longrightarrow \psi \ v' = \varphi \ v')$

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* $\mathcal{V} \longleftrightarrow$

(
 $\forall \varphi.$ *is-assignment* $\varphi \longrightarrow$
 $(\forall A \alpha. A \in \text{wffs}_\alpha \longrightarrow \mathcal{V} \varphi A \in \text{elts } (\mathcal{D} \alpha)) \wedge$ — closure condition, see note in page 186
 $(\forall x \alpha. \mathcal{V} \varphi (x_\alpha) = \varphi (x, \alpha)) \wedge$
 $(\forall c \alpha. \mathcal{V} \varphi (\{c\}_\alpha) = \mathcal{J} (c, \alpha)) \wedge$
 $(\forall A B \alpha \beta. A \in \text{wffs}_{\beta \rightarrow \alpha} \wedge B \in \text{wffs}_\beta \longrightarrow \mathcal{V} \varphi (A \cdot B) = (\mathcal{V} \varphi A) \cdot (\mathcal{V} \varphi B)) \wedge$
 $(\forall x B \alpha \beta. B \in \text{wffs}_\beta \longrightarrow \mathcal{V} \varphi (\lambda x_\alpha. B) = (\lambda z : \mathcal{D} \alpha. \mathcal{V} (\varphi((x, \alpha) := z)) B))$
)

lemma *wff-denotation-function-is-domain-respecting*:

assumes *is-wff-denotation-function* \mathcal{V}

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

and *is-assignment* φ

shows $\mathcal{V} \varphi A \in \text{elts } (\mathcal{D} \alpha)$

<proof>

lemma *wff-var-denotation*:

assumes *is-wff-denotation-function* \mathcal{V}

and *is-assignment* φ

shows $\mathcal{V} \varphi (x_\alpha) = \varphi (x, \alpha)$

<proof>

lemma *wff-Q-denotation*:

assumes *is-wff-denotation-function* \mathcal{V}

and *is-assignment* φ

shows $\mathcal{V} \varphi (Q_\alpha) = q_\alpha$

<proof>

lemma *wff-iota-denotation*:

assumes *is-wff-denotation-function* \mathcal{V}

and *is-assignment* φ

shows *is-unique-member-selector* ($\mathcal{V} \varphi \iota$)

<proof>

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 $\mathcal{V} \varphi (\{c\}_\alpha) = \mathcal{J} (c, \alpha)$

<proof>

lemma *wff-app-denotation*:

assumes *is-wff-denotation-function* \mathcal{V}

and *is-assignment* φ

and $A \in \text{wffs}_{\beta \rightarrow \alpha}$
and $B \in \text{wffs}_{\beta}$
shows $\mathcal{V} \varphi (A \cdot B) = \mathcal{V} \varphi A \cdot \mathcal{V} \varphi B$
 ⟨proof⟩

lemma *wff-abs-denotation*:

assumes *is-wff-denotation-function* \mathcal{V}
and *is-assignment* φ
and $B \in \text{wffs}_{\beta}$
shows $\mathcal{V} \varphi (\lambda x_{\alpha}. B) = (\lambda z : \mathcal{D} \alpha. \mathcal{V} (\varphi((x, \alpha) := z)) B)$
 ⟨proof⟩

lemma *wff-denotation-function-is-uniquely-determined*:

assumes *is-wff-denotation-function* \mathcal{V}
and *is-wff-denotation-function* \mathcal{V}'
and *is-assignment* φ
and $A \in \text{wffs}$
shows $\mathcal{V} \varphi A = \mathcal{V}' \varphi A$
 ⟨proof⟩

end

7.3 General models

type-synonym *model-structure* = $(\text{type} \Rightarrow V) \times (\text{con} \Rightarrow V) \times ((\text{var} \Rightarrow V) \Rightarrow \text{form} \Rightarrow V)$

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 $\mathcal{V} \varphi (\lambda x_{\alpha}. B) \cdot y = \mathcal{V} (\varphi((x, \alpha) := y)) B$
 ⟨proof⟩

lemma *conj-fun-is-domain-respecting*:

assumes *is-assignment* φ
shows $\mathcal{V} \varphi (\wedge_{o \rightarrow o \rightarrow o}) \in \text{elts} (\mathcal{D} (o \rightarrow o \rightarrow o))$
 ⟨proof⟩

lemma *fully-applied-conj-fun-is-domain-respecting*:

assumes *is-assignment* φ
and $\{x, y\} \subseteq \text{elts} (\mathcal{D} o)$

shows $\mathcal{V} \varphi (\wedge_{o \rightarrow o \rightarrow o}) \cdot x \cdot y \in \text{elts } (\mathcal{D} \ o)$
<proof>

lemma *imp-fun-denotation-is-domain-respecting:*

assumes *is-assignment* φ
shows $\mathcal{V} \varphi (\supset_{o \rightarrow o \rightarrow o}) \in \text{elts } (\mathcal{D} \ (o \rightarrow o \rightarrow o))$
<proof>

lemma *fully-applied-imp-fun-denotation-is-domain-respecting:*

assumes *is-assignment* φ
and $\{x, y\} \subseteq \text{elts } (\mathcal{D} \ o)$
shows $\mathcal{V} \varphi (\supset_{o \rightarrow o \rightarrow o}) \cdot x \cdot y \in \text{elts } (\mathcal{D} \ o)$
<proof>

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 $\mathcal{M} \equiv \text{case } \mathcal{M} \text{ of } (\mathcal{D}, \mathcal{J}, \mathcal{V}) \Rightarrow \text{standard-model } \mathcal{D} \ \mathcal{J} \ \mathcal{V}$

lemma *standard-model-is-general-model:*

assumes *is-standard-model* \mathcal{M}
shows *is-general-model* \mathcal{M}
<proof>

7.5 Validity

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

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

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

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**

$\models A \equiv \forall \mathcal{M}. \text{is-general-model } \mathcal{M} \longrightarrow \mathcal{M} \models A$

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

abbreviation *is-true-sentence-in* **where**

is-true-sentence-in $A \mathcal{M} \equiv \text{is-sentence } A \wedge \mathcal{M} \models_{\text{undefined}} A$ — assignments are not meaningful

abbreviation *is-false-sentence-in* **where**

is-false-sentence-in $A \mathcal{M} \equiv \text{is-sentence } A \wedge \neg \mathcal{M} \models_{\text{undefined}} A$ — 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 $\models_S A$

<proof>

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>

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 $\mathcal{V} \varphi A = \mathcal{V} \psi A$

<proof>

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>

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>

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 $\mathcal{V} \varphi (A \equiv^Q B) = \mathbf{T} \longleftrightarrow \mathcal{V} \varphi A = \mathcal{V} \varphi B$

<proof>

lemma (in *general-model*) *prop-5401-c*:

assumes $\varphi \rightsquigarrow \mathcal{D}$

shows $\mathcal{V} \varphi T_o = \mathbf{T}$

<proof>

lemma (in *general-model*) *prop-5401-d*:

assumes $\varphi \rightsquigarrow \mathcal{D}$

shows $\mathcal{V} \varphi F_o = \mathbf{F}$

<proof>

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>

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 $\mathcal{V} \varphi (A \wedge^Q B) = \mathcal{V} \varphi A \wedge \mathcal{V} \varphi B$

<proof>

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>

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 $\mathcal{V} \varphi (A \supset^{\mathcal{Q}} B) = \mathcal{V} \varphi A \supset \mathcal{V} \varphi B$

<proof>

lemma (in *general-model*) *forall-denotation*:

assumes $\varphi \rightsquigarrow \mathcal{D}$

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

shows $\mathcal{V} \varphi (\forall x_\alpha. A) = \mathbf{T} \iff (\forall z \in \text{elts}(\mathcal{D} \alpha). \mathcal{V}(\varphi((x, \alpha) := z)) A = \mathbf{T})$

<proof>

lemma *prop-5401-g*:

assumes *is-general-model* \mathcal{M}

and $\varphi \rightsquigarrow_M \mathcal{M}$

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

shows $\mathcal{M} \models_\varphi \forall x_\alpha. A \iff (\forall \psi. \psi \rightsquigarrow_M \mathcal{M} \wedge \psi \sim_{(x, \alpha)} \varphi \longrightarrow \mathcal{M} \models_\psi A)$

<proof>

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>

lemma *axiom-1-validity*:

shows $\models \mathfrak{g}_{o \rightarrow o} \cdot T_o \wedge^{\mathcal{Q}} \mathfrak{g}_{o \rightarrow o} \cdot F_o \equiv^{\mathcal{Q}} \forall \mathfrak{r}_o. \mathfrak{g}_{o \rightarrow o} \cdot \mathfrak{r}_o$ (is $\models ?A \equiv^{\mathcal{Q}} ?B$)

<proof>

lemma (in *general-model*) *axiom-2-validity-aux*:

assumes $\varphi \rightsquigarrow \mathcal{D}$

shows $\mathcal{V} \varphi ((\mathfrak{r}_\alpha =_\alpha \eta_\alpha) \supset^{\mathcal{Q}} (\mathfrak{h}_{\alpha \rightarrow o} \cdot \mathfrak{r}_\alpha \equiv^{\mathcal{Q}} \mathfrak{h}_{\alpha \rightarrow o} \cdot \eta_\alpha)) = \mathbf{T}$ (is $\mathcal{V} \varphi (?A \supset^{\mathcal{Q}} ?B) = \mathbf{T}$)

<proof>

lemma *axiom-2-validity*:

shows $\models (\mathfrak{r}_\alpha =_\alpha \eta_\alpha) \supset^{\mathcal{Q}} (\mathfrak{h}_{\alpha \rightarrow o} \cdot \mathfrak{r}_\alpha \equiv^{\mathcal{Q}} \mathfrak{h}_{\alpha \rightarrow o} \cdot \eta_\alpha)$ (is $\models ?A \supset^{\mathcal{Q}} ?B$)

<proof>

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>

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>

lemma (in *general-model*) *axiom-4-1-con-validity-aux*:

assumes $\varphi \rightsquigarrow \mathcal{D}$
and $A \in \text{wffs}_\alpha$
shows $\forall \varphi ((\lambda x_\alpha. \llbracket c \rrbracket_\beta) \cdot A =_\beta \llbracket c \rrbracket_\beta) = \mathbf{T}$
 $\langle \text{proof} \rangle$

lemma *axiom-4-1-con-validity*:
assumes $A \in \text{wffs}_\alpha$
shows $\models (\lambda x_\alpha. \llbracket c \rrbracket_\beta) \cdot A =_\beta \llbracket c \rrbracket_\beta$
 $\langle \text{proof} \rangle$

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 $\forall \varphi ((\lambda x_\alpha. y_\beta) \cdot A =_\beta y_\beta) = \mathbf{T}$
 $\langle \text{proof} \rangle$

lemma *axiom-4-1-var-validity*:
assumes $A \in \text{wffs}_\alpha$
and $(y, \beta) \neq (x, \alpha)$
shows $\models (\lambda x_\alpha. y_\beta) \cdot A =_\beta y_\beta$
 $\langle \text{proof} \rangle$

lemma (in *general-model*) *axiom-4-2-validity-aux*:
assumes $\varphi \rightsquigarrow \mathcal{D}$
and $A \in \text{wffs}_\alpha$
shows $\forall \varphi ((\lambda x_\alpha. x_\alpha) \cdot A =_\alpha A) = \mathbf{T}$
 $\langle \text{proof} \rangle$

lemma *axiom-4-2-validity*:
assumes $A \in \text{wffs}_\alpha$
shows $\models (\lambda x_\alpha. x_\alpha) \cdot A =_\alpha A$
 $\langle \text{proof} \rangle$

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 $\forall \varphi ((\lambda x_\alpha. B \cdot C) \cdot A =_\beta ((\lambda x_\alpha. B) \cdot A) \cdot ((\lambda x_\alpha. C) \cdot A)) = \mathbf{T}$
(is $\forall \varphi (?A =_\beta ?B) = \mathbf{T}$ **)
 $\langle \text{proof} \rangle$**

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** $\models ?A =_\beta ?B$ **)
 $\langle \text{proof} \rangle$**

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⟩

lemma *axiom-4-4-validity*:
assumes $A \in \text{wffs}_\alpha$ **and** $B \in \text{wffs}_\delta$
and $(y, \gamma) \notin \{(x, \alpha)\} \cup \text{vars } A$
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⟩

lemma (in general-model) *axiom-4-5-validity-aux*:
assumes $\varphi \rightsquigarrow \mathcal{D}$
and $A \in \text{wffs}_\alpha$ **and** $B \in \text{wffs}_\delta$
shows $\mathcal{V} \varphi ((\lambda x_\alpha. \lambda x_\alpha. B) \cdot A =_{\alpha \rightarrow \delta} (\lambda x_\alpha. B)) = \mathbf{T}$
 ⟨proof⟩

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⟩

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⟩

lemma *axiom-5-validity*:
shows $\models \iota \cdot (Q_i \cdot \eta_i) =_i \eta_i$
 ⟨proof⟩

lemma *axioms-validity*:
assumes $A \in \text{axioms}$
shows $\models A$
 ⟨proof⟩

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⟩

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>

lemma *individual-proof-step-validity*:
assumes *is-proof* \mathcal{S} and $A \in \text{lset } \mathcal{S}$
shows $\models A$
<proof>

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>

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 $\mathcal{M} \models hs \supset^Q_* P$
shows $\mathcal{M} \models P$
<proof>

8.3 Proposition 5402(a)

proposition *theoremhood-implies-validity*:
assumes *is-theorem* A
shows $\models A$
<proof>

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>

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

\langle *proof* \rangle

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

interpretation *singleton-standard-premodel*: *premodel* $\mathcal{D}^S \mathcal{J}^S$

\langle *proof* \rangle

fun *singleton-standard-wff-denotation-function* (\mathcal{V}^S) **where**

$\mathcal{V}^S \varphi (x_\alpha) = \varphi (x, \alpha)$

| $\mathcal{V}^S \varphi (\llbracket e \rrbracket_\alpha) = \mathcal{J}^S (c, \alpha)$

| $\mathcal{V}^S \varphi (A \cdot B) = (\mathcal{V}^S \varphi A) \cdot (\mathcal{V}^S \varphi B)$
| $\mathcal{V}^S \varphi (\lambda x_\alpha. A) = (\lambda z : \mathcal{D}^S \alpha. \mathcal{V}^S (\varphi((x, \alpha) := z)) A)$

lemma *singleton-standard-wff-denotation-function-closure:*

assumes *frame.is-assignment* $\mathcal{D}^S \varphi$

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

shows $\mathcal{V}^S \varphi A \in \text{elts} (\mathcal{D}^S \alpha)$

<proof>

interpretation *singleton-standard-model: standard-model* $\mathcal{D}^S \mathcal{J}^S \mathcal{V}^S$

<proof>

proposition *standard-model-existence:*

shows $\exists \mathcal{M}. \text{is-standard-model } \mathcal{M}$

<proof>

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>

proposition *\mathcal{Q}_0 -is-consistent:*

shows $\neg \mathcal{Q}_0\text{-is-inconsistent}$

<proof>

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>

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.