Metatheory of $\mathcal{Q}_0$

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

March 17, 2025

Abstract

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

Contents

1 Utilities

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

1.1 Utilities for lists

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

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

lemma *rev-induct2* [*consumes 1*, *case-names Nil snoc*]:
assumes *length xs* = *length ys*
and *P* [] []
and $\bigwedge x\ xs\ y\ ys.$ *length xs* = *length ys* $\Longrightarrow$ *P xs ys* $\Longrightarrow$ *P* (*xs* @ [*x*]) (*ys* @ [*y*])
shows *P xs ys*
⟨*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* (λ*v' 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** ‹→› *101*)

primrec *type-size* :: *type* ⇒ *nat* **where**
type-size i = *1*
| *type-size o* = *1*
| *type-size* (*α* → *β*) = *Suc* (*type-size α* + *type-size β*)

primrec *subtypes* :: *type* ⇒ *type set* **where**
subtypes i = {}
| *subtypes o* = {}
| *subtypes* (*α* → *β*) = {*α*, *β*} ∪ *subtypes α* ∪ *subtypes β*

lemma *subtype-size-decrease*:
assumes *α* ∈ *subtypes β*
shows *type-size α* < *type-size β*
⟨*proof*⟩

lemma *subtype-is-not-type*:
assumes *α* ∈ *subtypes β*
shows *α* ≠ *β*
⟨*proof*⟩

lemma *fun-type-atoms-in-subtypes*:
assumes *k* < *length ts*
shows *ts* ! *k* ∈ *subtypes* (*foldr* (→) *ts γ*)
⟨*proof*⟩

lemma *fun-type-atoms-neq-fun-type*:
assumes *k* < *length ts*
shows *ts* ! *k* ≠ *foldr* (→) *ts γ*
⟨*proof*⟩

2.2 Variables

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

type-synonym *var* = *nat* × *type*

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

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

lemma *fresh-var-existence*:
assumes *finite* (*vs* :: *var set*)
obtains *x* **where** (*x*, *α*) ∉ *vs*
⟨*proof*⟩

lemma *fresh-var-name-list-existence*:
assumes *finite* (*ns* :: *nat set*)
obtains *ns′* **where** *length ns′* = *n* **and** *distinct ns′* **and** *lset ns′* ∩ *ns* = {}
⟨*proof*⟩

lemma *fresh-var-list-existence*:
fixes *xs* :: *var list*
and *ns* :: *nat set*
assumes *finite ns*
obtains *vs′* :: *var list*
where *length vs′* = *length xs*
and *distinct vs′*
and *var-name* ‘ *lset vs′* ∩ (*ns* ∪ *var-name* ‘ *lset xs*) = {}
and *map var-type vs′* = *map var-type xs*
⟨*proof*⟩

2.3 Constants

type-synonym *con* = *nat* × *type*

2.4 Formulas

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

syntax
-FVar :: *nat* ⇒ *type* ⇒ *form* (‹-_› [*899*, *0*] *900*)
-FCon :: *nat* ⇒ *type* ⇒ *form* (‹{|-|}_› [*899*, *0*] *900*)
-FAbs :: *nat* ⇒ *type* ⇒ *form* ⇒ *form* (‹(*4*λ-_./ -)› [*0*, *0*, *104*] *104*)
syntax-consts
-FVar ⇌ *FVar* **and**
-FCon ⇌ *FCon* **and**
-FAbs ⇌ *FAbs*
translations

$x_\alpha \rightleftharpoons$ *CONST FVar* (x, α)
$\{\!|c|\!\}_\alpha \rightleftharpoons$ *CONST FCon* (c, α)
$\lambda x_\alpha.\ A \rightleftharpoons$ *CONST FAbs* (x, α) A

2.5 Generalized operators

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

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

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

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

fun *form-size* :: *form* ⇒ *nat* **where**
form-size (x_α) = *1*
| *form-size* $(\{\!|c|\!\}_\alpha)$ = *1*
| *form-size* (*A* · *B*) = *Suc* (*form-size A* + *form-size B*)
| *form-size* (λx_α. *A*) = *Suc* (*form-size A*)

fun *form-depth* :: *form* ⇒ *nat* **where**
form-depth (x_α) = *0*
| *form-depth* $(\{\!|c|\!\}_\alpha)$ = *0*
| *form-depth* (*A* · *B*) = *Suc* (*max* (*form-depth A*) (*form-depth B*))
| *form-depth* (λx_α. *A*) = *Suc* (*form-depth A*)

2.6 Subformulas

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

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

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

lemma *empty-is-position* [*simp*]:
shows [] ∈ *positions A*
⟨*proof*⟩

fun *subform-at* :: *form* ⇒ *position* ⇀ *form* **where**
subform-at A [] = *Some A*

| *subform-at* $(A \cdot B)$ $(\langle\langle \# p) = \mathit{subform\text{-}at}\ A\ p$
| *subform-at* $(A \cdot B)$ $(\rangle\rangle \# p) = \mathit{subform\text{-}at}\ B\ p$
| *subform-at* $(\lambda x_\alpha.\ A)$ $(\langle\langle \# p) = \mathit{subform\text{-}at}\ A\ p$
| *subform-at* - - = *None*

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

lemma *is-subform-at-alt-def*:
  shows $A' \preceq_p A$ = (*case subform-at A p of Some B* ⇒ *B* = *A'* | *None* ⇒ *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 $A \preceq_p @\ [\langle\langle]\ C$ **and** $B \preceq_p @\ [\rangle\rangle]\ C$
  ⟨*proof*⟩

lemma *subform-at-subforms-abs*:
  assumes λx_α. $A \preceq_p C$
  shows $A \preceq_p @\ [\langle\langle]\ C$
  ⟨*proof*⟩

lemma *is-subform-implies-in-positions*:
  assumes $B \preceq_p A$
  shows $p \in \mathit{positions}\ A$
  ⟨*proof*⟩

lemma *subform-size-decrease*:
  assumes $A \preceq_p B$ **and** $p \neq []$

shows *form-size* $A <$ *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 $\nexists B.\ B \preceq_{\text{»}} \#\ p\ \lambda x_\alpha.\ A$
⟨*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
$(A = B \cdot C \wedge p = []) \vee$
$(A \neq B \cdot C \wedge$
$(\exists p' \in \mathit{positions}\ B.\ p = \text{«} \# p' \wedge A \preceq_{p'} B) \vee (\exists p' \in \mathit{positions}\ C.\ p = \text{»} \# p' \wedge A \preceq_{p'} C))$
⟨*proof*⟩

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

lemma *leftmost-subform-in-generalized-app*:
shows $B \preceq_{\mathit{replicate}\ (\mathit{length}\ As)\ \text{«}} {}^{\cdot^{\mathcal{Q}}}{}_{\star}\ B\ As$
⟨*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 $p \in$ *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 $C \preceq_{drop\ (length\ p')\ p} B$
⟨*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* $C <$ *form-size* B
⟨*proof*⟩

definition *is-subform* :: *form* ⇒ *form* ⇒ *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 $p \in \mathit{positions}\ A \longleftrightarrow (\exists\, A'.\ A' \preceq_p A)$
⟨*proof*⟩

lemma *position-prefix-is-position*:
assumes $p \in \mathit{positions}\ A$ **and** $\mathit{prefix}\ p'\ p$
shows $p' \in \mathit{positions}\ A$
⟨*proof*⟩

2.7 Free and bound variables

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

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

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

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

end

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

lemma *vars-form-finiteness*:
fixes A :: *form*
shows *finite* (*vars* A)
⟨*proof*⟩

lemma *vars-form-set-finiteness*:
fixes S :: *form set*
assumes *finite* S
shows *finite* (*vars* S)
⟨*proof*⟩

lemma *form-var-names-finiteness*:

fixes $A :: form$
shows *finite* (*var-names* A)
⟨*proof*⟩

lemma *form-set-var-names-finiteness*:
fixes $S :: form\ set$
assumes *finite* S
shows *finite* (*var-names* S)
⟨*proof*⟩

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

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

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

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

end

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

lemma *free-vars-form-finiteness*:
fixes $A :: form$
shows *finite* (*free-vars* A)
⟨*proof*⟩

lemma *free-vars-of-generalized-app*:
shows *free-vars* $(\cdot^{\mathcal{Q}}{}_\star\ A\ Bs) =$ *free-vars* $A \cup$ *free-vars* (*lset* Bs)
⟨*proof*⟩

lemma *free-vars-of-generalized-abs*:
shows *free-vars* $(\lambda^{\mathcal{Q}}{}_\star\ vs\ A) =$ *free-vars* $A -$ *lset* vs
⟨*proof*⟩

lemma *free-vars-in-all-vars*:
fixes $A :: form$
shows *free-vars* $A \subseteq$ *vars* A
⟨*proof*⟩

lemma *free-vars-in-all-vars-set*:
fixes S :: *form set*
shows *free-vars* $S \subseteq$ *vars* S
⟨*proof*⟩

lemma *singleton-form-set-vars*:
shows *vars* $\{FVar\ y\} = \{y\}$
⟨*proof*⟩

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

lemma *vars-is-free-and-bound-vars*:
shows *vars* $A =$ *free-vars* $A \cup$ *bound-vars* A
⟨*proof*⟩

fun *binders-at* :: *form* ⇒ *position* ⇒ *var set* **where**
binders-at $(A \cdot B)$ $(« \#\ p) =$ *binders-at* $A\ p$
| *binders-at* $(A \cdot B)$ $(» \#\ p) =$ *binders-at* $B\ p$
| *binders-at* $(\lambda x_\alpha.\ A)$ $(« \#\ p) = \{(x, \alpha)\} \cup$ *binders-at* $A\ p$
| *binders-at* $A\ [] = \{\}$
| *binders-at* $A\ p = \{\}$

lemma *binders-at-concat*:
assumes $A' \preceq_p A$
shows *binders-at* $A\ (p\ @\ p') =$ *binders-at* $A\ p \cup$ *binders-at* $A'\ p'$
⟨*proof*⟩

2.8 Free and bound occurrences

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

lemma *occurs-at-alt-def*:
shows *occurs-at* $v\ []\ (FVar\ v') \longleftrightarrow (v = v')$
and *occurs-at* $v\ p\ (\{\!|c|\!\}_\alpha) \longleftrightarrow$ *False*
and *occurs-at* $v\ (« \#\ p)\ (A \cdot B) \longleftrightarrow$ *occurs-at* $v\ p\ A$
and *occurs-at* $v\ (» \#\ p)\ (A \cdot B) \longleftrightarrow$ *occurs-at* $v\ p\ B$
and *occurs-at* $v\ (« \#\ p)\ (\lambda x_\alpha.\ A) \longleftrightarrow$ *occurs-at* $v\ p\ A$
and *occurs-at* $v\ (d \#\ p)\ (FVar\ v') \longleftrightarrow$ *False*
and *occurs-at* $v\ (» \#\ p)\ (\lambda x_\alpha.\ A) \longleftrightarrow$ *False*
and *occurs-at* $v\ []\ (A \cdot B) \longleftrightarrow$ *False*
and *occurs-at* $v\ []\ (\lambda x_\alpha.\ A) \longleftrightarrow$ *False*
⟨*proof*⟩

definition *occurs* :: *var* ⇒ *form* ⇒ *bool* **where**

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

lemma *occurs-in-vars*:
assumes *occurs v A*
shows *v* $\in$ *vars A*
⟨*proof*⟩

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

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

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

lemma *in-scope-of-abs-in-left-app*:
shows *in-scope-of-abs v* (« # *p*) (*A* ▪ *B*) $\longleftrightarrow$ *in-scope-of-abs v p A*
⟨*proof*⟩

lemma *in-scope-of-abs-in-right-app*:
shows *in-scope-of-abs v* (» # *p*) (*A* ▪ *B*) $\longleftrightarrow$ *in-scope-of-abs v p B*
⟨*proof*⟩

lemma *in-scope-of-abs-in-app*:
assumes *in-scope-of-abs v p* (*A* ▪ *B*)
obtains *p'* **where** (*p* = « # *p'* $\wedge$ *in-scope-of-abs v p' A*) $\vee$ (*p* = » # *p'* $\wedge$ *in-scope-of-abs v p' B*)
⟨*proof*⟩

lemma *not-in-scope-of-abs-in-app*:
assumes
$\forall$ *p'*.
(*p* = « # *p'* $\longrightarrow$ $\neg$ *in-scope-of-abs v' p' A*)
$\wedge$
(*p* = » # *p'* $\longrightarrow$ $\neg$ *in-scope-of-abs v' p' B*)
shows $\neg$ *in-scope-of-abs v' p* (*A* ▪ *B*)
⟨*proof*⟩

lemma *in-scope-of-abs-in-abs*:
shows *in-scope-of-abs v* (« # *p*) (*FAbs v′ B*) ⟷ *v* = *v′* ∨ *in-scope-of-abs v p B*
⟨*proof*⟩

lemma *not-in-scope-of-abs-in-var*:
shows ¬ *in-scope-of-abs v p* (*FVar v′*)
⟨*proof*⟩

lemma *in-scope-of-abs-in-vars*:
assumes *in-scope-of-abs v p A*
shows *v* ∈ *vars A*
⟨*proof*⟩

lemma *binders-at-alt-def*:
assumes *p* ∈ *positions A*
shows *binders-at A p* = {*v* | *v*. *in-scope-of-abs v p A*}
⟨*proof*⟩

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

lemma *not-is-bound-at-in-var*:
shows ¬ *is-bound-at v p* (*FVar v′*)
⟨*proof*⟩

lemma *not-is-bound-at-in-con*:
shows ¬ *is-bound-at v p* (*FCon k*)
⟨*proof*⟩

lemma *is-bound-at-in-left-app*:
shows *is-bound-at v* (« # *p*) (*B* ▪ *C*) ⟷ *is-bound-at v p B*
⟨*proof*⟩

lemma *is-bound-at-in-right-app*:
shows *is-bound-at v* (» # *p*) (*B* ▪ *C*) ⟷ *is-bound-at v p C*
⟨*proof*⟩

lemma *is-bound-at-from-app*:
assumes *is-bound-at v p* (*B* ▪ *C*)
obtains *p′* **where** (*p* = « # *p′* ∧ *is-bound-at v p′ B*) ∨ (*p* = » # *p′* ∧ *is-bound-at v p′ C*)
⟨*proof*⟩

lemma *is-bound-at-from-abs*:
assumes *is-bound-at v* (« # *p*) (*FAbs v′ B*)
shows *v* = *v′* ∨ *is-bound-at v p B*
⟨*proof*⟩

lemma *is-bound-at-from-absE*:

assumes *is-bound-at v p* (*FAbs v′ B*)
obtains *p′* **where** $p = \langle\!\langle \# p'$ **and** $v = v' \vee$ *is-bound-at v p′ B*
⟨*proof*⟩

lemma *is-bound-at-to-abs*:
assumes $(v = v' \wedge$ *occurs-at v p B*$) \vee$ *is-bound-at v p B*
shows *is-bound-at v* $(\langle\!\langle \# p)$ (*FAbs v′ B*)
⟨*proof*⟩

lemma *is-bound-at-in-bound-vars*:
assumes $p \in$ *positions A*
and *is-bound-at v p A* $\vee$ $v \in$ *binders-at A p*
shows $v \in$ *bound-vars A*
⟨*proof*⟩

lemma *bound-vars-in-is-bound-at*:
assumes $v \in$ *bound-vars A*
obtains *p* **where** $p \in$ *positions A* **and** *is-bound-at v p A* $\vee$ $v \in$ *binders-at A p*
⟨*proof*⟩

lemma *bound-vars-alt-def*:
shows *bound-vars A* $= \{v \mid v\ p.\ p \in$ *positions A* $\wedge$ (*is-bound-at v p A* $\vee$ $v \in$ *binders-at A p*$)\}$
⟨*proof*⟩

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

lemma *is-free-at-in-var*:
shows *is-free-at v* [] (*FVar v′*) ⟷ $v = v'$
⟨*proof*⟩

lemma *not-is-free-at-in-con*:
shows ¬ *is-free-at v* [] $(\{\!|c|\!\}_\alpha)$
⟨*proof*⟩

lemma *is-free-at-in-left-app*:
shows *is-free-at v* $(\langle\!\langle \# p)$ $(B \cdot C)$ ⟷ *is-free-at v p B*
⟨*proof*⟩

lemma *is-free-at-in-right-app*:
shows *is-free-at v* $(\rangle\!\rangle \# p)$ $(B \cdot C)$ ⟷ *is-free-at v p C*
⟨*proof*⟩

lemma *is-free-at-from-app*:
assumes *is-free-at v p* $(B \cdot C)$
obtains *p′* **where** $(p = \langle\!\langle \# p' \wedge$ *is-free-at v p′ B*$) \vee (p = \rangle\!\rangle \# p' \wedge$ *is-free-at v p′ C*$)$
⟨*proof*⟩

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

assumes *is-free-at v* (« # *p*) (*FAbs v′ B*)
shows *is-free-at v 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 immeditately preceded by λ counts as a bound variable:

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

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

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

lemma *is-free-for-absent-var* [*intro*]:
assumes *v* ∉ *vars B*
shows *is-free-for A v B*
⟨*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* ▪ *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* ▪ *C*)
⟨*proof*⟩

lemma *is-free-for-in-app*:
shows *is-free-for A v* (*B* ▪ *C*) ⟷ *is-free-for A v B* ∧ *is-free-for A v C*
⟨*proof*⟩

lemma *is-free-for-to-abs* [*intro*]:
assumes *is-free-for A v B* **and** (*x*, *α*) ∉ *free-vars A*
shows *is-free-for A v* (λx_α. *B*)
⟨*proof*⟩

lemma *is-free-for-from-abs*:
assumes *is-free-for A v* (λx_α. *B*) **and** *v* ≠ (*x*, *α*)

shows *is-free-for A v B*
⟨*proof*⟩

lemma *closed-is-free-for* [*intro*]:
assumes *free-vars A* = {}
shows *is-free-for A v B*
⟨*proof*⟩

lemma *is-free-for-closed-form* [*intro*]:
assumes *free-vars B* = {}
shows *is-free-for A v B*
⟨*proof*⟩

lemma *is-free-for-alt-def*:
shows
is-free-for A v B
⟷
(
∄ *p*.
(
p ∈ *positions B* ∧ *is-free-at v p B* ∧ *p* ≠ [] ∧
(∃ *v'* ∈ *free-vars A*. ∃ *p' C*. *strict-prefix p' p* ∧ *FAbs v' C* ⪯$_{p'}$ *B*)
)
)
⟨*proof*⟩

lemma *binding-var-not-free-for-in-abs*:
assumes *is-free x B* **and** *x* ≠ *w*
shows ¬ *is-free-for* (*FVar w*) *x* (*FAbs w B*)
⟨*proof*⟩

lemma *absent-var-is-free-for* [*intro*]:
assumes *x* ∉ *vars A*
shows *is-free-for* (*FVar x*) *y A*
⟨*proof*⟩

lemma *form-is-free-for-absent-var* [*intro*]:
assumes *x* ∉ *vars A*
shows *is-free-for B x A*
⟨*proof*⟩

lemma *form-with-free-binder-not-free-for*:
assumes *v* ≠ *v'* **and** *v'* ∈ *free-vars A* **and** *v* ∈ *free-vars B*
shows ¬ *is-free-for A v* (*FAbs v' B*)
⟨*proof*⟩

2.10 Replacement of subformulas

inductive

is-replacement-at :: *form* ⇒ *position* ⇒ *form* ⇒ *form* ⇒ *bool*
(‹(4-⦇- ← -⦈ ▷ -)› [1000, 0, 0, 0] 900)
where
pos-found: $A⦇p \leftarrow C⦈ \triangleright C'$ **if** $p = []$ **and** $C = C'$
| *replace-left-app*: $(G \cdot H)⦇« \# p \leftarrow C⦈ \triangleright (G' \cdot H)$ **if** $p \in$ *positions* G **and** $G⦇p \leftarrow C⦈ \triangleright G'$
| *replace-right-app*: $(G \cdot H)⦇» \# p \leftarrow C⦈ \triangleright (G \cdot H')$ **if** $p \in$ *positions* H **and** $H⦇p \leftarrow C⦈ \triangleright H'$
| *replace-abs*: $(\lambda x_\gamma.\ E)⦇« \# p \leftarrow C⦈ \triangleright (\lambda x_\gamma.\ E')$ **if** $p \in$ *positions* E **and** $E⦇p \leftarrow C⦈ \triangleright E'$

lemma *is-replacement-at-implies-in-positions*:
assumes $C⦇p \leftarrow A⦈ \triangleright D$
shows $p \in$ *positions* C
⟨*proof*⟩

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

lemma *is-replacement-at-existence*:
assumes $p \in$ *positions* C
obtains D **where** $C⦇p \leftarrow A⦈ \triangleright D$
⟨*proof*⟩

lemma *is-replacement-at-minimal-change*:
assumes $C⦇p \leftarrow A⦈ \triangleright D$
shows $A \preceq_p D$
and $\forall p' \in$ *positions* $D.\ \neg$ *prefix* $p'\ p \wedge \neg$ *prefix* $p\ p' \longrightarrow$ *subform-at* $D\ p' =$ *subform-at* $C\ p'$
⟨*proof*⟩

lemma *is-replacement-at-binders*:
assumes $C⦇p \leftarrow A⦈ \triangleright D$
shows *binders-at* $D\ p =$ *binders-at* $C\ p$
⟨*proof*⟩

lemma *is-replacement-at-occurs*:
assumes $C⦇p \leftarrow A⦈ \triangleright D$
and $\neg$ *prefix* $p'\ p$ **and** $\neg$ *prefix* $p\ p'$
shows *occurs-at* $v\ p'\ C \longleftrightarrow$ *occurs-at* $v\ p'\ D$
⟨*proof*⟩

lemma *fresh-var-replacement-position-uniqueness*:
assumes $v \notin$ *vars* C
and $C⦇p \leftarrow FVar\ v⦈ \triangleright G$
and *occurs-at* $v\ p'\ G$
shows $p' = p$
⟨*proof*⟩

lemma *is-replacement-at-new-positions*:
assumes $C⦇p \leftarrow A⦈ \triangleright D$ **and** *prefix* $p\ p'$ **and** $p' \in$ *positions* D
obtains p'' **where** $p' = p\ @\ p''$ **and** $p'' \in$ *positions* A
⟨*proof*⟩

lemma *replacement-override*:
assumes $C\langle\!|p \leftarrow B|\!\rangle \rhd D$ **and** $C\langle\!|p \leftarrow A|\!\rangle \rhd F$
shows $D\langle\!|p \leftarrow A|\!\rangle \rhd F$
⟨*proof*⟩

lemma *leftmost-subform-in-generalized-app-replacement*:
shows $(\cdot^{@}{}_{\star}\ C\ As)\langle\!|replicate\ (length\ As)\ «\ \leftarrow D|\!\rangle \rhd (\cdot^{@}{}_{\star}\ D\ As)$
⟨*proof*⟩

2.11 Logical constants

abbreviation (*input*) $\mathfrak{x}$ **where** $\mathfrak{x} \equiv 0$
abbreviation (*input*) $\mathfrak{y}$ **where** $\mathfrak{y} \equiv Suc\ \mathfrak{x}$
abbreviation (*input*) $\mathfrak{z}$ **where** $\mathfrak{z} \equiv Suc\ \mathfrak{y}$
abbreviation (*input*) $\mathfrak{f}$ **where** $\mathfrak{f} \equiv Suc\ \mathfrak{z}$
abbreviation (*input*) $\mathfrak{g}$ **where** $\mathfrak{g} \equiv Suc\ \mathfrak{f}$
abbreviation (*input*) $\mathfrak{h}$ **where** $\mathfrak{h} \equiv Suc\ \mathfrak{g}$
abbreviation (*input*) $\mathfrak{c}$ **where** $\mathfrak{c} \equiv Suc\ \mathfrak{h}$
abbreviation (*input*) $\mathfrak{c}_Q$ **where** $\mathfrak{c}_Q \equiv Suc\ \mathfrak{c}$
abbreviation (*input*) $\mathfrak{c}_\iota$ **where** $\mathfrak{c}_\iota \equiv Suc\ \mathfrak{c}_Q$

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

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

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

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

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

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

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

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

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

shows P
⟨*proof*⟩

2.12 Definitions and abbreviations

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

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

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

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

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

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

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

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

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

lemma *innermost-replacement-in-generalized-forall*:
assumes $vs \neq []$
shows $(\forall^{\mathfrak{Q}}_\star\ vs\ C)\langle\!|foldr\ (\lambda\text{-}.\ (@)\ [»,«])\ vs\ []\leftarrow B|\!\rangle \rhd (\forall^{\mathfrak{Q}}_\star\ vs\ B)$
⟨*proof*⟩

lemma *false-is-forall*:
shows $F_o = \forall\mathfrak{x}_o.\ \mathfrak{x}_o$
⟨*proof*⟩

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

definition *conj-op* :: *form* ⇒ *form* ⇒ *form* (**infixl** ‹∧$^{\mathfrak{Q}}$› *131*) **where**

[*simp*]: $A \wedge^{\mathfrak{Q}} B = \wedge_{o\to o\to o} \cdot A \cdot B$

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

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

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

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

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

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

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

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

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

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

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

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

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

2.13 Well-formed formulas

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

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

abbreviation *wffs* :: *form set* **where**

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

lemma *is-wff-of-type-wffs-of-type-eq* [*pred-set-conv*]:
shows *is-wff-of-type* $\alpha = (\lambda f.\ f \in wffs_\alpha)$
⟨*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 *length As* = *length ts*
and $\forall\, k < length\ As.\ As\ !\ k \in wffs_{ts\ !\ k}$
and $B \in wffs_{foldr\ (\to)\ ts\ \beta}$
shows $\cdot^{\mathcal{Q}}_\star\ B\ As \in wffs_\beta$
⟨*proof*⟩

lemma *generalized-abs-wff* [*intro*]:
assumes $B \in wffs_\beta$
shows $\lambda^{\mathcal{Q}}_\star\ vs\ B \in wffs_{foldr\ (\to)\ (map\ snd\ vs)\ \beta}$
⟨*proof*⟩

lemma *Q-wff* [*intro*]:
shows $Q_\alpha \in wffs_{\alpha\to\alpha\to o}$
⟨*proof*⟩

lemma *iota-wff* [*intro*]:
shows $\iota \in wffs_{(i\to o)\to i}$
⟨*proof*⟩

lemma *equality-wff* [*intro*]:
assumes $A \in wffs_\alpha$ **and** $B \in wffs_\alpha$
shows $A =_\alpha B \in wffs_o$
⟨*proof*⟩

lemma *equivalence-wff* [*intro*]:
assumes $A \in wffs_o$ **and** $B \in wffs_o$
shows $A \equiv^{\mathcal{Q}} B \in wffs_o$
⟨*proof*⟩

lemma *true-wff* [*intro*]:
shows $T_o \in wffs_o$
⟨*proof*⟩

lemma *false-wff* [*intro*]:
shows $F_o \in wffs_o$
⟨*proof*⟩

lemma *pi-wff* [*intro*]:
shows $\prod \alpha \in \mathit{wffs}_{(\alpha \to o) \to o}$
⟨*proof*⟩

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

lemma *generalized-forall-wff* [*intro*]:
assumes $B \in \mathit{wffs}_o$
shows $\forall^{\mathfrak{Q}}_\star\ \mathit{vs}\ B \in \mathit{wffs}_o$
⟨*proof*⟩

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

lemma *conj-op-wff* [*intro*]:
assumes $A \in \mathit{wffs}_o$ **and** $B \in \mathit{wffs}_o$
shows $A \wedge^{\mathfrak{Q}} B \in \mathit{wffs}_o$
⟨*proof*⟩

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

lemma *imp-op-wff* [*intro*]:
assumes $A \in \mathit{wffs}_o$ **and** $B \in \mathit{wffs}_o$
shows $A \supset^{\mathfrak{Q}} B \in \mathit{wffs}_o$
⟨*proof*⟩

lemma *neg-wff* [*intro*]:
assumes $A \in \mathit{wffs}_o$
shows $\sim^{\mathfrak{Q}} A \in \mathit{wffs}_o$
⟨*proof*⟩

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

lemma *disj-op-wff* [*intro*]:
assumes $A \in \mathit{wffs}_o$ **and** $B \in \mathit{wffs}_o$
shows $A \vee^{\mathfrak{Q}} B \in \mathit{wffs}_o$
⟨*proof*⟩

lemma *exists-wff* [*intro*]:
assumes $A \in \mathit{wffs}_o$

shows $\exists\, x_\alpha.\ A \in \mathit{wffs}_o$
⟨*proof*⟩

lemma *inequality-wff* [*intro*]:
assumes $A \in \mathit{wffs}_\alpha$ **and** $B \in \mathit{wffs}_\alpha$
shows $A \neq_\alpha B \in \mathit{wffs}_o$
⟨*proof*⟩

lemma *wffs-from-app*:
assumes $A \cdot B \in \mathit{wffs}_\beta$
obtains α **where** $A \in \mathit{wffs}_{\alpha\to\beta}$ **and** $B \in \mathit{wffs}_\alpha$
⟨*proof*⟩

lemma *wffs-from-generalized-app*:
assumes $\cdot^{@}_\star\ B\ As \in \mathit{wffs}_\beta$
obtains *ts*
where *length ts* = *length As*
and $\forall\, k < \mathit{length}\ As.\ As\ !\ k \in \mathit{wffs}_{ts\ !\ k}$
and $B \in \mathit{wffs}_{\mathit{foldr}\ (\to)\ ts\ \beta}$
⟨*proof*⟩

lemma *wffs-from-abs*:
assumes $\lambda x_\alpha.\ A \in \mathit{wffs}_\gamma$
obtains β **where** $\gamma = \alpha\to\beta$ **and** $A \in \mathit{wffs}_\beta$
⟨*proof*⟩

lemma *wffs-from-equality*:
assumes $A =_\alpha B \in \mathit{wffs}_o$
shows $A \in \mathit{wffs}_\alpha$ **and** $B \in \mathit{wffs}_\alpha$
⟨*proof*⟩

lemma *wffs-from-equivalence*:
assumes $A \equiv^{@} B \in \mathit{wffs}_o$
shows $A \in \mathit{wffs}_o$ **and** $B \in \mathit{wffs}_o$
⟨*proof*⟩

lemma *wffs-from-forall*:
assumes $\forall\, x_\alpha.\ A \in \mathit{wffs}_o$
shows $A \in \mathit{wffs}_o$
⟨*proof*⟩

lemma *wffs-from-conj-fun*:
assumes $\wedge_{o\to o\to o} \cdot A \cdot B \in \mathit{wffs}_o$
shows $A \in \mathit{wffs}_o$ **and** $B \in \mathit{wffs}_o$
⟨*proof*⟩

lemma *wffs-from-conj-op*:
assumes $A \wedge^{@} B \in \mathit{wffs}_o$
shows $A \in \mathit{wffs}_o$ **and** $B \in \mathit{wffs}_o$

⟨*proof*⟩

lemma *wffs-from-imp-fun*:
assumes $\supset_{o\to o\to o} \cdot A \cdot B \in \mathit{wffs}_o$
shows $A \in \mathit{wffs}_o$ **and** $B \in \mathit{wffs}_o$
⟨*proof*⟩

lemma *wffs-from-imp-op*:
assumes $A \supset^{\mathfrak{Q}} B \in \mathit{wffs}_o$
shows $A \in \mathit{wffs}_o$ **and** $B \in \mathit{wffs}_o$
⟨*proof*⟩

lemma *wffs-from-neg*:
assumes $\sim^{\mathfrak{Q}} A \in \mathit{wffs}_o$
shows $A \in \mathit{wffs}_o$
⟨*proof*⟩

lemma *wffs-from-disj-fun*:
assumes $\vee_{o\to o\to o} \cdot A \cdot B \in \mathit{wffs}_o$
shows $A \in \mathit{wffs}_o$ **and** $B \in \mathit{wffs}_o$
⟨*proof*⟩

lemma *wffs-from-disj-op*:
assumes $A \vee^{\mathfrak{Q}} B \in \mathit{wffs}_o$
shows $A \in \mathit{wffs}_o$ **and** $B \in \mathit{wffs}_o$
⟨*proof*⟩

lemma *wffs-from-exists*:
assumes $\exists\, x_\alpha.\ A \in \mathit{wffs}_o$
shows $A \in \mathit{wffs}_o$
⟨*proof*⟩

lemma *wffs-from-inequality*:
assumes $A \neq_\alpha B \in \mathit{wffs}_o$
shows $A \in \mathit{wffs}_\alpha$ **and** $B \in \mathit{wffs}_\alpha$
⟨*proof*⟩

lemma *wff-has-unique-type*:
assumes $A \in \mathit{wffs}_\alpha$ **and** $A \in \mathit{wffs}_\beta$
shows $\alpha = \beta$
⟨*proof*⟩

lemma *wffs-of-type-o-induct* [*consumes 1*, *case-names Var Con App*]:
assumes $A \in \mathit{wffs}_o$
and $\bigwedge x.\ \mathcal{P}\ (x_o)$
and $\bigwedge c.\ \mathcal{P}\ (\{\!|c|\!\}_o)$
and $\bigwedge A\ B\ \alpha.\ A \in \mathit{wffs}_{\alpha\to o} \Longrightarrow B \in \mathit{wffs}_\alpha \Longrightarrow \mathcal{P}\ (A \cdot B)$
shows $\mathcal{P}\ A$
⟨*proof*⟩

lemma *diff-types-implies-diff-wffs*:
assumes $A \in wffs_\alpha$ **and** $B \in wffs_\beta$
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** $\forall\, C \in lset\ Cs.$ *is-free-for A v C*
shows *is-free-for A v* ($\blacksquare^{\mathcal{Q}}_\star$ *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** $(x, \alpha) \notin$ *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** *lset vs* $\cap$ *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* ($\sim^{\mathfrak{Q}}$ *B*)
⟨*proof*⟩

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

lemma *replacement-preserves-typing*:
assumes $C\langle\!|p \leftarrow B|\!\rangle \rhd D$
and $A \preceq_p C$
and $A \in wffs_\alpha$ **and** $B \in wffs_\alpha$
shows $C \in wffs_\beta \longleftrightarrow D \in wffs_\beta$
⟨*proof*⟩

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

Closed formulas and sentences:

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

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

2.14 Substitutions

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

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

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

lemma *empty-substitution-neutrality*:
shows S {$$} *A* = *A*
⟨*proof*⟩

lemma *substitution-preserves-typing*:
assumes *is-substitution* ϑ
and $A \in wffs_\alpha$
shows **S** ϑ $A \in wffs_\alpha$
⟨*proof*⟩

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

lemma *generalized-app-substitution*:
shows **S** ϑ $(\cdot^{\mathcal{Q}}{}_\star\ A\ Bs) = \cdot^{\mathcal{Q}}{}_\star$ (**S** ϑ A) (*map* $(\lambda B.$ **S** ϑ $B)$ Bs)
⟨*proof*⟩

lemma *generalized-abs-substitution*:
shows **S** ϑ $(\lambda^{\mathcal{Q}}{}_\star\ vs\ A) = \lambda^{\mathcal{Q}}{}_\star\ vs$ (**S** (*fmdrop-set* (*fmdom'* $\vartheta \cap$ *lset vs*) ϑ) A)
⟨*proof*⟩

lemma *generalized-forall-substitution*:
shows **S** ϑ $(\forall^{\mathcal{Q}}{}_\star\ vs\ A) = \forall^{\mathcal{Q}}{}_\star\ vs$ (**S** (*fmdrop-set* (*fmdom'* $\vartheta \cap$ *lset vs*) ϑ) A)
⟨*proof*⟩

lemma *singleton-substitution-simps*:
shows **S** $\{(x, \alpha) \rightarrowtail A\}$ $(y_\beta) =$ (*if* $(x, \alpha) \neq (y, \beta)$ *then* y_β *else* A)
and **S** $\{(x, \alpha) \rightarrowtail A\}$ $(\{\!|c|\!\}_\alpha) = \{\!|c|\!\}_\alpha$
and **S** $\{(x, \alpha) \rightarrowtail A\}$ $(B \cdot C) =$ (**S** $\{(x, \alpha) \rightarrowtail A\}$ B) $\cdot$ (**S** $\{(x, \alpha) \rightarrowtail A\}$ C)
and **S** $\{(x, \alpha) \rightarrowtail A\}$ $(\lambda y_\beta.\ B) = \lambda y_\beta.$ (*if* $(x, \alpha) = (y, \beta)$ *then* B *else* **S** $\{(x, \alpha) \rightarrowtail A\}$ B)
⟨*proof*⟩

lemma *substitution-preserves-freeness*:
assumes $y \notin$ *free-vars* A **and** $y \neq z$
shows $y \notin$ *free-vars* **S** $\{x \rightarrowtail FVar\ z\}$ A
⟨*proof*⟩

lemma *renaming-substitution-minimal-change*:
assumes $y \notin$ *vars* A **and** $y \neq z$
shows $y \notin$ *vars* (**S** $\{x \rightarrowtail FVar\ z\}$ A)

⟨*proof*⟩

lemma *free-var-singleton-substitution-neutrality*:
assumes $v \notin$ *free-vars* A
shows **S** $\{v \rightarrowtail B\}$ $A = A$
⟨*proof*⟩

lemma *identity-singleton-substitution-neutrality*:
shows **S** $\{v \rightarrowtail FVar\ v\}$ $A = A$
⟨*proof*⟩

lemma *free-var-in-renaming-substitution*:
assumes $x \neq y$
shows $(x, \alpha) \notin$ *free-vars* (**S** $\{(x, \alpha) \rightarrowtail y_\alpha\}$ B)
⟨*proof*⟩

lemma *renaming-substitution-preserves-form-size*:
shows *form-size* (**S** $\{v \rightarrowtail FVar\ v'\}$ A) = *form-size* A
⟨*proof*⟩

The following lemma corresponds to X5100 in [2]:

lemma *substitution-composability*:
assumes $v' \notin$ *vars* B
shows **S** $\{v' \rightarrowtail A\}$ **S** $\{v \rightarrowtail FVar\ v'\}$ B = **S** $\{v \rightarrowtail A\}$ B
⟨*proof*⟩

The following lemma corresponds to X5101 in [2]:

lemma *renaming-substitution-composability*:
assumes $z \notin$ *free-vars* A **and** *is-free-for* (*FVar* z) x A
shows **S** $\{z \rightarrowtail FVar\ y\}$ **S** $\{x \rightarrowtail FVar\ z\}$ A = **S** $\{x \rightarrowtail FVar\ y\}$ A
⟨*proof*⟩

lemma *absent-vars-substitution-preservation*:
assumes $v \notin$ *vars* A
and $\forall\, v' \in$ *fmdom'* ϑ. $v \notin$ *vars* (ϑ \$\$! v')
shows $v \notin$ *vars* (**S** ϑ A)
⟨*proof*⟩

lemma *substitution-free-absorption*:
assumes ϑ \$\$ v = *None* **and** $v \notin$ *free-vars* B
shows **S** $(\{v \rightarrowtail A\} +\!\!+_f \vartheta)$ B = **S** ϑ B
⟨*proof*⟩

lemma *substitution-absorption*:
assumes ϑ \$\$ v = *None* **and** $v \notin$ *vars* B
shows **S** $(\{v \rightarrowtail A\} +\!\!+_f \vartheta)$ B = **S** ϑ B
⟨*proof*⟩

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

assumes *is-free-for A x B*
and $y \notin vars\ B$
and $x \notin fmdom'\ \vartheta$
and $\forall\, v \in fmdom'\ \vartheta.\ y \notin vars\ (\vartheta\ \$\$!\ v)$
and $\forall\, v \in fmdom'\ \vartheta.$ *is-free-for* $(\vartheta\ \$\$!\ v)\ v\ B$
shows *is-free-for A y* $(\mathbf{S}\ (\{x \rightarrowtail FVar\ y\} +\!\!+_f \vartheta)\ B)$
⟨*proof*⟩

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

lemma *substitution-fusion*:
assumes *is-substitution* ϑ **and** *is-substitution* $\{v \rightarrowtail A\}$
and $\vartheta\ \$\$\ v = None$ **and** $\forall\, v' \in fmdom'\ \vartheta.\ v \notin vars\ (\vartheta\ \$\$!\ v')$
shows $\mathbf{S}\ \{v \rightarrowtail A\}\ \mathbf{S}\ \vartheta\ B = \mathbf{S}\ (\{v \rightarrowtail A\} +\!\!+_f \vartheta)\ B$
⟨*proof*⟩

lemma *updated-substitution-is-substitution*:
assumes $v \notin fmdom'\ \vartheta$ **and** *is-substitution* $(\vartheta(v \rightarrowtail A))$
shows *is-substitution* ϑ
⟨*proof*⟩

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

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

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

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

lemma *fresh-vars-substitution-unfolding*:
fixes *ps* :: (*var* × *form*) *list*
assumes $\vartheta =$ *fmap-of-list ps* **and** *is-renaming-substitution* ϑ
and *distinct* (*map fst ps*) **and** *distinct* (*map snd ps*)
and $vars\ (fmran'\ \vartheta) \cap (fmdom'\ \vartheta \cup vars\ B) = \{\}$
shows $\mathbf{S}\ \vartheta\ B = foldr\ (\lambda(x,\ y)\ C.\ \mathbf{S}\ \{x \rightarrowtail y\}\ C)\ ps\ B$
⟨*proof*⟩

lemma *free-vars-agreement-substitution-equality*:
assumes $fmdom'\ \vartheta = fmdom'\ \vartheta'$
and $\forall\, v \in free\text{-}vars\ A \cap fmdom'\ \vartheta.\ \vartheta\ \$\$!\ v = \vartheta'\ \$\$!\ v$
shows $\mathbf{S}\ \vartheta\ A = \mathbf{S}\ \vartheta'\ A$
⟨*proof*⟩

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

lemma *substitution-consolidation*:
assumes $v \notin$ *fmdom'* ϑ
and $\forall\, v' \in$ *fmdom'* ϑ. *is-free-for* (ϑ \$\$! v') v' B
shows **S** $\{v \rightarrowtail A\}$ **S** ϑ B = **S** ($\{v \rightarrowtail A\}$ $+\!\!+_f$ *fmmap* ($\lambda A'$. **S** $\{v \rightarrowtail A\}$ A') ϑ) B
⟨*proof*⟩

lemma *vars-range-substitution*:
assumes *is-substitution* ϑ
and $v \notin$ *vars* (*fmran'* ϑ)
shows $v \notin$ *vars* (*fmran'* (*fmdrop* w ϑ))
⟨*proof*⟩

lemma *excluded-var-from-substitution*:
assumes *is-substitution* ϑ
and $v \notin$ *fmdom'* ϑ
and $v \notin$ *vars* (*fmran'* ϑ)
and $v \notin$ *vars* A
shows $v \notin$ *vars* (**S** ϑ A)
⟨*proof*⟩

2.15 Renaming of bound variables

fun *rename-bound-var* :: *var* ⇒ *nat* ⇒ *form* ⇒ *form* **where**
rename-bound-var v y (x_α) = x_α
| *rename-bound-var* v y ($\{\!|c|\!\}_\alpha$) = $\{\!|c|\!\}_\alpha$
| *rename-bound-var* v y (B ▪ C) = *rename-bound-var* v y B ▪ *rename-bound-var* v y C
| *rename-bound-var* v y (λx_α. B) =
(
if (x, α) = v *then*
λy_α. **S** $\{(x, \alpha) \rightarrowtail y_\alpha\}$ (*rename-bound-var* v y B)
else
λx_α. (*rename-bound-var* v y B)
)

lemma *rename-bound-var-preserves-typing*:
assumes $A \in$ *wffs*$_\alpha$
shows *rename-bound-var* (y, γ) z $A \in$ *wffs*$_\alpha$
⟨*proof*⟩

lemma *old-bound-var-not-free-in-abs-after-renaming*:
assumes $A \in$ *wffs*$_\alpha$
and $z_\gamma \neq y_\gamma$
and (z, γ) $\notin$ *vars* A
shows (y, γ) $\notin$ *free-vars* (*rename-bound-var* (y, γ) z (λy_γ. A))
⟨*proof*⟩

lemma *rename-bound-var-free-vars*:
assumes $A \in \mathit{wffs}_\alpha$
and $z_\gamma \neq y_\gamma$
and $(z, \gamma) \notin \mathit{vars}\ A$
shows $(z, \gamma) \notin \mathit{free\text{-}vars}\ (\mathit{rename\text{-}bound\text{-}var}\ (y, \gamma)\ z\ A)$
⟨*proof*⟩

lemma *old-bound-var-not-free-after-renaming*:
assumes $A \in \mathit{wffs}_\alpha$
and $z_\gamma \neq y_\gamma$
and $(z, \gamma) \notin \mathit{vars}\ A$
and $(y, \gamma) \notin \mathit{free\text{-}vars}\ A$
shows $(y, \gamma) \notin \mathit{free\text{-}vars}\ (\mathit{rename\text{-}bound\text{-}var}\ (y, \gamma)\ z\ A)$
⟨*proof*⟩

lemma *old-bound-var-not-ocurring-after-renaming*:
assumes $A \in \mathit{wffs}_\alpha$
and $z_\gamma \neq y_\gamma$
shows $\neg\ \mathit{occurs\text{-}at}\ (y, \gamma)\ p\ (\mathbf{S}\ \{(y, \gamma) \rightarrowtail z_\gamma\}\ (\mathit{rename\text{-}bound\text{-}var}\ (y, \gamma)\ z\ A))$
⟨*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 $A \in \mathit{wffs}_\alpha$
and $z_\gamma \neq y_\gamma$
and $(z, \gamma) \notin \mathit{vars}\ A$
and $\mathit{occurs\text{-}at}\ (y, \gamma)\ p\ (\mathit{rename\text{-}bound\text{-}var}\ (y, \gamma)\ z\ A)$
shows $\neg\ \mathit{in\text{-}scope\text{-}of\text{-}abs}\ (z, \gamma)\ p\ (\mathit{rename\text{-}bound\text{-}var}\ (y, \gamma)\ z\ A)$
⟨*proof*⟩

lemma *is-free-for-in-rename-bound-var*:
assumes $A \in \mathit{wffs}_\alpha$
and $z_\gamma \neq y_\gamma$
and $(z, \gamma) \notin \mathit{vars}\ A$
shows $\mathit{is\text{-}free\text{-}for}\ (z_\gamma)\ (y, \gamma)\ (\mathit{rename\text{-}bound\text{-}var}\ (y, \gamma)\ z\ A)$
⟨*proof*⟩

lemma *renaming-substitution-preserves-bound-vars*:
shows $\mathit{bound\text{-}vars}\ (\mathbf{S}\ \{(y, \gamma) \rightarrowtail z_\gamma\}\ A) = \mathit{bound\text{-}vars}\ A$
⟨*proof*⟩

lemma *rename-bound-var-bound-vars*:
assumes $A \in \mathit{wffs}_\alpha$
and $z_\gamma \neq y_\gamma$
shows $(y, \gamma) \notin \mathit{bound\text{-}vars}\ (\mathit{rename\text{-}bound\text{-}var}\ (y, \gamma)\ z\ A)$
⟨*proof*⟩

lemma *old-var-not-free-not-occurring-after-rename*:

assumes $A \in \mathit{wffs}_\alpha$
and $z_\gamma \neq y_\gamma$
and $(y, \gamma) \notin \mathit{free\text{-}vars}\ A$
and $(z, \gamma) \notin \mathit{vars}\ A$
shows $(y, \gamma) \notin \mathit{vars}\ (\mathit{rename\text{-}bound\text{-}var}\ (y, \gamma)\ z\ A)$
⟨*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* ⇒ 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 ⇒ *bool* **where**
[*simp*]: *bool-from-V* = *inv bool-to-V*

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

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

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

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

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

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

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

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

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

and $((p \vee q) \supset r) = ((p \supset r) \wedge (q \supset r))$
and $((p \supset q) \vee r) = (p \supset q \vee r)$
and $(q \vee (p \supset r)) = (p \supset q \vee r)$
and $\sim (p \vee q) = \sim p \wedge \sim q$
and $\sim (p \wedge q) = \sim p \vee \sim q$
and $\sim (p \supset q) = p \wedge \sim q$
and $\sim p \vee q = (p \supset q)$
and $p \vee \sim q = (q \supset p)$
and $(p \supset q) = (\sim p) \vee q$
and $p \vee q = \sim p \supset q$
and $(p \equiv q) = (p \supset q) \wedge (q \supset p)$
and $(p \supset q) \wedge (\sim p \supset q) = q$
and $p = \mathbf{T} \Longrightarrow \neg (p = \mathbf{F})$
and $p = \mathbf{F} \Longrightarrow \neg (p = \mathbf{T})$
and $p = \mathbf{T} \vee p = \mathbf{F}$
⟨*proof*⟩

lemma *tv-cases* [*consumes 1*, *case-names top bottom*, *cases type*: *V*]:
assumes $p \in elts\ \mathbb{B}$
and $p = \mathbf{T} \Longrightarrow P$
and $p = \mathbf{F} \Longrightarrow P$
shows *P*
⟨*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: $T_o \in pwffs$
| *F-pwff*: $F_o \in pwffs$
| *var-pwff*: $p_o \in pwffs$
| *neg-pwff*: $\sim^{\mathfrak{Q}} A \in pwffs$ **if** $A \in pwffs$
| *conj-pwff*: $A \wedge^{\mathfrak{Q}} B \in pwffs$ **if** $A \in pwffs$ **and** $B \in pwffs$
| *disj-pwff*: $A \vee^{\mathfrak{Q}} B \in pwffs$ **if** $A \in pwffs$ **and** $B \in pwffs$
| *imp-pwff*: $A \supset^{\mathfrak{Q}} B \in pwffs$ **if** $A \in pwffs$ **and** $B \in pwffs$
| *eqv-pwff*: $A \equiv^{\mathfrak{Q}} B \in pwffs$ **if** $A \in pwffs$ **and** $B \in pwffs$

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

lemma *pwffs-distinctnesses* [*induct-simp*]:

shows $T_o \neq F_o$
and $T_o \neq p_o$
and $T_o \neq \sim^{\mathfrak{Q}} A$
and $T_o \neq A \wedge^{\mathfrak{Q}} B$
and $T_o \neq A \vee^{\mathfrak{Q}} B$
and $T_o \neq A \supset^{\mathfrak{Q}} B$
and $T_o \neq A \equiv^{\mathfrak{Q}} B$
and $F_o \neq p_o$
and $F_o \neq \sim^{\mathfrak{Q}} A$
and $F_o \neq A \wedge^{\mathfrak{Q}} B$
and $F_o \neq A \vee^{\mathfrak{Q}} B$
and $F_o \neq A \supset^{\mathfrak{Q}} B$
and $F_o \neq A \equiv^{\mathfrak{Q}} B$
and $p_o \neq \sim^{\mathfrak{Q}} A$
and $p_o \neq A \wedge^{\mathfrak{Q}} B$
and $p_o \neq A \vee^{\mathfrak{Q}} B$
and $p_o \neq A \supset^{\mathfrak{Q}} B$
and $p_o \neq A \equiv^{\mathfrak{Q}} B$
and $\sim^{\mathfrak{Q}} A \neq B \wedge^{\mathfrak{Q}} C$
and $\sim^{\mathfrak{Q}} A \neq B \vee^{\mathfrak{Q}} C$
and $\sim^{\mathfrak{Q}} A \neq B \supset^{\mathfrak{Q}} C$
and $\neg\ (B = F_o \wedge A = C) \Longrightarrow \sim^{\mathfrak{Q}} A \neq B \equiv^{\mathfrak{Q}} C$ — $\sim^{\mathfrak{Q}} A$ is the same as $F_o \equiv^{\mathfrak{Q}} A$
and $A \wedge^{\mathfrak{Q}} B \neq C \vee^{\mathfrak{Q}} D$
and $A \wedge^{\mathfrak{Q}} B \neq C \supset^{\mathfrak{Q}} D$
and $A \wedge^{\mathfrak{Q}} B \neq C \equiv^{\mathfrak{Q}} D$
and $A \vee^{\mathfrak{Q}} B \neq C \supset^{\mathfrak{Q}} D$
and $A \vee^{\mathfrak{Q}} B \neq C \equiv^{\mathfrak{Q}} D$
and $A \supset^{\mathfrak{Q}} B \neq C \equiv^{\mathfrak{Q}} D$
⟨*proof*⟩

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

lemma *pwff-from-neg-pwff* [*elim*!]:
assumes $\sim^{\mathfrak{Q}} A \in$ *pwffs*
shows $A \in$ *pwffs*
⟨*proof*⟩

lemma *pwffs-from-conj-pwff* [*elim*!]:
assumes $A \wedge^{\mathfrak{Q}} B \in$ *pwffs*
shows $\{A, B\} \subseteq$ *pwffs*
⟨*proof*⟩

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

assumes $A \vee^{\mathfrak{Q}} B \in \mathit{pwffs}$
shows $\{A, B\} \subseteq \mathit{pwffs}$
⟨*proof*⟩

lemma *pwffs-from-imp-pwff* [*elim*!]:
assumes $A \supset^{\mathfrak{Q}} B \in \mathit{pwffs}$
shows $\{A, B\} \subseteq \mathit{pwffs}$
⟨*proof*⟩

lemma *pwffs-from-eqv-pwff* [*elim*!]:
assumes $A \equiv^{\mathfrak{Q}} B \in \mathit{pwffs}$
shows $\{A, B\} \subseteq \mathit{pwffs}$
⟨*proof*⟩

lemma *pwffs-subset-of-wffso*:
shows $\mathit{pwffs} \subseteq \mathit{wffs}_o$
⟨*proof*⟩

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

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

lemma *is-free-for-in-pwff* [*intro*]:
assumes $A \in \mathit{pwffs}$
and $v \in \mathit{free\text{-}vars}\ A$
shows $\mathit{is\text{-}free\text{-}for}\ B\ v\ A$
⟨*proof*⟩

4.2 Semantics

Assignment of truth values to propositional variables:

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

Denotation of a pwff:

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

[*iff*]: *is-pwff-denotation-function* $\mathcal{V} \longleftrightarrow$
(
$\forall \varphi$. *is-tv-assignment* $\varphi \longrightarrow$
(
$\mathcal{V}\ \varphi\ T_o = \mathbf{T} \wedge$
$\mathcal{V}\ \varphi\ F_o = \mathbf{F} \wedge$
$(\forall p.\ \mathcal{V}\ \varphi\ (p_o) = \varphi\ p) \wedge$
$(\forall A.\ A \in \mathit{pwffs} \longrightarrow \mathcal{V}\ \varphi\ (\sim^{\mathfrak{Q}} A) = \sim \mathcal{V}\ \varphi\ A) \wedge$
$(\forall A\ B.\ A \in \mathit{pwffs} \wedge B \in \mathit{pwffs} \longrightarrow \mathcal{V}\ \varphi\ (A \wedge^{\mathfrak{Q}} B) = \mathcal{V}\ \varphi\ A \wedge \mathcal{V}\ \varphi\ B) \wedge$
$(\forall A\ B.\ A \in \mathit{pwffs} \wedge B \in \mathit{pwffs} \longrightarrow \mathcal{V}\ \varphi\ (A \vee^{\mathfrak{Q}} B) = \mathcal{V}\ \varphi\ A \vee \mathcal{V}\ \varphi\ B) \wedge$
$(\forall A\ B.\ A \in \mathit{pwffs} \wedge B \in \mathit{pwffs} \longrightarrow \mathcal{V}\ \varphi\ (A \supset^{\mathfrak{Q}} B) = \mathcal{V}\ \varphi\ A \supset \mathcal{V}\ \varphi\ B) \wedge$
$(\forall A\ B.\ A \in \mathit{pwffs} \wedge B \in \mathit{pwffs} \longrightarrow \mathcal{V}\ \varphi\ (A \equiv^{\mathfrak{Q}} B) = \mathcal{V}\ \varphi\ A \equiv \mathcal{V}\ \varphi\ B)$
)
)

lemma *pwff-denotation-is-truth-value*:
assumes $A \in \mathit{pwffs}$
and *is-tv-assignment* φ
and *is-pwff-denotation-function* $\mathcal{V}$
shows $\mathcal{V}\ \varphi\ A \in \mathit{elts}\ \mathbb{B}$
⟨*proof*⟩

lemma *closed-pwff-is-meaningful-regardless-of-assignment*:
assumes $A \in \mathit{pwffs}$
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* $\varphi\ (\sim^{\mathfrak{Q}} A)\ (\sim\ b_A)$ **if** $\mathcal{V}_B$-*graph* $\varphi\ A\ b_A$
| $\mathcal{V}_B$-*graph-conj*: $\mathcal{V}_B$-*graph* $\varphi\ (A \wedge^{\mathfrak{Q}} B)\ (b_A \wedge b_B)$ **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* $\varphi\ (A \vee^{\mathfrak{Q}} B)\ (b_A \vee b_B)$ **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* $\varphi\ (A \supset^{\mathfrak{Q}} B)\ (b_A \supset b_B)$ **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* $\varphi\ (A \equiv^{\mathfrak{Q}} B)\ (b_A \equiv b_B)$ **if** $\mathcal{V}_B$-*graph* $\varphi\ A\ b_A$ **and** $\mathcal{V}_B$-*graph* $\varphi\ B\ b_B$ **and** $A \neq F_o$

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

lemma $\mathcal{V}_B$-*graph-denotation-is-truth-value* [*elim*!]:
assumes $\mathcal{V}_B$-*graph* $\varphi\ A\ b$
and *is-tv-assignment* φ
shows $b \in \mathit{elts}\ \mathbb{B}$
⟨*proof*⟩

lemma $\mathcal{V}_B$*-graph-denotation-uniqueness*:
assumes $A \in$ *pwffs*
and *is-tv-assignment* φ
and $\mathcal{V}_B$*-graph* φ A b **and** $\mathcal{V}_B$*-graph* φ A b'
shows $b = b'$
⟨*proof*⟩

lemma $\mathcal{V}_B$*-graph-denotation-existence*:
assumes $A \in$ *pwffs*
and *is-tv-assignment* φ
shows $\exists\, b.\ \mathcal{V}_B$*-graph* φ A b
⟨*proof*⟩

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

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

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

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

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

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

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

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

assumes *is-tv-assignment* φ
shows $\mathcal{V}_B\ \varphi\ (p_o) = \varphi\ p$
⟨*proof*⟩

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

lemma $\mathcal{V}_B$*-disj* [$\mathcal{V}_B$*-simps*]:
assumes $A \in$ *pwffs* **and** $B \in$ *pwffs*
and *is-tv-assignment* φ
shows $\mathcal{V}_B\ \varphi\ (A \vee^{\mathfrak{Q}} B) = \mathcal{V}_B\ \varphi\ A \vee \mathcal{V}_B\ \varphi\ B$
⟨*proof*⟩

lemma $\mathcal{V}_B$*-conj* [$\mathcal{V}_B$*-simps*]:
assumes $A \in$ *pwffs* **and** $B \in$ *pwffs*
and *is-tv-assignment* φ
shows $\mathcal{V}_B\ \varphi\ (A \wedge^{\mathfrak{Q}} B) = \mathcal{V}_B\ \varphi\ A \wedge \mathcal{V}_B\ \varphi\ B$
⟨*proof*⟩

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

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

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

lemma *pwff-denotation-function-existence*:
shows *is-pwff-denotation-function* $\mathcal{V}_B$
⟨*proof*⟩

Tautologies:

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

lemma *tautology-is-wffo*:
assumes *is-tautology* A
shows $A \in$ *wffs*$_o$
⟨*proof*⟩

lemma *propositional-implication-reflexivity-is-tautology*:
shows *is-tautology* $(p_o \supset^{\mathfrak{Q}} p_o)$
⟨*proof*⟩

lemma *propositional-principle-of-simplification-is-tautology*:
shows *is-tautology* $(p_o \supset^{\mathfrak{Q}} (r_o \supset^{\mathfrak{Q}} p_o))$
⟨*proof*⟩

lemma *closed-pwff-denotation-uniqueness*:
assumes $A \in$ *pwffs* **and** *free-vars* $A = \{\}$
obtains b **where** $\forall \varphi.$ *is-tv-assignment* $\varphi \longrightarrow \mathcal{V}_B\ \varphi\ A = b$
⟨*proof*⟩

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

lemma *pwff-substitution-in-pwffs*:
assumes $A \in$ *pwffs* **and** $B \in$ *pwffs*
shows S $\{(p,\ o) \rightarrowtail A\}\ B \in$ *pwffs*
⟨*proof*⟩

lemma *pwff-substitution-denotation*:
assumes $A \in$ *pwffs* **and** $B \in$ *pwffs*
and *is-tv-assignment* φ
shows $\mathcal{V}_B\ \varphi$ (**S** $\{(p,\ o) \rightarrowtail A\}\ B$) $= \mathcal{V}_B\ (\varphi(p := \mathcal{V}_B\ \varphi\ A))\ B$
⟨*proof*⟩

lemma *pwff-substitution-tautology-preservation*:
assumes *is-tautology* B **and** $A \in$ *pwffs*
and $(p,\ o) \in$ *free-vars* B
shows *is-tautology* (**S** $\{(p,\ o) \rightarrowtail A\}\ B$)
⟨*proof*⟩

lemma *closed-pwff-substitution-free-vars*:
assumes $A \in$ *pwffs* **and** $B \in$ *pwffs*
and *free-vars* $A = \{\}$
and $(p,\ o) \in$ *free-vars* B
shows *free-vars* (**S** $\{(p,\ o) \rightarrowtail A\}\ B$) $=$ *free-vars* $B - \{(p,\ o)\}$ (**is** ‹*free-vars* (**S** *?ϑ* B) = -›)
⟨*proof*⟩

Substitution in a pwff:

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

Tautologous pwff:

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

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

lemma *implication-reflexivity-is-tautologous*:
assumes $A \in \mathit{wffs}_o$
shows *is-tautologous* $(A \supset^{\mathfrak{Q}} A)$
⟨*proof*⟩

lemma *principle-of-simplification-is-tautologous*:
assumes $A \in \mathit{wffs}_o$ **and** $B \in \mathit{wffs}_o$
shows *is-tautologous* $(A \supset^{\mathfrak{Q}} (B \supset^{\mathfrak{Q}} A))$
⟨*proof*⟩

lemma *pseudo-modus-tollens-is-tautologous*:
assumes $A \in \mathit{wffs}_o$ **and** $B \in \mathit{wffs}_o$
shows *is-tautologous* $((A \supset^{\mathfrak{Q}} \sim^{\mathfrak{Q}} B) \supset^{\mathfrak{Q}} (B \supset^{\mathfrak{Q}} \sim^{\mathfrak{Q}} A))$
⟨*proof*⟩

end

5 Proof System

theory *Proof-System*
imports
Syntax
begin

5.1 Axioms

inductive-set
axioms :: *form set*
where
axiom-1:
$\mathfrak{g}_{o\to o} \cdot T_o \wedge^{\mathfrak{Q}} \mathfrak{g}_{o\to o} \cdot F_o \equiv^{\mathfrak{Q}} \forall \mathfrak{x}_o.\ \mathfrak{g}_{o\to o} \cdot \mathfrak{x}_o \in \mathit{axioms}$
| *axiom-2*:
$(\mathfrak{x}_\alpha =_\alpha \mathfrak{y}_\alpha) \supset^{\mathfrak{Q}} (\mathfrak{h}_{\alpha\to o} \cdot \mathfrak{x}_\alpha \equiv^{\mathfrak{Q}} \mathfrak{h}_{\alpha\to o} \cdot \mathfrak{y}_\alpha) \in \mathit{axioms}$
| *axiom-3*:
$(\mathfrak{f}_{\alpha\to\beta} =_{\alpha\to\beta} \mathfrak{g}_{\alpha\to\beta}) \equiv^{\mathfrak{Q}} \forall \mathfrak{x}_\alpha.\ (\mathfrak{f}_{\alpha\to\beta} \cdot \mathfrak{x}_\alpha =_\beta \mathfrak{g}_{\alpha\to\beta} \cdot \mathfrak{x}_\alpha) \in \mathit{axioms}$
| *axiom-4-1-con*:
$(\lambda x_\alpha.\ \{\!|c|\!\}_\beta) \cdot A =_\beta \{\!|c|\!\}_\beta \in \mathit{axioms}$ **if** $A \in \mathit{wffs}_\alpha$

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

lemma *axioms-are-wffs-of-type-o*:
  shows $\mathit{axioms} \subseteq \mathit{wffs}_o$
  ⟨*proof*⟩

5.2 Inference rule R

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

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

5.3 Proof and derivability

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

lemma *derivable-form-is-wffso*:
  assumes *is-derivable A*
  shows $A \in \mathit{wffs}_o$
  ⟨*proof*⟩

definition *is-proof-step* :: *form list* ⇒ *nat* ⇒ *bool* **where**
  [*iff*]: *is-proof-step* $\mathcal{S}$ i' ⟷

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

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

lemma *common-prefix-is-subproof*:
assumes *is-proof* $(\mathcal{S} \mathbin{@} \mathcal{S}_1)$
and $i' < \mathit{length}\ \mathcal{S}$
shows *is-proof-step* $(\mathcal{S} \mathbin{@} \mathcal{S}_2)\ i'$
⟨*proof*⟩

lemma *added-suffix-proof-preservation*:
assumes *is-proof* $\mathcal{S}$
and $i' < \mathit{length}\ (\mathcal{S} \mathbin{@} \mathcal{S}') - \mathit{length}\ \mathcal{S}'$
shows *is-proof-step* $(\mathcal{S} \mathbin{@} \mathcal{S}')\ i'$
⟨*proof*⟩

lemma *append-proof-step-is-proof*:
assumes *is-proof* $\mathcal{S}$
and *is-proof-step* $(\mathcal{S} \mathbin{@} [A])\ (\mathit{length}\ (\mathcal{S} \mathbin{@} [A]) - 1)$
shows *is-proof* $(\mathcal{S} \mathbin{@} [A])$
⟨*proof*⟩

lemma *added-prefix-proof-preservation*:
assumes *is-proof* $\mathcal{S}'$
and $i' \in \{\mathit{length}\ \mathcal{S}..{<}\mathit{length}\ (\mathcal{S} \mathbin{@} \mathcal{S}')\}$
shows *is-proof-step* $(\mathcal{S} \mathbin{@} \mathcal{S}')\ i'$
⟨*proof*⟩

lemma *proof-but-last-is-proof*:
assumes *is-proof* $(\mathcal{S} \mathbin{@} [A])$
shows *is-proof* $\mathcal{S}$
⟨*proof*⟩

lemma *proof-prefix-is-proof*:
assumes *is-proof* $(\mathcal{S}_1 \mathbin{@} \mathcal{S}_2)$
shows *is-proof* $\mathcal{S}_1$
⟨*proof*⟩

lemma *single-axiom-is-proof*:
assumes $A \in \mathit{axioms}$
shows *is-proof* $[A]$
⟨*proof*⟩

lemma *proofs-concatenation-is-proof*:
assumes *is-proof* $\mathcal{S}_1$ **and** *is-proof* $\mathcal{S}_2$
shows *is-proof* $(\mathcal{S}_1 \mathbin{@} \mathcal{S}_2)$
⟨*proof*⟩

lemma *elem-of-proof-is-wffo*:
assumes *is-proof* $\mathcal{S}$ **and** $A \in$ *lset* $\mathcal{S}$
shows $A \in$ *wffs*$_o$
⟨*proof*⟩

lemma *axiom-prepended-to-proof-is-proof*:
assumes *is-proof* $\mathcal{S}$
and $A \in$ *axioms*
shows *is-proof* ([A] @ $\mathcal{S}$)
⟨*proof*⟩

lemma *axiom-appended-to-proof-is-proof*:
assumes *is-proof* $\mathcal{S}$
and $A \in$ *axioms*
shows *is-proof* ($\mathcal{S}$ @ [A])
⟨*proof*⟩

lemma *rule-R-app-appended-to-proof-is-proof*:
assumes *is-proof* $\mathcal{S}$
and $i_C <$ *length* $\mathcal{S}$ **and** $\mathcal{S}$! $i_C = C$
and $i_E <$ *length* $\mathcal{S}$ **and** $\mathcal{S}$! $i_E = E$
and *is-rule-R-app p D C E*
shows *is-proof* ($\mathcal{S}$ @ [D])
⟨*proof*⟩

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

lemma *proof-prefix-is-proof-of-last*:
assumes *is-proof* ($\mathcal{S}$ @ $\mathcal{S}'$) **and** $\mathcal{S} \neq []$
shows *is-proof-of* $\mathcal{S}$ (*last* $\mathcal{S}$)
⟨*proof*⟩

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

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

lemma *proof-form-is-theorem*:
assumes *is-proof* $\mathcal{S}$ **and** $\mathcal{S} \neq []$
and $i' <$ *length* $\mathcal{S}$
shows *is-theorem* ($\mathcal{S}$! i')
⟨*proof*⟩

theorem *derivable-form-is-theorem*:
assumes *is-derivable A*
shows *is-theorem A*
⟨*proof*⟩

theorem *theorem-is-derivable-form*:
assumes *is-theorem A*
shows *is-derivable A*
⟨*proof*⟩

theorem *theoremhood-derivability-equivalence*:
shows *is-theorem A* ⟷ *is-derivable A*
⟨*proof*⟩

lemma *theorem-is-wffo*:
assumes *is-theorem A*
shows $A \in wffs_o$
⟨*proof*⟩

lemma *equality-reflexivity*:
assumes $A \in wffs_\alpha$
shows *is-theorem* $(A =_\alpha A)$ (**is** *is-theorem ?A*$_2$)
⟨*proof*⟩

lemma *equality-reflexivity′*:
assumes $A \in wffs_\alpha$
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* ⇒ *form* ⇒ *'a* ⇒ *var set* **where**
[*simp*]: *capture-exposed-vars-at p A* $\mathcal{X}$ =
$\{(x, \beta) \mid x\ \beta\ p'\ E.$ *strict-prefix p′ p* ∧ $\lambda x_\beta.\ E \preceq_{p'} A \wedge (x, \beta) \in$ *free-vars* $\mathcal{X}\}$

lemma *capture-exposed-vars-at-alt-def*:
assumes *p* ∈ *positions A*
shows *capture-exposed-vars-at p A* $\mathcal{X}$ = *binders-at A p* ∩ *free-vars* $\mathcal{X}$
⟨*proof*⟩

Inference rule R′:

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

lemma *rule-R′-side-condition-alt-def*:
fixes $\mathcal{H}$ :: *form set*
assumes $C \in wffs_\alpha$

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

lemma *rule-R′-preserves-typing*:
assumes *is-rule-R′-app* $\mathcal{H}$ p D C E
shows $C \in$ *wffs*$_o \longleftrightarrow D \in$ *wffs*$_o$
⟨*proof*⟩

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

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

lemma *hyp-derivable-form-is-wffso*:
assumes *is-derivable-from-hyps* $\mathcal{H}$ A

shows $A \in \mathit{wffs}_o$
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

lemma *hyp-proof-prefix-is-hyp-proof*:

assumes *is-hyp-proof* $\mathcal{H}$ $\mathcal{S}_1$ ($\mathcal{S}_2$ @ $\mathcal{S}_2{}'$)
shows *is-hyp-proof* $\mathcal{H}$ $\mathcal{S}_1$ $\mathcal{S}_2$
⟨*proof*⟩

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

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

lemma *hyp-proofs-from-concatenation-is-hyp-proof*:
assumes *is-hyp-proof* $\mathcal{H}$ $\mathcal{S}_1$ $\mathcal{S}_1{}'$ **and** *is-hyp-proof* $\mathcal{H}$ $\mathcal{S}_2$ $\mathcal{S}_2{}'$
shows *is-hyp-proof* $\mathcal{H}$ ($\mathcal{S}_1$ @ $\mathcal{S}_2$) ($\mathcal{S}_1{}'$ @ $\mathcal{S}_2{}'$)
⟨*proof*⟩

lemma *elem-of-hyp-proof-is-wffo*:
assumes *is-hyps* $\mathcal{H}$
and *lset* $\mathcal{S}_1 \subseteq$ *wffs*$_o$
and *is-hyp-proof* $\mathcal{H}$ $\mathcal{S}_1$ $\mathcal{S}_2$
and $A \in$ *lset* $\mathcal{S}_2$
shows $A \in$ *wffs*$_o$
⟨*proof*⟩

lemma *hyp-prepended-to-hyp-proof-is-hyp-proof*:
assumes *is-hyp-proof* $\mathcal{H}$ $\mathcal{S}_1$ $\mathcal{S}_2$
and $A \in \mathcal{H}$
shows *is-hyp-proof* $\mathcal{H}$ $\mathcal{S}_1$ ([A] @ $\mathcal{S}_2$)
⟨*proof*⟩

lemma *hyp-appended-to-hyp-proof-is-hyp-proof*:
assumes *is-hyp-proof* $\mathcal{H}$ $\mathcal{S}_1$ $\mathcal{S}_2$
and $A \in \mathcal{H}$
shows *is-hyp-proof* $\mathcal{H}$ $\mathcal{S}_1$ ($\mathcal{S}_2$ @ [A])
⟨*proof*⟩

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

lemma *thm-prepended-to-hyp-proof-is-hyp-proof*:
assumes *is-hyp-proof* $\mathcal{H}$ $\mathcal{S}_1$ $\mathcal{S}_2$
and $A \in$ *lset* $\mathcal{S}_1$
shows *is-hyp-proof* $\mathcal{H}$ $\mathcal{S}_1$ ([A] @ $\mathcal{S}_2$)

⟨*proof*⟩

lemma *thm-appended-to-hyp-proof-is-hyp-proof*:
assumes *is-hyp-proof* $\mathcal{H}$ $\mathcal{S}_1$ $\mathcal{S}_2$
and $A \in$ *lset* $\mathcal{S}_1$
shows *is-hyp-proof* $\mathcal{H}$ $\mathcal{S}_1$ ($\mathcal{S}_2$ @ [A])
⟨*proof*⟩

lemma *rule-R′-app-appended-to-hyp-proof-is-hyp-proof*:
assumes *is-hyp-proof* $\mathcal{H}$ $\mathcal{S}'$ $\mathcal{S}$
and $i_C <$ *length* $\mathcal{S}$ **and** $\mathcal{S}$! $i_C = C$
and $i_E <$ *length* $\mathcal{S}$ **and** $\mathcal{S}$! $i_E = E$
and *is-rule-R′-app* $\mathcal{H}$ p D C E
shows *is-hyp-proof* $\mathcal{H}$ $\mathcal{S}'$ ($\mathcal{S}$ @ [D])
⟨*proof*⟩

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

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

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

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

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

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

abbreviation *is-derivable-from-no-hyps* (‹⊢ -› [*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 $A \in$ *axioms*
  shows $\vdash A$
  ⟨*proof*⟩

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

lemma *rule-R* [*consumes 2*, *case-names occ-subform replacement*]:
  assumes $\vdash C$ **and** $\vdash A =_\alpha B$
  and $A \preceq_p C$ **and** $C\langle\!|p \leftarrow B|\!\rangle \rhd D$
  shows $\vdash D$
⟨*proof*⟩

lemma *rule-R′* [*consumes 2*, *case-names occ-subform replacement no-capture*]:
  assumes $\mathcal{H} \vdash C$ **and** $\mathcal{H} \vdash A =_\alpha B$
  and $A \preceq_p C$ **and** $C\langle\!|p \leftarrow B|\!\rangle \rhd D$
  and *rule-R′-side-condition* $\mathcal{H}$ p D C $(A =_\alpha B)$
  shows $\mathcal{H} \vdash D$
⟨*proof*⟩

end

6 Elementary Logic

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

unbundle *no funcset-syntax*
notation *funcset* (**infixr** ‹↠› *60*)

6.1 Proposition 5200

proposition *prop-5200*:
  assumes $A \in \mathit{wffs}_\alpha$
  shows $\vdash A =_\alpha A$

⟨*proof*⟩

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

6.2 Proposition 5201 (Equality Rules)

proposition *prop-5201-1*:
assumes $\mathcal{H} \vdash A$ **and** $\mathcal{H} \vdash A \equiv^{\mathfrak{Q}} B$
shows $\mathcal{H} \vdash B$
⟨*proof*⟩

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 $\mathcal{H} \vdash A =_{\alpha\to\beta} B$ **and** $\mathcal{H} \vdash C =_\alpha D$
shows $\mathcal{H} \vdash A \centerdot C =_\beta B \centerdot D$
⟨*proof*⟩

proposition *prop-5201-5*:
assumes $\mathcal{H} \vdash A =_{\alpha\to\beta} B$ **and** $C \in \mathit{wffs}_\alpha$
shows $\mathcal{H} \vdash A \centerdot C =_\beta B \centerdot C$
⟨*proof*⟩

proposition *prop-5201-6*:
assumes $\mathcal{H} \vdash C =_\alpha D$ **and** $A \in \mathit{wffs}_{\alpha\to\beta}$
shows $\mathcal{H} \vdash A \centerdot C =_\beta A \centerdot D$
⟨*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 $p \in \mathit{positions}\ C$ **and** $A \preceq_p C$ **and** $C\langle\!|p \leftarrow B|\!\rangle \rhd D$
and $\mathcal{H} \vdash C$
shows $\mathcal{H} \vdash D$
⟨*proof*⟩

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

6.4 Proposition 5203

proposition *prop-5203*:
assumes $A \in \mathit{wffs}_\alpha$ **and** $B \in \mathit{wffs}_\beta$
and $\forall\, v \in \mathit{vars}\ A.\ \neg\ \mathit{is\text{-}bound}\ v\ B$
shows $\vdash\ (\lambda x_\alpha.\ B) \cdot A =_\beta \mathbf{S}\ \{(x,\ \alpha) \rightarrowtail A\}\ B$
⟨*proof*⟩

6.5 Proposition 5204

proposition *prop-5204*:
assumes $A \in \mathit{wffs}_\alpha$ **and** $B \in \mathit{wffs}_\beta$ **and** $C \in \mathit{wffs}_\beta$
and $\vdash\ B =_\beta C$
and $\forall\, v \in \mathit{vars}\ A.\ \neg\ \mathit{is\text{-}bound}\ v\ B \wedge \neg\ \mathit{is\text{-}bound}\ v\ C$
shows $\vdash\ \mathbf{S}\ \{(x,\ \alpha) \rightarrowtail A\}\ (B =_\beta C)$
⟨*proof*⟩

6.6 Proposition 5205 (η-conversion)

proposition *prop-5205*:
shows $\vdash\ \mathsf{f}_{\alpha\to\beta} =_{\alpha\to\beta} (\lambda y_\alpha.\ \mathsf{f}_{\alpha\to\beta} \cdot y_\alpha)$
⟨*proof*⟩

6.7 Proposition 5206 (α-conversion)

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

lemmas α = *prop-5206*

6.8 Proposition 5207 (β-conversion)

context
begin

private lemma *bound-var-renaming-equality*:
assumes $A \in \mathit{wffs}_\alpha$
and $z_\gamma \neq y_\gamma$
and $(z,\ \gamma) \notin \mathit{vars}\ A$
shows $\vdash\ A =_\alpha \mathit{rename\text{-}bound\text{-}var}\ (y,\ \gamma)\ z\ A$
⟨*proof*⟩

proposition *prop-5207*:

assumes $A \in \mathit{wffs}_\alpha$ **and** $B \in \mathit{wffs}_\beta$
and *is-free-for* A $(x,\ \alpha)$ B
shows $\vdash (\lambda x_\alpha.\ B) \centerdot A =_\beta \mathbf{S}\ \{(x,\ \alpha) \rightarrowtail A\}\ B$
⟨*proof*⟩

end

6.9 Proposition 5208

proposition *prop-5208*:
assumes $\mathit{vs} \neq []$ **and** $B \in \mathit{wffs}_\beta$
shows $\vdash \centerdot^{\mathcal{Q}}_\star\ (\lambda^{\mathcal{Q}}_\star\ \mathit{vs}\ B)\ (\mathit{map}\ \mathit{FVar}\ \mathit{vs}) =_\beta B$
⟨*proof*⟩

6.10 Proposition 5209

proposition *prop-5209*:
assumes $A \in \mathit{wffs}_\alpha$ **and** $B \in \mathit{wffs}_\beta$ **and** $C \in \mathit{wffs}_\beta$
and $\vdash B =_\beta C$
and *is-free-for* A $(x,\ \alpha)$ $(B =_\beta C)$
shows $\vdash \mathbf{S}\ \{(x,\ \alpha) \rightarrowtail A\}\ (B =_\beta C)$
⟨*proof*⟩

6.11 Proposition 5210

proposition *prop-5210*:
assumes $B \in \mathit{wffs}_\beta$
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 $\vdash (A =_\alpha B) \wedge^{\mathfrak{Q}} (C =_\beta D)$
⟨*proof*⟩

6.15 Proposition 5214

proposition *prop-5214*:
shows $\vdash T_o \wedge^{\mathfrak{Q}} F_o =_o F_o$
⟨*proof*⟩

6.16 Proposition 5215 (Universal Instantiation)

proposition *prop-5215*:
assumes $\mathcal{H} \vdash \forall x_\alpha.\ B$ **and** $A \in \mathit{wffs}_\alpha$
and *is-free-for* A $(x,\ \alpha)$ B
shows $\mathcal{H} \vdash \mathbf{S}\ \{(x,\ \alpha) \rightarrowtail A\}\ B$
⟨*proof*⟩

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

6.17 Proposition 5216

proposition *prop-5216*:
assumes $A \in \mathit{wffs}_o$
shows $\vdash (T_o \wedge^{\mathfrak{Q}} A) =_o A$
⟨*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 $A \in \mathit{wffs}_o$
shows $\vdash (T_o =_o A) =_o A$
⟨*proof*⟩

6.20 Proposition 5219 (Rule T)

proposition *prop-5219-1*:
assumes $A \in \mathit{wffs}_o$
shows $\mathcal{H} \vdash A \longleftrightarrow \mathcal{H} \vdash T_o =_o A$
⟨*proof*⟩

proposition *prop-5219-2*:
assumes $A \in \mathit{wffs}_o$
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 $\vdash (\lambda x_\alpha.\ T_o) =_{\alpha\to o} (\lambda z_\alpha.\ T_o)$
⟨*proof*⟩

proposition *prop-5220*:
assumes $\mathcal{H} \vdash A$
and $(x,\ \alpha) \notin$ *free-vars* $\mathcal{H}$
shows $\mathcal{H} \vdash \forall\, x_\alpha.\ A$
⟨*proof*⟩

end

lemmas *Gen* = *prop-5220*

proposition *generalized-Gen*:
assumes $\mathcal{H} \vdash A$
and *lset vs* $\cap$ *free-vars* $\mathcal{H} = \{\}$
shows $\mathcal{H} \vdash \forall^{\mathfrak{Q}}_{\star}$ *vs* A
⟨*proof*⟩

6.22 Proposition 5221 (Substitution)

context
begin

private lemma *prop-5221-aux*:
assumes $\mathcal{H} \vdash B$
and $(x,\ \alpha) \notin$ *free-vars* $\mathcal{H}$
and *is-free-for* A $(x,\ \alpha)$ B
and $A \in \mathit{wffs}_\alpha$
shows $\mathcal{H} \vdash \mathbf{S}\ \{(x,\ \alpha) \rightarrowtail A\}\ B$
⟨*proof*⟩

proposition *prop-5221*:
assumes $\mathcal{H} \vdash B$
and *is-substitution* ϑ
and $\forall\, v \in \mathit{fmdom}'\ \vartheta.$ *var-name* $v \notin$ *free-var-names* $\mathcal{H} \wedge$ *is-free-for* $(\vartheta$ \$\$! $v)$ v B
and $\vartheta \neq \{\$\$\}$
shows $\mathcal{H} \vdash \mathbf{S}\ \vartheta\ B$
⟨*proof*⟩

end

lemmas *Sub* = *prop-5221*

6.23 Proposition 5222 (Rule of Cases)

lemma *forall-α-conversion*:
assumes $A \in \mathit{wffs}_o$
and $(z,\ \beta) \notin \mathit{free\text{-}vars}\ A$
and *is-free-for* $(z_\beta)\ (x,\ \beta)\ A$
shows $\vdash \forall x_\beta.\ A =_o \forall z_\beta.\ \mathbf{S}\ \{(x,\ \beta) \rightarrowtail z_\beta\}\ A$
⟨*proof*⟩

proposition *prop-5222*:
assumes $\mathcal{H} \vdash \mathbf{S}\ \{(x,\ o) \rightarrowtail T_o\}\ A$ **and** $\mathcal{H} \vdash \mathbf{S}\ \{(x,\ o) \rightarrowtail F_o\}\ A$
and $A \in \mathit{wffs}_o$
shows $\mathcal{H} \vdash A$
⟨*proof*⟩

lemmas *Cases* = *prop-5222*

6.24 Proposition 5223

proposition *prop-5223*:
shows $\vdash (T_o \supset^{\mathcal{Q}} \mathfrak{y}_o) =_o \mathfrak{y}_o$
⟨*proof*⟩

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

6.25 Proposition 5224 (Modus Ponens)

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

lemmas *MP* = *prop-5224*

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

6.26 Proposition 5225

proposition *prop-5225*:
shows $\vdash \prod \alpha \cdot \mathfrak{f}_{\alpha\rightarrow o} \supset^{\mathcal{Q}} \mathfrak{f}_{\alpha\rightarrow o} \cdot \mathfrak{x}_\alpha$

⟨*proof*⟩

6.27 Proposition 5226

proposition *prop-5226*:
assumes $A \in \mathit{wffs}_\alpha$ **and** $B \in \mathit{wffs}_o$
and *is-free-for* A $(x,\ \alpha)$ B
shows $\vdash \forall x_\alpha.\ B \supset^{\mathcal{Q}} \mathbf{S}\ \{(x,\ \alpha) \rightarrowtail A\}\ B$
⟨*proof*⟩

6.28 Proposition 5227

corollary *prop-5227*:
shows $\vdash F_o \supset^{\mathcal{Q}} \mathfrak{x}_o$
⟨*proof*⟩

corollary *generalized-prop-5227*:
assumes $A \in \mathit{wffs}_o$
shows $\vdash F_o \supset^{\mathcal{Q}} A$
⟨*proof*⟩

6.29 Proposition 5228

proposition *prop-5228*:
shows $\vdash (T_o \supset^{\mathcal{Q}} T_o) =_o T_o$
and $\vdash (T_o \supset^{\mathcal{Q}} F_o) =_o F_o$
and $\vdash (F_o \supset^{\mathcal{Q}} T_o) =_o T_o$
and $\vdash (F_o \supset^{\mathcal{Q}} F_o) =_o T_o$
⟨*proof*⟩

6.30 Proposition 5229

lemma *false-in-conj-provability*:
assumes $A \in \mathit{wffs}_o$
shows $\vdash F_o \wedge^{\mathcal{Q}} A \equiv^{\mathcal{Q}} F_o$
⟨*proof*⟩

proposition *prop-5229*:
shows $\vdash (T_o \wedge^{\mathcal{Q}} T_o) =_o T_o$
and $\vdash (T_o \wedge^{\mathcal{Q}} F_o) =_o F_o$
and $\vdash (F_o \wedge^{\mathcal{Q}} T_o) =_o F_o$
and $\vdash (F_o \wedge^{\mathcal{Q}} F_o) =_o F_o$
⟨*proof*⟩

6.31 Proposition 5230

proposition *prop-5230*:
shows $\vdash (T_o \equiv^{\mathcal{Q}} T_o) =_o T_o$
and $\vdash (T_o \equiv^{\mathcal{Q}} F_o) =_o F_o$
and $\vdash (F_o \equiv^{\mathcal{Q}} T_o) =_o F_o$

and $\vdash (F_o \equiv^{\mathfrak{Q}} F_o) =_o T_o$
⟨*proof*⟩

6.32 Proposition 5231

proposition *prop-5231*:
shows $\vdash \sim^{\mathfrak{Q}} T_o =_o F_o$
and $\vdash \sim^{\mathfrak{Q}} F_o =_o T_o$
⟨*proof*⟩

6.33 Proposition 5232

lemma *disj-op-alt-def-provability*:
assumes $A \in \mathit{wffs}_o$ **and** $B \in \mathit{wffs}_o$
shows $\vdash A \vee^{\mathfrak{Q}} B =_o \sim^{\mathfrak{Q}} (\sim^{\mathfrak{Q}} A \wedge^{\mathfrak{Q}} \sim^{\mathfrak{Q}} B)$
⟨*proof*⟩

context begin

private lemma *prop-5232-aux*:
assumes $\vdash \sim^{\mathfrak{Q}} (A \wedge^{\mathfrak{Q}} B) =_o C$
and $\vdash \sim^{\mathfrak{Q}} A' =_o A$ **and** $\vdash \sim^{\mathfrak{Q}} B' =_o B$
shows $\vdash A' \vee^{\mathfrak{Q}} B' =_o C$
⟨*proof*⟩

proposition *prop-5232*:
shows $\vdash (T_o \vee^{\mathfrak{Q}} T_o) =_o T_o$
and $\vdash (T_o \vee^{\mathfrak{Q}} F_o) =_o T_o$
and $\vdash (F_o \vee^{\mathfrak{Q}} T_o) =_o T_o$
and $\vdash (F_o \vee^{\mathfrak{Q}} F_o) =_o F_o$
⟨*proof*⟩

end

6.34 Proposition 5233

context begin

private lemma *lem-prop-5233-no-free-vars*:
assumes $A \in \mathit{pwffs}$ **and** $\mathit{free\text{-}vars}\ A = \{\}$
shows $(\forall \varphi.\ \mathit{is\text{-}tv\text{-}assignment}\ \varphi \longrightarrow \mathcal{V}_B\ \varphi\ A = \mathbf{T}) \longrightarrow \vdash A =_o T_o$ (**is** $?A_T \longrightarrow$ -)
and $(\forall \varphi.\ \mathit{is\text{-}tv\text{-}assignment}\ \varphi \longrightarrow \mathcal{V}_B\ \varphi\ A = \mathbf{F}) \longrightarrow \vdash A =_o F_o$ (**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 $[A^1 \wedge \cdots \wedge A^n] \supset B$ is tautologous, then clearly $A^1 \supset (\ldots (A^n \supset B) \ldots)$ is also tautologous. Since this is not clear to us, we prove instead the version of Rule P found in [1]:

proposition *tautologous-horn-clause-is-hyp-derivable*:
assumes *is-hyps* $\mathcal{H}$ **and** *is-hyps* $\mathcal{G}$
and $\forall A \in \mathcal{G}.\ \mathcal{H} \vdash A$
and *lset* $hs = \mathcal{G}$
and *is-tautologous* $(hs \supset^{\mathcal{Q}}_{\star} B)$
shows $\mathcal{H} \vdash B$
⟨*proof*⟩

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

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

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

6.36 Proposition 5235

proposition *prop-5235*:
assumes $A \in$ *pwffs* **and** $B \in$ *pwffs*
and $(x,\ \alpha) \notin$ *free-vars* A
shows $\vdash \forall x_\alpha.\ (A \vee^{\mathcal{Q}} B) \supset^{\mathcal{Q}} (A \vee^{\mathcal{Q}} \forall x_\alpha.\ B)$
⟨*proof*⟩

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

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

context begin

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

proposition *prop-5237*:
assumes *is-hyps* $\mathcal{H}$
and $\mathcal{H} \vdash A \supset^{\mathcal{Q}} B$
and $(x,\ \alpha) \notin$ *free-vars* $(\{A\} \cup \mathcal{H})$
shows $\mathcal{H} \vdash A \supset^{\mathcal{Q}} (\forall x_\alpha.\ B)$

⟨*proof*⟩

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

corollary *generalized-prop-5237*:
assumes *is-hyps* $\mathcal{H}$
and $\mathcal{H} \vdash A \supset^{\mathcal{Q}} B$
and $\forall\, v \in S.\ v \notin$ *free-vars* $(\{A\} \cup \mathcal{H})$
and *lset vs* $= S$
shows $\mathcal{H} \vdash A \supset^{\mathcal{Q}} (\forall\,{}^{\mathcal{Q}}{}_{\star}\ vs\ B)$
⟨*proof*⟩

end

6.38 Proposition 5238

context begin

private lemma *prop-5238-aux*:
assumes $A \in wffs_\alpha$ **and** $B \in wffs_\alpha$
shows $\vdash ((\lambda x_\beta.\ A) =_{\beta\to\alpha} (\lambda x_\beta.\ B)) \equiv^{\mathcal{Q}} \forall\, x_\beta.\ (A =_\alpha B)$
⟨*proof*⟩

proposition *prop-5238*:
assumes $vs \neq []$ **and** $A \in wffs_\alpha$ **and** $B \in wffs_\alpha$
shows $\vdash \lambda^{\mathcal{Q}}{}_{\star}\ vs\ A =_{foldr\ (\to)\ (map\ var\text{-}type\ vs)\ \alpha}\ \lambda^{\mathcal{Q}}{}_{\star}\ vs\ B \equiv^{\mathcal{Q}} \forall\,{}^{\mathcal{Q}}{}_{\star}\ vs\ (A =_\alpha B)$
⟨*proof*⟩

end

6.39 Proposition 5239

lemma *replacement-derivability*:
assumes $C \in wffs_\beta$
and $A \preceq_p C$
and $\vdash A =_\alpha B$
and $C\langle\!|p \leftarrow B|\!\rangle \rhd D$
shows $\vdash C =_\beta D$
⟨*proof*⟩

context
begin

private lemma *prop-5239-aux-1*:
assumes $p \in$ *positions* $(\cdot^{\mathcal{Q}}{}_{\star}\ (FVar\ v)\ (map\ FVar\ vs))$
and $p \neq$ *replicate* $(length\ vs)$ «
shows
$(\exists\, A\ B.\ A \cdot B \preceq_p (\cdot^{\mathcal{Q}}{}_{\star}\ (FVar\ v)\ (map\ FVar\ vs)))$
$\vee$

$(\exists\, v \in \mathit{lset}\ \mathit{vs}.\ \mathit{occurs\text{-}at}\ v\ p\ (\cdot^{\mathcal{Q}}_{\star}\ (\mathit{FVar}\ v)\ (\mathit{map}\ \mathit{FVar}\ \mathit{vs})))$
⟨*proof*⟩ **lemma** *prop-5239-aux-2*:
assumes $t \notin \mathit{lset}\ \mathit{vs} \cup \mathit{vars}\ C$
and $C\langle\!|p \leftarrow (\cdot^{\mathcal{Q}}_{\star}\ (\mathit{FVar}\ t)\ (\mathit{map}\ \mathit{FVar}\ \mathit{vs}))|\!\rangle \rhd G$
and $C\langle\!|p \leftarrow (\cdot^{\mathcal{Q}}_{\star}\ (\lambda^{\mathcal{Q}}_{\star}\ \mathit{vs}\ A)\ (\mathit{map}\ \mathit{FVar}\ \mathit{vs}))|\!\rangle \rhd G'$
shows $\mathbf{S}\ \{t \rightarrowtail \lambda^{\mathcal{Q}}_{\star}\ \mathit{vs}\ A\}\ G = G'$ (**is** ‹$\mathbf{S}\ ?\vartheta\ G = G'$›)
⟨*proof*⟩ **lemma** *prop-5239-aux-3*:
assumes $t \notin \mathit{lset}\ \mathit{vs} \cup \mathit{vars}\ \{A, C\}$
and $C\langle\!|p \leftarrow (\cdot^{\mathcal{Q}}_{\star}\ (\mathit{FVar}\ t)\ (\mathit{map}\ \mathit{FVar}\ \mathit{vs}))|\!\rangle \rhd G$
and $\mathit{occurs\text{-}at}\ t\ p'\ G$
shows $p' = p\ @\ \mathit{replicate}\ (\mathit{length}\ \mathit{vs})\ «\ (\mathbf{is}\ ‹p' = ?p_t›)$
⟨*proof*⟩ **lemma** *prop-5239-aux-4*:
assumes $t \notin \mathit{lset}\ \mathit{vs} \cup \mathit{vars}\ \{A, C\}$
and $A \preceq_p C$
and $\mathit{lset}\ \mathit{vs} \supseteq \mathit{capture\text{-}exposed\text{-}vars\text{-}at}\ p\ C\ A$
and $C\langle\!|p \leftarrow (\cdot^{\mathcal{Q}}_{\star}\ (\mathit{FVar}\ t)\ (\mathit{map}\ \mathit{FVar}\ \mathit{vs}))|\!\rangle \rhd G$
shows $\mathit{is\text{-}free\text{-}for}\ (\lambda^{\mathcal{Q}}_{\star}\ \mathit{vs}\ A)\ t\ G$
⟨*proof*⟩

proposition *prop-5239*:
assumes $\mathit{is\text{-}rule\text{-}R\text{-}app}\ p\ D\ C\ (A =_{\alpha} B)$
and $\mathit{lset}\ \mathit{vs} =$
$\{(x, \beta) \mid x\ \beta\ p'\ E.\ \mathit{strict\text{-}prefix}\ p'\ p \wedge \lambda x_{\beta}.\ E \preceq_{p'} C \wedge (x, \beta) \in \mathit{free\text{-}vars}\ (A =_{\alpha} B)\}$
shows $\vdash \forall^{\mathcal{Q}}_{\star}\ \mathit{vs}\ (A =_{\alpha} B) \supset^{\mathcal{Q}} (C \equiv^{\mathcal{Q}} D)$
⟨*proof*⟩

end

6.40 Theorem 5240 (Deduction Theorem)

lemma *pseudo-rule-R-is-tautologous*:
assumes $C \in \mathit{wffs}_o$ **and** $D \in \mathit{wffs}_o$ **and** $E \in \mathit{wffs}_o$ **and** $H \in \mathit{wffs}_o$
shows $\mathit{is\text{-}tautologous}\ (((H \supset^{\mathcal{Q}} C) \supset^{\mathcal{Q}} ((H \supset^{\mathcal{Q}} E) \supset^{\mathcal{Q}} ((E \supset^{\mathcal{Q}} (C \equiv^{\mathcal{Q}} D)) \supset^{\mathcal{Q}} (H \supset^{\mathcal{Q}} D)))))$
⟨*proof*⟩

syntax
$\mathit{-HypDer} :: \mathit{form} \Rightarrow \mathit{form\ set} \Rightarrow \mathit{form} \Rightarrow \mathit{bool}\ (‹\text{-},\text{-} \vdash \text{-}›\ [50, 50, 50]\ 50)$
syntax-consts
$\mathit{-HypDer} \rightleftharpoons \mathit{is\text{-}derivable\text{-}from\text{-}hyps}$
translations
$\mathcal{H}, H \vdash P \rightharpoonup \mathcal{H} \cup \{H\} \vdash P$

theorem *thm-5240*:
assumes *finite* $\mathcal{H}$
and $\mathcal{H}, H \vdash P$
shows $\mathcal{H} \vdash H \supset^{\mathcal{Q}} P$
⟨*proof*⟩

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

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

corollary *generalized-deduction-theorem*:
assumes *finite* $\mathcal{H}$ **and** *finite* $\mathcal{H}'$
and $\mathcal{H} \cup \mathcal{H}' \vdash P$
and *lset hs* $= \mathcal{H}'$
shows $\mathcal{H} \vdash hs \supset^{\mathcal{Q}}_{\star} P$
⟨*proof*⟩

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 $A \in wffs_\alpha$ **and** $B \in wffs_o$
and $\mathcal{H} \vdash \mathbf{S}\ \{(x, \alpha) \rightarrowtail A\}\ B$
and *is-free-for* $A\ (x, \alpha)\ B$
shows $\mathcal{H} \vdash \exists\, x_\alpha.\ B$
⟨*proof*⟩

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

6.43 Proposition 5243 (Comprehension Theorem)

context
begin

private lemma *prop-5243-aux*:
assumes $\cdot^{\mathcal{Q}}_{\star}\ B\ (map\ FVar\ vs) \in wffs_\gamma$
and $B \in wffs_\beta$
and $k <$ *length vs*
shows $\beta \neq$ *var-type* $(vs\ !\ k)$
⟨*proof*⟩

proposition *prop-5243*:
assumes $B \in wffs_\beta$
and $\gamma =$ *foldr* $(\rightarrow)$ $(map\ var\text{-}type\ vs)\ \beta$
and $(u, \gamma) \notin$ *free-vars* B
shows $\vdash \exists\, u_\gamma.\ \forall^{\mathcal{Q}}_{\star}\ vs\ ((\cdot^{\mathcal{Q}}_{\star}\ u_\gamma\ (map\ FVar\ vs)) =_\beta B)$
⟨*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 $A \in \mathit{wffs}_o$ **and** $B \in \mathit{wffs}_o$
  and $(x,\ \alpha) \notin \mathit{free\text{-}vars}\ A$
  shows $\vdash \forall\, x_\alpha.\ (B \supset^{\mathrm{Q}} A) \supset^{\mathrm{Q}} (\exists\, x_\alpha.\ B \supset^{\mathrm{Q}} A)$
⟨*proof*⟩

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

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

6.45 Proposition 5245 (Rule C)

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

proposition *prop-5245*:
  assumes $\mathcal{H} \vdash \exists\, x_\alpha.\ B$
  and $\mathcal{H},$ **S** $\{(x,\ \alpha) \rightarrowtail y_\alpha\}\ B \vdash A$
  and *is-free-for* $(y_\alpha)\ (x,\ \alpha)\ B$
  and $(y,\ \alpha) \notin \mathit{free\text{-}vars}\ (\mathcal{H} \cup \{\exists\, x_\alpha.\ B,\ A\})$
  shows $\mathcal{H} \vdash A$
⟨*proof*⟩

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

end

7 Semantics

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

unbundle *no funcset-syntax*

notation *funcset* (**infixr** ‹↠› *60*)

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

notation *app* (**infixl** ‹·› *300*)

syntax
-vlambda :: *pttrn* ⇒ $V \Rightarrow (V \Rightarrow V) \Rightarrow V$ (‹(3**λ**-:- **./** -)› [*0*, *0*, *3*] *3*)
syntax-consts
-vlambda ⇌ *VLambda*
translations
$\boldsymbol{\lambda} x : A.\ f \rightleftharpoons CONST\ VLambda\ A\ (\lambda x.\ f)$

lemma *vlambda-extensionality*:
assumes $\bigwedge x.\ x \in elts\ A \Longrightarrow f\ x = g\ x$
shows $(\boldsymbol{\lambda} x : A.\ f\ x) = (\boldsymbol{\lambda} x : A.\ g\ x)$
⟨*proof*⟩

7.1 Frames

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

lemma *function-domainD*:
assumes $f \in elts\ (\mathcal{D}\ (\alpha \to \beta))$
shows $f \in elts\ (\mathcal{D}\ \alpha \longmapsto \mathcal{D}\ \beta)$
⟨*proof*⟩

lemma *vlambda-from-function-domain*:
assumes $f \in elts\ (\mathcal{D}\ (\alpha \to \beta))$
obtains b **where** $f = (\boldsymbol{\lambda} x : \mathcal{D}\ \alpha.\ b\ x)$ **and** $\forall x \in elts\ (\mathcal{D}\ \alpha).\ b\ x \in elts\ (\mathcal{D}\ \beta)$
⟨*proof*⟩

lemma *app-is-domain-respecting*:
assumes $f \in elts\ (\mathcal{D}\ (\alpha \to \beta))$ **and** $x \in elts\ (\mathcal{D}\ \alpha)$
shows $f \cdot x \in elts\ (\mathcal{D}\ \beta)$
⟨*proof*⟩

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

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

lemma *one-element-function-is-domain-respecting*:
shows $\{x\}_\alpha \in elts\ (\mathcal{D}\ \alpha \longmapsto \mathcal{D}\ o)$

⟨*proof*⟩

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

lemma *one-element-function-injectivity*:
assumes $\{x,\ x'\} \subseteq \mathit{elts}\ (\mathcal{D}\ i)$ **and** $\{x\}_i = \{x'\}_i$
shows $x = x'$
⟨*proof*⟩

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

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

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

lemma *identity-relation-is-domain-respecting*:
shows $q_\alpha \in \mathit{elts}\ (\mathcal{D}\ \alpha \longmapsto \mathcal{D}\ \alpha \longmapsto \mathcal{D}\ o)$
⟨*proof*⟩

lemma *q-is-equality*:
assumes $\{x,\ y\} \subseteq \mathit{elts}\ (\mathcal{D}\ \alpha)$
shows $(q_\alpha) \cdot x \cdot y = \mathbf{T} \longleftrightarrow x = y$
⟨*proof*⟩

Unique member selector:

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

Assignment:

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

end

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

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

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

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

7.2 Pre-models (interpretations)

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

locale *premodel* = *frame* +
  fixes $\mathcal{J}$ :: *con* ⇒ *V*
  assumes *Q-denotation*: ∀ α. $\mathcal{J}$ (*Q-constant-of-type* α) = q_α
  and *ι-denotation*: *is-unique-member-selector* ($\mathcal{J}$ *iota-constant*)
  and *non-logical-constant-denotation*: ∀ *c* α. ¬ *is-logical-constant* (*c*, α) ⟶ $\mathcal{J}$ (*c*, α) ∈ *elts* ($\mathcal{D}$ α)
begin

Wff denotation function:

definition *is-wff-denotation-function* :: ((*var* ⇒ *V*) ⇒ *form* ⇒ *V*) ⇒ *bool* **where**
  [*iff*]: *is-wff-denotation-function* $\mathcal{V}$ ⟷
    (
      ∀ φ. *is-assignment* φ ⟶
        (∀ *A* α. *A* ∈ *wffs*$_\alpha$ ⟶ $\mathcal{V}$ φ *A* ∈ *elts* ($\mathcal{D}$ α)) ∧ — closure condition, see note in page 186
        (∀ *x* α. $\mathcal{V}$ φ (x_α) = φ (*x*, α)) ∧
        (∀ *c* α. $\mathcal{V}$ φ (⦇*c*⦈$_\alpha$) = $\mathcal{J}$ (*c*, α)) ∧
        (∀ *A B* α β. *A* ∈ *wffs*$_{\beta\to\alpha}$ ∧ *B* ∈ *wffs*$_\beta$ ⟶ $\mathcal{V}$ φ (*A* ▪ *B*) = ($\mathcal{V}$ φ *A*) ▪ ($\mathcal{V}$ φ *B*)) ∧
        (∀ *x B* α β. *B* ∈ *wffs*$_\beta$ ⟶ $\mathcal{V}$ φ (λx_α. *B*) = (**λ***z* **:** $\mathcal{D}$ α**.** $\mathcal{V}$ (φ((*x*, α) := *z*)) *B*))
)

lemma *wff-denotation-function-is-domain-respecting*:
  assumes *is-wff-denotation-function* $\mathcal{V}$
  and *A* ∈ *wffs*$_\alpha$
  and *is-assignment* φ
  shows $\mathcal{V}$ φ *A* ∈ *elts* ($\mathcal{D}$ α)
  ⟨*proof*⟩

lemma *wff-var-denotation*:
  assumes *is-wff-denotation-function* $\mathcal{V}$
  and *is-assignment* φ
  shows $\mathcal{V}$ φ (x_α) = φ (*x*, α)
  ⟨*proof*⟩

lemma *wff-Q-denotation*:
  assumes *is-wff-denotation-function* $\mathcal{V}$
  and *is-assignment* φ
  shows $\mathcal{V}$ φ (Q_α) = q_α
  ⟨*proof*⟩

lemma *wff-iota-denotation*:
  assumes *is-wff-denotation-function* $\mathcal{V}$
  and *is-assignment* φ
  shows *is-unique-member-selector* ($\mathcal{V}$ φ ι)
  ⟨*proof*⟩

lemma *wff-non-logical-constant-denotation*:
  assumes *is-wff-denotation-function* $\mathcal{V}$

and *is-assignment* φ
and $\neg$ *is-logical-constant* $(c,\ \alpha)$
shows $\mathcal{V}\ \varphi\ (\{\!|c|\!\}_\alpha) = \mathcal{J}\ (c,\ \alpha)$
⟨*proof*⟩

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

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

end

7.3 General models

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

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

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

lemma *mixed-beta-conversion*:
assumes *is-assignment* φ
and $y \in elts\ (\mathcal{D}\ \alpha)$
and $B \in wffs_\beta$
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 $\mathcal{V}\ \varphi\ (\wedge_{o\to o\to o}) \in$ *elts* $(\mathcal{D}\ (o\to o\to o))$
⟨*proof*⟩

lemma *fully-applied-conj-fun-is-domain-respecting*:
assumes *is-assignment* φ
and $\{x, y\} \subseteq$ *elts* $(\mathcal{D}\ o)$
shows $\mathcal{V}\ \varphi\ (\wedge_{o\to o\to o}) \cdot x \cdot y \in$ *elts* $(\mathcal{D}\ o)$
⟨*proof*⟩

lemma *imp-fun-denotation-is-domain-respecting*:
assumes *is-assignment* φ
shows $\mathcal{V}\ \varphi\ (\supset_{o\to o\to o}) \in$ *elts* $(\mathcal{D}\ (o\to o\to o))$
⟨*proof*⟩

lemma *fully-applied-imp-fun-denotation-is-domain-respecting*:
assumes *is-assignment* φ
and $\{x, y\} \subseteq$ *elts* $(\mathcal{D}\ o)$
shows $\mathcal{V}\ \varphi\ (\supset_{o\to o\to o}) \cdot x \cdot y \in$ *elts* $(\mathcal{D}\ o)$
⟨*proof*⟩

end

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

7.4 Standard models

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

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

lemma *standard-model-is-general-model*:
assumes *is-standard-model* $\mathcal{M}$
shows *is-general-model* $\mathcal{M}$
⟨*proof*⟩

7.5 Validity

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

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

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

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

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

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

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

abbreviation *is-false-sentence-in* **where**
  is-false-sentence-in $A\ \mathcal{M} \equiv$ *is-sentence* $A \wedge \neg\ \mathcal{M} \models_{undefined} A$ — assignments are not meaningful

abbreviation *is-model-for* **where**
  is-model-for $\mathcal{M}\ \mathcal{G} \equiv \forall A \in \mathcal{G}.\ \mathcal{M} \models A$

lemma *general-validity-in-standard-validity*:
  assumes $\models A$
  shows $\models_S A$
  ⟨*proof*⟩

end

8 Soundness

theory *Soundness*
  imports
    Elementary-Logic
    Semantics
begin

unbundle *no funcset-syntax*
notation *funcset* (**infixr** ‹↠› *60*)

8.1 Proposition 5400

proposition (**in** *general-model*) *prop-5400*:
  assumes $A \in wffs_\alpha$
  and $\varphi \rightsquigarrow \mathcal{D}$ **and** $\psi \rightsquigarrow \mathcal{D}$
  and $\forall v \in$ *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 α
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 $A \in \mathit{wffs}_\alpha$
and $B \in \mathit{wffs}_\beta$
shows $\mathcal{V}\ \varphi\ ((\lambda x_\alpha.\ B) \cdot A) = \mathcal{V}\ (\varphi((x,\ \alpha) := \mathcal{V}\ \varphi\ A))\ B$
⟨*proof*⟩

lemma (**in** *general-model*) *prop-5401-b*:
assumes $\varphi \rightsquigarrow \mathcal{D}$
and $A \in \mathit{wffs}_\alpha$
and $B \in \mathit{wffs}_\alpha$
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 $A \in \mathit{wffs}_o$
and $B \in \mathit{wffs}_o$
shows $\mathcal{V}\ \varphi\ (A \equiv^{\mathcal{Q}} B) = \mathbf{T} \longleftrightarrow \mathcal{V}\ \varphi\ A = \mathcal{V}\ \varphi\ B$
⟨*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 $\{x,\ y\} \subseteq \mathit{elts}\ (\mathcal{D}\ o)$
shows $\mathcal{V}\ \varphi\ (\wedge_{o\to o\to o}) \cdot x \cdot y = (\mathit{if}\ x = \mathbf{T} \wedge y = \mathbf{T}\ \mathit{then}\ \mathbf{T}\ \mathit{else}\ \mathbf{F})$
⟨*proof*⟩

corollary (**in** *general-model*) *prop-5401-e′*:
assumes $\varphi \rightsquigarrow \mathcal{D}$
and $A \in \mathit{wffs}_o$ **and** $B \in \mathit{wffs}_o$
shows $\mathcal{V}\ \varphi\ (A \wedge^{\mathcal{Q}} B) = \mathcal{V}\ \varphi\ A \wedge \mathcal{V}\ \varphi\ B$

⟨*proof*⟩

lemma (**in** *general-model*) *prop-5401-f*:
assumes $\varphi \rightsquigarrow \mathcal{D}$
and $\{x,\ y\} \subseteq$ *elts* $(\mathcal{D}\ o)$
shows $\mathcal{V}\ \varphi\ (\supset_{o\to o\to o}) \cdot x \cdot y = (\textit{if}\ x = \mathbf{T} \wedge y = \mathbf{F}\ \textit{then}\ \mathbf{F}\ \textit{else}\ \mathbf{T})$
⟨*proof*⟩

corollary (**in** *general-model*) *prop-5401-f′*:
assumes $\varphi \rightsquigarrow \mathcal{D}$
and $A \in \textit{wffs}_o$ **and** $B \in \textit{wffs}_o$
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 $A \in \textit{wffs}_o$
shows $\mathcal{V}\ \varphi\ (\forall x_\alpha.\ A) = \mathbf{T} \longleftrightarrow (\forall z \in \textit{elts}\ (\mathcal{D}\ \alpha).\ \mathcal{V}\ (\varphi((x,\ \alpha) := z))\ A = \mathbf{T})$
⟨*proof*⟩

lemma *prop-5401-g*:
assumes *is-general-model* $\mathcal{M}$
and $\varphi \rightsquigarrow_M \mathcal{M}$
and $A \in \textit{wffs}_o$
shows $\mathcal{M} \models_\varphi \forall x_\alpha.\ A \longleftrightarrow (\forall \psi.\ \psi \rightsquigarrow_M \mathcal{M} \wedge \psi \sim_{(x,\ \alpha)} \varphi \longrightarrow \mathcal{M} \models_\psi A)$
⟨*proof*⟩

lemma (**in** *general-model*) *axiom-1-validity-aux*:
assumes $\varphi \rightsquigarrow \mathcal{D}$
shows $\mathcal{V}\ \varphi\ (\mathfrak{g}_{o\to o} \cdot T_o \wedge^{\mathcal{Q}} \mathfrak{g}_{o\to o} \cdot F_o \equiv^{\mathcal{Q}} \forall \mathfrak{x}_o.\ \mathfrak{g}_{o\to o} \cdot \mathfrak{x}_o) = \mathbf{T}$ (**is** $\mathcal{V}\ \varphi\ (?A \equiv^{\mathcal{Q}} ?B) = \mathbf{T}$)
⟨*proof*⟩

lemma *axiom-1-validity*:
shows $\models \mathfrak{g}_{o\to o} \cdot T_o \wedge^{\mathcal{Q}} \mathfrak{g}_{o\to o} \cdot F_o \equiv^{\mathcal{Q}} \forall \mathfrak{x}_o.\ \mathfrak{g}_{o\to o} \cdot \mathfrak{x}_o$ (**is** $\models ?A \equiv^{\mathcal{Q}} ?B$)
⟨*proof*⟩

lemma (**in** *general-model*) *axiom-2-validity-aux*:
assumes $\varphi \rightsquigarrow \mathcal{D}$
shows $\mathcal{V}\ \varphi\ ((\mathfrak{x}_\alpha =_\alpha \mathfrak{y}_\alpha) \supset^{\mathcal{Q}} (\mathfrak{h}_{\alpha\to o} \cdot \mathfrak{x}_\alpha \equiv^{\mathcal{Q}} \mathfrak{h}_{\alpha\to o} \cdot \mathfrak{y}_\alpha)) = \mathbf{T}$ (**is** $\mathcal{V}\ \varphi\ (?A \supset^{\mathcal{Q}} ?B) = \mathbf{T}$)
⟨*proof*⟩

lemma *axiom-2-validity*:
shows $\models (\mathfrak{x}_\alpha =_\alpha \mathfrak{y}_\alpha) \supset^{\mathcal{Q}} (\mathfrak{h}_{\alpha\to o} \cdot \mathfrak{x}_\alpha \equiv^{\mathcal{Q}} \mathfrak{h}_{\alpha\to o} \cdot \mathfrak{y}_\alpha)$ (**is** $\models ?A \supset^{\mathcal{Q}} ?B$)
⟨*proof*⟩

lemma (**in** *general-model*) *axiom-3-validity-aux*:
assumes $\varphi \rightsquigarrow \mathcal{D}$
shows $\mathcal{V}\ \varphi\ ((\mathfrak{f}_{\alpha\to\beta} =_{\alpha\to\beta} \mathfrak{g}_{\alpha\to\beta}) \equiv^{\mathcal{Q}} \forall \mathfrak{x}_\alpha.\ (\mathfrak{f}_{\alpha\to\beta} \cdot \mathfrak{x}_\alpha =_\beta \mathfrak{g}_{\alpha\to\beta} \cdot \mathfrak{x}_\alpha)) = \mathbf{T}$
(**is** $\mathcal{V}\ \varphi\ (?A \equiv^{\mathcal{Q}} ?B) = \mathbf{T}$)

⟨*proof*⟩

lemma *axiom-3-validity*:
shows $\models (\mathfrak{f}_{\alpha\to\beta} =_{\alpha\to\beta} \mathfrak{g}_{\alpha\to\beta}) \equiv^{\mathcal{Q}} \forall \mathfrak{x}_\alpha.\ (\mathfrak{f}_{\alpha\to\beta} \cdot \mathfrak{x}_\alpha =_\beta \mathfrak{g}_{\alpha\to\beta} \cdot \mathfrak{x}_\alpha)$ (**is** $\models$ *?A* $\equiv^{\mathcal{Q}}$ *?B*)
⟨*proof*⟩

lemma (**in** *general-model*) *axiom-4-1-con-validity-aux*:
assumes $\varphi \rightsquigarrow \mathcal{D}$
and $A \in \mathit{wffs}_\alpha$
shows $\mathcal{V}\ \varphi\ ((\lambda x_\alpha.\ \{\!|c|\!\}_\beta) \cdot A =_\beta \{\!|c|\!\}_\beta) = \mathbf{T}$
⟨*proof*⟩

lemma *axiom-4-1-con-validity*:
assumes $A \in \mathit{wffs}_\alpha$
shows $\models (\lambda x_\alpha.\ \{\!|c|\!\}_\beta) \cdot A =_\beta \{\!|c|\!\}_\beta$
⟨*proof*⟩

lemma (**in** *general-model*) *axiom-4-1-var-validity-aux*:
assumes $\varphi \rightsquigarrow \mathcal{D}$
and $A \in \mathit{wffs}_\alpha$
and $(y,\ \beta) \neq (x,\ \alpha)$
shows $\mathcal{V}\ \varphi\ ((\lambda x_\alpha.\ y_\beta) \cdot A =_\beta y_\beta) = \mathbf{T}$
⟨*proof*⟩

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

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

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

lemma (**in** *general-model*) *axiom-4-3-validity-aux*:
assumes $\varphi \rightsquigarrow \mathcal{D}$
and $A \in \mathit{wffs}_\alpha$ **and** $B \in \mathit{wffs}_{\gamma\to\beta}$ **and** $C \in \mathit{wffs}_\gamma$
shows $\mathcal{V}\ \varphi\ ((\lambda x_\alpha.\ B \cdot C) \cdot A =_\beta ((\lambda x_\alpha.\ B) \cdot A) \cdot ((\lambda x_\alpha.\ C) \cdot A)) = \mathbf{T}$
(**is** $\mathcal{V}\ \varphi\ (\mathit{?A} =_\beta \mathit{?B}) = \mathbf{T}$)
⟨*proof*⟩

lemma *axiom-4-3-validity*:

assumes $A \in \mathit{wffs}_\alpha$ **and** $B \in \mathit{wffs}_{\gamma\to\beta}$ **and** $C \in \mathit{wffs}_\gamma$
shows $\models (\lambda x_\alpha.\ B \cdot C) \cdot A =_\beta ((\lambda x_\alpha.\ B) \cdot A) \cdot ((\lambda x_\alpha.\ C) \cdot A)$ (**is** $\models \mathit{?A} =_\beta \mathit{?B}$)
⟨*proof*⟩

lemma (**in** *general-model*) *axiom-4-4-validity-aux*:
assumes $\varphi \rightsquigarrow \mathcal{D}$
and $A \in \mathit{wffs}_\alpha$ **and** $B \in \mathit{wffs}_\delta$
and $(y,\ \gamma) \notin \{(x,\ \alpha)\} \cup \mathit{vars}\ A$
shows $\mathcal{V}\ \varphi\ ((\lambda x_\alpha.\ \lambda y_\gamma.\ B) \cdot A =_{\gamma\to\delta} (\lambda y_\gamma.\ (\lambda x_\alpha.\ B) \cdot A)) = \mathbf{T}$
(**is** $\mathcal{V}\ \varphi\ (\mathit{?A} =_{\gamma\to\delta} \mathit{?B}) = \mathbf{T}$)
⟨*proof*⟩

lemma *axiom-4-4-validity*:
assumes $A \in \mathit{wffs}_\alpha$ **and** $B \in \mathit{wffs}_\delta$
and $(y,\ \gamma) \notin \{(x,\ \alpha)\} \cup \mathit{vars}\ A$
shows $\models (\lambda x_\alpha.\ \lambda y_\gamma.\ B) \cdot A =_{\gamma\to\delta} (\lambda y_\gamma.\ (\lambda x_\alpha.\ B) \cdot A)$ (**is** $\models \mathit{?A} =_{\gamma\to\delta} \mathit{?B}$)
⟨*proof*⟩

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

lemma *axiom-4-5-validity*:
assumes $A \in \mathit{wffs}_\alpha$ **and** $B \in \mathit{wffs}_\delta$
shows $\models (\lambda x_\alpha.\ \lambda x_\alpha.\ B) \cdot A =_{\alpha\to\delta} (\lambda x_\alpha.\ B)$
⟨*proof*⟩

lemma (**in** *general-model*) *axiom-5-validity-aux*:
assumes $\varphi \rightsquigarrow \mathcal{D}$
shows $\mathcal{V}\ \varphi\ (\iota \cdot (Q_i \cdot \mathfrak{y}_i) =_i \mathfrak{y}_i) = \mathbf{T}$
⟨*proof*⟩

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

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

lemma (**in** *general-model*) *rule-R-validity-aux*:
assumes $A \in \mathit{wffs}_\alpha$ **and** $B \in \mathit{wffs}_\alpha$
and $\forall \varphi.\ \varphi \rightsquigarrow \mathcal{D} \longrightarrow \mathcal{V}\ \varphi\ A = \mathcal{V}\ \varphi\ B$
and $C \in \mathit{wffs}_\beta$ **and** $C' \in \mathit{wffs}_\beta$
and $p \in \mathit{positions}\ C$ **and** $A \preceq_p C$ **and** $C\langle\!|p \leftarrow B|\!\rangle \rhd C'$
shows $\forall \varphi.\ \varphi \rightsquigarrow \mathcal{D} \longrightarrow \mathcal{V}\ \varphi\ C = \mathcal{V}\ \varphi\ C'$

⟨*proof*⟩

lemma *rule-R-validity*:
assumes $C \in \mathit{wffs}_o$ **and** $C' \in \mathit{wffs}_o$ **and** $E \in \mathit{wffs}_o$
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** $A \in \mathit{lset}\ \mathcal{S}$
shows $\models A$
⟨*proof*⟩

lemma *semantic-modus-ponens*:
assumes *is-general-model* $\mathcal{M}$
and $A \in \mathit{wffs}_o$ **and** $B \in \mathit{wffs}_o$
and $\mathcal{M} \models A \supset^{\mathcal{Q}} B$
and $\mathcal{M} \models A$
shows $\mathcal{M} \models B$
⟨*proof*⟩

lemma *generalized-semantic-modus-ponens*:
assumes *is-general-model* $\mathcal{M}$
and *lset hs* $\subseteq \mathit{wffs}_o$
and $\forall H \in \mathit{lset}\ \mathit{hs}.\ \mathcal{M} \models H$
and $P \in \mathit{wffs}_o$
and $\mathcal{M} \models \mathit{hs} \supset^{\mathcal{Q}}_{\star} P$
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* ⇒ *bool* **where**
[*iff*]: *is-inconsistent-set* $\mathcal{G}$ ⟷ $\mathcal{G} \vdash F_o$

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

definition *is-wffo-consistent-with* :: *form* ⇒ *form set* ⇒ *bool* **where**
[*iff*]: *is-wffo-consistent-with B* $\mathcal{G}$ ⟷ ¬ *is-inconsistent-set* ($\mathcal{G} \cup \{B\}$)

9.1 Existence of a standard model

We construct a standard model in which $\mathcal{D}\ i$ is the set $\{0\}$:

primrec *singleton-standard-domain-family* (‹$\mathcal{D}^S$›) **where**
$\mathcal{D}^S\ i = 1$ — i.e., $\mathcal{D}^S\ i$ = *ZFC-in-HOL.set* $\{0\}$
| $\mathcal{D}^S\ o = \mathbb{B}$
| $\mathcal{D}^S\ (\alpha \to \beta) = \mathcal{D}^S\ \alpha \longmapsto \mathcal{D}^S\ \beta$

interpretation *singleton-standard-frame*: *frame* $\mathcal{D}^S$
⟨*proof*⟩

definition *singleton-standard-constant-denotation-function* (‹$\mathcal{J}^S$›) **where**
[*simp*]: $\mathcal{J}^S\ k =$
(
if
$\exists \beta$. *is-Q-constant-of-type k* β
then
let β = *type-of-Q-constant k in* ${q_\beta}^{\mathcal{D}^S}$
else
if
is-iota-constant k
then
$\boldsymbol{\lambda} z : \mathcal{D}^S\ (i \to o).\ 0$
else
case k of (c, α) ⇒ *SOME z*. $z \in$ *elts* $(\mathcal{D}^S\ \alpha)$
)

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

fun *singleton-standard-wff-denotation-function* (‹$\mathcal{V}^S$›) **where**
  $\mathcal{V}^S$ φ (x_α) = φ (x, α)
| $\mathcal{V}^S$ φ ({|c|}$_\alpha$) = $\mathcal{J}^S$ (c, α)
| $\mathcal{V}^S$ φ (A ⋅ B) = ($\mathcal{V}^S$ φ A) ⋅ ($\mathcal{V}^S$ φ B)
| $\mathcal{V}^S$ φ (λx_α. A) = (**λ**z **:** $\mathcal{D}^S$ α**.** $\mathcal{V}^S$ (φ((x, α) := z)) A)

lemma *singleton-standard-wff-denotation-function-closure*:
  assumes *frame.is-assignment* $\mathcal{D}^S$ φ
  and $A \in$ *wffs*$_\alpha$
  shows $\mathcal{V}^S$ φ $A \in$ *elts* ($\mathcal{D}^S$ α)
⟨*proof*⟩

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

proposition *standard-model-existence*:
  shows ∃ $\mathcal{M}$. *is-standard-model* $\mathcal{M}$
  ⟨*proof*⟩

9.2 Theorem 5403 (Consistency Theorem)

proposition *model-existence-implies-set-consistency*:
  assumes *is-hyps* $\mathcal{G}$
  and ∃ $\mathcal{M}$. *is-general-model* $\mathcal{M}$ ∧ *is-model-for* $\mathcal{M}$ $\mathcal{G}$
  shows ¬ *is-inconsistent-set* $\mathcal{G}$
⟨*proof*⟩

proposition $\mathcal{Q}_0$*-is-consistent*:
  shows ¬ $\mathcal{Q}_0$*-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* $\mathcal{G}$ ⟷ (∀ $A \in$ (*wffs*$_o$). $\mathcal{G}$ ⊢ 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.