Soundness of the Q0 proof system for higher-order logic

Anders Schlichtkrull

Abstract

This entry formalizes the Q0 proof system for higher-order logic (also known as simple type theory) from the book "An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof" by Peter B. Andrews [And13] together with the system's soundness. Most of the used theorems and lemmas are also from his book. The soundness proof is with respect to general models and as a consequence also holds for standard models. Higher-order logic is given a semantics by porting to Isabelle/HOL the specification of set theory from the CakeML project [KAMO14, KAMO16]. The independently developed AFP entry "Metatheory of Q0" by Javier Díaz [Día23] also formalizes Q0 in Isabelle/HOL. I highly recommend the reader to also take a look at his formalization!

Contents

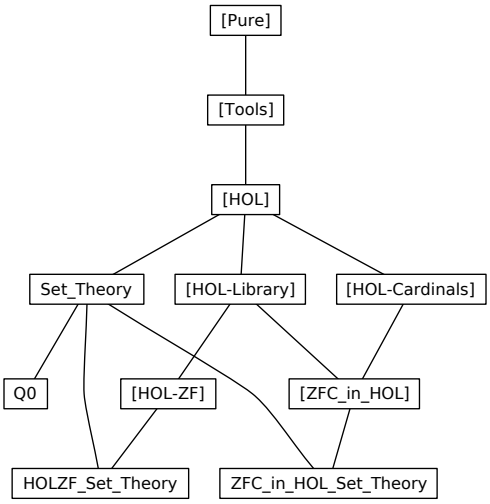

Figure 1: Theory dependency graph

1 Introduction

This entry formalizes the Q0 proof system for higher-order logic (also called simple type theory) and its soundness. Both the system and most of the proven lemmas and theorems are from Peter B. Andrews' book [And13]. In the book's chapter on type theory, Andrews explains that type theory was invented by Russell [Rus08] and that Whitehead and Russell [WR13] showed that fundamental parts of mathematics could be formalized in type theory. As influences on the type theory presented in Andrews' chapter he mentions Church [Chu40], Henkin [Hen50, Hen63] and earlier works by himself [And63, And72]. The present Isabelle formalization of higher-order logic is given a semantics by using a port to Isabelle/HOL of CakeML's specification of set theory [KAMO16]. The specification is formalized as a locale that fixes a type of set theoretic sets and a number of functions (powerset, union, separation) and the set membership predicate. The soundness proof is with respect to general models and as a consequence it also holds for standard models. Variables are implemented simply using named variables rather than e.g. De Bruijn indices or nominal techniques. It is well-known that named variables require the definition of substitution to rename variables, but since substitution is derived in Q0 rather than built into the proof system this complication is essentially circumvented in the present formalization. As a curiosity the present AFP entry also proves that the set theory specification is fulfilled by the sets axiomatized in Isabelle/HOLZF [Obu06] and also by the sets axiomatized by the AFP entry on Zermelo Fraenkel Set Theory in Higher-Order Logic [Pau19]. The theory files of the present entry appeared in the IsaFoL effort [Sch23].

In the literature we find other formalizations of the metatheory of higher-order logics and type theories. Independently from my work here, Javier Díaz also formalizes Q0 in Isabelle/HOL. His work is available in the AFP entry "Metatheory of Q0" [Día23]. I highly recommend the reader to take a look also at his very thorough formalization! Arthan specifies HOL [Art14a, Art14b, Art02] in ProofPower but does not prove soundness. John Harrison formalizes the soundness of the proof system of HOL Light [Har06] in HOL Light and Kumar et al. [KAMO14, KAMO16, AMKS22, AM23] formalize it in HOL4 as part of the CakeML project including also definitions and a verified implementation. Roßkopf and Nipkow [NR21a, NR21b, RN23] formalize a proof system for terms in Isabelle's metalogic together with an implementation of a proof checker and prove that the proof checker indeed implements this proof system. There are also a number of works using Coq to formalize metatheory of the calculus of constructions and the calculus of inductive constructions [Bar96b, Bar96a, BW96, Bar97, SBF+20]. Worth mentioning is also the formalization in Coq of second-order logic which also formalizes general models [KK22].

There are plenty of opportunities to go further with the present formalization. The present formalization proves soundness of Q0 theorems (i.e. $\vdash A$ implies $\models A$), but not soundness of Q0 derivability (i.e. $M \models G$ and $G \vdash A$ implies $M \models A$). Other interesting lemmas and theorems from Andrews' book could also be proved such as e.g. derived inference rules and completeness.

2 Set Theory

theory Set_Theory **imports** Main **begin**

2.1 CakeML License

2.2 Set theory specification

This formal document is the set theory from https://github.com/CakeML/cakeml/blob/master/candle/set-theory/set-SpecScript.sml ported to Isabelle/HOL and extended.

```
locale set_theory =
  fixes mem :: "'s ⇒ 's ⇒ bool" (infix ‹∈:› 67)
  fixes sub :: "'s ⇒ ('s ⇒ bool) ⇒ 's" (infix ‹suchthat› 67)
  fixes pow :: "'s ⇒ 's"
  fixes uni :: "'s ⇒ 's" (‹⋃:_› [900] 900)
  fixes upair :: "'s ⇒ 's ⇒ 's" (infix ‹+:› 67)
  assumes extensional: "⋀x y. x = y ⟷ (∀a. a ∈: x ⟷ a ∈: y)"
  assumes mem_sub[simp]: "⋀x P a. a ∈: (x suchthat P) ⟷ a ∈: x ∧ P a"
  assumes mem_pow[simp]: "⋀x a. a ∈: (pow x) ⟷ (∀b. b ∈: a ⟶ b ∈: x)"
  assumes mem_uni[simp]: "⋀x a. a ∈: ⋃:x ⟷ (∃b. a ∈: b ∧ b ∈: x)"
  assumes mem_upair[simp]: "⋀x y a. a ∈: (x +: y) ⟷ a = x ∨ a = y"
begin


lemma seperation_unique:
  assumes "∀x P a. a ∈: (sub2 x P) ⟷ a ∈: x ∧ P a"
  shows "sub2 = sub"
⟨proof⟩


lemma pow_unique:
  assumes "∀x a. a ∈: (pow2 x) ⟷ (∀b. b ∈: a ⟶ b ∈: x)"
  shows "pow2 = pow"
  ⟨proof⟩


lemma uni_unique:
```

assumes "∀x a. a ∈: uni2 x ⟷ (∃b. a ∈: b ∧ b ∈: x)"
shows "uni2 = uni"
⟨*proof*⟩

lemma upair_unique:
assumes "∀x y a. a ∈: upair2 x y ⟷ a = x ∨ a = y"
shows "upair2 = upair"
⟨*proof*⟩

definition empt :: 's (‹∅›) **where**
"∅ = undefined suchthat (λx. False)"

lemma mem_empty[simp]: "¬x ∈: ∅"
⟨*proof*⟩

definition unit :: "'s ⇒ 's" **where**
"unit x = x +: x"

lemma mem_unit[simp]: "x ∈: (unit y) ⟷ x = y"
⟨*proof*⟩

lemma unit_inj: "unit x = unit y ⟷ x = y"
⟨*proof*⟩

definition one :: 's **where**
"one = unit ∅"

lemma mem_one[simp]: "x ∈: one ⟷ x = ∅"
⟨*proof*⟩

definition two :: 's **where**
"two = ∅ +: one"

lemma mem_two[simp]: "∀x. x ∈: two ⟷ x = ∅ ∨ x = one"
⟨*proof*⟩

definition pair :: "'s ⇒ 's ⇒ 's" (**infix** ‹,:› 50) **where**
"(x,:y) = (unit x) +: (x +: y)"

lemma upair_inj:
"a +: b = c +: d ⟷ a = c ∧ b = d ∨ a = d ∧ b = c"
⟨*proof*⟩

lemma unit_eq_upair:
"unit x = y +: z ⟷ x = y ∧ y = z"
⟨*proof*⟩

lemma pair_inj:
"(a,:b) = (c,:d) ⟷ a = c ∧ b = d"
⟨*proof*⟩

definition binary_uni (**infix** ‹∪:› 67) **where**
"x ∪: y = ⋃: (x +: y)"

lemma mem_binary_uni[simp]:
"a ∈: (x ∪: y) ⟷ a ∈: x ∨ a ∈: y"
⟨*proof*⟩

definition product :: "'s ⇒ 's ⇒ 's" (**infix** ‹×:› 67) **where**
"x ×: y = (pow (pow (x ∪: y)) suchthat (λa. ∃b c. b ∈: x ∧ c ∈: y ∧ a = (b,:c)))"

lemma mem_product[simp]:
"a ∈: (x ×: y) ⟷ (∃b c. a = (b,:c) ∧ b ∈: x ∧ c ∈: y)"
⟨*proof*⟩

definition relspace **where**
"relspace x y = pow (x ×: y)"

definition funspace **where**
"funspace x y =
(relspace x y suchthat
(λf. ∀a. a ∈: x ⟶ (∃!b. (a,:b) ∈: f)))"

definition "apply" :: "'s ⇒ 's ⇒ 's" (**infixl** ‹·› 68) **where**
"(x·y) = (SOME a. (y,:a) ∈: x)"

definition boolset **where**
"boolset ≡ two"

definition true **where**
"true = ∅"

definition false **where**
"false = one"

lemma true_neq_false:
"true ≠ false"
⟨*proof*⟩

lemma mem_boolset[simp]:
"x ∈: boolset ⟷ ((x = true) ∨ (x = false))"
⟨*proof*⟩

definition boolean :: "bool ⇒ 's" **where**
"boolean b = (if b then true else false)"

lemma boolean_in_boolset:
"boolean b ∈: boolset"
⟨*proof*⟩

lemma boolean_eq_true:
"boolean b = true ⟷ b"
⟨*proof*⟩

definition "holds s x ⟷ s · x = true"

definition abstract **where**
"abstract doma rang f = ((doma ×: rang) suchthat (λx. ∃a. x = (a,:f a)))"

lemma apply_abstract[simp]:
"x ∈: s ⟹ f x ∈: t ⟹ abstract s t f · x = f x"
⟨*proof*⟩

lemma apply_abstract_matchable:
"x ∈: s ⟹ f x ∈: t ⟹ f x = u ⟹ abstract s t f · x = u"
⟨*proof*⟩

lemma apply_in_rng:
assumes "x ∈: s"
assumes "f ∈: funspace s t"
shows "f · x ∈: t"
⟨*proof*⟩

lemma abstract_in_funspace[simp]:
"(∀x. x ∈: s ⟶ f x ∈: t) ⟹ abstract s t f ∈: funspace s t"
⟨*proof*⟩

lemma abstract_in_funspace_matchable:
"(∀x. x ∈: s ⟶ f x ∈: t) ⟹ fs = funspace s t ⟹ abstract s t f ∈: fs"
⟨*proof*⟩

lemma abstract_eq:
assumes "∀x. x ∈: s ⟶ f x ∈: t1 ∧ g x ∈: t2 ∧ f x = g x"
shows "abstract s t1 f = abstract s t2 g"
⟨*proof*⟩

lemma abstract_extensional:
assumes "∀x. x ∈: s ⟶ f x ∈: t ∧ f x = g x"
shows "abstract s t f = abstract s t g"
⟨*proof*⟩

lemma abstract_extensional':
assumes "⋀x. x ∈: s ⟹ f x ∈: t"
assumes "⋀x. x ∈: s ⟹ f x = g x"
shows "abstract s t f = abstract s t g"
⟨*proof*⟩

lemma abstract_cong:
assumes "∀x. x ∈: s ⟶ f x ∈: t ∧ g x ∈: t"
assumes "abstract s t f = abstract s t g"
assumes "x ∈: s"
shows "f x = g x"
⟨*proof*⟩

lemma abstract_cong_specific:

assumes "x ∈: s"
assumes "f x ∈: t"
assumes "abstract s t f = abstract s t g"
assumes "g x ∈: t"
shows "f x = g x"
⟨*proof*⟩

lemma abstract_iff_extensional:
assumes "∀x. x ∈: s ⟶ f x ∈: t ∧ g x ∈: t"
shows "(abstract s t f = abstract s t g) ⟷ (∀x. x ∈: s ⟶ f x ∈: t ∧ f x = g x)"
⟨*proof*⟩

lemma in_funspace_abstract[simp]:
assumes "z ∈: funspace s t"
shows "∃f. z = abstract s t f ∧ (∀x. x ∈: s ⟶ f x ∈: t)"
⟨*proof*⟩

theorem axiom_of_choice:
assumes "∀a. a ∈: x ⟶ (∃b. b ∈: a)"
shows "∃f. ∀a. a ∈: x ⟶ f · a ∈: a"
⟨*proof*⟩

definition is_infinite **where**
"is_infinite s = infinite {a. a ∈: s}"

lemma funspace_inhabited:
"(∃x. x ∈: s) ⟹ (∃x. x ∈: t) ⟹ (∃f. f ∈: funspace s t)"
⟨*proof*⟩

fun tuple **where**
"tuple [] = ∅" |
"tuple (a#as) = (a,: tuple as)"

lemma pair_not_empty:
"(x,:y) ≠ ∅"
⟨*proof*⟩

lemma tuple_empty:
"tuple ls = ∅ ⟷ ls = []"
⟨*proof*⟩

lemma tuple_inj:
"tuple l1 = tuple l2 ⟷ l1 = l2"
⟨*proof*⟩

fun bigcross **where**
"bigcross [] = one" |
"bigcross (a#as) = a ×: (bigcross as)"

lemma mem_bigcross[simp]:
"x ∈: (bigcross ls) ⟷ (∃xs. x = tuple xs ∧ list_all2 mem xs ls)"
⟨*proof*⟩

definition subs :: "'s ⇒ 's ⇒ bool" (**infix** ‹⊆:› 67) **where**

```
  "x ⊆: y ⟷ x ∈: pow y"
```

definition one_elem_fun :: "'s ⇒ 's ⇒ 's" **where**
```
  "one_elem_fun x d = abstract d boolset (λy. boolean (x=y))"
```

definition iden :: "'s ⇒ 's" **where**
```
  "iden D = abstract D (funspace D boolset) (λx. one_elem_fun x D)"
```

lemma apply_id[simp]:
```
  assumes A_in_D: "A ∈: D"
  assumes B_in_D: "B ∈: D"
  shows "iden D · A · B = boolean (A = B)"
```
⟨*proof*⟩

lemma apply_id_true[simp]:
```
  assumes A_in_D: "A ∈: D"
  assumes B_in_D: "B ∈: D"
  shows "iden D · A · B = true ⟷ A = B"
```
⟨*proof*⟩

lemma apply_if_pair_in:
```
  assumes "(a1,: a2) ∈: f"
  assumes "f ∈: funspace s t"
  shows "f · a1 = a2"
```
⟨*proof*⟩

lemma funspace_app_unique:
```
  assumes "f ∈: funspace s t"
  assumes "(a1,: a2) ∈: f"
  assumes "(a1,: a3) ∈: f"
  shows "a3 = a2"
```
⟨*proof*⟩

lemma funspace_extensional:
```
  assumes "f ∈: funspace s t"
  assumes "g ∈: funspace s t"
  assumes "∀x. x ∈: s ⟶ f · x = g · x"
  shows "f = g"
```
⟨*proof*⟩

lemma funspace_difference_witness:
```
  assumes "f ∈: funspace s t"
  assumes "g ∈: funspace s t"
  assumes "f ≠ g"
  shows "∃z. z ∈: s ∧ f · z ≠ g · z"
```
⟨*proof*⟩

end

end

3 Isabelle/HOLZF lives up to CakeML's set theory specification

theory HOLZF_Set_Theory **imports** "HOL-ZF.MainZF" Set_Theory **begin**

interpretation set_theory Elem Sep Power Sum Upair
⟨*proof*⟩

end

4 ZFC_in_HOL lives up to CakeML's set theory specification

theory ZFC_in_HOL_Set_Theory **imports** ZFC_in_HOL.ZFC_in_HOL Set_Theory **begin**

interpretation set_theory "λx y. x ∈ elts y" "λx::V. λP. (inv elts) ({y. y ∈ elts x ∧ P y})" VPow
"(λY. SOME Z. elts Z = ⋃(elts ` (elts Y)))" "λx y::V. (inv elts) {x, y}"
⟨*proof*⟩

end

5 Q0 abbreviations

theory Q0
imports Set_Theory
abbrevs "App" = "·"
and "Abs" = "[λ_:_. _]"
and "Eql" = "[_ =_= _]"
and "Con" = "∧"
and "Forall" = "[∀_:_. _]"
and "Imp" = "⟶"
and "Fun" = "⇐"
begin

lemma arg_cong3: "a = b ⟹ c = d ⟹ e = f ⟹ h a c e = h b d f"
⟨*proof*⟩

6 Syntax and typing

datatype type_sym =
Ind |
Tv |
Fun type_sym type_sym (**infixl** ‹⇐› 80)

type_synonym var_sym = string
type_synonym cst_sym = string

datatype trm =
Var var_sym type_sym |
Cst cst_sym type_sym |
App trm trm (**infixl** ‹·› 80) |
Abs var_sym type_sym trm (‹[λ_:_. _]› [80,80,80])

fun vars :: "trm ⇒ (var_sym * type_sym) set" **where**
"vars (Var x α) = {(x,α)}"
| "vars (Cst _ _) = {}"
| "vars (A · B) = vars A ∪ vars B"
| "vars ([λx:α. A]) = {(x,α)} ∪ vars A"

fun frees :: "trm ⇒ (var_sym * type_sym) set" **where**
"frees (Var x α) = {(x,α)}"
| "frees (Cst _ _) = {}"
| "frees (A · B) = frees A ∪ frees B"
| "frees ([λx:α. A]) = frees A - {(x,α)}"

lemma frees_subset_vars:
"frees A ⊆ vars A"
⟨*proof*⟩

inductive wff :: "type_sym ⇒ trm ⇒ bool" **where**
wff_Var: "wff α (Var _ α)"
| wff_Cst: "wff α (Cst _ α)"
| wff_App: "wff (α ⇐ β) A ⟹ wff β B ⟹ wff α (A · B)"
| wff_Abs: "wff α A ⟹ wff (α ⇐ β) [λx:β. A]"

```
fun type_of :: "trm ⇒ type_sym" where
  "type_of (Var x α) = α"
| "type_of (Cst c α) = α"
| "type_of (A · B) =
     (case type_of A of β ⇐ α ⇒ β)"
| "type_of [λx:α. A] = (type_of A) ⇐ α"

lemma type_of[simp]:
  "wff α A ⟹ type_of A = α"
  ⟨proof⟩

lemma wff_Var'[simp, code]:
  "wff β (Var x α) ⟷ β = α"
  ⟨proof⟩

lemma wff_Cst'[simp, code]:
  "wff β (Cst c α) ⟷ β = α"
  ⟨proof⟩

lemma wff_App'[simp]:
  "wff α (A · B) ⟷ (∃β. wff (α ⇐ β) A ∧ wff β B)"
⟨proof⟩

lemma wff_Abs'[simp]:
  "wff γ ([λx:α. A]) ⟷ (∃β. wff β A ∧ γ = β ⇐ α)"
⟨proof⟩

lemma wff_Abs_type_of[code]:
  "wff γ [λx:α. A] ⟷ (wff (type_of A) A ∧ γ = (type_of A) ⇐ α)"
⟨proof⟩

lemma wff_App_type_of[code]:
  "wff γ ((A · B)) ⟷ (wff (type_of A) A ∧ wff (type_of B) B ∧ type_of A = γ ⇐ (type_of B))"
⟨proof⟩

lemma unique_type:
  "wff β A ⟹ wff α A ⟹ α = β"
⟨proof⟩
```

7 Replacement

```
inductive replacement :: "trm ⇒ trm ⇒ trm ⇒ trm ⇒ bool" where
  replace: "replacement A B A B"
| replace_App_left: "replacement A B C E ⟹ replacement A B (C · D) (E · D)"
| replace_App_right: "replacement A B D E ⟹ replacement A B (C · D) (C · E)"
| replace_Abs: "replacement A B C D ⟹ replacement A B [λx:α. C] [λx:α. D]"

lemma replacement_preserves_type:
  assumes "replacement A B C D"
  assumes "wff α A"
  assumes "wff α B"
  assumes "wff β C"
  shows "wff β D"
  ⟨proof⟩

lemma replacement_preserved_type:
  assumes "replacement A B C D"
  assumes "wff α A"
  assumes "wff α B"
  assumes "wff β D"
  shows "wff β C"
  ⟨proof⟩
```

8 Defined wffs

8.1 Common expressions

abbreviation (input) Var_yi (‹y_i›) **where**
"y_i == Cst ''y'' Ind"

abbreviation (input) Var_xo (‹x_o›) **where**
"x_o == Var ''x'' Tv"

abbreviation (input) Var_yo (‹y_o›) **where**
"y_o == Var ''y'' Tv"

abbreviation (input) Fun_oo (‹oo›) **where**
"oo == Tv ⇐ Tv"

abbreviation (input) Fun_ooo (‹ooo›) **where**
"ooo == oo ⇐ Tv"

abbreviation (input) Var_goo (‹g_{oo}›) **where**
"g_{oo} == Var ''g'' oo"

abbreviation (input) Var_gooo (‹g_{ooo}›) **where**
"g_{ooo} == Var ''g'' ooo"

8.2 Equality symbol

abbreviation QQ :: "type_sym ⇒ trm" (‹Q›) **where**
"Q α ≡ Cst ''Q'' α"

8.3 Description or selection operator

abbreviation $\iota\iota$ :: "trm" (‹$\boldsymbol{\iota}$›) **where**
"$\boldsymbol{\iota}$ ≡ Cst ''i'' (Ind ⇐ (Tv ⇐ Ind))"

8.4 Equality

definition Eql :: "trm ⇒ trm ⇒ type_sym ⇒ trm" **where**
"Eql A B α ≡ (Q (Tv ⇐ α ⇐ α)) · A · B"

abbreviation Eql' :: "trm ⇒ type_sym ⇒ trm ⇒ trm" (‹[_ =_= _]› [89]) **where**
"[A =α= B] ≡ Eql A B α"

definition LHS **where**
"LHS EqlAB = (case EqlAB of (_ · A · _) ⇒ A)"

lemma LHS_def2[simp]: "LHS [A =α= B] = A"
⟨*proof*⟩

definition RHS **where**
"RHS EqlAB = (case EqlAB of (_ · B) ⇒ B)"

lemma RHS_def2[simp]: "RHS ([A =α= B]) = B"
⟨*proof*⟩

lemma wff_Eql[simp]:
"wff α A ⟹ wff α B ⟹ wff Tv [A =α= B]"
⟨*proof*⟩

lemma wff_Eql_iff[simp]:
"wff β [A =α= B] ⟷ wff α A ∧ wff α B ∧ β = Tv"
⟨*proof*⟩

8.5 Truth

definition T :: trm **where**
"T ≡ [(Q ooo) =ooo= (Q ooo)]"

lemma wff_T[simp]: "wff Tv T"
⟨*proof*⟩

lemma wff_T_iff[simp]: "wff α T ⟷ α = Tv"
⟨*proof*⟩

8.6 Falsity

abbreviation F :: trm **where**
"F ≡ [[λ''x'':Tv. T] = oo= [λ''x'':Tv. x_o]]"

lemma wff_F[simp]: "wff Tv F"
⟨*proof*⟩

lemma wff_F_iff[simp]: "wff α F ⟷ α = Tv"
⟨*proof*⟩

8.7 Pi

definition PI :: "type_sym ⇒ trm" **where**
"PI α ≡ (Q (Tv ⇐ (Tv ⇐ α) ⇐ (Tv ⇐ α))) · [λ ''x'':α. T]"

lemma wff_PI[simp]: "wff (Tv ⇐ (Tv ⇐ α)) (PI α)"
⟨*proof*⟩

lemma wff_PI_subterm[simp]: "wff (Tv ⇐ α) [λ ''x'':α. T]"
⟨*proof*⟩

lemma wff_PI_subterm_iff[simp]:
"wff β [λ ''x'':α. T] ⟷ β = (Tv ⇐ α)"
⟨*proof*⟩

8.8 Forall

definition Forall :: "string ⇒ type_sym ⇒ trm ⇒ trm" (‹[∀_:_. _]› [80,80,80]) **where**
"[∀x:α. A] = (PI α) · [λx:α. A]"

lemma wff_Forall[simp]: "wff Tv A ⟹ wff Tv [∀x:α. A]"
⟨*proof*⟩

lemma wff_Forall_iff[simp]: "wff β [∀x:α. A] ⟷ wff Tv A ∧ β = Tv"
⟨*proof*⟩

8.9 Conjunction symbol

definition Con_sym :: trm **where**
"Con_sym ≡
[λ''x'':Tv. [λ''y'':Tv.
[[λ''g'':ooo. g_{ooo} · T · T] =Tv ⇐ ooo= [λ''g'':ooo. g_{ooo} · x_o · y_o]]
]]"

lemma wff_Con_sym[simp]: "wff ooo Con_sym"
⟨*proof*⟩

lemma wff_Con_sym'[simp]: "wff α Con_sym ⟷ α = ooo"
⟨*proof*⟩

lemma wff_Con_sym_subterm0[simp]:
"wff Tv A ⟹ wff Tv B ⟹ wff (Tv ⇐ ooo) [λ''g'':ooo. g_{ooo} · A · B]"

⟨*proof*⟩

lemma wff_Con_sym_subterm0_iff[simp]:
"wff β [λ''g'':ooo. g_{ooo} · A · B] ⟷ wff Tv A ∧ wff Tv B ∧ β = (Tv ⇐ ooo)"
⟨*proof*⟩

lemma wff_Con_sym_subterm1[simp]:
"wff Tv [[λ''g'':ooo. g_{ooo} · T · T] =(Tv ⇐ ooo)= [λ''g'':ooo. g_{ooo} · x_o · y_o]]"
⟨*proof*⟩

lemma wff_Con_sym_subterm1_iff[simp]:
"wff α [[λ''g'':ooo. g_{ooo} · T · T] =(Tv ⇐ ooo)= [λ''g'':ooo. g_{ooo} · x_o · y_o]] ⟷ α = Tv"
⟨*proof*⟩

lemma wff_Con_sym_subterm2[simp]:
"wff oo [λ ''y'':Tv. [[λ''g'':ooo. g_{ooo} · T · T] =(Tv ⇐ ooo)= [λ''g'':ooo. g_{ooo} · x_o · y_o]]]"
⟨*proof*⟩

lemma wff_Con_sym_subterm2_iff[simp]:
"wff α [λ ''y'':Tv. [[λ''g'':ooo. g_{ooo} · T · T] =(Tv ⇐ ooo)= [λ''g'':ooo. g_{ooo} · x_o · y_o]]] ⟷ α = oo"
⟨*proof*⟩

8.10 Conjunction

definition Con :: "trm ⇒ trm ⇒ trm" (**infix** ‹∧› 80) **where**
"A ∧ B = Con_sym · A · B"

lemma wff_Con[simp]: "wff Tv A ⟹ wff Tv B ⟹ wff Tv (A ∧ B)"
⟨*proof*⟩

lemma wff_Con_iff[simp]: "wff α (A ∧ B) ⟷ wff Tv A ∧ wff Tv B ∧ α = Tv"
⟨*proof*⟩

8.11 Implication symbol

definition Imp_sym :: trm **where**
"Imp_sym ≡ [λ''x'':Tv. [λ''y'':Tv. [x_o =Tv= (x_o ∧ y_o)]]]"

lemma wff_Imp_sym[simp]:
"wff ooo Imp_sym"
⟨*proof*⟩

lemma wff_Imp_sym_iff[simp]:
"wff α Imp_sym ⟷ α = ooo"
⟨*proof*⟩

lemma wff_Imp_sym_subterm0[simp]:
"wff Tv (x_o ∧ y_o)"
⟨*proof*⟩

lemma wff_Imp_sym_subterm0_iff[simp]:
"wff α (x_o ∧ y_o) ⟷ α = Tv"
⟨*proof*⟩

lemma wff_Imp_sym_subterm1[simp]:
"wff Tv [x_o =Tv= (x_o ∧ y_o)]"
⟨*proof*⟩

lemma wff_Imp_sym_subterm1_iff[simp]:
"wff α [x_o =Tv= (x_o ∧ y_o)] ⟷ α = Tv"
⟨*proof*⟩

lemma wff_Imp_sym_subterm2[simp]:
"wff oo [λ ''y'':Tv. [x_o =Tv= (x_o ∧ y_o)]]"
⟨*proof*⟩

lemma wff_Imp_sym_subterm2_iff[simp]:
"wff α [λ ''y'':Tv. [x_o =Tv= (x_o $\wedge$ y_o)]] $\longleftrightarrow$ α = oo"
⟨*proof*⟩

8.12 Implication

definition Imp :: "trm ⇒ trm ⇒ trm" (**infix** ‹⟶› 80) **where**
"A ⟶ B = Imp_sym · A · B"

lemma wff_Imp[simp]: "wff Tv A ⟹ wff Tv B ⟹ wff Tv (A ⟶ B)"
⟨*proof*⟩

lemma wff_Imp_iff[simp]: "wff α (A ⟶ B) $\longleftrightarrow$ wff Tv A $\wedge$ wff Tv B $\wedge$ α = Tv"
⟨*proof*⟩

9 The Q0 proof system

definition axiom_1 :: trm **where**
"axiom_1 ≡ [((g_{oo} · T) $\wedge$ (g_{oo} · F)) =Tv= [$\forall$ ''x'':Tv. g_{oo} · x_o]]"

lemma wff_axiom_1[simp]: "wff Tv axiom_1"
⟨*proof*⟩

definition axiom_2 :: "type_sym ⇒ trm" **where**
"axiom_2 α ≡
[(Var ''x'' α) =α= (Var ''y'' α)] ⟶
[((Var ''h'' (Tv ⇐ α)) · (Var ''x'' α)) =Tv= ((Var ''h'' (Tv ⇐ α)) · (Var ''y'' α))]"

definition axiom_3 :: "type_sym ⇒ type_sym ⇒ trm" **where**
"axiom_3 α β ≡
[[(Var ''f'' (α ⇐ β)) =α ⇐ β= (Var ''g'' (α ⇐ β))] =Tv=
[$\forall$ ''x'':β. [((Var ''f'' (α ⇐ β)) · (Var ''x'' β)) =α= ((Var ''g'' (α ⇐ β)) · (Var ''x'' β))]]]"

definition axiom_4_1 :: "var_sym ⇒ type_sym ⇒ trm ⇒ type_sym ⇒ trm ⇒ trm" **where**
"axiom_4_1 x α B β A ≡ [([λx:α. B] · A) =β= B]"

definition axiom_4_1_side_condition :: "var_sym ⇒ type_sym ⇒ trm ⇒ type_sym ⇒ trm ⇒ bool" **where**
"axiom_4_1_side_condition x α B β A ≡ ($\exists$c. B = Cst c β) $\vee$ ($\exists$y. B = Var y β $\wedge$ (x $\neq$ y $\vee$ α $\neq$ β))"

definition axiom_4_2 :: "var_sym ⇒ type_sym ⇒ trm ⇒ trm" **where**
"axiom_4_2 x α A = [([λx:α. Var x α] · A) =α= A]"

definition axiom_4_3 :: "var_sym ⇒ type_sym ⇒ trm ⇒
type_sym ⇒ type_sym ⇒ trm ⇒ trm ⇒ trm" **where**
"axiom_4_3 x α B β γ C A = [([λx:α. B · C] · A) =β= (([λx:α. B] · A) · ([λx:α. C] · A))]"

definition axiom_4_4 :: "var_sym ⇒ type_sym ⇒ var_sym ⇒ type_sym ⇒ trm ⇒ type_sym ⇒ trm ⇒ trm" **where**
"axiom_4_4 x α y γ B δ A = [([λx:α. [λy:γ. B]] · A) =δ ⇐ γ= [λy:γ. [λx:α. B] · A]]"

definition axiom_4_4_side_condition :: "var_sym ⇒ type_sym ⇒ var_sym ⇒ type_sym ⇒ trm ⇒ type_sym ⇒ trm ⇒
bool" **where**
"axiom_4_4_side_condition x α y γ B δ A ≡ (x $\neq$ y $\vee$ α $\neq$ γ) $\wedge$ (y, γ) $\notin$ vars A"

definition axiom_4_5 :: "var_sym ⇒ type_sym ⇒ trm ⇒ type_sym ⇒ trm ⇒ trm" **where**
"axiom_4_5 x α B δ A = [([λx:α. [λx:α. B]] · A) =δ ⇐ α = [λx:α. B]]"

definition axiom_5 **where**
"axiom_5 = [(ι · ((Q (Tv ⇐ Ind ⇐ Ind)) · y_i)) =Ind= y_i]"

inductive axiom :: "trm ⇒ bool" **where**
by_axiom_1:
"axiom axiom_1"
| by_axiom_2:

```
  "axiom (axiom_2 α)"
| by_axiom_3:
  "axiom (axiom_3 α β)"
| by_axiom_4_1:
  "wff α A ⟹
   wff β B ⟹
   axiom_4_1_side_condition x α B β A ⟹
   axiom (axiom_4_1 x α B β A)"
| by_axiom_4_2:
  "wff α A ⟹
   axiom (axiom_4_2 x α A)"
| by_axiom_4_3:
  "wff α A ⟹
   wff (β ⇐ γ) B ⟹
   wff γ C ⟹
   axiom (axiom_4_3 x α B β γ C A)"
| by_axiom_4_4:
  "wff α A ⟹
   wff δ B ⟹
   axiom_4_4_side_condition x α y γ B δ A ⟹
   axiom (axiom_4_4 x α y γ B δ A)"
| by_axiom_4_5:
  "wff α A ⟹
   wff δ B ⟹
   axiom (axiom_4_5 x α B δ A)"
| by_axiom_5:
  "axiom (axiom_5)"

inductive rule_R :: "trm ⇒ trm ⇒ trm ⇒ bool" where
  "replacement A B C D ⟹ rule_R C ([A =α= B]) D"

definition "proof" :: "trm ⇒ trm list ⇒ bool" where
  "proof A p ⟷ (p ≠ [] ∧ last p = A ∧
                  (∀i<length p. axiom (p ! i)
                ∨ (∃j<i. ∃k<i. rule_R (p ! j) (p ! k) (p ! i))))"


inductive "theorem" :: "trm ⇒ bool" where
  by_axiom: "axiom A ⟹ theorem A"
| by_rule_R: "theorem A ⟹ theorem B ⟹ rule_R A B C ⟹ theorem C"


definition axiom_4_1_variant_cst :: "var_sym ⇒ type_sym ⇒ var_sym ⇒ type_sym ⇒ trm ⇒ trm" where
  "axiom_4_1_variant_cst x α c β A ≡ [([λx:α. Cst c β] · A) =β= (Cst c β)]"

definition axiom_4_1_variant_var :: "var_sym ⇒ type_sym ⇒ var_sym ⇒ type_sym ⇒ trm ⇒ trm" where
  "axiom_4_1_variant_var x α y β A ≡  [([λx:α. Var y β] · A) =β= Var y β]"

definition axiom_4_1_variant_var_side_condition :: "var_sym ⇒ type_sym ⇒ var_sym ⇒ type_sym ⇒ trm ⇒ bool"
where
  "axiom_4_1_variant_var_side_condition x α y β A ≡ x ≠ y ∨ α ≠ β"
```

10 Semantics

```
type_synonym 's frame = "type_sym ⇒ 's"

type_synonym 's denotation = "cst_sym ⇒ type_sym ⇒ 's"

type_synonym 's asg = "var_sym * type_sym ⇒ 's"

definition agree_off_asg :: "'s asg ⇒ 's asg ⇒ var_sym ⇒ type_sym ⇒ bool" where
  "agree_off_asg φ ψ x α ⟷ (∀y β. (y≠x ∨ β ≠ α) ⟶ φ (y,β) = ψ (y,β))"

lemma agree_off_asg_def2:
```

```
  "agree_off_asg ψ φ x α ⟷ (∃xa. φ((x, α) := xa) = ψ)"
  ⟨proof⟩

lemma agree_off_asg_disagree_var_sym[simp]:
  "agree_off_asg ψ φ x α ⟹ x ≠ y ⟹ ψ(y,β) = φ(y,β)"
  ⟨proof⟩

lemma agree_off_asg_disagree_type_sym[simp]:
  "agree_off_asg ψ φ x α ⟹ α ≠ β ⟹ ψ(y,β) = φ(y,β)"
  ⟨proof⟩


context set_theory
begin

definition wf_frame :: "'s frame ⇒ bool" where
  "wf_frame D ⟷ D Tv = boolset ∧ (∀α β. D (α ⇐ β) ⊆: funspace (D β) (D α)) ∧ (∀α. D α ≠ ∅)"

definition inds :: "'s frame ⇒ 's" where
  "inds Fr = Fr Ind"

inductive wf_interp :: "'s frame ⇒ 's denotation ⇒ bool" where
  "wf_frame D ⟹
   ∀c α. I c α ∈: D α ⟹
   ∀α. I ''Q'' (Tv ⇐ α ⇐ α) = iden (D α) ⟹
   (I ''i'' (Ind ⇐ (Tv ⇐ Ind))) ∈: funspace (D (Tv ⇐ Ind)) (D Ind) ⟹
   ∀x. x ∈: D Ind ⟶ (I ''i'' (Ind ⇐ (Tv ⇐ Ind))) · one_elem_fun x (D Ind) = x ⟹
   wf_interp D I"

definition asg_into_frame :: "'s asg ⇒ 's frame ⇒ bool" where
  "asg_into_frame φ D ⟷ (∀x α. φ (x, α) ∈: D α)"

abbreviation(input) asg_into_interp :: "'s asg ⇒ 's frame ⇒ 's denotation ⇒ bool" where
  "asg_into_interp φ D I ≡ asg_into_frame φ D"


fun val :: "'s frame ⇒ 's denotation ⇒ 's asg ⇒ trm ⇒ 's" where
  "val D I φ (Var x α) = φ (x,α)"
| "val D I φ (Cst c α) = I c α"
| "val D I φ (A · B) = val D I φ A · val D I φ B"
| "val D I φ [λx:α. B] = abstract (D α) (D (type_of B)) (λz. val D I (φ((x,α):=z)) B)"

fun general_model :: "'s frame ⇒ 's denotation ⇒ bool" where
  "general_model D I ⟷ wf_interp D I ∧ (∀φ A α. asg_into_interp φ D I ⟶ wff α A ⟶ val D I φ A ∈: D α)"

fun standard_model :: "'s frame ⇒ 's denotation ⇒ bool" where
  "standard_model D I ⟷ wf_interp D I ∧ (∀α β. D (α ⇐ β) = funspace (D β) (D α))"

lemma asg_into_frame_fun_upd:
  assumes "asg_into_frame φ D"
  assumes "xa ∈: D α"
  shows "asg_into_frame (φ((x, α) := xa)) D"
  ⟨proof⟩

lemma asg_into_interp_fun_upd:
  assumes "general_model D I"
  assumes "asg_into_interp φ D I"
  assumes "wff α A"
  shows "asg_into_interp (φ((x, α) := val D I φ A)) D I"
  ⟨proof⟩

lemma standard_model_is_general_model:
  assumes "standard_model D I"
  shows "general_model D I"
```

⟨*proof*⟩

abbreviation agree_on_asg :: "'s asg ⇒ 's asg ⇒ var_sym ⇒ type_sym ⇒ bool" **where**
"agree_on_asg φ ψ x α == (φ (x, α) = ψ (x, α))"

proposition prop_5400:
assumes "general_model D I"
assumes "asg_into_interp φ D I"
assumes "asg_into_interp ψ D I"
assumes "wff α A"
assumes "∀(x,α) ∈ frees A. agree_on_asg φ ψ x α"
shows "val D I φ A = val D I ψ A"
⟨*proof*⟩

abbreviation satisfies :: "'s frame ⇒ 's denotation ⇒ 's asg ⇒ trm ⇒ bool" **where**
"satisfies D I φ A ≡ (val D I φ A = true)"

definition valid_in_model :: "'s frame ⇒ 's denotation ⇒ trm ⇒ bool" **where**
"valid_in_model D I A ≡ (∀φ. asg_into_interp φ D I ⟶ val D I φ A = true)"

definition valid_general :: "trm ⇒ bool" **where**
"valid_general A ≡ ∀D I. general_model D I ⟶ valid_in_model D I A"

definition valid_standard :: "trm ⇒ bool" **where**
"valid_standard A ≡ ∀D I. standard_model D I ⟶ valid_in_model D I A"

11 Semantics of defined wffs

lemma lemma_5401_a:
assumes "general_model D I"
assumes "asg_into_interp φ D I"
assumes "wff α A" "wff β B"
shows "val D I φ ([λx:α. B] · A) = val D I (φ((x,α):=val D I φ A)) B"
⟨*proof*⟩

lemma lemma_5401_b_variant_1:
assumes "general_model D I"
assumes "asg_into_interp φ D I"
assumes "wff α A" "wff α B"
shows "val D I φ ([A =α= B]) = (boolean (val D I φ A = val D I φ B))"
⟨*proof*⟩

lemma lemma_5401_b:
assumes "general_model D I"
assumes "asg_into_interp φ D I"
assumes "wff α A" "wff α B"
shows "val D I φ ([A =α= B]) = true ⟷ val D I φ A = val D I φ B"
⟨*proof*⟩

lemma lemma_5401_b_variant_2: — Just a reformulation of lemma_5401_b's directions
assumes "general_model D I"
assumes "asg_into_interp φ D I"
assumes "wff α A" "wff α B"
assumes "val D I φ A = val D I φ B"
shows "val D I φ ([A =α= B]) = true"
⟨*proof*⟩

lemma lemma_5401_b_variant_3: — Just a reformulation of lemma_5401_b's directions

assumes "general_model D I"
assumes "asg_into_interp φ D I"
assumes "wff α A" "wff α B"
assumes "val D I φ A $\neq$ val D I φ B"
shows "val D I φ ([A =α= B]) = false"
⟨*proof*⟩

lemma lemma_5401_c:
assumes "general_model D I"
assumes "asg_into_interp φ D I"
shows "val D I φ T = true"
⟨*proof*⟩

lemma lemma_5401_d:
assumes "general_model D I"
assumes "asg_into_interp φ D I"
shows "val D I φ F = false"
⟨*proof*⟩

lemma asg_into_interp_fun_upd_true:
assumes "general_model D I"
assumes "asg_into_interp φ D I"
shows "asg_into_interp (φ((x, Tv) := true)) D I"
⟨*proof*⟩

lemma asg_into_interp_fun_upd_false:
assumes "general_model D I"
assumes "asg_into_interp φ D I"
shows "asg_into_interp (φ((x, Tv) := false)) D I"
⟨*proof*⟩

lemma lemma_5401_e_1:
assumes "general_model D I"
assumes "asg_into_interp φ D I"
shows "(val D I φ Con_sym) · true · true = true"
⟨*proof*⟩

lemma lemma_5401_e_2:
assumes "general_model D I"
assumes "asg_into_interp φ D I"
assumes "y = true $\vee$ y = false"
shows "(val D I φ Con_sym) · false · y = false"
⟨*proof*⟩

lemma lemma_5401_e_3:
assumes "general_model D I"
assumes "asg_into_interp φ D I"
assumes "x = true $\vee$ x = false"
shows "(val D I φ Con_sym) · x · false = false"
⟨*proof*⟩

lemma lemma_5401_e_variant_1:
assumes "asg_into_interp φ D I"
assumes "general_model D I"
assumes "y = true $\vee$ y = false"
assumes "x = true $\vee$ x = false"
shows "(val D I φ Con_sym) · x · y = boolean (x = true $\wedge$ y = true)"
⟨*proof*⟩

lemma asg_into_interp_is_true_or_false:
assumes "general_model D I"
assumes "asg_into_interp φ D I"
shows "φ (x, Tv) = true $\vee$ φ (x, Tv) = false"
⟨*proof*⟩

lemma wff_Tv_is_true_or_false:
assumes "asg_into_interp φ D I"
assumes "general_model D I"
assumes "wff Tv A"
shows "val D I φ A = true $\vee$ val D I φ A = false"
⟨*proof*⟩

lemma lemma_5401_e_variant_2:
assumes "asg_into_interp φ D I"
assumes "general_model D I"
assumes "wff Tv A"
assumes "wff Tv B"
shows "(val D I φ (A $\wedge$ B)) = boolean (satisfies D I φ A $\wedge$ satisfies D I φ B)"
⟨*proof*⟩

lemma lemma_5401_f_1:
assumes "general_model D I"
assumes "asg_into_interp φ D I"
assumes "y = true $\vee$ y = false"
shows "(val D I φ Imp_sym) · false · y = true"
⟨*proof*⟩

lemma lemma_5401_f_2:
assumes "general_model D I"
assumes "asg_into_interp φ D I"
assumes "x = true $\vee$ x = false"
shows "(val D I φ Imp_sym) · x · true = true"
⟨*proof*⟩

lemma lemma_5401_f_3:
assumes "general_model D I"
assumes "asg_into_interp φ D I"
shows "(val D I φ Imp_sym) · true · false = false"
⟨*proof*⟩

lemma lemma_5401_f_variant_1:
assumes "asg_into_interp φ D I"
assumes "general_model D I"
assumes "x = true $\vee$ x = false"
assumes "y = true $\vee$ y = false"
shows "(val D I φ Imp_sym) · x · y = boolean (x = true $\longrightarrow$ y = true)"
⟨*proof*⟩

lemma lemma_5401_f_variant_2:
assumes "asg_into_interp φ D I"
assumes "general_model D I"
assumes "wff Tv A"
assumes "wff Tv B"
shows "(val D I φ (A $\longrightarrow$ B)) = boolean (satisfies D I φ A $\longrightarrow$ satisfies D I φ B)"
⟨*proof*⟩

lemma lemma_5401_g:
assumes "general_model D I"
assumes "asg_into_interp φ D I"
assumes "wff Tv A"
shows "satisfies D I φ [$\forall$x:α. A] $\longleftrightarrow$
$(\forall \psi$. asg_into_interp ψ D I $\longrightarrow$ agree_off_asg ψ φ x α $\longrightarrow$ satisfies D I ψ A)"
⟨*proof*⟩

theorem lemma_5401_g_variant_1:
assumes "asg_into_interp φ D I"
assumes "general_model D I"
assumes "wff Tv A"
shows "val D I φ [$\forall$x:α. A] =
boolean $(\forall \psi$. asg_into_interp ψ D I $\longrightarrow$ agree_off_asg ψ φ x α $\longrightarrow$ satisfies D I ψ A)"
⟨*proof*⟩

12 Soundness

lemma fun_sym_asg_to_funspace:
assumes "asg_into_frame φ D"
assumes "general_model D I"
shows "φ (f, α $\Leftarrow$ β) $\in$: funspace (D β) (D α)"
⟨*proof*⟩

lemma fun_sym_interp_to_funspace:
assumes "asg_into_frame φ D"
assumes "general_model D I"
shows "I f (α $\Leftarrow$ β) $\in$: funspace (D β) (D α)"
⟨*proof*⟩

theorem theorem_5402_a_rule_R:
assumes A_eql_B: "valid_general ([A =α= B])"
assumes "valid_general C"
assumes "rule_R C ([A =α= B]) C'"
assumes "wff α A"
assumes "wff α B"
assumes "wff β C"
shows "valid_general C'"
⟨*proof*⟩

theorem Fun_Tv_Tv_frame_subs_funspace:
assumes "general_model D I"
assumes "asg_into_interp φ D I"
shows "D oo $\subseteq$: funspace (boolset) (boolset)"
⟨*proof*⟩

theorem theorem_5402_a_axiom_1_variant:
assumes "general_model D I"
assumes "asg_into_interp φ D I"
shows "satisfies D I φ axiom_1"
⟨*proof*⟩

theorem theorem_5402_a_axiom_1: "valid_general axiom_1"
⟨*proof*⟩

theorem theorem_5402_a_axiom_2_variant:
assumes "general_model D I"
assumes "asg_into_interp φ D I"

shows "satisfies D I φ (axiom_2 α)"
⟨*proof*⟩

theorem theorem_5402_a_axiom_2: "valid_general (axiom_2 α)"
⟨*proof*⟩

theorem theorem_5402_a_axiom_3_variant:
assumes "general_model D I"
assumes "asg_into_interp φ D I"
shows "satisfies D I φ (axiom_3 α β)"
⟨*proof*⟩

theorem theorem_5402_a_axiom_3: "valid_general (axiom_3 α β)"
⟨*proof*⟩

theorem theorem_5402_a_axiom_4_1_variant_cst:
assumes "general_model D I"
assumes "asg_into_interp φ D I"
assumes "wff α A"
shows "satisfies D I φ (axiom_4_1_variant_cst x α c β A)"
⟨*proof*⟩

theorem theorem_5402_a_axiom_4_1_variant_var:
assumes "general_model D I"
assumes "asg_into_interp φ D I"
assumes "wff α A"
assumes "axiom_4_1_variant_var_side_condition x α y β A"
shows "satisfies D I φ (axiom_4_1_variant_var x α y β A)"
⟨*proof*⟩

theorem theorem_5402_a_axiom_4_1:
assumes "asg_into_interp φ D I"
assumes "general_model D I"
assumes "axiom_4_1_side_condition x α y β A"
assumes "wff α A"
shows "satisfies D I φ (axiom_4_1 x α y β A)"
⟨*proof*⟩

theorem theorem_5402_a_axiom_4_2:
assumes "general_model D I"
assumes "asg_into_interp φ D I"
assumes "wff α A"
shows "satisfies D I φ (axiom_4_2 x α A)"
⟨*proof*⟩

theorem theorem_5402_a_axiom_4_3:
assumes "general_model D I"
assumes "asg_into_interp φ D I"
assumes "wff α A"
assumes "wff (β $\Leftarrow$ γ) B"
assumes "wff γ C"
shows "satisfies D I φ (axiom_4_3 x α B β γ C A)"
⟨*proof*⟩

lemma lemma_to_help_with_theorem_5402_a_axiom_4_4:
assumes lambda_eql_lambda_lambda:

```
    "⋀z. z ∈: D γ ⟹ val D I ψ [λy:γ. B] · z = val D I φ [λy:γ. [λx:α. B] · A] · z"
  assumes ψ_eql: "ψ = φ((x, α) := val D I φ A)"
  assumes "asg_into_frame φ D"
  assumes "general_model D I"
  assumes "axiom_4_4_side_condition x α y γ B δ A"
  assumes "wff α A"
  assumes "wff δ B"
  shows "val D I ψ [λy:γ. B] = val D I φ [λy:γ. [λx:α. B] · A]"
⟨proof⟩


theorem theorem_5402_a_axiom_4_4:
  assumes "general_model D I"
  assumes "asg_into_interp φ D I"
  assumes "axiom_4_4_side_condition x α y γ B δ A"
  assumes "wff α A"
  assumes "wff δ B"
  shows "satisfies D I φ (axiom_4_4 x α y γ B δ A)"
⟨proof⟩


theorem theorem_5402_a_axiom_4_5:
  assumes "general_model D I"
  assumes "asg_into_interp φ D I"
  assumes "wff α A"
  assumes "wff δ B"
  shows "satisfies D I φ (axiom_4_5 x α B δ A)"
⟨proof⟩


theorem theorem_5402_a_axiom_5:
  assumes "general_model D I"
  assumes "asg_into_interp φ D I"
  shows "satisfies D I φ (axiom_5)"
⟨proof⟩

lemma theorem_isa_Tv:
  assumes "theorem A"
  shows "wff Tv A"
  ⟨proof⟩


theorem theorem_5402_a_general:
  assumes "theorem A"
  shows "valid_general A"
  ⟨proof⟩


theorem theorem_5402_a_standard:
  assumes "theorem A"
  shows "valid_standard A"
  ⟨proof⟩

end

end
```

References

[AM23] Oskar Abrahamsson and Magnus O. Myreen. Fast, verified computation for Candle. In Adam Naumowicz and René Thiemann, editors, *14th International Conference on Interactive Theorem Proving, ITP 2023, July 31 to August 4, 2023, Białystok, Poland*, volume 268 of *LIPIcs*, pages 4:1–4:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[AMKS22] Oskar Abrahamsson, Magnus O. Myreen, Ramana Kumar, and Thomas Sewell. Candle: A verified implementation of HOL Light. In June Andronick and Leonardo de Moura, editors, *13th International Conference on Interactive Theorem Proving, ITP 2022, August 7-10, 2022, Haifa, Israel*, volume 237 of *LIPIcs*, pages 3:1–3:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[And63] Peter Andrews. A reduction of the axioms for the theory of propositional types. *Fundamenta Mathematicae*, 52:345–350, 1963.

[And72] Peter B Andrews. General models and extensionality. *The Journal of Symbolic Logic*, 37(2):395–397, 1972.

[And13] Peter B. Andrews. *An introduction to mathematical logic and type theory: to truth through proof*, volume 27 of *Applied Logic Series*. Springer Science & Business Media, 2nd edition, 2013.

[Art02] Rob Arthan. HOL formalised: Deductive system, 1993, revised 2002. http://www.lemma-one.com/ProofPower/specs/specs.html.

[Art14a] Rob Arthan. HOL formalised: Language and overview, 1993, revised 2014. http://www.lemma-one.com/ProofPower/specs/specs.html.

[Art14b] Rob Arthan. HOL formalised: Semantics, 1993, revised 2014. http://www.lemma-one.com/ProofPower/specs/specs.html.

[Bar96a] Bruno Barras. Coq en Coq. Research Report RR-3026, INRIA, 1996. Projet COQ, https://inria.hal.science/inria-00073667.

[Bar96b] Bruno Barras. Verification of the interface of a small proof system in Coq. In Eduardo Giménez and Christine Paulin-Mohring, editors, *Types for Proofs and Programs, International Workshop TYPES'96, Aussois, France, December 15-19, 1996, Selected Papers*, volume 1512 of *Lecture Notes in Computer Science*, pages 28–45. Springer, 1996.

[Bar97] Bruno Barras. coq-in-coq. In *coq-contribs*, 1997. https://github.com/coq-contribs/coq-in-coq.

[BW96] Bruno Barras and Benjamin Werner. Coq in Coq (manuscript). Technical report, 1996. http://www.lix.polytechnique.fr/Labo/Bruno.Barras/publi/coqincoq.pdf.

[Chu40] Alonzo Church. A formulation of the simple theory of types. *The journal of symbolic logic*, 5(2):56–68, 1940.

[Día23] Javier Díaz. Metatheory of Q0. *Archive of Formal Proofs*, November 2023. https://isa-afp.org/entries/Q0_Metatheory.html, Formal proof development.

[Har06] John Harrison. Towards self-verification of HOL Light. In Ulrich Furbach and Natarajan Shankar, editors, *Automated Reasoning, Third International Joint Conference, IJCAR 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, volume 4130 of *Lecture Notes in Computer Science*, pages 177–191. Springer, 2006.

[Hen50] Leon Henkin. Completeness in the theory of types. *The Journal of Symbolic Logic*, 15(2):81–91, 1950.

[Hen63] Leon Henkin. A theory of propositional types. *Fundamenta Mathematicae*, 52:323–344, 1963.

[KAMO14] Ramana Kumar, Rob Arthan, Magnus O. Myreen, and Scott Owens. HOL with definitions: Semantics, soundness, and a verified implementation. In Gerwin Klein and Ruben Gamboa, editors, *Interactive Theorem Proving - 5th International Conference, ITP 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, volume 8558 of *Lecture Notes in Computer Science*, pages 308–324. Springer, 2014.

[KAMO16] Ramana Kumar, Rob Arthan, Magnus O. Myreen, and Scott Owens. Self-formalisation of higher-order logic - semantics, soundness, and a verified implementation. *J. Autom. Reason.*, 56(3):221–259, 2016.

[KK22] Mark Koch and Dominik Kirst. Undecidability, incompleteness, and completeness of second-order logic in coq. In Andrei Popescu and Steve Zdancewic, editors, *CPP '22: 11th ACM SIGPLAN International Conference on Certified Programs and Proofs, Philadelphia, PA, USA, January 17 - 18, 2022*, pages 274–290. ACM, 2022.

[NR21a] Tobias Nipkow and Simon Roßkopf. Isabelle's metalogic: Formalization and proof checker. In André Platzer and Geoff Sutcliffe, editors, *Automated Deduction - CADE 28 - 28th International Conference on Automated Deduction, Virtual Event, July 12-15, 2021, Proceedings*, volume 12699 of *Lecture Notes in Computer Science*, pages 93–110. Springer, 2021.

[NR21b] Tobias Nipkow and Simon Roßkopf. Isabelle's metalogic: Formalization and proof checker. *Archive of Formal Proofs*, April 2021. https://isa-afp.org/entries/Metalogic_ProofChecker.html, Formal proof development.

[Obu06] Steven Obua. Partizan games in Isabelle/HOLZF. In Kamel Barkaoui, Ana Cavalcanti, and Antonio Cerone, editors, *Theoretical Aspects of Computing - ICTAC 2006, Third International Colloquium, Tunis, Tunisia, November 20-24, 2006, Proceedings*, volume 4281 of *Lecture Notes in Computer Science*, pages 272–286. Springer, 2006.

[Pau19] Lawrence C. Paulson. Zermelo Fraenkel set theory in higher-order logic. *Archive of Formal Proofs*, October 2019. https://isa-afp.org/entries/ZFC_in_HOL.html, Formal proof development.

[RN23] Simon Roßkopf and Tobias Nipkow. A formalization and proof checker for Isabelle's metalogic. *J. Autom. Reason.*, 67(1):1, 2023.

[Rus08] Bertrand Russell. Mathematical logic as based on the theory of types. *American journal of mathematics*, 30(3):222–262, 1908.

[SBF+20] Matthieu Sozeau, Simon Boulier, Yannick Forster, Nicolas Tabareau, and Théo Winterhalter. Coq Coq correct! verification of type checking and erasure for Coq, in Coq. *Proc. ACM Program. Lang.*, 4(POPL):8:1–8:28, 2020.

[Sch23] Anders Schlichtkrull. Q0. *IsaFoL*, Jan 4th 2022 - Nov 2023. Now at GitHub https://github.com/IsaFoL/IsaFoL/tree/master/Q0 but formerly at BitBucket https://bitbucket.org/isafol/isafol/src/master/Q0/. Formal proof development.

[WR13] Alfred North Whitehead and Bertrand Russell. *Principia Mathematica*. Cambridge University Press, Cambridge England, 1913. 3 volumes; first edition 1913, second edition 1927.