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].

Contents

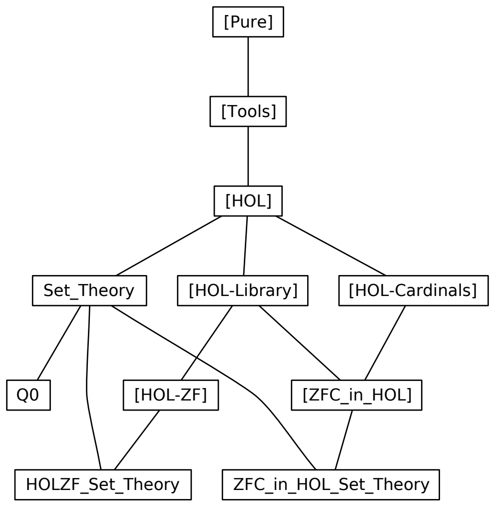

Figure 1: Theory dependency graph

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].

In the literature we find other formalizations of the metatheory of higher-order logics and type theories. 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/setSpecScript.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
  fix x
  show "sub2 x = (suchthat) x"
    using assms extensional by auto
qed


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

```
  using assms extensional by auto


lemma uni_unique:
  assumes "∀x a. a ∈: uni2 x ⟷ (∃b. a ∈: b ∧ b ∈: x)"
  shows "uni2 = uni"
  using assms extensional by auto


lemma upair_unique:
  assumes "∀x y a. a ∈: upair2 x y ⟷ a = x ∨ a = y"
  shows "upair2 = upair"
proof
  fix x
  show "upair2 x = (+:) x"
    using assms extensional by auto
qed


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


lemma mem_empty[simp]: "¬x ∈: ∅"
  unfolding empt_def using mem_sub by auto


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


lemma mem_unit[simp]: "x ∈: (unit y) ⟷ x = y"
  unfolding unit_def using mem_upair by auto


lemma unit_inj: "unit x = unit y ⟷ x = y"
  using extensional unfolding unit_def by auto


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


lemma mem_one[simp]: "x ∈: one ⟷ x = ∅"
  unfolding one_def by auto


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


lemma mem_two[simp]: "∀x. x ∈: two ⟷ x = ∅ ∨ x = one"
  unfolding two_def by auto


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"
  using extensional by auto
```

```
lemma unit_eq_upair:
  "unit x = y +: z ⟷ x = y ∧ y = z"
  using extensional mem_unit mem_upair by metis


lemma pair_inj:
  "(a,:b) = (c,:d) ⟷ a = c ∧ b = d"
  using pair_def upair_inj unit_inj unit_eq_upair by metis


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


lemma mem_binary_uni[simp]:
  "a ∈: (x ∪: y) ⟷ a ∈: x ∨ a ∈: y"
  unfolding binary_uni_def by auto


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)"
  using product_def pair_def by auto


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"
  using true_def false_def extensional one_def by auto


lemma mem_boolset[simp]:
  "x ∈: boolset ⟷ ((x = true) ∨ (x = false))"
  using true_def false_def boolset_def by auto
```

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


lemma boolean_in_boolset:
  "boolean b ∈: boolset"
  using boolean_def one_def true_def false_def by auto


lemma boolean_eq_true:
  "boolean b = true ⟷ b"
  using boolean_def true_neq_false by auto


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"
  using apply_def abstract_def pair_inj by auto


lemma apply_abstract_matchable:
  "x ∈: s ⟹ f x ∈: t ⟹ f x = u ⟹ abstract s t f · x = u"
  using apply_abstract by auto


lemma apply_in_rng:
  assumes "x ∈: s"
  assumes "f ∈: funspace s t"
  shows "f · x ∈: t"
proof -
  from assms have "f ∈: (relspace s t suchthat (λf. ∀a. a ∈: s ⟶ (∃!b. (a ,: b) ∈: f)))"
    unfolding funspace_def by auto
  then have f_p: "f ∈: relspace s t ∧ (∀a. a ∈: s ⟶ (∃!b. (a ,: b) ∈: f))"
    by auto
  then have fxf: "(x ,: f · x) ∈: f"
    using someI assms apply_def by metis
  from f_p have "f ∈: pow (s ×: t)"
    unfolding relspace_def by auto
  then have "(x ,: f · x) ∈: (s ×: t)"
    using fxf by auto
  then show ?thesis
    using pair_inj by auto
qed


lemma abstract_in_funspace[simp]:
  "(∀x. x ∈: s ⟶ f x ∈: t) ⟹ abstract s t f ∈: funspace s t"
  using funspace_def relspace_def abstract_def pair_inj by auto


lemma abstract_in_funspace_matchable:
  "(∀x. x ∈: s ⟶ f x ∈: t) ⟹ fs = funspace s t ⟹ abstract s t f ∈: fs"
  using abstract_in_funspace by auto


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 (rule iffD2[OF extensional], rule)
  fix a
  from assms show "a ∈: abstract s t1 f = a ∈: abstract s t2 g"
    unfolding abstract_def using pair_inj by auto
qed

lemma abstract_extensional:
  assumes "∀x. x ∈: s ⟶ f x ∈: t ∧ f x = g x"
  shows "abstract s t f = abstract s t g"
proof (rule iffD2[OF extensional], rule)
  fix a
  from assms show "a ∈: abstract s t f = a ∈: abstract s t g"
    unfolding abstract_def using pair_inj by auto
qed

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 (rule iffD2[OF extensional], rule)
  fix a
  from assms show "a ∈: abstract s t f = a ∈: abstract s t g"
    unfolding abstract_def using pair_inj by auto
qed

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"
  using assms
  by (metis apply_abstract_matchable)

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 -
  have "f x = abstract s t f · x"
    using apply_abstract[of x s f t]
    using assms by auto
  also have "... = abstract s t g · x"
    using assms by auto
  also have "... = g x"
    using apply_abstract[of x s g t]
    using assms by auto
  finally show ?thesis
    by auto
qed
thm set_eq_iff
thm fun_cong
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)"
  using assms abstract_cong
    abstract_extensional by meson


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

```
  define f where "f = (λx. SOME y. (x,:y) ∈: z)"
  have "∀x. x ∈: z ⟷ x ∈: (abstract s t f)"
  proof (rule, rule)
    fix x
    assume xz: "x ∈: z"
    moreover
    have "∀b. b ∈: z ⟶ (∃ba c. b = (ba ,: c) ∧ ba ∈: s ∧ c ∈: t)"
      using xz assms
      unfolding funspace_def relspace_def by (auto)
    moreover
    have "∀a. a ∈: s ⟶ (∃!b. (a ,: b) ∈: z)"
      using xz using assms
      unfolding funspace_def relspace_def by auto
    ultimately
    have "∃a. x = (a ,: f a)"
      using assms f_def someI_ex unfolding funspace_def relspace_def by (metis (no_types, lifting))
    then show "x ∈: abstract s t f"
      using xz assms unfolding funspace_def relspace_def abstract_def by auto
  next
    fix x
    assume xf: "x ∈: abstract s t f"
    then have "(∃b c. x = (b ,: c) ∧ b ∈: s ∧ c ∈: t)"
      unfolding abstract_def by simp
    moreover
    have "(∃a. x = (a ,: (SOME y. (a ,: y) ∈: z)))"
      using xf f_def unfolding abstract_def by simp
    moreover
    have "(∀a. a ∈: s ⟶ (∃!b. (a ,: b) ∈: z))"
      using assms unfolding funspace_def by simp
    ultimately
    show "x ∈: z"
      using f_def by (metis pair_inj someI_ex)
  qed
  then have "z = abstract s t f"
    using extensional by auto
  moreover
  from f_def have "∀x. x ∈: s ⟶ f x ∈: t"
    using apply_in_rng assms local.apply_def by force
  ultimately
  show ?thesis
    by auto
qed


theorem axiom_of_choice:
  assumes "∀a. a ∈: x ⟶ (∃b. b ∈: a)"
  shows "∃f. ∀a. a ∈: x ⟶ f · a ∈: a"
proof -
  define f where "f = (λa. SOME b. mem b a)"
  define fa where "fa = abstract x (uni x) f"

  have "∀a. a ∈: x ⟶ fa · a ∈: a"
  proof (rule, rule)
    fix a
    assume "a ∈: x"
    moreover
    have "f a ∈: ⋃:x"
      by (metis (full_types) assms calculation f_def mem_uni someI_ex)
    moreover
    have "f a ∈: a"
      using assms calculation(1) f_def someI_ex by force
    ultimately
    show "fa · a ∈: a"
      unfolding fa_def using apply_abstract by auto
```

```
  qed
  then show ?thesis
    by auto
qed


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)"
  apply (rule_tac x="abstract s t (λx. SOME x. x ∈: t)" in exI)
  using abstract_in_funspace
  using someI by metis


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


lemma pair_not_empty:
  "(x,:y) ≠ ∅"
  apply rule
  unfolding extensional using mem_empty pair_def mem_upair by metis


lemma tuple_empty:
  "tuple ls = ∅ ⟷ ls = []"
  using pair_not_empty by (cases ls) auto


lemma tuple_inj:
  "tuple l1 = tuple l2 ⟷ l1 = l2"
proof (induction l1 arbitrary: l2)
  case Nil
  then show ?case
    using tuple_empty by metis
next
  case (Cons a l1)
  then show ?case
    using pair_not_empty pair_inj by (metis tuple.elims tuple.simps(2))
qed


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 (induction ls arbitrary: x)
  case Nil
  then show ?case
    using mem_one mem_product by simp
next
  case (Cons l ls)
  show ?case
  proof
    assume "x ∈: bigcross (l # ls)"
    then obtain b c where bc_p: "x = (b ,: c) ∧ b ∈: l ∧ c ∈: bigcross ls "
      by auto
```

```
    then obtain xs' where "c = tuple xs' ∧ list_all2 (∈:) xs' ls"
      using Cons[of c] by auto
    then have "x = tuple (b#xs') ∧ list_all2 (∈:) (b#xs') (l # ls)"
      using bc_p by auto
    then show "∃xs. x = tuple xs ∧ list_all2 (∈:) xs (l # ls)"
      by metis
  next
    assume "∃xs. x = tuple xs ∧ list_all2 (∈:) xs (l # ls)"
    then obtain xs where "x = tuple xs ∧ list_all2 (∈:) xs (l # ls)"
      by auto
    then obtain xs where "x = tuple xs ∧ list_all2 (∈:) xs (l # ls)"
      by auto
    then show "x ∈: bigcross (l # ls)"
      using Cons list.distinct(1) list.rel_cases mem_product by fastforce
  qed
qed

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 -
  have abstract_D: "abstract D boolset (λy. boolean (A = y)) ∈: funspace D boolset"
    using boolean_in_boolset by auto
  have bool_in_two: "boolean (A = B) ∈: boolset"
    using boolean_in_boolset by blast
  have "(boolean (A = B)) = (abstract D boolset (λy. boolean (A = y)) · B)"
    using apply_abstract[of B D "λy. boolean (A = y)" two] B_in_D bool_in_two by auto
  also
  have "... = (abstract D (funspace D boolset) (λx. abstract D boolset (λy. boolean (x = y))) · A) · B"
    using A_in_D abstract_D
      apply_abstract[of A D "λx. abstract D boolset (λy. boolean (x = y))" "funspace D boolset"]
    by auto
  also
  have "... = iden D · A · B"
    unfolding iden_def one_elem_fun_def ..
  finally
  show ?thesis
    by auto
qed

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"
  using assms using boolean_def using true_neq_false by auto

lemma apply_if_pair_in:
  assumes "(a1,: a2) ∈: f"
  assumes "f ∈: funspace s t"
  shows "f · a1 = a2"
  using assms
  by (smt abstract_def apply_abstract mem_product pair_inj set_theory.in_funspace_abstract
      set_theory.mem_sub set_theory_axioms)
```

```
lemma funspace_app_unique:
  assumes "f ∈: funspace s t"
  assumes "(a1,: a2) ∈: f"
  assumes "(a1,: a3) ∈: f"
  shows "a3 = a2"
  using assms apply_if_pair_in by blast

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 -
  have "⋀a. a ∈: f ⟹ a ∈: g"
  proof -
    fix a
    assume af: "a ∈: f"
    from af have "∃a1 a2. a1 ∈:s ∧ a2 ∈: t ∧ (a1 ,: a2) = a"
      using assms unfolding funspace_def apply_def using relspace_def by force
    then obtain a1 a2 where a12:
      "a1 ∈:s ∧ a2 ∈: t ∧ (a1 ,: a2) = a"
      by blast
    then have "∃a3. a2 ∈: t ∧ (a1 ,: a3) ∈: g"
      using assms(2) funspace_def by auto
    then obtain a3 where a3: "a2 ∈: t ∧ (a1 ,: a3) ∈: g"
      by auto
    then have "a3 = a2"
      using a12 af assms(1,2,3) apply_if_pair_in by auto
    then show "a ∈: g"
      using a12 a3 by blast
  qed
  moreover
  have "⋀a. a ∈: g ⟹ a ∈: f"
  proof -
    fix a
    assume ag: "a ∈: g"
    then have "∃a1 a2. a1 ∈:s ∧ a2 ∈: t ∧ (a1 ,: a2) = a"
      using assms unfolding funspace_def apply_def using relspace_def by force
    then obtain a1 a2 where a12:
      "a1 ∈:s ∧ a2 ∈: t ∧ (a1 ,: a2) = a"
      by blast
    then have "∃a3. a2 ∈: t ∧ (a1 ,: a3) ∈: f"
      using assms(1) funspace_def by auto
    then obtain a3 where a3: "a2 ∈: t ∧ (a1 ,: a3) ∈: f"
      by auto
    then have "a3 = a2"
      using a12 ag assms(1,2,3) apply_if_pair_in by auto
    then show "a ∈: f"
      using a3 a12 by blast
  qed
  ultimately
  show ?thesis
    using iffD2[OF extensional] by metis
qed

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"
  using assms(1,2,3) funspace_extensional by blast
```

```
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
  using Ext Sep Power subset_def Sum Upair by unfold_locales auto

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}"
  apply unfold_locales
  subgoal for x y
    apply blast
    done
  subgoal for x P a
    apply (rule iffI)
    subgoal
      using mem_Collect_eq set_of_elts ZFC_in_HOL.set_def subsetI small_iff smaller_than_small
      apply smt
      done
    subgoal
      using elts_0 elts_of_set empty_iff f_inv_into_f mem_Collect_eq set_of_elts small_def
        small_elts subset_eq subset_iff_less_eq_V zero_V_def
      apply (smt ZFC_in_HOL.set_def down subsetI)
      done
    done
  subgoal for x a
    apply blast
    done
  subgoal for x a
    apply (metis (mono_tags) UN_iff elts_Sup small_elts tfl_some)
    done
  subgoal for x y a
    apply (metis ZFC_in_HOL.set_def elts_of_set insert_iff singletonD small_upair)
    done
  done

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"
```

```
  by auto
```

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"
  by (induction A) auto

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"
  by (induction rule: wff.induct) auto

lemma wff_Var'[simp, code]:
  "wff β (Var x α) ⟷ β = α"
  using wff.cases wff_Var by auto

lemma wff_Cst'[simp, code]:
  "wff β (Cst c α) ⟷ β = α"
  using wff.cases wff_Cst by auto

lemma wff_App'[simp]:
  "wff α (A · B) ⟷ (∃β. wff (α ⇐ β) A ∧ wff β B)"
proof
  assume "wff α (A · B)"
  then show "∃β. wff (α ⇐ β) A ∧ wff β B"
```

```
    using wff.cases by fastforce
next
  assume "∃β. wff (α ⇐ β) A ∧ wff β B"
  then show "wff α (A · B)"
    using wff_App by auto
qed

lemma wff_Abs'[simp]:
  "wff γ ([λx:α. A]) ⟷ (∃β. wff β A ∧ γ = β ⇐ α)"
proof
  assume "wff γ [λx:α. A]"
  then show "∃β. wff β A ∧ γ = β ⇐ α"
    using wff.cases by blast
next
  assume "∃β. wff β A ∧ γ = β ⇐ α"
  then show "wff γ [λx:α. A]"
    using wff_Abs by auto
qed

lemma wff_Abs_type_of[code]:
  "wff γ [λx:α. A] ⟷ (wff (type_of A) A ∧ γ = (type_of A) ⇐ α)"
proof
  assume "wff γ [λx:α. A]"
  then show "wff (type_of A) A ∧ γ = (type_of A) ⇐ α"
    using wff.cases by auto
next
  assume "wff (type_of A) A ∧ γ = (type_of A) ⇐ α"
  then show "wff γ [λx:α. A]"
    using wff_Abs by auto
qed

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
  assume "wff γ (A · B)"
  then show "wff (type_of A) A ∧ wff (type_of B) B ∧ type_of A = γ ⇐ (type_of B)"
    by auto
next
  assume "wff (type_of A) A ∧ wff (type_of B) B ∧ type_of A = γ ⇐ (type_of B)"
  then show "wff γ (A · B)"
    by (metis wff_App')
qed

lemma unique_type:
  "wff β A ⟹ wff α A ⟹ α = β"
proof (induction arbitrary: α rule: wff.induct)
  case (wff_Var α' y)
  then show ?case
    by simp
next
  case (wff_Cst α' c)
  then show ?case
    by simp
next
  case (wff_App α' β A B)
  then show ?case
    using wff_App' by blast
next
  case (wff_Abs β A α x)
  then show ?case
    using wff_Abs_type_of by blast
qed
```

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"
  using assms
proof (induction arbitrary: α β rule: replacement.induct)
  case (replace A B)
  then show ?case
    using unique_type by auto
next
  case (replace_App_left A B C E D)
  then have "∃β'. wff (β ⇐ β') C"
    by auto
  then obtain β' where wff_C: "wff (β ⇐ β') C"
    by auto
  then have e: "wff (β ⇐ β') E"
    using replace_App_left by auto
  define α' where "α' = β ⇐ β'"
  have "wff β' D"
    using wff_C unique_type replace_App_left.prems(3) by auto
  then have "wff β (E · D)"
    using e by auto
  then show ?case
    by auto
next
  case (replace_App_right A B D E C)
  have "∃β'. wff (β ⇐ β') C"
    using replace_App_right.prems(3) by auto
  then obtain β' where wff_C: "wff (β ⇐ β') C"
    by auto
  have wff_E: "wff β' E"
    using wff_C unique_type replace_App_right by fastforce
  define α' where α': "α' = β ⇐ β'"
  have "wff β (C · E)"
    using wff_C wff_E by auto
  then show ?case
    using α' by auto
next
  case (replace_Abs A B C D x α')
  then have "∃β'. wff β' D"
    by auto
  then obtain β' where wff_D: "wff β' D"
    by auto
  have β: "β = β' ⇐ α'"
    using wff_D unique_type replace_Abs by auto
  have "wff (β' ⇐ α') ([λx:α'. D])"
    using wff_D by auto
  then show ?case
    using β by auto
qed

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"
  using assms
proof (induction arbitrary: α β rule: replacement.induct)
  case (replace A B)
  then show ?case
    using unique_type by auto
next
  case (replace_App_left A B C E D)
  then obtain γ where γ: "wff (β ⇐ γ) E ∧ wff γ D"
    by auto
  then have "wff (β ⇐ γ) C"
    using replace_App_left by auto
  then show ?case
    using γ by auto
next
  case (replace_App_right A B D E C)
  then obtain γ where γ: "wff (β ⇐ γ) C ∧ wff γ E"
    by auto
  then have "wff γ D"
    using replace_App_right by auto
  then show ?case
    using γ by auto
next
  case (replace_Abs A B C D x α')
  then obtain γ where "wff γ D"
    by auto
  then show ?case
    using unique_type replace_Abs by auto
qed
```

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 ιι :: "trm" ("ι") where
  "ι ≡ 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"
  unfolding LHS_def Eql_def by auto

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

lemma RHS_def2[simp]: "RHS ([A =α= B]) = B"
  unfolding RHS_def Eql_def by auto

lemma wff_Eql[simp]:
  "wff α A ⟹ wff α B ⟹ wff Tv [A =α= B]"
  unfolding Eql_def by force

lemma wff_Eql_iff[simp]:
  "wff β [A =α= B] ⟷ wff α A ∧ wff α B ∧ β = Tv"
  using Eql_def by auto
```

8.5 Truth

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

lemma wff_T[simp]: "wff Tv T"
  unfolding T_def by auto

lemma wff_T_iff[simp]: "wff α T ⟷ α = Tv"
  using unique_type wff_T by blast
```

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"
  by auto

lemma wff_F_iff[simp]: "wff α F ⟷ α = Tv"
  using unique_type wff_F by blast
```

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)"
  unfolding PI_def by auto

lemma wff_PI_subterm[simp]: "wff (Tv ⇐ α) [λ ''x'':α. T]"
  by auto
```

```
lemma wff_PI_subterm_iff[simp]:
  "wff β [λ ''x'':α. T] ⟷ β = (Tv ⇐ α)"
  by auto
```

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]"
  unfolding Forall_def by force

lemma wff_Forall_iff[simp]: "wff β [∀x:α. A] ⟷ wff Tv A ∧ β = Tv"
proof
  assume "wff β [∀x:α. A]"
  then show "wff Tv A ∧ β = Tv"
    by (smt Forall_def unique_type type_sym.inject wff_Abs' wff_App' wff_PI)
next
  assume "wff Tv A ∧ β = Tv"
  then show "wff β [∀x:α. A]"
    unfolding Forall_def by force
qed
```

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"
  unfolding Con_sym_def by auto

lemma wff_Con_sym'[simp]: "wff α Con_sym ⟷ α = ooo"
  unfolding Con_sym_def by auto


lemma wff_Con_sym_subterm0[simp]:
  "wff Tv A ⟹ wff Tv B ⟹ wff (Tv ⇐ ooo) [λ''g'':ooo. g_ooo · A · B]"
  by force

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
  assume wff: "wff β [λ''g'':ooo. g_ooo · A · B]"
  then have "wff Tv A"
    by auto
  moreover
  from wff have "wff Tv B"
    by auto
  moreover
  from wff have "β = Tv ⇐ ooo"
    by auto
  ultimately show "wff Tv A ∧ wff Tv B ∧ β = Tv ⇐ ooo"
    by auto
next
  assume "wff Tv A ∧ wff Tv B ∧ β = Tv ⇐ ooo"
  then show "wff β [λ''g'':ooo. g_ooo · A · B]"
    by force
qed

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]]"
```

by auto

lemma wff_Con_sym_subterm1_iff[simp]:
"wff α [[$\boldsymbol{\lambda}$''g'':ooo. g_{ooo} · T · T] =(Tv $\Leftarrow$ ooo)= [$\boldsymbol{\lambda}$''g'':ooo. g_{ooo} · x_o · y_o]] $\longleftrightarrow$ α = Tv"
using unique_type wff_Con_sym_subterm1 **by** blast

lemma wff_Con_sym_subterm2[simp]:
"wff oo [$\boldsymbol{\lambda}$ ''y'':Tv. [[$\boldsymbol{\lambda}$''g'':ooo. g_{ooo} · T · T] =(Tv $\Leftarrow$ ooo)= [$\boldsymbol{\lambda}$''g'':ooo. g_{ooo} · x_o · y_o]]]"
by auto

lemma wff_Con_sym_subterm2_iff[simp]:
"wff α [$\boldsymbol{\lambda}$ ''y'':Tv. [[$\boldsymbol{\lambda}$''g'':ooo. g_{ooo} · T · T] =(Tv $\Leftarrow$ ooo)= [$\boldsymbol{\lambda}$''g'':ooo. g_{ooo} · x_o · y_o]]] $\longleftrightarrow$ α = oo"
by auto

8.10 Conjunction

definition Con :: "trm $\Rightarrow$ trm $\Rightarrow$ trm" (**infix** "$\wedge$" 80) **where**
"A $\wedge$ B = Con_sym · A · B"

lemma wff_Con[simp]: "wff Tv A $\Longrightarrow$ wff Tv B $\Longrightarrow$ wff Tv (A $\wedge$ B)"
unfolding Con_def **by** auto

lemma wff_Con_iff[simp]: "wff α (A $\wedge$ B) $\longleftrightarrow$ wff Tv A $\wedge$ wff Tv B $\wedge$ α = Tv"
unfolding Con_def **by** auto

8.11 Implication symbol

definition Imp_sym :: trm **where**
"Imp_sym $\equiv$ [$\boldsymbol{\lambda}$''x'':Tv. [$\boldsymbol{\lambda}$''y'':Tv. [x_o =Tv= (x_o $\wedge$ y_o)]]]"

lemma wff_Imp_sym[simp]:
"wff ooo Imp_sym"
unfolding Imp_sym_def **by** auto

lemma wff_Imp_sym_iff[simp]:
"wff α Imp_sym $\longleftrightarrow$ α = ooo"
unfolding Imp_sym_def **by** auto

lemma wff_Imp_sym_subterm0[simp]:
"wff Tv (x_o $\wedge$ y_o)"
by auto

lemma wff_Imp_sym_subterm0_iff[simp]:
"wff α (x_o $\wedge$ y_o) $\longleftrightarrow$ α = Tv"
by auto

lemma wff_Imp_sym_subterm1[simp]:
"wff Tv [x_o =Tv= (x_o $\wedge$ y_o)]"
by auto

lemma wff_Imp_sym_subterm1_iff[simp]:
"wff α [x_o =Tv= (x_o $\wedge$ y_o)] $\longleftrightarrow$ α = Tv"
using unique_type wff_Imp_sym_subterm1 **by** blast

lemma wff_Imp_sym_subterm2[simp]:
"wff oo [$\boldsymbol{\lambda}$ ''y'':Tv. [x_o =Tv= (x_o $\wedge$ y_o)]]"
by auto

lemma wff_Imp_sym_subterm2_iff[simp]:
"wff α [$\boldsymbol{\lambda}$ ''y'':Tv. [x_o =Tv= (x_o $\wedge$ y_o)]] $\longleftrightarrow$ α = oo"
by auto

8.12 Implication

definition Imp :: "trm $\Rightarrow$ trm $\Rightarrow$ trm" (**infix** "$\longrightarrow$" 80) **where**

```
  "A ⟶ B = Imp_sym · A · B"

lemma wff_Imp[simp]: "wff Tv A ⟹ wff Tv B ⟹ wff Tv (A ⟶ B)"
  unfolding Imp_def by auto

lemma wff_Imp_iff[simp]: "wff α (A ⟶ B) ⟷ wff Tv A ∧ wff Tv B ∧ α = Tv"
  using Imp_def by auto
```

9 The Q0 proof system

```
definition axiom_1 :: trm where
  "axiom_1 ≡ [((g_oo · T) ∧ (g_oo · F)) =Tv= [∀''x'':Tv. g_oo · x_o]]"

lemma wff_axiom_1[simp]: "wff Tv axiom_1"
  unfolding axiom_1_def by auto

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=
       [∀''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 ≡ (∃c. B = Cst c β) ∨ (∃y. B = Var y β ∧ (x ≠ y ∨ α ≠ 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 ≠ y ∨ α ≠ γ) ∧ (y, γ) ∉ 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)"
  unfolding agree_off_asg_def by force

lemma agree_off_asg_disagree_var_sym[simp]:
  "agree_off_asg ψ φ x α ⟹ x ≠ y ⟹ ψ(y,β) = φ(y,β)"
  unfolding agree_off_asg_def by auto

lemma agree_off_asg_disagree_type_sym[simp]:
```

```
  "agree_off_asg ψ φ x α ⟹ α ≠ β ⟹ ψ(y,β) = φ(y,β)"
  unfolding agree_off_asg_def by auto


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"
  using assms unfolding asg_into_frame_def by auto

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"
  using assms asg_into_frame_fun_upd by auto

lemma standard_model_is_general_model:
  assumes "standard_model D I"
  shows "general_model D I"
proof -
  have "wf_interp D I"
    using assms by auto
  moreover
  have "wff α A ⟹ asg_into_interp φ D I ⟹ val D I φ A ∈: D α" for φ α A
  proof (induction arbitrary: φ rule: wff.induct)
    case (wff_Var α uu)
    then show ?case
```

```
      unfolding asg_into_frame_def using assms by auto
  next
    case (wff_Cst α uv)
    then show ?case
      using assms using wf_interp.simps by auto
  next
    case (wff_App α β A B)
    then show ?case
      using apply_in_rng assms by fastforce
  next
    case (wff_Abs β A α x)
    then show ?case
      using assms abstract_in_funspace asg_into_frame_fun_upd by force
  qed
  ultimately
  have "general_model D I"
    unfolding general_model.simps by auto
  then show "general_model D I"
    by auto
qed

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"
  using assms(4) assms(1-3,5)
proof (induction arbitrary: φ ψ rule: wff.induct)
  case (wff_Var α x)
  then show ?case by auto
next
  case (wff_Cst α c)
  then show ?case by auto
next
  case (wff_App α β A B)
  show ?case
    using wff_App(1,2,5,6,7,8) wff_App(3,4)[of φ ψ] by auto
next
  case (wff_Abs β A α x)
  have "abstract (D α) (D β) (λz. val D I (φ((x, α) := z)) A) =
        abstract (D α) (D β) (λz. val D I (ψ((x, α) := z)) A)"
  proof (rule abstract_extensional, rule, rule)
    fix xa
    assume "xa ∈: D α"
    then have "val D I (φ((x, α) := xa)) A ∈: D β"
      using wff_Abs asg_into_frame_fun_upd by auto
    moreover
    {
      have "asg_into_frame (ψ((x, α) := xa)) D"
        using ‹xa ∈: D α› asg_into_frame_fun_upd wff_Abs by auto
      moreover
      have "asg_into_frame (φ((x, α) := xa)) D"
        using ‹xa ∈: D α› asg_into_frame_fun_upd wff_Abs by auto
      moreover
      have "(∀y∈frees A. (φ((x, α) := xa)) y = (ψ((x, α) := xa)) y)"
        using wff_Abs by auto
      ultimately
      have "val D I (φ((x, α) := xa)) A = val D I (ψ((x, α) := xa)) A"
```

```
      using assms wff_Abs by (smt case_prodI2)
    }
    ultimately
    show "val D I (φ((x, α) := xa)) A ∈: D β ∧ val D I (φ((x, α) := xa)) A = val D I (ψ((x, α) := xa)) A"
      by auto
  qed
  then show ?case
    using wff_Abs by auto
qed


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 -
  have val_A: "val D I φ A ∈: D α"
    using assms  by simp
  have "asg_into_interp (φ((x, α) := val D I φ A)) D I"
    using assms asg_into_frame_fun_upd  by force
  then have val_B: "val D I (φ((x, α) := val D I φ A)) B ∈: D β"
    using assms by simp

  have "val D I φ ([λx:α. B] · A) =
          (abstract (D α) (D β) (λz. val D I (φ((x, α) := z)) B)) · val D I φ A"
    using assms by auto
  also
  have "... = val D I (φ((x, α) := val D I φ A)) B"
    using apply_abstract val_A val_B by auto
  finally
  show ?thesis
    by auto
qed


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 -
  have "val D I φ ([A =α= B]) = (I ''Q'' (Tv ⇐ α ⇐ α)) · val D I φ A · val D I φ B"
    unfolding Eql_def by auto
  have "... = (iden (D α)) · val D I φ A · val D I φ B"
    using assms general_model.simps wf_interp.simps by auto
  also
  have "... = boolean (val D I φ A = val D I φ B)"
    using apply_id using assms general_model.simps by blast
  finally show ?thesis
```

```
    unfolding Eql_def by simp
qed


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"
  using lemma_5401_b_variant_1[OF assms] boolean_eq_true by auto


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"
  using assms(5) lemma_5401_b[OF assms(1,2,3,4)] by auto


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 ≠ val D I φ B"
  shows "val D I φ ([A =α= B]) = false"
  using assms(5) lemma_5401_b_variant_1[OF assms(1,2,3,4)] by (simp add: boolean_def)


lemma lemma_5401_c:
  assumes "general_model D I"
  assumes "asg_into_interp φ D I"
  shows "val D I φ T = true"
  using assms lemma_5401_b[OF assms(1,2)] unfolding T_def by auto


lemma lemma_5401_d:
  assumes "general_model D I"
  assumes "asg_into_interp φ D I"
  shows "val D I φ F = false"
proof -
  have "iden boolset ∈: D ooo"
    using assms general_model.simps wf_interp.simps wf_frame_def by metis
  then have "(val D I φ [λ''x'':Tv. T]) · false ≠ (val D I φ [λ''x'':Tv. x_o]) · false"
    using assms wf_interp.simps wf_frame_def true_neq_false
      apply_id[of "iden boolset" "(D ooo)" "iden boolset"]
    unfolding boolean_def Eql_def T_def by auto
  then have neqLR: "val D I φ [λ''x'':Tv. T] ≠ val D I φ [λ''x'':Tv. x_o]"
    by metis
  have "val D I φ F = boolean (val D I φ ([λ''x'':Tv. T]) = val D I φ [λ''x'':Tv. x_o])"
    using lemma_5401_b_variant_1[OF assms(1,2),
        of "oo" "([λ''x'':Tv. T])" "[λ''x'':Tv. x_o]"] assms
    by auto
  also
  have "... = boolean False"
    using neqLR by auto
  also
  have "... = false"
    unfolding boolean_def by auto
  finally
  show ?thesis
    by auto
qed
```

```
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"
  using asg_into_interp_fun_upd[OF assms wff_T, of x] lemma_5401_c[OF assms(1,2)] by auto

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"
  using asg_into_interp_fun_upd[OF assms wff_F, of x] lemma_5401_d[OF assms] by auto


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 -
  define φ' where "φ' ≡ φ((''x'',Tv) := val D I φ T)"
  define φ'' where "φ'' ≡ φ'((''y'',Tv) :=  val D I φ' T)"
  define φ''' where "φ''' ≡ λz. φ''((''g'', ooo) := z)"
  have φ'_asg_into: "asg_into_interp φ' D I"
    unfolding φ'_def using asg_into_interp_fun_upd[OF assms wff_T] by blast
  have φ''_asg_into: "asg_into_interp φ'' D I"
    unfolding φ''_def using asg_into_interp_fun_upd[OF assms(1) φ'_asg_into wff_T] by blast

  have "(val D I φ Con_sym) · true · true = val D I φ (Con_sym · T · T)"
    using lemma_5401_c[OF assms(1,2)] by auto
  also
  have "... = val D I φ ([λ''x'':Tv. [λ''y'':Tv. [[λ''g'':ooo. g_ooo · T · T] =(Tv ⇐ ooo)= [λ''g'':ooo. g_ooo
· x_o · y_o]]]] · T · T)"
    unfolding Con_sym_def by auto
  also
  have "... = (val D I φ (([λ''x'':Tv. [λ''y'':Tv. [[λ''g'':ooo. g_ooo · T · T] =(Tv ⇐ ooo)= [λ''g'':ooo. g_ooo
· x_o · y_o]]]] · T))) · val D I φ T"
    by simp
  also
  have "... = (val D I (φ((''x'',Tv) := val D I φ T)) (([λ''y'':Tv. [[λ''g'':ooo. g_ooo · T · T] =(Tv ⇐ ooo)=
[λ''g'':ooo. g_ooo · x_o · y_o]]]))) · val D I φ T"
    by (metis lemma_5401_a[OF assms(1,2)] wff_Con_sym_subterm2 wff_T)
  also
  have "... = (val D I φ' (([λ''y'':Tv. [[λ''g'':ooo. g_ooo · T · T] =(Tv ⇐ ooo)= [λ''g'':ooo. g_ooo · x_o · y_o]]])))
· val D I φ T"
    unfolding φ'_def ..
  also
  have "... = (val D I φ' (([λ''y'':Tv. [[λ''g'':ooo. g_ooo · T · T] =(Tv ⇐ ooo)= [λ''g'':ooo. g_ooo · x_o · y_o]]])))
· val D I φ' T"
    using φ'_asg_into assms(2) lemma_5401_c[OF assms(1)] by auto
  also
  have "... = (val D I φ' ([λ''y'':Tv. [[λ''g'':ooo. g_ooo · T · T] =(Tv ⇐ ooo)= [λ''g'':ooo. g_ooo · x_o · y_o]]]
· T))"
    by simp
  also
  have "... = (val D I (φ'((''y'',Tv) :=  val D I φ' T)) ([[λ''g'':ooo. g_ooo · T · T] =(Tv ⇐ ooo)= [λ''g'':ooo.
g_ooo · x_o · y_o]]))"
    by (meson φ'_asg_into assms(1) lemma_5401_a[OF assms(1)] wff_Con_sym_subterm1 wff_T)
  also
  have "... = (val D I φ'' ([[λ''g'':ooo. g_ooo · T · T] =(Tv ⇐ ooo)= [λ''g'':ooo. g_ooo · x_o · y_o]]))"
    unfolding φ''_def ..
  also
  have "... = true"
  proof (rule lemma_5401_b_variant_2[OF assms(1)])
    show "wff (Tv ⇐ ooo) [λ''g'':ooo. g_ooo · T · T]"
```

```
    by auto
next
  show "wff (Tv ⇐ ooo) [λ''g'':ooo. g_ooo · x_o · y_o]"
    by auto
next
  show "asg_into_frame φ'' D"
    using φ''_asg_into by blast
next
  have "val D I φ'' [λ''g'':ooo. g_ooo · T · T] = abstract (D ooo) (D Tv)
              (λz. val D I (φ''((''g'', ooo) := z))
                 (g_ooo · T · T))"
    by (simp only: val.simps(4) type_of.simps type_sym.case)
  also
  have "... = abstract (D ooo) (D Tv)
              (λz. val D I (φ''' z) (g_ooo · T · T))"
    unfolding φ'''_def ..
  also
  have "... = abstract (D ooo) (D Tv)
              (λz. val D I (φ''' z) g_ooo · val D I (φ''' z) T
                 · val D I (φ''' z) T)"
    unfolding val.simps(3) ..
  also
  have "... = abstract (D ooo) (D Tv)
              (λz. val D I (φ''' z) g_ooo · true · true)"
  proof (rule abstract_extensional')
    fix x
    assume "x ∈: D ooo"
    then have "val D I (φ''' x) (g_ooo · T · T) ∈: D Tv"
      using φ'''_def φ''_asg_into asg_into_frame_fun_upd assms(1)
        general_model.elims(2) type_sym.inject wff_Abs_type_of wff_Con_sym_subterm0 wff_T
      by (metis wff_App wff_Var)
    then show "val D I (φ''' x) g_ooo · val D I (φ''' x) T ·
               val D I (φ''' x) T
             ∈: D Tv"
      by simp
  next
    fix x
    assume "x ∈: D ooo"
    then have "val D I (φ''' x) T = true"
      unfolding φ'''_def using  φ''_asg_into asg_into_frame_fun_upd
        lemma_5401_c[OF assms(1)] by blast
    then show "val D I (φ''' x) g_ooo · val D I (φ''' x) T ·
               val D I (φ''' x) T =
             val D I (φ''' x) g_ooo · true · true" by auto
  qed
  also
  have "... = abstract (D ooo) (D Tv)
              (λz. val D I (φ''' z) g_ooo ·
                     val D I (φ''' z) x_o · val D I (φ''' z) y_o)"
  proof (rule abstract_extensional')
    fix x
    assume x_in_D: "x ∈: D ooo"
    then have "val D I (φ''' x) (g_ooo · T · T) ∈: D Tv"
      using φ'''_def φ''_asg_into asg_into_frame_fun_upd assms(1)
        general_model.elims(2) type_sym.inject wff_Abs_type_of wff_Con_sym_subterm0 wff_T
      by (metis wff_App wff_Var)
    then have "val D I (φ''' x) g_ooo · val D I (φ''' x) T ·
               val D I (φ''' x) T ∈: D Tv"
      by simp
    then show "val D I (φ''' x) g_ooo · true · true ∈: D Tv"
      by (metis φ'''_def φ''_asg_into lemma_5401_c[OF assms(1)] asg_into_frame_fun_upd x_in_D)
  next
    fix x
    assume x_in_D: "x ∈: D ooo"
```

```
      then have "val D I (φ''' x) x_o = true"
        unfolding φ'''_def φ''_def φ'_def using lemma_5401_c[OF assms(1,2)] by auto
      moreover
      from x_in_D have "val D I (φ''' x) y_o = true"
        unfolding φ'''_def φ''_def using φ'_asg_into lemma_5401_c[OF assms(1)] by auto
      ultimately
      show "val D I (φ''' x) g_ooo · true · true =
        val D I (φ''' x)
          g_ooo ·
            val D I (φ''' x) x_o · val D I (φ''' x) y_o"
        by auto
    qed
    also
    have "... = abstract (D ooo) (D Tv) (λz. val D I (φ''' z)
               (g_ooo · x_o · y_o))"
      unfolding val.simps(3) ..
    also
    have "... = abstract (D ooo) (D Tv)
               (λz. val D I (φ''((''g'', ooo) := z))
                      (g_ooo · x_o · y_o))"
      unfolding φ'''_def ..
    also
    have "... = val D I φ'' [λ''g'':ooo.
                             g_ooo · x_o · y_o]"
      by (simp only: val.simps(4) type_of.simps type_sym.case)
    finally
    show "val D I φ'' [λ''g'':ooo. g_ooo · T · T] = val D I φ'' [λ''g'':ooo. g_ooo · x_o · y_o]"
      .
  qed
  finally show ?thesis .
qed


lemma lemma_5401_e_2:
  assumes "general_model D I"
  assumes "asg_into_interp φ D I"
  assumes "y = true ∨ y = false"
  shows "(val D I φ Con_sym) · false · y = false"
proof -
  define give_x :: trm where "give_x = [λ''y'':Tv. x_o]"
  define give_fst :: trm where "give_fst = [λ ''x'':Tv. give_x]"
  define val_give_fst :: 's where "val_give_fst = val D I φ give_fst"
  have wff_give_x: "wff oo give_x"
    unfolding give_x_def by auto

  have "⋀a b. a ∈: D Tv ⟹
             b ∈: D Tv ⟹
             val D I (φ((''x'', Tv) := a)) give_x ∈: D (type_of give_x)"
    using wff_give_x asg_into_frame_def assms(1,2) by auto
  moreover
  have "⋀a b. a ∈: D Tv ⟹ b ∈: D Tv ⟹ val D I (φ((''x'', Tv) := a)) give_x · b = a"
    unfolding give_x_def by auto
  ultimately
  have "⋀a b. a ∈: D Tv ⟹
             b ∈: D Tv ⟹
             abstract (D Tv) (D (type_of give_x)) (λz. val D I (φ((''x'', Tv) := z)) give_x) · a · b
             = a"
    by auto
  then have val_give_fst_simp: "⋀a b. a ∈: D Tv ⟹ b ∈: D Tv ⟹ val_give_fst · a · b = a"
    unfolding val_give_fst_def give_fst_def by auto

  have wff_give_fst: "wff ooo give_fst"
    unfolding give_fst_def give_x_def by auto
  then have val_give_fst_fun: "val_give_fst ∈: D ooo"
```

```
    unfolding val_give_fst_def using assms by auto

  have "val D I (φ((''x'', Tv) := false,
                (''y'', Tv) := y,
                (''g'', ooo) := val_give_fst)) T ∈: D Tv"
    by (smt Pair_inject wff_give_fst assms(1,2,3) fun_upd_twist general_model.simps
        asg_into_interp_fun_upd[OF assms(1,2)] asg_into_interp_fun_upd_true[OF assms(1)]
        asg_into_interp_fun_upd_false[OF assms(1)] type_sym.distinct(5) val_give_fst_def wff_T)
  then have val_give_fst_D:
    "val_give_fst · val D I (φ((''x'', Tv) := false,
                              (''y'', Tv) := y,
                              (''g'', ooo) := val_give_fst)) T ·
                    val D I (φ((''x'', Tv) := false,
                              (''y'', Tv) := y,
                              (''g'', ooo) := val_give_fst)) T
      ∈: D Tv"
    using val_give_fst_simp[of
        "val D I (φ((''x'', Tv) := false,
                  (''y'', Tv) := y,
                  (''g'', ooo) := val_give_fst)) T"
        "val D I (φ((''x'', Tv) := false,
                  (''y'', Tv) := y,
                  (''g'', ooo) := val_give_fst)) T" ]
    by auto

  have false_y_TV: "false ∈: D Tv ∧ y ∈: D Tv"
    using assms(1) assms(3) wf_frame_def wf_interp.simps by auto
  then have val_give_fst_in_D: "val_give_fst · false · y ∈: D Tv"
    using val_give_fst_simp by auto

  have "true ∈: D Tv"
    by (metis assms(1) assms(2) general_model.simps lemma_5401_c[OF assms(1,2)] wff_T)
  from this val_give_fst_in_D false_y_TV have "val_give_fst · true · true ≠ val_give_fst · false · y"
    using val_give_fst_simp true_neq_false by auto
  then have val_give_fst_not_false:
    "val_give_fst · val D I (φ((''x'', Tv) := false,
                            (''y'', Tv) := y,
                            (''g'', ooo) := val_give_fst)) T
                 · val D I (φ((''x'', Tv) := false,
                            (''y'', Tv) := y,
                            (''g'', ooo) := val_give_fst)) T
     ≠ val_give_fst · false · y"
    using asg_into_frame_fun_upd assms(1) assms(2) lemma_5401_c false_y_TV val_give_fst_fun by auto
  have Con_sym_subterm0TT_neq_Con_sym_subterm0xy:
    "val D I (φ((''x'', Tv) := false, (''y'', Tv) := y)) [λ''g'':ooo. g_ooo · T · T] ≠
     val D I (φ((''x'', Tv) := false, (''y'', Tv) := y)) [λ''g'':ooo. g_ooo · x_o · y_o]"
    using abstract_cong_specific[of
        val_give_fst
        "(D ooo)"
        "(λz. z · val D I (φ((''x'', Tv) := false,
                          (''y'', Tv) := y,
                          (''g'', ooo) := z)) T
               · val D I (φ((''x'', Tv) := false,
                          (''y'', Tv) := y,
                          (''g'', ooo) := z)) T)"
        "(D Tv)"
        "(λz. z · false · y)"]
    using val_give_fst_fun val_give_fst_D val_give_fst_in_D val_give_fst_not_false by auto

  have "asg_into_frame (φ((''x'', Tv) := false, (''y'', Tv) := y)) D"
    using asg_into_interp_fun_upd_false[OF assms(1)] general_model.simps[of D I] assms wff_Con_sym_subterm1
      asg_into_interp_fun_upd_true[OF assms(1)] by auto
  then have val_Con_sym_subterm1: "val D I (φ((''x'', Tv) := false, (''y'', Tv) := y)) [[λ''g'':ooo. g_ooo · T
· T] =(Tv ⇐ ooo)= [λ''g'':ooo. g_ooo · x_o · y_o]] = false"
```

```
    using Con_sym_subterm0TT_neq_Con_sym_subterm0xy lemma_5401_b_variant_3[OF assms(1)]
    by auto

  have "y ∈: D Tv"
    using general_model.simps lemma_5401_d[OF assms(1,2)] wff_F assms
    by (metis lemma_5401_c[OF assms(1,2)] wff_T)
  moreover
  have "val D I (φ((''x'', Tv) := false, (''y'', Tv) := y)) [[λ''g'':ooo. g_ooo · T · T] =(Tv ⇐ ooo)= [λ''g'':ooo.
g_ooo · x_o · y_o]] ∈: D Tv"
    using asg_into_interp_fun_upd_false[OF assms(1)] general_model.simps[of D I] assms wff_Con_sym_subterm1
      asg_into_interp_fun_upd_true[OF assms(1)] by auto
  moreover
  have "val D I (φ((''x'', Tv) := false, (''y'', Tv) := y)) [[λ''g'':ooo. g_ooo · T · T] =(Tv ⇐ ooo)= [λ''g'':ooo.
g_ooo · x_o · y_o]] = false"
    using val_Con_sym_subterm1 by auto
  ultimately
  have val_y: "(val D I (φ((''x'', Tv) := false)) [λ ''y'':Tv. [[λ''g'':ooo. g_ooo · T · T] =(Tv ⇐ ooo)= [λ''g'':ooo.
g_ooo · x_o · y_o]]]) · y = false"
    by simp

  have 11: "val D I (φ((''x'', Tv) := false)) [λ ''y'':Tv. [[λ''g'':ooo. g_ooo · T · T] =(Tv ⇐ ooo)= [λ''g'':ooo.
g_ooo · x_o · y_o]]] ∈: D oo"
    using asg_into_interp_fun_upd_false[OF assms(1,2)] general_model.simps[of D I] assms
      wff_Con_sym_subterm2 by blast
  moreover
  have "val D I (φ((''x'', Tv) := false)) [λ ''y'':Tv. [[λ''g'':ooo. g_ooo · T · T] =(Tv ⇐ ooo)= [λ''g'':ooo.
g_ooo · x_o · y_o]]] · y = false"
    using val_y by auto
  ultimately
  have "(val D I φ [λ''x'':Tv. [λ ''y'':Tv. [[λ''g'':ooo. g_ooo · T · T] =(Tv ⇐ ooo)= [λ''g'':ooo. g_ooo · x_o ·
y_o]]]]) · false · y = false"
    using false_y_TV by simp
  then show "(val D I φ Con_sym) · false · y = false"
    unfolding Con_sym_def by auto
qed


lemma lemma_5401_e_3:
  assumes "general_model D I"
  assumes "asg_into_interp φ D I"
  assumes "x = true ∨ x = false"
  shows "(val D I φ Con_sym) · x · false = false"
proof -

  define give_y :: trm where "give_y = ([λ ''y'':Tv. y_o])"
  define give_snd :: trm where "give_snd = [λ ''x'':Tv. give_y]"
  define val_give_snd :: 's where "val_give_snd = val D I φ give_snd"
  have wff_give_y: "wff oo give_y"
    unfolding give_y_def by auto

  have  "⋀a b. a ∈: D Tv ⟹ b ∈: D Tv ⟹ a ∈: D Tv"
    by simp
  moreover
  have "⋀a b. a ∈: D Tv ⟹ b ∈: D Tv ⟹ val D I (φ((''x'', Tv) := a)) give_y ∈: D (type_of give_y)"
    using wff_give_y asg_into_frame_def assms(1) assms(2) by auto
  moreover
  have "⋀a b. a ∈: D Tv ⟹ b ∈: D Tv ⟹ val D I (φ((''x'', Tv) := a)) give_y · b = b"
    unfolding give_y_def by auto
  ultimately
  have val_give_snd_simp: "⋀a b. a ∈: D Tv ⟹ b ∈: D Tv ⟹ val_give_snd · a · b = b"
    unfolding val_give_snd_def give_snd_def by auto

  have wff_give_snd: "wff ooo give_snd"
    unfolding give_snd_def give_y_def by auto
```

```
  then have val_give_snd_in_D: "val_give_snd ∈: D ooo"
    unfolding val_give_snd_def using assms
    by auto

  then have "val D I (φ((''x'', Tv) := x,
                  (''y'', Tv) := false,
                  (''g'', ooo) := val_give_snd)) T ∈: D Tv"
    by (smt Pair_inject wff_give_snd assms(1,2,3)
        fun_upd_twist general_model.simps asg_into_interp_fun_upd[OF assms(1,2)]
        asg_into_interp_fun_upd_false[OF assms(1)] asg_into_interp_fun_upd_true[OF assms(1)]
        type_sym.distinct(5) val_give_snd_def wff_T)
  then have val_give_snd_app_in_D:
    "val_give_snd · val D I (φ((''x'', Tv) := x,
                             (''y'', Tv) := false,
                             (''g'', ooo) := val_give_snd)) T
                 · val D I (φ((''x'', Tv) := x,
                             (''y'', Tv) := false,
                             (''g'', ooo) := val_give_snd)) T
     ∈: D Tv"
    using val_give_snd_simp[of
        "val D I (φ((''x'', Tv) := x,
                   (''y'', Tv) := false,
                   (''g'', ooo) := val_give_snd)) T"
        "val D I (φ((''x'', Tv) := x,
                    (''y'', Tv) := false,
                    (''g'', ooo) := val_give_snd)) T" ]
    by auto

  have false_and_x_in_D: "false ∈: D Tv ∧ x ∈: D Tv"
    using assms(1,3) wf_frame_def wf_interp.simps by auto
  then have val_give_snd_app_x_false_in_D: "val_give_snd · x · false ∈: D Tv"
    using val_give_snd_simp by auto

  have "true ∈: D Tv"
    by (metis assms(1) assms(2) general_model.simps lemma_5401_c[OF assms(1,2)] wff_T)
  then have "val_give_snd · true · true ≠ val_give_snd · x · false"
    using val_give_snd_simp true_neq_false val_give_snd_app_in_D false_and_x_in_D by auto
  then have
    "val_give_snd · val D I (φ((''x'', Tv) := x,
                             (''y'', Tv) := false,
                             (''g'', ooo) := val_give_snd)) T
                 · val D I (φ((''x'', Tv) := x,
                             (''y'', Tv) := false,
                             (''g'', ooo) := val_give_snd)) T ≠
     val_give_snd · x · false"
    using asg_into_frame_fun_upd assms(1) assms(2) lemma_5401_c false_and_x_in_D val_give_snd_in_D
    by auto
  then have "val D I (φ((''x'', Tv) := x, (''y'', Tv) := false)) [λ''g'':ooo. g_ooo · T · T] ≠
        val D I (φ((''x'', Tv) := x, (''y'', Tv) := false))
          [λ''g'':ooo. g_ooo · x_o · y_o]"
    using abstract_cong_specific[of
        val_give_snd
        "(D ooo)"
        "(λz. z · val D I (φ((''x'', Tv) := x, (''y'', Tv) := false, (''g'', ooo) := z))
             T · val D I (φ((''x'', Tv) := x, (''y'', Tv) := false, (''g'', ooo) := z)) T)"
        "(D Tv)"
        "(λz. z · x · false)"
        ]
    using val_give_snd_in_D val_give_snd_app_x_false_in_D val_give_snd_app_in_D by auto
  then have "val D I (φ((''x'', Tv) := x, (''y'', Tv) := false)) [[λ''g'':ooo. g_ooo · T · T] =(Tv ⇐ ooo)= [λ''g'':oo
g_ooo · x_o · y_o]] = false"
    using asg_into_frame_fun_upd assms(1,2) lemma_5401_b_variant_3 false_and_x_in_D by auto
  then have val_Con_sym_subterm2_false: "(val D I (φ((''x'', Tv) := x)) [λ ''y'':Tv. [[λ''g'':ooo. g_ooo · T ·
T] =(Tv ⇐ ooo)= [λ''g'':ooo. g_ooo · x_o · y_o]]]) · false = false"
```

```
    using false_and_x_in_D by simp

  have "x ∈: D Tv"
    by (simp add: false_and_x_in_D)
  moreover
  have "val D I (φ((''x'', Tv) := x)) [λ ''y'':Tv. [[λ''g'':ooo. g_ooo · T · T] =(Tv ⇐ ooo)= [λ''g'':ooo. g_ooo
· x_o · y_o]]] ∈: D oo"
    by (metis assms(1,3) general_model.simps lemma_5401_c[OF assms(1,2)]
        asg_into_interp_fun_upd[OF assms(1,2)] asg_into_interp_fun_upd_false[OF assms(1,2)]
        wff_Con_sym_subterm2 wff_T)
  moreover
  have "val D I (φ((''x'', Tv) := x)) [λ ''y'':Tv. [[λ''g'':ooo. g_ooo · T · T] =(Tv ⇐ ooo)= [λ''g'':ooo. g_ooo
· x_o · y_o]]] · false = false"
    using val_Con_sym_subterm2_false by auto
  ultimately
  have "(val D I φ [λ''x'':Tv. [λ ''y'':Tv. [[λ''g'':ooo. g_ooo · T · T] =(Tv ⇐ ooo)= [λ''g'':ooo. g_ooo · x_o ·
y_o]]]]) · x · false = false"
    by auto
  then show "(val D I φ Con_sym) · x · false = false"
    unfolding Con_sym_def by auto
qed


lemma lemma_5401_e_variant_1:
  assumes "asg_into_interp φ D I"
  assumes "general_model D I"
  assumes "y = true ∨ y = false"
  assumes "x = true ∨ x = false"
  shows "(val D I φ Con_sym) · x · y = boolean (x = true ∧ y = true)"
proof (cases "y = true")
  case True
  note True_outer = this
  then show ?thesis
  proof (cases "x = true")
    case True
    then show ?thesis
      using True_outer assms lemma_5401_e_1 unfolding boolean_def by auto
  next
    case False
    then show ?thesis
      using True_outer assms  lemma_5401_e_2 unfolding boolean_def by auto
  qed
next
  case False
  note False_outer = this
  then show ?thesis
  proof (cases "x = true")
    case True
    then show ?thesis
      using False_outer assms lemma_5401_e_3 unfolding boolean_def by auto
  next
    case False
    then show ?thesis
      using False_outer assms lemma_5401_e_2 unfolding boolean_def by auto
  qed
qed

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 ∨ φ (x, Tv) = false"
proof -
  have "wff Tv (Var x Tv)"
    by auto
```

```
  then have "val D I φ (Var x Tv) ∈: D Tv"
    using assms general_model.simps by blast
  then have "val D I φ (Var x Tv) ∈: boolset"
    using assms unfolding general_model.simps wf_interp.simps wf_frame_def by auto
  then show ?thesis
    using mem_boolset by simp
qed

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 ∨ val D I φ A = false"
proof -
  have "val D I φ A ∈: D Tv"
    using assms by auto
  then have "val D I φ A ∈: boolset"
    using assms unfolding general_model.simps wf_interp.simps wf_frame_def by force
  then show ?thesis
    using mem_boolset by blast
qed


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 ∧ B)) = boolean (satisfies D I φ A ∧ satisfies D I φ B)"
  using assms wff_Tv_is_true_or_false[of φ D I] lemma_5401_e_variant_1 unfolding Con_def by auto


lemma lemma_5401_f_1:
  assumes "general_model D I"
  assumes "asg_into_interp φ D I"
  assumes "y = true ∨ y = false"
  shows "(val D I φ Imp_sym) · false · y = true"
proof -
  define Imp_subterm2 :: trm where
    "Imp_subterm2 ≡ [λ ''y'':Tv. [x_o =Tv= (x_o ∧ y_o)]]"

  have val_Imp_subterm0_false: "val D I (φ((''x'', Tv) := false, (''y'', Tv) := y)) (x_o ∧ y_o) = false"
    using assms asg_into_interp_fun_upd_false[OF assms(1)] asg_into_interp_fun_upd_true[OF assms(1)]
      boolean_def lemma_5401_e_variant_2 by auto

  have "asg_into_frame (φ((''x'', Tv) := false, (''y'', Tv) := y)) D"
    using assms(1, 2, 3) lemma_5401_c[OF assms(1)] asg_into_interp_fun_upd wff_T
      asg_into_interp_fun_upd_false by metis
  then have "val D I (φ((''x'', Tv) := false, (''y'', Tv) := y)) [x_o =Tv= (x_o ∧ y_o)] = true"
    using lemma_5401_b_variant_1[OF assms(1)] val_Imp_subterm0_false unfolding boolean_def by auto
  then have val_Imp_subterm2_true: "(val D I (φ((''x'', Tv) := false)) [λ ''y'':Tv. [x_o =Tv= (x_o ∧ y_o)]]) · y
= true"
    using assms(1,3) wf_frame_def wf_interp.simps by auto

  have "val D I φ [λ ''x'':Tv. [λ ''y'':Tv. [x_o =Tv= (x_o ∧ y_o)]]] · false · y = true"
  proof -
    have "false ∈: D Tv"
      by (metis asg_into_frame_def asg_into_interp_fun_upd_false assms(1) assms(2) fun_upd_same)
    then have "val D I (φ((''x'', Tv) := false)) [λ ''y'':Tv. [x_o =Tv= (x_o ∧ y_o)]] = val D I φ [λ''x'':Tv. [λ
''y'':Tv. [x_o =Tv= (x_o ∧ y_o)]]] · false"
      using asg_into_interp_fun_upd_false assms(1,2) Imp_subterm2_def[symmetric] wff_Imp_sym_subterm2_iff by
force
    then show ?thesis
      by (metis val_Imp_subterm2_true)
```

```
  qed
  then show "(val D I φ Imp_sym) · false · y = true"
    unfolding Imp_sym_def Imp_subterm2_def by auto
qed


lemma lemma_5401_f_2:
  assumes "general_model D I"
  assumes "asg_into_interp φ D I"
  assumes "x = true ∨ x = false"
  shows "(val D I φ Imp_sym) · x · true = true"
proof -
  have asg: "asg_into_frame (φ((''x'', Tv) := x, (''y'', Tv) := true)) D"
    using assms(1,2,3) asg_into_interp_fun_upd_false asg_into_interp_fun_upd_true[OF assms(1)] by blast
  then have "val D I (φ((''x'', Tv) := x, (''y'', Tv) := true)) (xo ∧ yo) = x"
    using lemma_5401_e_variant_2 assms unfolding boolean_def by auto
  then have val_Imp_subterm1_true: "val D I (φ((''x'', Tv) := x, (''y'', Tv) := true)) [xo =Tv= (xo ∧ yo)] =
true"
    using asg lemma_5401_b_variant_1[OF assms(1)] boolean_eq_true  by auto

  have val_Imp_subterm2_true: "val D I (φ((''x'', Tv) := x)) [λ ''y'':Tv. [xo =Tv= (xo ∧ yo)]] · true = true"
    using val_Imp_subterm1_true assms(1) wf_frame_def wf_interp.simps by auto

  have "x ∈: D Tv"
    by (metis asg_into_frame_def assms(1) assms(3) fun_upd_same asg_into_interp_fun_upd_false
        asg_into_interp_fun_upd_true[OF assms(1,2)])
  moreover
  have "val D I (φ((''x'', Tv) := x)) [λ ''y'':Tv. [xo =Tv= (xo ∧ yo)]] ∈: D oo"
    using wff_Imp_sym_subterm2
    by (metis assms(1,2,3) general_model.simps lemma_5401_c[OF assms(1,2)]
        asg_into_interp_fun_upd[OF assms(1,2)] asg_into_interp_fun_upd_false wff_T)
  ultimately
  have "(val D I φ [λ''x'':Tv. [λ ''y'':Tv. [xo =Tv= (xo ∧ yo)]]]) · x · true = true"
    using val_Imp_subterm2_true by auto
  then show "(val D I φ Imp_sym) · x · true = true"
    unfolding Imp_sym_def by auto
qed


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 -
  have asg: "asg_into_frame (φ((''x'', Tv) := true, (''y'', Tv) := false)) D"
    by (meson assms(1,2) asg_into_interp_fun_upd_false asg_into_interp_fun_upd_true)
  moreover
  have "false = true ∨ false = false"
    unfolding boolean_def by auto
  moreover
  have "boolean (true = true ∧ false = true) = false"
    unfolding boolean_def by auto
  ultimately
  have 3: "val D I (φ((''x'', Tv) := true, (''y'', Tv) := false)) (xo ∧ yo) = false"
    using lemma_5401_e_variant_2 assms by auto
  then have Imp_subterm1_false: "val D I (φ((''x'', Tv) := true, (''y'', Tv) := false)) [xo =Tv= (xo ∧ yo)] =
false"
    using subst lemma_5401_b_variant_1[OF assms(1)] asg boolean_def by auto

  have asdff: "wff Tv [xo =Tv= (xo ∧ yo)]"
    by auto

  have false_Tv: "false ∈: D Tv"
    using assms(1) wf_frame_def wf_interp.simps by auto
```

```
  moreover
  have "val D I (φ((''x'', Tv) := true, (''y'', Tv) := false)) [xₒ =Tv= (xₒ ∧ yₒ)] ∈: D Tv"
    by (simp add: Imp_subterm1_false false_Tv)
  moreover
  have "val D I (φ((''x'', Tv) := true, (''y'', Tv) := false)) [xₒ =Tv= (xₒ ∧ yₒ)] = false"
    using Imp_subterm1_false by auto
  ultimately
  have Imp_subterm2_app_false: "val D I (φ((''x'', Tv) := true)) [λ ''y'':Tv. [xₒ =Tv= (xₒ ∧ yₒ)]] · false =
false"
    by auto

  have wff_Imp_sym_subterm2: "wff oo [λ ''y'':Tv. [xₒ =Tv= (xₒ ∧ yₒ)]]"
    by auto

  have "(val D I φ [λ ''x'':Tv. [λ ''y'':Tv. [xₒ =Tv= (xₒ ∧ yₒ)]]]) · true · false = false"
  proof -
    have "true ∈: D Tv"
      by (metis assms(1) assms(2) general_model.simps lemma_5401_c[OF assms(1,2)] wff_T)
    moreover
    have "val D I (φ((''x'', Tv) := true)) [λ ''y'':Tv. [xₒ =Tv= (xₒ ∧ yₒ)]] ∈: D oo"
      using wff_Imp_sym_subterm2
      by (metis assms(1) general_model.simps asg_into_interp_fun_upd_true[OF assms(1,2)])
    moreover
    have "val D I (φ((''x'', Tv) := true)) [λ ''y'':Tv. [xₒ =Tv= (xₒ ∧ yₒ)]] · false = false"
      using Imp_subterm2_app_false by auto
    ultimately
    show ?thesis
      by auto
  qed
  then show "(val D I φ Imp_sym) · true · false = false"
    unfolding Imp_sym_def by auto
qed


lemma lemma_5401_f_variant_1:
  assumes "asg_into_interp φ D I"
  assumes "general_model D I"
  assumes "x = true ∨ x = false"
  assumes "y = true ∨ y = false"
  shows "(val D I φ Imp_sym) · x · y = boolean (x = true ⟶ y = true)"
proof (cases "y = true")
  case True
  note True_outer = this
  then show ?thesis
  proof (cases "x = true")
    case True
    then show ?thesis
      using True_outer assms lemma_5401_f_2 unfolding boolean_def by auto
  next
    case False
    then show ?thesis
      using True_outer assms  lemma_5401_f_2 unfolding boolean_def by auto
  qed
next
  case False
  note False_outer = this
  then show ?thesis
  proof (cases "x = true")
    case True
    then show ?thesis
      using False_outer assms lemma_5401_f_3 unfolding boolean_def by auto
  next
    case False
    then show ?thesis
```

```
      using False_outer assms lemma_5401_f_1 unfolding boolean_def by auto
  qed
qed


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 ⟶ B)) = boolean (satisfies D I φ A ⟶ satisfies D I φ B)"
  using assms unfolding Imp_def
  by (simp add: lemma_5401_f_variant_1 wff_Tv_is_true_or_false)


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 [∀x:α. A] ⟷
        (∀ψ. asg_into_interp ψ D I ⟶ agree_off_asg ψ φ x α ⟶ satisfies D I ψ A)"
proof -
  have "wff (Tv ⇐ α) [λ ''x'':α. T]"
    by auto
  then have PI_subterm_in_D: "val D I φ [λ ''x'':α. T] ∈: D (Tv ⇐ α)"
    using assms(1,2) general_model.simps by blast

  have "wff (Tv ⇐ α) ([λx:α. A])"
    using assms by auto
  moreover
  have "∀φ. asg_into_frame φ D ⟶ (∀A α. wff α A ⟶ val D I φ A ∈: D α)"
    using assms(1) by auto
  then have "∀t cs. val D I φ [λcs:t. A] ∈: D (Tv ⇐ t)"
    using wff_Abs assms(1,2,3) by presburger
  then have "abstract (D α) (D Tv) (λu. val D I (φ((x, α) := u)) A) ∈: D (Tv ⇐ α)"
    using assms(3) by simp
  ultimately
  have val_lambda_A: "val D I φ ([λx:α. A]) ∈: D (Tv ⇐ α)"
    using assms by auto

  have true_and_A_in_D: "∀z. z ∈: D α ⟶ true ∈: D Tv ∧ val D I (φ((x, α) := z)) A ∈: D Tv"
    by (metis assms(1,2,3) general_model.simps lemma_5401_c[OF assms(1,2)] asg_into_frame_fun_upd wff_T)

  have "satisfies D I φ [∀ x: α. A] ⟷ val D I φ [∀x: α. A] = true"
    by auto
  moreover have "... ⟷ val D I φ (PI α) · val D I φ [λx:α. A] = true"
    unfolding Forall_def by simp
  moreover have "... ⟷ I ''Q'' ((Tv ⇐ (Tv ⇐ α)) ⇐ (Tv ⇐ α))
                           · val D I φ [λ ''x'':α. T] · val D I φ [λx:α. A] =
                        true"
    unfolding PI_def by simp
  moreover have "... ⟷ (iden (D (Tv ⇐ α))) · val D I φ [λ ''x'':α. T] · val D I φ [λx:α. A] =
                        true"
    unfolding PI_def using wf_interp.simps assms by simp
  moreover have "... ⟷ val D I φ [λ''x'':α. T] = val D I φ [λx:α. A]"
    using PI_subterm_in_D val_lambda_A apply_id_true by auto
  moreover have "... ⟷ abstract (D α) (D Tv) (λz. val D I (φ((''x'', α) := z)) T) = val D I φ [λx:α. A]"
    using assms wff_T by simp
  moreover have "... ⟷ abstract (D α) (D Tv) (λz. true) = val D I φ [λx:α. A]"
  proof -
    have "∀x. x ∈: D α ⟶ val D I (φ((''x'', α) := x)) T ∈: D Tv ∧ true ∈: D Tv"
      using true_and_A_in_D assms(1,2)  asg_into_frame_fun_upd by auto
    moreover
    have "∀x. x ∈: D α ⟶ val D I (φ((''x'', α) := x)) T ∈: D Tv ∧ satisfies D I (φ((''x'', α) := x)) T"
```

```
      using true_and_A_in_D assms(1) assms(2) lemma_5401_c[OF assms(1)] asg_into_frame_fun_upd by auto
    ultimately
    have "abstract (D α) (D Tv) (λz. val D I (φ((''x'', α) := z)) T) = abstract (D α) (D Tv) (λz. true)"
      using abstract_extensional by auto
    then show ?thesis
      by auto
  qed
  moreover have "... ⟷ abstract (D α) (D Tv) (λz. true) = val D I φ ([λx:α. A])"
    by auto
  moreover have "... ⟷ abstract (D α) (D Tv) (λz. true) =
                        abstract (D α) (D Tv) (λz. val D I (φ((x, α) := z)) A)"
    using assms by simp
  moreover have "... ⟷ (∀z. z ∈: D α ⟶ true ∈: D Tv ∧ true = val D I (φ((x, α) := z)) A)"
  proof -
    have "∀z. z ∈: D α ⟶ true ∈: D Tv ∧ val D I (φ((x, α) := z)) A ∈: D Tv"
      using true_and_A_in_D by auto
    then show ?thesis
      using abstract_iff_extensional by auto
  qed
  moreover have "... ⟷ (∀z. z ∈: D α ⟶ true = val D I (φ((x, α) := z)) A)"
    by (metis assms(1,2) general_model.simps lemma_5401_c[OF assms(1,2)] wff_T)
  moreover have "... ⟷ (∀z. z ∈: D α ⟶ satisfies D I (φ((x, α) := z)) A)"
    by auto
  moreover have "... ⟷ (∀ψ. asg_into_interp ψ D I ⟶ agree_off_asg ψ φ x α ⟶ satisfies D I ψ A)"
  proof -
    show ?thesis
    proof
      assume A_sat: "∀z. z ∈: D α ⟶ satisfies D I (φ((x, α) := z)) A"
      show "∀ψ. asg_into_frame ψ D ⟶ agree_off_asg ψ φ x α ⟶ satisfies D I ψ A"
      proof (rule; rule; rule)
        fix ψ
        assume a1: "asg_into_frame ψ D"
        assume a2: "agree_off_asg ψ φ x α"
        have "∃xa. (φ((x, α) := xa)) = ψ"
          using a1 a2 agree_off_asg_def2 by blast
        then show "satisfies D I ψ A"
          using A_sat a1 a2 by (metis asg_into_frame_def fun_upd_same)
      qed
    next
      assume "∀ψ. asg_into_frame ψ D ⟶ agree_off_asg ψ φ x α ⟶ satisfies D I ψ A"
      then show "∀z. z ∈: D α ⟶ satisfies D I (φ((x, α) := z)) A"
        using asg_into_frame_fun_upd asg_into_interp_fun_upd[OF assms(1,2)] assms(2) fun_upd_other
        unfolding agree_off_asg_def by auto
    qed
  qed
  ultimately show ?thesis
    by auto
qed


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 [∀x:α. A] =
        boolean (∀ψ. asg_into_interp ψ D I ⟶ agree_off_asg ψ φ x α ⟶ satisfies D I ψ A)"
proof -
  have "val D I φ [∀x:α. A] = (if satisfies D I φ [∀x:α. A] then true else false)"
    using assms wff_Forall wff_Tv_is_true_or_false by metis
  then show ?thesis
    using assms lemma_5401_g[symmetric] unfolding boolean_def by auto
qed
```

12 Soundness

```
lemma fun_sym_asg_to_funspace:
  assumes "asg_into_frame φ D"
  assumes "general_model D I"
  shows "φ (f, α ⇐ β) ∈: funspace (D β) (D α)"
proof -
  have "wff (α ⇐ β) (Var f (α ⇐ β))"
    by auto
  then have "val D I φ (Var f (α ⇐ β)) ∈: D (α ⇐ β)"
    using assms
    using general_model.simps by blast
  then show "φ (f, α ⇐ β) ∈: funspace (D β) (D α)"
    using assms unfolding general_model.simps wf_interp.simps wf_frame_def
    by (simp add: subs_def)
qed

lemma fun_sym_interp_to_funspace:
  assumes "asg_into_frame φ D"
  assumes "general_model D I"
  shows "I f (α ⇐ β) ∈: funspace (D β) (D α)"
proof -
  have "wff (α ⇐ β) (Cst f (α ⇐ β))"
    by auto
  then have "val D I φ (Cst f (α ⇐ β)) ∈: D (α ⇐ β)"
    using assms general_model.simps by blast
  then show "I f (α ⇐ β) ∈: funspace (D β) (D α)"
    using assms subs_def unfolding general_model.simps wf_interp.simps wf_frame_def by auto
qed


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'"
  unfolding valid_general_def
proof (rule allI, rule allI, rule impI)
  fix D :: "type_sym ⇒ 's" and I :: "char list ⇒ type_sym ⇒ 's"
  assume DI: "general_model D I"
  then have "valid_in_model D I ([A =α= B])"
    using A_eql_B unfolding valid_general_def by auto
  then have x: "∀φ. asg_into_frame φ D ⟶ (val D I φ A = val D I φ B)"
    unfolding valid_in_model_def using lemma_5401_b[OF DI, of _ α A B ] assms(4,5) by auto
  have r: "replacement A B C C'"
    using assms(3) using Eql_def rule_R.cases by fastforce
  from r have "∀φ. asg_into_frame φ D ⟶ (val D I φ C = val D I φ C')"
    using x assms(4,5,6)
  proof (induction arbitrary: β rule: replacement.induct)
    case (replace A B)
    then show ?case by auto
  next
    case (replace_App_left A B C E D')
    define α' where "α' = type_of C"
    define β' where "β' = type_of D'"
    show ?case
    proof (rule, rule)
      fix φ
      assume asg: "asg_into_frame φ D"
      have α': "α' = β ⇐ β'"
        using trm.distinct(11) trm.distinct(3,7) trm.inject(3) replace_App_left.prems(4) wff.simps
```

```
      by (metis α'_def β'_def wff_App_type_of)
    from asg have "wff α' C"
      using replace_App_left trm.distinct(3,7,11) trm.inject(3) wff.simps
      by (metis α' β'_def type_of wff_App')
    then have "val D I φ C = val D I φ E"
      using asg replace_App_left by auto
    then show "val D I φ (C · D') = val D I φ (E · D')"
      using α' by auto
  qed
next
  case (replace_App_right A B D' E C)
  define α' where "α' = type_of C"
  define β' where "β' = type_of D'"
  show ?case
  proof (rule, rule)
    fix φ
    assume asg: "asg_into_frame φ D"
    have α': "α' = β ⇐ β'"
      using trm.distinct(11) trm.distinct(3) trm.distinct(7) trm.inject(3)
        replace_App_right.prems(4) wff.simps by (metis α'_def β'_def type_of wff_App')
    from asg have "wff β' D'"
      using β'_def replace_App_right.prems(4) by fastforce
    then have "val D I φ D' = val D I φ E"
      using asg replace_App_right by auto
    then show "val D I φ (C · D') = val D I φ (C · E)"
      using α' by auto
  qed
next
  case (replace_Abs A B C D' x α')
  define β' where "β' = type_of C"
  show ?case
  proof (rule, rule)
    fix φ
    assume asg: "asg_into_frame φ D"
    then have val_C_eql_val_D':
      "∀z. z ∈: D α' ⟶ val D I (φ((x, α') := z)) C = val D I (φ((x, α') := z)) D'"
      using asg replace_App_right
      by (metis trm.distinct(11) trm.distinct(5) trm.distinct(9) trm.inject(4)
          asg_into_frame_fun_upd replace_Abs.IH replace_Abs.prems(1) replace_Abs.prems(2)
          replace_Abs.prems(3) replace_Abs.prems(4) wff.cases)

    have val_C_eql_val_D'_type:
      "∀z. z ∈: D α' ⟶
              val D I (φ((x, α') := z)) C ∈: D (type_of C) ∧
                val D I (φ((x, α') := z)) C = val D I (φ((x, α') := z)) D'"
    proof (rule; rule)
      fix z
      assume a2: "z ∈: D α'"
      have "val D I (φ((x, α') := z)) C ∈: D (type_of C)"
        using DI asg a2 asg_into_frame_fun_upd replace_Abs.prems(4) by auto
      moreover
      have "val D I (φ((x, α') := z)) C = val D I (φ((x, α') := z)) D'"
        using a2 val_C_eql_val_D' replace_Abs by auto
      ultimately
      show
        "val D I (φ((x, α') := z)) C ∈: D (type_of C) ∧
        val D I (φ((x, α') := z)) C = val D I (φ((x, α') := z)) D'"
        by auto
    qed
    have "wff (type_of C) D'"
      using replacement_preserves_type replace_Abs.hyps replace_Abs.prems(2)
        replace_Abs.prems(3) replace_Abs.prems(4) wff_Abs_type_of by blast
    then have same_type:
      "abstract (D α') (D (type_of C)) (λz. val D I (φ((x, α') := z)) D') =
```

```
        abstract (D α') (D (type_of D')) (λz. val D I (φ((x, α') := z)) D')"
      using type_of by presburger
    then show "val D I φ [λx:α'. C] = val D I φ ([λx:α'. D'])"
      using val_C_eql_val_D'_type same_type
        abstract_extensional[of _ _ _ "λxa. val D I (φ((x, α') := xa)) D'"]
      by (simp add: val_C_eql_val_D'_type same_type)
  qed
qed
then show "valid_in_model D I C'"
  using assms(2) DI unfolding valid_in_model_def valid_general_def by auto
qed

theorem Fun_Tv_Tv_frame_subs_funspace:
  assumes "general_model D I"
  assumes "asg_into_interp φ D I"
  shows "D oo ⊆: funspace (boolset) (boolset)"
  by (metis assms(1) general_model.elims(2) wf_frame_def wf_interp.simps)


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 (cases "(φ (''g'',oo)) · true = true ∧ (φ (''g'',oo)) · false = true")
  case True
  then have val: "val D I φ ((g_oo · T) ∧ (g_oo · F)) = true"
    using assms lemma_5401_e_variant_2
    by (auto simp add: boolean_eq_true lemma_5401_c[OF assms(1,2)] lemma_5401_d[OF assms(1,2)])
  have "∀ψ. asg_into_frame ψ D ⟶
          agree_off_asg ψ φ ''x'' Tv ⟶
          satisfies D I ψ (g_oo · x_o)"
  proof (rule; rule; rule)
    fix ψ
    assume ψ: "asg_into_frame ψ D" "agree_off_asg ψ φ ''x'' Tv"
    then have "ψ (''x'', Tv) = true ∨ ψ (''x'', Tv) = false"
      using asg_into_interp_is_true_or_false assms
      by auto
    then show " satisfies D I ψ (g_oo · x_o)"
      using True ψ unfolding agree_off_asg_def by auto
  qed
  then have  "val D I φ ([∀''x'':Tv. (g_oo · x_o)]) = true"
    using lemma_5401_g using assms by auto
  then show ?thesis
    unfolding axiom_1_def
    using lemma_5401_b[OF assms(1,2)] val by auto
next
  case False
  have "φ (''g'', oo) ∈: D oo"
    using assms
    by (simp add: asg_into_frame_def)
  then have 0: "φ (''g'', oo) ∈: funspace (D Tv) (D Tv)"
    using assms(1) assms(2) fun_sym_asg_to_funspace by blast

  from False have "(φ (''g'', oo) · true ≠ true ∨ φ (''g'', oo) · false ≠ true)"
    by auto
  then have "∃z. φ (''g'', oo) · z = false ∧ z ∈: D Tv"
  proof
    assume a: "φ (''g'', oo) · true ≠ true"
    have "φ (''g'', oo) · true ∈: boolset"
      by (metis "0" apply_abstract assms(1) boolset_def general_model.elims(2) in_funspace_abstract
          mem_two true_def wf_frame_def wf_interp.simps)
    from this a have "φ (''g'', oo) · true = false ∧ true ∈: D Tv"
      using assms(1) wf_frame_def wf_interp.simps by auto
    then show "∃z. φ (''g'', oo) · z = false ∧ z ∈: D Tv"
```

```
      by auto
  next
    assume a: "φ (''g'', oo) · false ≠ true"
    have "φ (''g'', oo) · false ∈: boolset"
      by (metis "0" apply_abstract assms(1) boolset_def general_model.elims(2) in_funspace_abstract
          mem_two false_def wf_frame_def wf_interp.simps)
    from this a have "φ (''g'', oo) · false = false ∧ false ∈: D Tv"
      using assms(1) wf_frame_def wf_interp.simps by auto
    then show "∃z. φ (''g'', oo) · z = false ∧ z ∈: D Tv"
      by auto
  qed
  then obtain z where z_p: "φ (''g'', oo) · z = false ∧ z ∈: D Tv"
    by auto
  have "boolean (satisfies D I φ (g_oo · T)
          ∧ satisfies D I φ (g_oo · F)) = false"
    using False
    by (smt boolean_def val.simps(1) val.simps(3) lemma_5401_c[OF assms(1,2)]
        lemma_5401_d[OF assms(1,2)])
  then have 1: "val D I φ (
        (g_oo · T) ∧
        (g_oo · F)) = false"
    using lemma_5401_e_variant_2 assms by auto
  have 3: "asg_into_frame (φ((''x'', Tv) := z)) D ∧
    agree_off_asg (φ((''x'', Tv) := z)) φ ''x'' Tv ∧
    φ (''g'', oo) · (φ((''x'', Tv) := z)) (''x'', Tv) ≠ true"
    using z_p Pair_inject agree_off_asg_def2 asg_into_frame_fun_upd assms(2) true_neq_false by fastforce
  then have 2: "val D I φ ([∀''x'':Tv. (g_oo · x_o)]) = false"
    using  lemma_5401_g_variant_1 assms boolean_def by auto
  then show ?thesis
    unfolding axiom_1_def using 1 2 lemma_5401_b_variant_2[OF assms(1,2)] by auto
qed


theorem theorem_5402_a_axiom_1: "valid_general axiom_1"
  using theorem_5402_a_axiom_1_variant unfolding valid_general_def valid_in_model_def by auto


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 (cases "φ(''x'',α) = φ(''y'',α)")
  case True
  have "val D I φ ((Var ''h'' (Tv ⇐ α)) · (Var ''x'' α)) =
          (φ (''h'', (Tv ⇐ α))) · φ (''x'', α)"
    using assms by auto
  also
  have "... = φ (''h'', (Tv ⇐ α)) · φ (''y'', α)"
    using True by auto
  also
  have "... = val D I φ ((Var ''h'' (Tv ⇐ α)) · (Var ''y'' α))"
    using assms by auto
  finally
  show ?thesis
    unfolding axiom_2_def
    using lemma_5401_f_variant_2 assms lemma_5401_b_variant_1[OF assms(1,2)] boolean_def by auto
next
  case False
  have "asg_into_frame φ D"
    using assms(2) by blast
  moreover
  have "general_model D I"
    using assms(1) by blast
  ultimately
```

```
  have
    "boolean (satisfies D I φ [Var ''x'' α =α= Var ''y'' α] ⟶
       satisfies D I φ
         [(Var ''h'' (Tv ⇐ α) · Var ''x'' α) =Tv= Var ''h'' (Tv ⇐ α) · Var ''y'' α]) =
       true"
    using boolean_eq_true lemma_5401_b by auto
  then
  show ?thesis
    using assms lemma_5401_f_variant_2 unfolding axiom_2_def by auto
qed


theorem theorem_5402_a_axiom_2: "valid_general (axiom_2 α)"
  using theorem_5402_a_axiom_2_variant unfolding valid_general_def valid_in_model_def by auto


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 (cases "φ (''f'',α ⇐ β) = φ (''g'',α ⇐ β)")
  case True
  {
    fix ψ
    assume agree: "agree_off_asg ψ φ ''x'' β"
    assume asg: "asg_into_interp ψ D I"
    have "val D I ψ ((Var ''f'' (α ⇐ β)) · (Var ''x'' β)) = ψ (''f'',α ⇐ β) · ψ (''x'', β)"
      by auto
    also
    have "... = φ (''f'',α ⇐ β) · ψ (''x'', β)"
      using agree by auto
    also
    have "... = φ (''g'',α ⇐ β) · ψ (''x'', β)"
      using True by auto
    also
    have "... = ψ (''g'',α ⇐ β) · ψ (''x'', β)"
      using agree by auto
    also
    have "... = val D I ψ ((Var ''g'' (α ⇐ β)) · (Var ''x'' β))"
      by auto
    finally
    have
      "val D I ψ
            ([((Var ''f'' (α ⇐ β)) · (Var ''x'' β)) =α= ((Var ''g'' (α ⇐ β)) · (Var ''x'' β))])
       = true"
      using lemma_5401_b_variant_1[OF assms(1)] assms agree asg boolean_eq_true by auto
  }
  then have "satisfies D I φ
        ([∀''x'':β. [(Var ''f'' (α ⇐ β) · Var ''x'' β) =α= Var ''g'' (α ⇐ β) · Var ''x'' β]])"
    using assms lemma_5401_g by force
  moreover
  have "satisfies D I φ [Var ''f'' (α ⇐ β) =α ⇐ β= Var ''g'' (α ⇐ β)]"
    using True assms using lemma_5401_b_variant_2 wff_Var by auto
  ultimately
  show ?thesis
    using axiom_3_def lemma_5401_b_variant_2 assms by auto
next
  case False
  then have "∃z. z ∈: D β ∧ φ (''f'', α ⇐ β) · z ≠ φ (''g'', α ⇐ β) · z"
    using funspace_difference_witness[of "φ (''f'', α ⇐ β)" "D β" "D α" "φ (''g'', α ⇐ β)"]
      assms(1,2) fun_sym_asg_to_funspace by blast
  then obtain z where
    zβ: "z ∈: D β" and
    z_neq: "φ (''f'', α ⇐ β) · z ≠ φ (''g'', α ⇐ β) · z"
```

```
    by auto
  define ψ where "ψ = (φ((''x'',β):= z))"
  have agree: "agree_off_asg ψ φ ''x'' β"
    using ψ_def agree_off_asg_def2 by blast
  have asg: "asg_into_interp ψ D I"
    using zβ ψ_def asg_into_frame_fun_upd assms(2) by blast
  have "val D I ψ ((Var ''f'' (α ⇐ β)) · (Var ''x'' β)) = ψ (''f'',α ⇐ β) · ψ (''x'', β)"
    by auto
  moreover
  have "... = φ (''f'',α ⇐ β) · z"
    by (simp add: ψ_def)
  moreover
  have "... ≠ φ (''g'',α ⇐ β) · z"
    using False z_neq by blast
  moreover
  have "φ (''g'',α ⇐ β) · z = ψ (''g'',α ⇐ β) · ψ (''x'', β)"
    by (simp add: ψ_def)
  moreover
  have "... = val D I ψ ((Var ''g'' (α ⇐ β)) · (Var ''x'' β))"
    by auto
  ultimately
  have
    "val D I ψ
            ([((Var ''f'' (α ⇐ β)) · (Var ''x'' β)) =α= ((Var ''g'' (α ⇐ β)) · (Var ''x'' β))])
       = false"
    by (metis asg assms(1) lemma_5401_b_variant_3 wff_App wff_Var)
  have "val D I φ
        ([∀''x'':β. [(Var ''f'' (α ⇐ β) · Var ''x'' β) =α= Var ''g'' (α ⇐ β) · Var ''x'' β]]) = false"
    by (smt (verit) ‹val D I ψ [(Var ''f'' (α ⇐ β) · Var ''x'' β) =α= Var ''g'' (α ⇐ β) · Var ''x'' β] =
false›
        agree asg assms(1,2) lemma_5401_g wff_App wff_Eql wff_Forall wff_Tv_is_true_or_false wff_Var)
  moreover
  have "val D I φ [Var ''f'' (α ⇐ β) =α ⇐ β= Var ''g'' (α ⇐ β)] = false"
    using False assms(1,2) lemma_5401_b_variant_3 wff_Var by auto
  ultimately show ?thesis
    using assms(1,2) axiom_3_def lemma_5401_b by auto
qed


theorem theorem_5402_a_axiom_3: "valid_general (axiom_3 α β)"
  using theorem_5402_a_axiom_3_variant unfolding valid_general_def valid_in_model_def by auto


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 -
  let ?ψ = "φ((x,α):=val D I φ A)"
  have "val D I φ ([λx:α. (Cst c β)] · A) = val D I ?ψ (Cst c β)"
    by (rule lemma_5401_a[of _ _ _ _ _ β]; use assms in auto)
  then have "val D I φ ([λx:α. Cst c β] · A) = val D I φ (Cst c β)"
    by auto
  moreover
  have "wff β ([λx:α. Cst c β] · A)"
    using assms by auto
  ultimately
  show ?thesis
    unfolding axiom_4_1_variant_cst_def
    using lemma_5401_b_variant_2[OF assms(1,2)] by auto
qed
```

```
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 -
  let ?ψ = "φ((x,α):=val D I φ A)"
  have  "val D I φ ([λx:α. (Var y β)] · A) = val D I ?ψ (Var y β)"
    by (rule lemma_5401_a[of _ _ _ _ _  β], use assms in auto)
  then have "val D I φ ([λx:α. Var y β] · A) = val D I φ (Var y β)"
    using assms unfolding axiom_4_1_variant_var_side_condition_def by auto
  moreover
  have "wff β ([λx:α. Var y β] · A)"
    using assms by auto
  moreover
  have "wff β (Var y β)"
    using assms by auto
  ultimately
  show ?thesis
    unfolding axiom_4_1_variant_var_def
    using lemma_5401_b_variant_2[OF assms(1,2)] by auto
qed


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)"
  using assms theorem_5402_a_axiom_4_1_variant_cst theorem_5402_a_axiom_4_1_variant_var
  unfolding axiom_4_1_variant_cst_def axiom_4_1_variant_var_side_condition_def
    axiom_4_1_side_condition_def axiom_4_1_variant_var_def
    axiom_4_1_def by auto


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 -
  let ?ψ = "φ((x,α):=val D I φ A)"
  have "wff α ([λx:α. Var x α] · A)"
    using assms by auto
  moreover
  have "wff α A"
    using assms by auto
  moreover
  have "val D I φ ([λx:α. Var x α] · A) = val D I φ A"
    using lemma_5401_a[of _ _ _ _ _ α _ _] assms by auto
  ultimately
  show ?thesis
    unfolding axiom_4_2_def by (rule lemma_5401_b_variant_2[OF assms(1,2)])
qed


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 ⇐ γ) B"
  assumes "wff γ C"
```

```
  shows "satisfies D I φ (axiom_4_3 x α B β γ C A)"
proof -
  let ?ψ = "φ((x,α):=val D I φ A)"
  let ?E = "B · C"

  have "val D I φ (LHS (axiom_4_3 x α B β γ C A)) = val D I ?ψ ?E"
    by (metis LHS_def2 assms(3) assms(4) assms(5) axiom_4_3_def lemma_5401_a[OF assms(1,2)] wff_App)
  moreover
  have "... = val D I ?ψ (B · C)"
    by simp
  moreover
  have "... = (val D I ?ψ B) · val D I ?ψ C"
    by simp
  moreover
  have "... = (val D I φ ([λx:α. B] · A)) · val D I φ (App [λx :α. C] A)"
    by (metis assms(3) assms(4) assms(5) lemma_5401_a[OF assms(1,2)])
  moreover
  have "... = val D I φ (RHS (axiom_4_3 x α B β γ C A))"
    unfolding axiom_4_3_def by auto
  ultimately
  have "val D I φ (LHS (axiom_4_3 x α B β γ C A)) = val D I φ (RHS (axiom_4_3 x α B β γ C A))"
    by auto
  then have "val D I φ ([λx:α. B · C] · A) = val D I φ ([λx:α. B] · A · ([λx:α. C] · A))"
    unfolding axiom_4_3_def by auto
  moreover
  have "wff β ([λx:α. B · C] · A)"
    using assms by auto
  moreover
  have "wff β ([λx:α. B] · A · ([λx:α. C] · A))"
    using assms by auto
  ultimately
  show ?thesis
    unfolding axiom_4_3_def using lemma_5401_b_variant_2[OF assms(1,2)] by auto
qed

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 -
  {
    fix e
    assume e_in_D: "e ∈: D γ"
    then have "val D I (ψ((y, γ) := e)) B ∈: D (type_of B)"
      using asg_into_frame_fun_upd assms(3,4,6,7) ψ_eql by auto
    then have val_lambda_B: "val D I ψ [λy:γ. B] · e = val D I (ψ((y, γ) := e)) B"
      using e_in_D by auto
    have
      "val D I φ [λy:γ. [λx:α. B] · A] · e =
       abstract (D α) (D (type_of B))
         (λz. val D I (φ((y, γ) := e, (x, α) := z)) B) · val D I (φ((y, γ) := e)) A"
      using apply_abstract e_in_D asg_into_frame_fun_upd assms(3,4,6,7) by auto
    then have "val D I (ψ((y, γ) := e)) B =
        abstract (D α) (D (type_of B))
         (λz. val D I (φ((y, γ) := e, (x, α) := z)) B) · val D I (φ((y, γ) := e)) A"
      using val_lambda_B lambda_eql_lambda_lambda e_in_D by metis
  }
  note val_eql_abstract = this
```

```
  have
    "∀e. e ∈: D γ ⟶
            val D I (ψ((y, γ) := e)) B ∈: D (type_of B) ∧
            val D I (ψ((y, γ) := e)) B =
            abstract (D α) (D (type_of B))
              (λza. val D I (φ((y, γ) := e, (x, α) := za)) B) · val D I (φ((y, γ) := e)) A"
    using asg_into_frame_fun_upd assms(3,4,6,7) ψ_eql val_eql_abstract by auto
  then have
    "abstract (D γ) (D (type_of B)) (λz. val D I (ψ((y, γ) := z)) B) =
     abstract (D γ) (D (type_of B))
       (λz. abstract (D α) (D (type_of B))
         (λza. val D I (φ((y, γ) := z, (x, α) := za)) B) · val D I (φ((y, γ) := z)) A)"
    by (rule abstract_extensional)
  then show ?thesis
    by auto
qed


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 -
  define ψ where "ψ = φ((x,α):=val D I φ A)"
  let ?E = "[λy:γ. B]"
  have fr: "(y, γ) ∉ vars A"
    using assms(3) axiom_4_4_side_condition_def by blast
  {
    fix z
    assume z_in_D: "z ∈: D γ"
    define φ' where "φ' = φ((y,γ) := z)"
    have "asg_into_frame φ' D"
      using assms z_in_D unfolding asg_into_frame_def φ'_def by auto
    moreover
    have "∀(x, α)∈vars A. agree_on_asg φ φ' x α"
      using fr unfolding φ'_def by auto
    ultimately
    have "val D I φ A = val D I φ' A"
      using prop_5400[OF assms(1), of _ _ α] assms(2) assms(4) frees_subset_vars by blast
    then have Az: "φ'((x,α):=(val D I φ' A)) = ψ((y,γ):=z)"
      using assms(3) unfolding axiom_4_4_side_condition_def
      by (simp add: fun_upd_twist φ'_def ψ_def)
    then have "abstract (D γ) (D (type_of B)) (λz. val D I (ψ((y, γ) := z)) B) · z =
               val D I (ψ((y, γ) := z)) B"
      using apply_abstract_matchable assms(1,2,4,5) type_of asg_into_frame_fun_upd
        general_model.elims(2) z_in_D by (metis φ'_def)
    then have "(val D I ψ ?E) · z = (val D I (ψ((y,γ):=z)) B)"
      by auto
    moreover
    have "... = val D I φ' ([λx:α. B] · A)"
      using assms(1,2,4,5) asg_into_frame_fun_upd lemma_5401_a z_in_D
      by (metis Az φ'_def)
    moreover
    have "... = val D I φ [λy:γ. [λx:α. B] · A] · z"
    proof -
      have valA: "val D I φ' A ∈: D α"
        using φ'_def asg_into_frame_fun_upd z_in_D assms by simp
      have valB: "val D I (φ'((x, α) := val D I φ' A)) B ∈: D (type_of B)"
        using asg_into_frame_fun_upd z_in_D assms by (simp add: Az ψ_def)
      have valA': "val D I (φ((y, γ) := z)) A ∈: D α"
```

```
        using z_in_D assms asg_into_frame_fun_upd valA unfolding ψ_def φ'_def
        by blast
      have valB':
        "val D I (φ((y, γ) := z, (x, α) := val D I (φ((y, γ) := z)) A)) B
         ∈: D (type_of B)"
        using asg_into_frame_fun_upd z_in_D assms valB φ'_def by blast
      have
        "val D I (φ'((x, α) := val D I φ' A)) B =
         val D I (φ((y, γ) := z, (x, α) := val D I (φ((y, γ) := z)) A)) B"
        unfolding ψ_def φ'_def by (metis apply_abstract asg_into_frame_fun_upd)
      then have valB_eql_abs:
        "val D I (φ'((x, α) := val D I φ' A)) B =
         abstract (D α) (D (type_of B))
           (λza. val D I (φ((y, γ) := z, (x, α) := za)) B) · val D I (φ((y, γ) := z)) A"
        using valA' valB' by auto
      then have "abstract (D α) (D (type_of B))
             (λza. val D I (φ((y, γ) := z, (x, α) := za)) B) · val D I (φ((y, γ) := z)) A
           ∈: D (type_of B)"
        using valB assms z_in_D by auto
      then have
        "val D I (φ'((x, α) := val D I φ' A)) B =
         abstract (D γ) (D (type_of B))
           (λz. abstract (D α) (D (type_of B))
             (λza. val D I (φ((y, γ) := z, (x, α) := za)) B) · val D I (φ((y, γ) := z)) A) · z"
        using z_in_D valB_eql_abs by auto
      then show "val D I φ' ([λx:α. B] · A) = val D I φ [λy:γ. [λx:α. B] · A] · z"
        using valA valB by auto
    qed
    ultimately
    have "val D I ψ [λy:γ. B] · z = val D I φ [λy:γ. [λx:α. B] · A] · z"
      by simp
  }
  note lambda_eql_lambda_lambda = this
  have equal_funs: "val D I ψ ?E = val D I φ ([λy:γ. ([λx:α. B]) · A])"
    using lambda_eql_lambda_lambda ψ_def assms lemma_to_help_with_theorem_5402_a_axiom_4_4 by metis
  have "val D I φ ([λx:α. [λy:γ. B]] · A) = val D I φ [λy:γ. [λx:α. B] · A]"
    using equal_funs by (metis ψ_def assms(1,2,4,5) lemma_5401_a wff_Abs)
  then have "satisfies D I φ [([λx:α. [λy:γ. B]] · A) =δ ⇐ γ= [λy:γ. [λx:α. B] · A]]"
    using lemma_5401_b[OF assms(1,2)] assms by auto
  then show ?thesis
    unfolding axiom_4_4_def .
qed


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 -
  define ψ where "ψ = φ((x,α):=val D I φ A)"
  let ?E = "[λx:α. B]"

  {
    assume val: "∀φ. asg_into_frame φ D ⟶ (∀A α. wff α A ⟶ val D I φ A ∈: D α)"
    assume asg: "asg_into_frame φ D"
    assume wffA: "wff α A"
    assume wffB: "wff δ B"
    have valA: "val D I φ A ∈: D α"
      using val asg wffA by blast
    have "∀t cs. val D I φ [λcs:t. B] ∈: D (δ ⇐ t)"
      using val asg wffB wff_Abs by blast
    then have "abstract (D α) (D (δ ⇐ α))
```

```
                (λu. abstract (D α) (D δ) (λu. val D I (φ((x, α) := u)) B)) · val D I φ A =
              abstract (D α) (D δ) (λu. val D I (φ((x, α) := u)) B)"
      using valA wffB by simp
  }
  note abstract_eql = this

  have "val D I ψ ?E = val D I φ ?E"
    using prop_5400[OF assms(1), of _ _ "δ ⇐ α"] ψ_def assms(2) by auto
  then show ?thesis
    unfolding axiom_4_5_def using lemma_5401_b[OF assms(1,2)] assms abstract_eql by auto
qed


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 -
  have iden_eql: "iden (D Ind) · I ''y'' Ind = one_elem_fun (I ''y'' Ind) (D Ind)"
  proof -
    have "I ''y'' Ind ∈: D Ind"
      using assms unfolding  general_model.simps wf_interp.simps[simplified] iden_def one_elem_fun_def
      by auto
    moreover
    have "abstract (D Ind) boolset (λy. boolean (I ''y'' Ind = y)) ∈: funspace (D Ind) boolset"
      using boolean_in_boolset by auto
    ultimately
    show ?thesis
      unfolding iden_def one_elem_fun_def by auto
  qed

  have "val D I φ (ι · ((Q (Tv ⇐ Ind ⇐ Ind)) · yᵢ)) =
          val D I φ ι · val D I φ ((Q (Tv ⇐ Ind ⇐ Ind)) · yᵢ)"
    by auto
  moreover
  have "... = val D I φ yᵢ"
    using assms iden_eql unfolding general_model.simps wf_interp.simps[simplified] by auto
  ultimately
  show ?thesis
    unfolding axiom_5_def using lemma_5401_b[OF assms(1,2)] by auto
qed

lemma theorem_isa_Tv:
  assumes "theorem A"
  shows "wff Tv A"
  using assms proof (induction)
  case (by_axiom A)
  then show ?case
  proof (induction)
    case by_axiom_1
    then show ?case
      unfolding axiom_1_def by auto
  next
    case (by_axiom_2 α)
    then show ?case
      unfolding axiom_2_def by auto
  next
    case (by_axiom_3 α β)
    then show ?case
      unfolding axiom_3_def by auto
  next
    case (by_axiom_4_1 α A β B x)
    then show ?case
      unfolding axiom_4_1_def by auto
```

```
  next
    case (by_axiom_4_2 α A x)
    then show ?case
      unfolding axiom_4_2_def by auto
  next
    case (by_axiom_4_3 α A β γ B C x)
    then show ?case
      unfolding axiom_4_3_def by auto
  next
    case (by_axiom_4_4 α A δ B x y γ)
    then show ?case
      unfolding axiom_4_4_def by auto
  next
    case (by_axiom_4_5 α A δ B x)
    then show ?case
      unfolding axiom_4_5_def by auto
  next
    case by_axiom_5
    then show ?case
      unfolding axiom_5_def by auto
  qed
next
  case (by_rule_R A B C)
  then show ?case
    by (smt replacement_preserves_type rule_R.cases wff_Eql_iff)
qed


theorem theorem_5402_a_general:
  assumes "theorem A"
  shows "valid_general A"
  using assms
proof (induction)
  case (by_axiom A)
  then show ?case
  proof (induction)
    case by_axiom_1
    then show ?case
      using theorem_5402_a_axiom_1 by auto
  next
    case (by_axiom_2 α)
    then show ?case
      using theorem_5402_a_axiom_2 by auto
  next
    case (by_axiom_3 α β)
    then show ?case
      using theorem_5402_a_axiom_3 by auto
  next
    case (by_axiom_4_1 α A β B x)
    then show ?case
      using theorem_5402_a_axiom_4_1
      unfolding valid_general_def valid_in_model_def by auto
  next
    case (by_axiom_4_2 α A x)
    then show ?case
      using theorem_5402_a_axiom_4_2
      unfolding valid_general_def valid_in_model_def by auto
  next
    case (by_axiom_4_3 α A β γ B C x)
    then show ?case
      using theorem_5402_a_axiom_4_3
      unfolding valid_general_def valid_in_model_def by auto
  next
    case (by_axiom_4_4 α A δ B x y γ)
```

```
      then show ?case
        using theorem_5402_a_axiom_4_4
        unfolding valid_general_def valid_in_model_def by auto
  next
    case (by_axiom_4_5 α A δ B x)
    then show ?case
      using theorem_5402_a_axiom_4_5
      unfolding valid_general_def valid_in_model_def by auto
  next
    case by_axiom_5
    then show ?case
      using theorem_5402_a_axiom_5
      unfolding valid_general_def valid_in_model_def by auto
  qed
next
  case (by_rule_R C AB C')
  then have C_isa_Tv: "wff Tv C"
    using theorem_isa_Tv by blast
  have "∃A B β. AB = [A =β= B] ∧ wff β A ∧ wff β B"
    using by_rule_R rule_R.simps theorem_isa_Tv by fastforce
  then obtain A B β where A_B_β_p: "AB = [A =β= B] ∧ wff β A ∧ wff β B"
    by blast
  then have R: "rule_R C [A =β= B] C'"
    using by_rule_R by auto
  then have "replacement A B C C'"
    using Eql_def rule_R.cases by fastforce
  show ?case
    using theorem_5402_a_rule_R[of A B β C C' Tv] by_rule_R.IH R
      A_B_β_p C_isa_Tv by auto
qed


theorem theorem_5402_a_standard:
  assumes "theorem A"
  shows "valid_standard A"
  using theorem_5402_a_general assms standard_model_is_general_model valid_general_def
    valid_standard_def by blast

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. Springer Science & Business Media, 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.

[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.

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