```
(* ********************************************************************************** *)
theory t
imports Complex_Main
begin
(* ********************************************************************************** *)

(* The meaning of operator ≡ can possibly be derived from operator ⟹ using
   rule equal_intr_rule. *)

thm equal_intr_rule (*                                                          (I.8)
  o¦ ⟦PROP phi ⟹ PROP psi; PROP psi ⟹ PROP phi⟧ ⟹ PROP phi ≡ PROP psi. *)

theorem
  "   (A ⟹ B)
  ⟹ (B ⟹ A)
  ⟹ (Trueprop A ≡ Trueprop B)"
by(rule equal_intr_rule)                                                        (I.8)

(* The meaning of operator ⋀ can possibily be derived from operator ≡ using
   rule triv_forall_equality, so the meaning of ⋀ can possibly be derived from
   ⟹ using equal_intr_rule. (THIS IS BOGUS. I don't know what all I can get
   out of triv_forall_equality, but it served its purpose for now.) *)

thm triv_forall_equality (*                                                     (I.9)
  o¦ (⋀x. PROP V) ≡ PROP V. *)

theorem
  "((⋀x. PROP V) ≡ PROP V)"
apply(rule equal_intr_rule)(*                                                   (I.8)
  2g¦1¦ (⋀x. PROP V) ⟹ PROP V ¦2¦ ⋀x. PROP V ⟹ PROP V.
    This is to show that apply(rule equal_intr_rule) will produces goals
  instead of completing the proof when no assumptions are given which match
  the assumptions of equal_intr_rule. This information and that of the next
  comment are used to set the stage for a comment below on the proof of
  atomize_all. *)
oops

theorem
  "   ((⋀x. PROP V) ⟹ PROP V)
  ⟹ (PROP V ⟹ (⋀x. PROP V))
  ⟹ ((⋀x. PROP V) ≡ PROP V)"
by(rule equal_intr_rule)(*                                                      (I.8)
  The two assumptions of equal_intr_rule are provided this time. Rather than
  produce two goals, rule now completes the proof in one step, using only the
  rule equal_intr_rule. *)

(* Now consider the theorem atomize_all, it uses ⋀ and ≡, but not ⟹. If
   atomize_all can be proved only using equal_intr_rule, then it seems the
   perspective is justified that all Isabelle/HOL logical meaning is a result
   of prop, schematic variables, ⟹, and nothing more, or at least, "something
   more" does not have to include ⋀ and ≡. (NOT ALL BOGUS, BUT BOGUS.) *)

thm atomize_all (*                                                              (I.14)
  o¦ (⋀x. P x) ≡ ∀x. P x. *)

theorem
  "   ((⋀x. P x) ⟹ (∀x. P x))
  ⟹ ((∀x. P x) ⟹ (⋀x. P x))
  ⟹ ((⋀x. P x) ≡ Trueprop (∀x. P x))"
apply(rule equal_intr_rule)(*                                                   (I.8)
```

```
  2g¦1¦ ⟦(⋀x. P x) ⟹ ∀x. P x; ⋀x. All P ⟹ P x⟧ ⟹ (⋀x. P x) ⟹ ∀x. P x
    ¦2¦ ⋀x. ⟦(⋀x. P x) ⟹ ∀x. P x; ⋀x. All P ⟹ P x⟧ ⟹ ∀x. P x ⟹ P x.
    Life is not simple. The right assumptions were provided, but rule did not
  use them. Instead, rule produced two goals. The conclusion can still be proved,
  but it cannot be completed in one step with equal_intr_rule.
    It might be that Trueprop operating on (∀x. P x) both automatically and
  explicitly causes complications. Having to use simp below ends up being good.
  It shows that ⋀ and ≡ appear to have a life of their own when it comes to
  rewriting. *)
using[[simp_trace]]
apply(simp only: triv_forall_equality)(*                                    (I.9)
  2g¦1¦ ⟦True ⟹ ∀x. True; All P ⟹ True⟧ ⟹ (⋀x. P x) ⟹ ∀x. True
    ¦2¦ ⋀x. ⟦(⋀x. P x) ⟹ ∀x. P x; ⋀x. All P ⟹ P x⟧ ⟹ ∀x. P x ⟹ P x.
    Now it starts to get complicated. From the simp trace, some rewriting is
  being done which is based on ⟹. For example, from the second assumption
  given, a rewrite rule is created:
      [1]SIMPLIFIER INVOKED ON THE FOLLOWING TERM:
      ((⋀x. P x) ⟹ ∀x. P x) ⟹ (⋀x. All P ⟹ P x) ⟹ (⋀x. P x) ⟹ ∀x. P x
      [1]Adding rewrite rule "??.unknown":
      All P ⟹ P ?x1 ≡ True.
  There is a "perspective" problem with this. The simplifier appears to be using
  the presence of the symbol ⋀ to decide that this is a rewrite rule. Addition-
  ally, the symbol ≡ is being used to state the new rewrite rule.
    Also, triv_forall_equality is being used to do some rewriting. For example,
  there is the following rewriting:
      [1]Applying instance of rewrite rule "Pure.triv_forall_equality":
      (⋀x. PROP ?V) ≡ PROP ?V
      [1]Rewriting:
      (⋀x. True) ≡ True.
  Again, it appears that the meta-logic symbols ⋀ and ≡ are used independently
  of ⟹. If simp is using ⟹ to determine these equivalencies, it is not making
  it known. *)
apply(simp only: simp_thms(35))(*                                           (I.10)
  1g¦1¦ ⋀x. ⟦(⋀x. P x) ⟹ ∀x. P x; ⋀x. All P ⟹ P x⟧ ⟹ ∀x. P x ⟹ P x.
    The rewriting ∀x. True ≡ True is done to complete the previous goal. *)
by(simp only: triv_forall_equality)(*                                       (I.9)
  And finally, more rewriting similar to what was done with the first use of
  triv_forall_equality. *)

(* ************************************************************************** *)
end
(* ************************************************************************** *)
```

I HOL/Pure Operators, Theorems, Glossary, Etc.

TEST

Test (I.8).

HOL Operators

⇒, => **(I.1)** Meta-logic function operator, [Wen13, 161].
`tycon "fun", typ "type ⇒ type ⇒ type", Mixfix ("(_/ \⇒ _)", [1, 0], 0).`
The notation *['a, 'a, 'a] ⇒ 'a* means *'a ⇒ ('a ⇒ ('a ⇒ 'a))*.

⋀, !! **(I.2)** Meta-logic universal quantifier, [Wen13, 182].
`Binding.name "all", typ "('a ⇒ prop) ⇒ prop", Binder ("!!", 0, 0).`

⟹, ==> **(I.3)** Meta-logic implication, [Wen13, 181].
`Binding.name "==>", typ "prop ⇒ prop ⇒ prop", Mixfix ("(_/ ==> _)", [2, 1], 1).`

≡, == **(I.4)** Meta-logic equivalence, [Wen13, 180].
`Binding.name "==", typ "'a ⇒ 'a ⇒ prop", Infix ("==", 2).`

Trueprop (I.5) *judgement*, [NPW13a, 60].
`("(_)" 5), (Trueprop::(bool ⇒ prop).`

∀, !, ALL, All **(I.6)** *consts*, *defs*, object-logic quantifier, [NPW13a, 76].
`(binder "∀" 10), (All::('a ⇒ bool) ⇒ bool), "All(P) ≡ (P = (λx. True))".`
Note that *ALL* is notation for the function *All*.

=, eq **(I.7)** *axiomatization*, equals, [NPW13a, 65].
`(infixl "=" 50), (eq::'a ⇒ 'a ⇒ bool).`
The notation <-> and ⟷ are also notation for *eq*, but have a priority of 25 instead of 50. The function *eq* is defined when it is used with *axiomatization*, such as in HOL.thy at lines 159-161, 166, 173, and 708, in Set.thy at lines 16-17, and in Nat.thy at lines 25-26.

Pure Rules

Theorem statements are mostly taken from the output of the *thm* command, with only a minimal amount of information shown.

equal_intr_rule (I.8) *val*, [Pau13, 567],
`⟦PROP phi ⟹ PROP psi; PROP psi ⟹ PROP phi⟧ ⟹ PROP phi ≡ PROP psi.`
Source comment: Introduction rule for == as a meta-theorem.

triv_forall_equality (I.9) *val*, [Pau13, 543],
`(⋀x. PROP V) ≡ PROP V.`

HOL Simp Rules

simp_thms (I.10) *lemma*, [NPW13a, 938]

939..01) not_not: "(¬ ¬P) = P"
940..02) Not_eq_iff: "((¬P) = (¬Q)) = (P = Q)"
942..03) "(P ≠ Q) = (P = (¬Q))"
943..04) "(P ∨ ¬P) = True"
.....05) "(¬P ∨ P) = True"
944..06) "(x = x) = True"
945..07) not_True_eq_False [code]: "(¬True) = False"
946..08) not_False_eq_True [code]: "(¬False) = True"
948..09) "(¬P) ≠ P"
.....10) "P ≠ (¬P)"
949..11) "(True=P) = P"
950..12) eq_True: "(P = True) = P"
951..13) "(False=P) = (¬P)"
952..14) eq_False: "(P = False) = (¬P)"
954..15) "(True ⟶ P) = P"
.....16) "(False ⟶ P) = True"
955..17) "(P ⟶ True) = True"
.....18) "(P ⟶ P) = True"
956..19) "(P ⟶ False) = (¬P)"
.....20) "(P ⟶ ¬P) = (¬P)"
957..21) "(P ∧ True) = P"
.....22) "(True ∧ P) = P"
958..23) "(P ∧ False) = False"
.....24) "(False ∧ P) = False"
959..25) "(P ∧ P) = P"
.....26) "(P ∧ (P ∧ Q)) = (P ∧ Q)"
960..27) "(P ∧ ¬P) = False"
.....28) "(¬P ∧ P) = False"
961..29) "(P ∨ True) = True"
.....30) "(True ∨ P) = True"
962..31) "(P ∨ False) = P"
.....32) "(False ∨ P) = P"
963..33) "(P ∨ P) = P"
.....34) "(P ∨ (P ∨ Q)) = (P ∨ Q)"
964..35) "(∀x. P) = P"
.....36) "(∃x. P) = P"
.....37) "∃x. x = t"
.....38) "∃x. t = x"
966..39) "⋀P. (∃x. x = t ∧ P(x)) = P(t)"
967..40) "⋀P. (∃x. t = x ∧ P(x)) = P(t)"
968..41) "⋀P. (∀x. x = t ⟶ P(x)) = P(t)"
969..42) "⋀P. (∀x. t = x ⟶ P(x)) = P(t)".

trans (I.11) ⟦r = s; s = t⟧ ⟹ r = t, (I.36).

A

allI (I.12) *lemma*, [NPW13a, 306],
(⋀x. P x) ⟹ ∀x. P x.

assumption (I.13) The *assumption* method proves a subgoal by finding a matching assumption [NPW13b, 69]. "All given facts are guaranteed to participate in the refinement; this means there may be only 0 or 1 in the first place. [Terminal proof step *qed*] concludes any remaining sub-goals by assumption, so structured proofs usually need not quote the *assumption* method at all'' [WC13, 122].

atomize_all (I.14) *lemma*, [NPW13a, 710],
(⋀x. P x) ≡ Trueprop (∀x. P x).

D

disjE (I.15) *lemma*, *[atomize]*, [NPW13a, 464],
⟦P ∨ Q; P ⟹ R; Q ⟹ R⟧ ⟹ R.

E

eq_reflection (I.16) *axiomatization*, [NPW13a, 708],
x = y ⟹ x ≡ y.

eqTrueE (I.17) *lemma*, [NPW13a, 300],
P = True ⟹ P.

eqTrueI (I.18) *lemma*, [NPW13a, 297],
P ⟹ P = True.

ext (I.19) *axiomatization*, [NPW13a, 161],
(⋀x. f x = g x) ⟹ (λx. f x) = (λx. g x).

F

FalseE (I.20) *lemma*, [NPW13a, 335],
False ⟹ P.

I

iff (I.21) *axiomatization*, [NPW13a, 172],
(P ⟶ Q) ⟶ (Q ⟶ P) ⟶ P = Q.

iffE (I.22) *lemma*, [NPW13a, 286],
⟦P = Q; ⟦P ⟶ Q; Q ⟶ P⟧ ⟹ R⟧ ⟹ R.

iffI (I.23) *lemma*, [NPW13a, 270],
$[\![P \Longrightarrow Q;\ Q \Longrightarrow P]\!] \Longrightarrow P = Q$.

impE (I.24) *lemma*, [NPW13a, 376],
$[\![P \longrightarrow Q;\ P;\ Q \Longrightarrow R]\!] \Longrightarrow R$.

impI (I.25) *axiomatization*, [NPW13a, 169],
$(P \Longrightarrow Q) \Longrightarrow P \longrightarrow Q$.

M

mp (I.26) *axiomatization*, [NPW13a, 170],
$[\![P \longrightarrow Q;\ P]\!] \Longrightarrow Q$.

N

notE (I.27) *lemma*, [NPW13a, 364],
$[\![\neg P;\ P]\!] \Longrightarrow R$.

notI (I.28) *lemma*, [NPW13a, 346],
$(P \Longrightarrow \mathrm{False}) \Longrightarrow \neg P$.

R

refl (I.29) *axiomatization*, [NPW13a, 159],
$t = t$.

rule (I.30) Use *apply(rule thm)* when the conclusion of *thm* matches the conclusion of the current goal [Tow13]. New goals will then be created that correspond with the premises of *thm*. Example: If you want to prove $A \longleftrightarrow B$, then use *apply(rule iffI)* (I.23). The new goals $A \Longrightarrow B$ and $B \Longrightarrow A$ will then be created.

When n rules are applied with *apply(rule* $r_1 \ldots r_n$*)*, then *rule* tries to apply, from left to right [Wen02, 60], each r_i in a backwards manner, where, if possible, r_i is reduced with available facts before it is applied to the goal. Consequently, if no facts reduce a r_i that is used, then *rule* acts as an introduction rule, where if facts reduce a r_i which is used, then *rule* acts as an elimination rule.

If *rule* is used with no r_i then appropriate rules are tried which have previously been declared in a theory with *intro*, *elim*, and *dest* [WC13, 122].

S

spec (I.31) *lemma*, [NPW13a, 309],
$\forall x.\ P\ x \Longrightarrow P\ x$.

subst (I.32) *axiomatization*, [NPW13a, 160],
$[\![s = t;\ P\ s]\!] \Longrightarrow P\ t$.

ssubst (I.33) *lemma*, [NPW13a, 207],
$[\![t = s;\ P\ s]\!] \Longrightarrow P\ t$.

sym (I.34) *lemma*, [NPW13a, 204],
$s = t \Longrightarrow t = s$.

T

the_eq_trivial (I.35) *axiomatization*, [NPW13a, 166],
$(\text{THE } x.\ x = a) = a$.

trans (I.36) *lemma*, [NPW13a, 210],
$[\![r = s;\ s = t]\!] \Longrightarrow r = t$.

True_or_False (I.37) *axiomatization*, [NPW13a, 173],
$P = \text{True} \vee P = \text{False}$.

TrueI (I.38) *lemma*, [NPW13a, 294],
True.

J Other Thy Operators, Theorems, Etc.

Set Operators

{}, **empty (J.1)** *abbreviation*, empty set, [NPW13c, 134].
```
("{}"), (empty::'a set), "{} ≡ bot".
```

{x}, {x,xs}, **insert (J.2)** *definition*, finite set, [NPW13c, 137].
```
("{(_)}"), "{x, xs}" == "CONST insert x {xs}", "{x}" == "CONST insert x {}",
(insert::'a ⇒ 'a set ⇒ 'a set), "insert a B = {x. x = a ∨ x ∈ B}".
```

⊂, <, **subset (J.3)** *abbreviation*, subset, [NPW13c, 149].
```
("(_/ ⊂ _)" [51, 51] 50), (subset::'a set ⇒ 'a set ⇒ bool), "subset ≡ less".
```

C

CollectI (J.4) *lemma*, [NPW13c, 57],
```
P a ⟹ a ∈ {x. P x}.
```

Bibliography

[NPW13a] Tobias Nipkow, Lawrence C. Paulson, and Makarius Wenzel. *HOL: The basis of Higher-Order Logic*. University of Cambridge and Technische Universität München, isabelle.in.tum.de, February 2013. http://isabelle.in.tum.de/repos/isabelle/file/d90218288d51/src/HOL/HOL.thy.

[NPW13b] Tobias Nipkow, Lawrence C. Paulson, and Makarius Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*. Springer-Verlag, Berlin, February 2013. http://isabelle.in.tum.de/website-Isabelle2013/documentation.html.

[NPW13c] Tobias Nipkow, Lawrence C. Paulson, and Makarius Wenzel. *Set theory for higher-order logic*. University of Cambridge and Technische Universität München, isabelle.in.tum.de, February 2013. http://isabelle.in.tum.de/repos/isabelle/file/d90218288d51/src/HOL/Set.thy.

[Pau13] Lawrence C. Paulson. *Pure/drule.ML*. University of Cambridge and Technische Universität München, isabelle.in.tum.de, February 2013. http://isabelle.in.tum.de/repos/isabelle/file/d90218288d51/src/Pure/drule.ML.

[Tow13] Henry Towsner. *Isabelle / Proof General Cheat Sheet credited to Towsner by Avigad*. www.phil.cmu.edu/~avigad/formal, July 2013. http://www.phil.cmu.edu/~avigad/formal/FormalCheatSheet.pdf.

[WC13] Makarius Wenzel and Contributors. *The Isabelle/Isar Reference Manual*. University of Cambridge and Technische Universität München, isabelle.in.tum.de, February 2013. http://isabelle.in.tum.de/website-Isabelle2013/documentation.html.

[Wen02] Makarius Wenzel. *Isabelle/Isar — a versatile environment for human-readable formal proof documents*. Université Paris-Sud, www.lri.fr/~wenzel, 2002. http://tumb1.biblio.tu-muenchen.de/publ/diss/in/2002/wenzel.html.

[Wen13] Makarius Wenzel. *Pure/pure_thy.ML*. University of Cambridge and Technische Universität München, isabelle.in.tum.de, February 2013. http://isabelle.in.tum.de/repos/isabelle/file/d90218288d51/src/Pure/pure_thy.ML.

Contents (Detailed)