Miscellaneous HOL Examples

April 19, 2009

Contents

1 An experimental alternative numeral representation.

theory *Numeral*
imports *Int Inductive*
begin

1.1 The *num* type

datatype *num* = *One* | *Dig0 num* | *Dig1 num*

Increment function for type *Numeral.num*

primrec
  inc :: *num* ⇒ *num*
where
  inc One = *Dig0 One*
| *inc* (*Dig0 x*) = *Dig1 x*
| *inc* (*Dig1 x*) = *Dig0* (*inc x*)

Converting between type *Numeral.num* and type *nat*

primrec
  nat-of-num :: *num* ⇒ *nat*
where
  nat-of-num One = *Suc 0*
| *nat-of-num* (*Dig0 x*) = *nat-of-num x* + *nat-of-num x*
| *nat-of-num* (*Dig1 x*) = *Suc* (*nat-of-num x* + *nat-of-num x*)

primrec
  num-of-nat :: *nat* ⇒ *num*
where
  num-of-nat 0 = *One*
| *num-of-nat* (*Suc n*) = (*if 0* < *n then inc* (*num-of-nat n*) *else One*)

lemma *nat-of-num-pos*: *0* < *nat-of-num x*
  ⟨*proof*⟩

lemma *nat-of-num-neq-0*: *nat-of-num x* ≠ *0*
  ⟨*proof*⟩

lemma *nat-of-num-inc*: *nat-of-num* (*inc x*) = *Suc* (*nat-of-num x*)
  ⟨*proof*⟩

lemma *num-of-nat-double*:
  0 < *n* ⟹ *num-of-nat* (*n* + *n*) = *Dig0* (*num-of-nat n*)
  ⟨*proof*⟩

Type *Numeral.num* is isomorphic to the strictly positive natural numbers.

lemma *nat-of-num-inverse*: *num-of-nat* (*nat-of-num x*) = *x*

⟨*proof*⟩

lemma *num-of-nat-inverse*: $0 < n \Longrightarrow$ *nat-of-num* (*num-of-nat* n) $= n$
⟨*proof*⟩

lemma *num-eq-iff*: $x = y \longleftrightarrow$ *nat-of-num* $x =$ *nat-of-num* y
⟨*proof*⟩

lemma *num-induct* [*case-names One inc*]:
fixes P :: *num* ⇒ *bool*
assumes *One*: P *One*
and *inc*: $\bigwedge x.\ P\ x \Longrightarrow P$ (*inc* x)
shows $P\ x$
⟨*proof*⟩

From now on, there are two possible models for *Numeral.num*: as positive naturals (rule *num-induct*) and as digit representation (rules *num.induct*, *num.cases*).

It is not entirely clear in which context it is better to use the one or the other, or whether the construction should be reversed.

1.2 Numeral operations

⟨*ML*⟩

instantiation *num* :: {*plus*,*times*,*ord*}
begin

definition *plus-num* :: *num* ⇒ *num* ⇒ *num* **where**
[*code del*]: $m + n =$ *num-of-nat* (*nat-of-num* $m +$ *nat-of-num* n)

definition *times-num* :: *num* ⇒ *num* ⇒ *num* **where**
[*code del*]: $m * n =$ *num-of-nat* (*nat-of-num* $m *$ *nat-of-num* n)

definition *less-eq-num* :: *num* ⇒ *num* ⇒ *bool* **where**
[*code del*]: $m \leq n \longleftrightarrow$ *nat-of-num* $m \leq$ *nat-of-num* n

definition *less-num* :: *num* ⇒ *num* ⇒ *bool* **where**
[*code del*]: $m < n \longleftrightarrow$ *nat-of-num* $m <$ *nat-of-num* n

instance ⟨*proof*⟩

end

lemma *nat-of-num-add*: *nat-of-num* $(x + y) =$ *nat-of-num* $x +$ *nat-of-num* y
⟨*proof*⟩

lemma *nat-of-num-mult*: *nat-of-num* $(x * y) =$ *nat-of-num* $x *$ *nat-of-num* y
⟨*proof*⟩

lemma *Dig-plus* [*numeral*, *simp*, *code*]:
$One + One = Dig0\ One$
$One + Dig0\ m = Dig1\ m$
$One + Dig1\ m = Dig0\ (m + One)$
$Dig0\ n + One = Dig1\ n$
$Dig0\ n + Dig0\ m = Dig0\ (n + m)$
$Dig0\ n + Dig1\ m = Dig1\ (n + m)$
$Dig1\ n + One = Dig0\ (n + One)$
$Dig1\ n + Dig0\ m = Dig1\ (n + m)$
$Dig1\ n + Dig1\ m = Dig0\ (n + m + One)$
⟨*proof*⟩

lemma *Dig-times* [*numeral*, *simp*, *code*]:
$One * One = One$
$One * Dig0\ n = Dig0\ n$
$One * Dig1\ n = Dig1\ n$
$Dig0\ n * One = Dig0\ n$
$Dig0\ n * Dig0\ m = Dig0\ (n * Dig0\ m)$
$Dig0\ n * Dig1\ m = Dig0\ (n * Dig1\ m)$
$Dig1\ n * One = Dig1\ n$
$Dig1\ n * Dig0\ m = Dig0\ (n * Dig0\ m + m)$
$Dig1\ n * Dig1\ m = Dig1\ (n * Dig1\ m + m)$
⟨*proof*⟩

lemma *Dig-eq*:
$One = One \longleftrightarrow True$
$One = Dig0\ n \longleftrightarrow False$
$One = Dig1\ n \longleftrightarrow False$
$Dig0\ m = One \longleftrightarrow False$
$Dig1\ m = One \longleftrightarrow False$
$Dig0\ m = Dig0\ n \longleftrightarrow m = n$
$Dig0\ m = Dig1\ n \longleftrightarrow False$
$Dig1\ m = Dig0\ n \longleftrightarrow False$
$Dig1\ m = Dig1\ n \longleftrightarrow m = n$
⟨*proof*⟩

lemma *less-eq-num-code* [*numeral*, *simp*, *code*]:
$One \leq n \longleftrightarrow True$
$Dig0\ m \leq One \longleftrightarrow False$
$Dig1\ m \leq One \longleftrightarrow False$
$Dig0\ m \leq Dig0\ n \longleftrightarrow m \leq n$
$Dig0\ m \leq Dig1\ n \longleftrightarrow m \leq n$
$Dig1\ m \leq Dig1\ n \longleftrightarrow m \leq n$
$Dig1\ m \leq Dig0\ n \longleftrightarrow m < n$
⟨*proof*⟩

lemma *less-num-code* [*numeral*, *simp*, *code*]:
$m < One \longleftrightarrow False$

$One < One \longleftrightarrow False$
$One < Dig0\ n \longleftrightarrow True$
$One < Dig1\ n \longleftrightarrow True$
$Dig0\ m < Dig0\ n \longleftrightarrow m < n$
$Dig0\ m < Dig1\ n \longleftrightarrow m \leq n$
$Dig1\ m < Dig1\ n \longleftrightarrow m < n$
$Dig1\ m < Dig0\ n \longleftrightarrow m < n$
⟨*proof*⟩

Rules using *One* and *inc* as constructors

lemma *add-One*: $x + One = inc\ x$
⟨*proof*⟩

lemma *add-inc*: $x + inc\ y = inc\ (x + y)$
⟨*proof*⟩

lemma *mult-One*: $x * One = x$
⟨*proof*⟩

lemma *mult-inc*: $x * inc\ y = x * y + x$
⟨*proof*⟩

A double-and-decrement function

primrec *DigM* :: *num* ⇒ *num* **where**
$DigM\ One = One$
| $DigM\ (Dig0\ n) = Dig1\ (DigM\ n)$
| $DigM\ (Dig1\ n) = Dig1\ (Dig0\ n)$

lemma *DigM-plus-one*: $DigM\ n + One = Dig0\ n$
⟨*proof*⟩

lemma *add-One-commute*: $One + n = n + One$
⟨*proof*⟩

lemma *one-plus-DigM*: $One + DigM\ n = Dig0\ n$
⟨*proof*⟩

Squaring and exponentiation

primrec *square* :: *num* ⇒ *num* **where**
$square\ One = One$
| $square\ (Dig0\ n) = Dig0\ (Dig0\ (square\ n))$
| $square\ (Dig1\ n) = Dig1\ (Dig0\ (square\ n + n))$

primrec *pow* :: *num* ⇒ *num* ⇒ *num*
where
$pow\ x\ One = x$
| $pow\ x\ (Dig0\ y) = square\ (pow\ x\ y)$
| $pow\ x\ (Dig1\ y) = x * square\ (pow\ x\ y)$

1.3 Binary numerals

We embed binary representations into a generic algebraic structure using *of-num*.

class *semiring-numeral* = *semiring* + *monoid-mult*
begin

primrec *of-num* :: *num* ⇒ *'a* **where**
of-num-one [*numeral*]: *of-num One* = *1*
| *of-num* (*Dig0 n*) = *of-num n* + *of-num n*
| *of-num* (*Dig1 n*) = *of-num n* + *of-num n* + *1*

lemma *of-num-inc*: *of-num* (*inc x*) = *of-num x* + *1*
⟨*proof*⟩

declare *of-num.simps* [*simp del*]

end

ML stuff and syntax.

⟨*ML*⟩

syntax
-Numerals :: *xnum* ⇒ *'a* (-)

⟨*ML*⟩

1.4 Class-specific numeral rules

of-num is a morphism.

1.4.1 Class *semiring-numeral*

context *semiring-numeral*
begin

abbreviation *Num1* ≡ *of-num One*

Alas, there is still the duplication of *1*::*'a*, thought the duplicated *0*::*'b* has disappeared. We could get rid of it by replacing the constructor *1*::*'a* in *Numeral.num* by two constructors *two* and *three*, resulting in a further blow-up. But it could be worth the effort.

lemma *of-num-plus-one* [*numeral*]:
of-num n + *1* = *of-num* (*n* + *One*)
⟨*proof*⟩

lemma *of-num-one-plus* [*numeral*]:
1 + *of-num n* = *of-num* (*n* + *One*)

⟨*proof*⟩

lemma *of-num-plus* [*numeral*]:
of-num m + of-num n = of-num (m + n)
⟨*proof*⟩

lemma *of-num-times-one* [*numeral*]:
*of-num n * 1 = of-num n*
⟨*proof*⟩

lemma *of-num-one-times* [*numeral*]:
*1 * of-num n = of-num n*
⟨*proof*⟩

lemma *of-num-times* [*numeral*]:
*of-num m * of-num n = of-num (m * n)*
⟨*proof*⟩

end

1.4.2 Structures with a zero: class *semiring-1*

context *semiring-1*
begin

subclass *semiring-numeral* ⟨*proof*⟩

lemma *of-nat-of-num* [*numeral*]: *of-nat (of-num n) = of-num n*
⟨*proof*⟩

declare *of-nat-1* [*numeral*]

lemma *Dig-plus-zero* [*numeral*]:
0 + 1 = 1
0 + of-num n = of-num n
1 + 0 = 1
of-num n + 0 = of-num n
⟨*proof*⟩

lemma *Dig-times-zero* [*numeral*]:
*0 * 1 = 0*
*0 * of-num n = 0*
*1 * 0 = 0*
*of-num n * 0 = 0*
⟨*proof*⟩

end

lemma *nat-of-num-of-num*: *nat-of-num = of-num*

⟨*proof*⟩

1.4.3 Equality: class *semiring-char-0*

context *semiring-char-0*
begin

lemma *of-num-eq-iff* [*numeral*]:
  of-num m = *of-num n* ⟷ *m* = *n*
  ⟨*proof*⟩

lemma *of-num-eq-one-iff* [*numeral*]:
  of-num n = *1* ⟷ *n* = *One*
⟨*proof*⟩

lemma *one-eq-of-num-iff* [*numeral*]:
  1 = *of-num n* ⟷ *n* = *One*
  ⟨*proof*⟩

end

1.4.4 Comparisons: class *ordered-semidom*

Could be perhaps more general than here.

context *ordered-semidom*
begin

lemma *of-num-pos* [*numeral*]: *0* < *of-num n*
  ⟨*proof*⟩

lemma *of-num-less-eq-iff* [*numeral*]: *of-num m* ≤ *of-num n* ⟷ *m* ≤ *n*
⟨*proof*⟩

lemma *of-num-less-eq-one-iff* [*numeral*]: *of-num n* ≤ *1* ⟷ *n* = *One*
⟨*proof*⟩

lemma *one-less-eq-of-num-iff* [*numeral*]: *1* ≤ *of-num n*
⟨*proof*⟩

lemma *of-num-less-iff* [*numeral*]: *of-num m* < *of-num n* ⟷ *m* < *n*
⟨*proof*⟩

lemma *of-num-less-one-iff* [*numeral*]: ¬ *of-num n* < *1*
⟨*proof*⟩

lemma *one-less-of-num-iff* [*numeral*]: *1* < *of-num n* ⟷ *n* ≠ *One*
⟨*proof*⟩

lemma *of-num-nonneg* [*numeral*]: *0* ≤ *of-num n*

⟨*proof*⟩

lemma *of-num-less-zero-iff* [*numeral*]: ¬ *of-num n* < *0*
⟨*proof*⟩

lemma *of-num-le-zero-iff* [*numeral*]: ¬ *of-num n* ≤ *0*
⟨*proof*⟩

end

context *ordered-idom*
begin

lemma *minus-of-num-less-of-num-iff*: − *of-num m* < *of-num n*
⟨*proof*⟩

lemma *minus-of-num-less-one-iff*: − *of-num n* < *1*
⟨*proof*⟩

lemma *minus-one-less-of-num-iff*: − *1* < *of-num n*
⟨*proof*⟩

lemma *minus-one-less-one-iff*: − *1* < *1*
⟨*proof*⟩

lemma *minus-of-num-le-of-num-iff*: − *of-num m* ≤ *of-num n*
⟨*proof*⟩

lemma *minus-of-num-le-one-iff*: − *of-num n* ≤ *1*
⟨*proof*⟩

lemma *minus-one-le-of-num-iff*: − *1* ≤ *of-num n*
⟨*proof*⟩

lemma *minus-one-le-one-iff*: − *1* ≤ *1*
⟨*proof*⟩

lemma *of-num-le-minus-of-num-iff*: ¬ *of-num m* ≤ − *of-num n*
⟨*proof*⟩

lemma *one-le-minus-of-num-iff*: ¬ *1* ≤ − *of-num n*
⟨*proof*⟩

lemma *of-num-le-minus-one-iff*: ¬ *of-num n* ≤ − *1*
⟨*proof*⟩

lemma *one-le-minus-one-iff*: ¬ *1* ≤ − *1*
⟨*proof*⟩

lemma *of-num-less-minus-of-num-iff*: $\neg$ *of-num m* $<$ $-$ *of-num n*
⟨*proof*⟩

lemma *one-less-minus-of-num-iff*: $\neg$ *1* $<$ $-$ *of-num n*
⟨*proof*⟩

lemma *of-num-less-minus-one-iff*: $\neg$ *of-num n* $<$ $-$ *1*
⟨*proof*⟩

lemma *one-less-minus-one-iff*: $\neg$ *1* $<$ $-$ *1*
⟨*proof*⟩

lemmas *le-signed-numeral-special* [*numeral*] =
minus-of-num-le-of-num-iff
minus-of-num-le-one-iff
minus-one-le-of-num-iff
minus-one-le-one-iff
of-num-le-minus-of-num-iff
one-le-minus-of-num-iff
of-num-le-minus-one-iff
one-le-minus-one-iff

lemmas *less-signed-numeral-special* [*numeral*] =
minus-of-num-less-of-num-iff
minus-of-num-less-one-iff
minus-one-less-of-num-iff
minus-one-less-one-iff
of-num-less-minus-of-num-iff
one-less-minus-of-num-iff
of-num-less-minus-one-iff
one-less-minus-one-iff

end

1.4.5 Structures with subtraction: class *semiring-1-minus*

class *semiring-minus* = *semiring* + *minus* + *zero* +
assumes *minus-inverts-plus1*: $a + b = c \Longrightarrow c - b = a$
assumes *minus-minus-zero-inverts-plus1*: $a + b = c \Longrightarrow b - c = 0 - a$
begin

lemma *minus-inverts-plus2*: $a + b = c \Longrightarrow c - a = b$
⟨*proof*⟩

lemma *minus-minus-zero-inverts-plus2*: $a + b = c \Longrightarrow a - c = 0 - b$
⟨*proof*⟩

end

class *semiring-1-minus* = *semiring-1* + *semiring-minus*
begin

lemma *Dig-of-num-pos*:
  assumes $k + n = m$
  shows *of-num* m − *of-num* n = *of-num* k
  ⟨*proof*⟩

lemma *Dig-of-num-zero*:
  shows *of-num* n − *of-num* n = *0*
  ⟨*proof*⟩

lemma *Dig-of-num-neg*:
  assumes $k + m = n$
  shows *of-num* m − *of-num* n = *0* − *of-num* k
  ⟨*proof*⟩

lemmas *Dig-plus-eval* =
  of-num-plus of-num-eq-iff Dig-plus refl [*of One*, *THEN eqTrueI*] *num.inject*

⟨*ML*⟩

lemma *Dig-of-num-minus-zero* [*numeral*]:
  of-num n − *0* = *of-num* n
  ⟨*proof*⟩

lemma *Dig-one-minus-zero* [*numeral*]:
  1 − *0* = *1*
  ⟨*proof*⟩

lemma *Dig-one-minus-one* [*numeral*]:
  1 − *1* = *0*
  ⟨*proof*⟩

lemma *Dig-of-num-minus-one* [*numeral*]:
  of-num (*Dig0* n) − *1* = *of-num* (*DigM* n)
  of-num (*Dig1* n) − *1* = *of-num* (*Dig0* n)
  ⟨*proof*⟩

lemma *Dig-one-minus-of-num* [*numeral*]:
  1 − *of-num* (*Dig0* n) = *0* − *of-num* (*DigM* n)
  1 − *of-num* (*Dig1* n) = *0* − *of-num* (*Dig0* n)
  ⟨*proof*⟩

end

1.4.6 Structures with negation: class *ring-1*

context *ring-1*

begin

subclass *semiring-1-minus*
⟨*proof*⟩

lemma *Dig-zero-minus-of-num* [*numeral*]:
0 − of-num n = − of-num n
⟨*proof*⟩

lemma *Dig-zero-minus-one* [*numeral*]:
0 − 1 = − 1
⟨*proof*⟩

lemma *Dig-uminus-uminus* [*numeral*]:
− (− of-num n) = of-num n
⟨*proof*⟩

lemma *Dig-plus-uminus* [*numeral*]:
of-num m + − of-num n = of-num m − of-num n
− of-num m + of-num n = of-num n − of-num m
− of-num m + − of-num n = − (of-num m + of-num n)
of-num m − − of-num n = of-num m + of-num n
− of-num m − of-num n = − (of-num m + of-num n)
− of-num m − − of-num n = of-num n − of-num m
⟨*proof*⟩

lemma *Dig-times-uminus* [*numeral*]:
− of-num n ∗ of-num m = − (of-num n ∗ of-num m)
of-num n ∗ − of-num m = − (of-num n ∗ of-num m)
− of-num n ∗ − of-num m = of-num n ∗ of-num m
⟨*proof*⟩

lemma *of-int-of-num* [*numeral*]: *of-int* (*of-num n*) = *of-num n*
⟨*proof*⟩

declare *of-int-1* [*numeral*]

end

1.4.7 Structures with exponentiation

lemma *of-num-square*: *of-num* (*square x*) = *of-num x ∗ of-num x*
⟨*proof*⟩

lemma *of-num-pow*:
(*of-num* (*pow x y*)::′*a*::{*semiring-numeral*,*recpower*}) = *of-num x ^ of-num y*
⟨*proof*⟩

lemma *power-of-num* [*numeral*]:

of-num x ^ of-num y = (of-num (pow x y)::'a::{semiring-numeral,recpower})
⟨*proof*⟩

lemma *power-zero-of-num* [*numeral*]:
0 ^ of-num n = (0::'a::{semiring-0,recpower})
⟨*proof*⟩

lemma *power-minus-one-double*:
(− 1) ^ (n + n) = (1::'a::{ring-1,recpower})
⟨*proof*⟩

lemma *power-minus-Dig0* [*numeral*]:
fixes *x :: 'a::{ring-1,recpower}*
shows *(− x) ^ of-num (Dig0 n) = x ^ of-num (Dig0 n)*
⟨*proof*⟩

lemma *power-minus-Dig1* [*numeral*]:
fixes *x :: 'a::{ring-1,recpower}*
shows *(− x) ^ of-num (Dig1 n) = − (x ^ of-num (Dig1 n))*
⟨*proof*⟩

declare *power-one* [*numeral*]

1.4.8 Greetings to *nat*.

instance *nat :: semiring-1-minus* ⟨*proof*⟩

lemma *Suc-of-num* [*numeral*]: *Suc (of-num n) = of-num (n + One)*
⟨*proof*⟩

lemma *nat-number*:
1 = Suc 0
of-num One = Suc 0
of-num (Dig0 n) = Suc (of-num (DigM n))
of-num (Dig1 n) = Suc (of-num (Dig0 n))
⟨*proof*⟩

declare *diff-0-eq-0* [*numeral*]

1.5 Code generator setup for *int*

definition *Pls :: num ⇒ int* **where**
[*simp*, *code post*]: *Pls n = of-num n*

definition *Mns :: num ⇒ int* **where**
[*simp*, *code post*]: *Mns n = − of-num n*

code-datatype *0::int Pls Mns*

lemmas [*code inline*] = *Pls-def* [*symmetric*] *Mns-def* [*symmetric*]

definition *sub* :: *num* ⇒ *num* ⇒ *int* **where**
[*simp*, *code del*]: *sub m n* = (*of-num m* − *of-num n*)

definition *dup* :: *int* ⇒ *int* **where**
[*code del*]: *dup k* = *2* ∗ *k*

lemma *Dig-sub* [*code*]:
sub One One = *0*
sub (*Dig0 m*) *One* = *of-num* (*DigM m*)
sub (*Dig1 m*) *One* = *of-num* (*Dig0 m*)
sub One (*Dig0 n*) = − *of-num* (*DigM n*)
sub One (*Dig1 n*) = − *of-num* (*Dig0 n*)
sub (*Dig0 m*) (*Dig0 n*) = *dup* (*sub m n*)
sub (*Dig1 m*) (*Dig1 n*) = *dup* (*sub m n*)
sub (*Dig1 m*) (*Dig0 n*) = *dup* (*sub m n*) + *1*
sub (*Dig0 m*) (*Dig1 n*) = *dup* (*sub m n*) − *1*
⟨*proof*⟩

lemma *dup-code* [*code*]:
dup 0 = *0*
dup (*Pls n*) = *Pls* (*Dig0 n*)
dup (*Mns n*) = *Mns* (*Dig0 n*)
⟨*proof*⟩

lemma [*code*, *code del*]:
(*1* :: *int*) = *1*
(*op* + :: *int* ⇒ *int* ⇒ *int*) = *op* +
(*uminus* :: *int* ⇒ *int*) = *uminus*
(*op* − :: *int* ⇒ *int* ⇒ *int*) = *op* −
(*op* ∗ :: *int* ⇒ *int* ⇒ *int*) = *op* ∗
(*eq-class.eq* :: *int* ⇒ *int* ⇒ *bool*) = *eq-class.eq*
(*op* ≤ :: *int* ⇒ *int* ⇒ *bool*) = *op* ≤
(*op* < :: *int* ⇒ *int* ⇒ *bool*) = *op* <
⟨*proof*⟩

lemma *one-int-code* [*code*]:
1 = *Pls One*
⟨*proof*⟩

lemma *plus-int-code* [*code*]:
k + *0* = (*k*::*int*)
0 + *l* = (*l*::*int*)
Pls m + *Pls n* = *Pls* (*m* + *n*)
Pls m − *Pls n* = *sub m n*
Mns m + *Mns n* = *Mns* (*m* + *n*)
Mns m − *Mns n* = *sub n m*
⟨*proof*⟩

lemma *uminus-int-code* [*code*]:
uminus 0 = (0::int)
uminus (Pls m) = Mns m
uminus (Mns m) = Pls m
⟨*proof*⟩

lemma *minus-int-code* [*code*]:
k − 0 = (k::int)
0 − l = uminus (l::int)
Pls m − Pls n = sub m n
Pls m − Mns n = Pls (m + n)
Mns m − Pls n = Mns (m + n)
Mns m − Mns n = sub n m
⟨*proof*⟩

lemma *times-int-code* [*code*]:
k ∗ 0 = (0::int)
0 ∗ l = (0::int)
Pls m ∗ Pls n = Pls (m ∗ n)
Pls m ∗ Mns n = Mns (m ∗ n)
Mns m ∗ Pls n = Mns (m ∗ n)
Mns m ∗ Mns n = Pls (m ∗ n)
⟨*proof*⟩

lemma *eq-int-code* [*code*]:
eq-class.eq 0 (0::int) ⟷ True
eq-class.eq 0 (Pls l) ⟷ False
eq-class.eq 0 (Mns l) ⟷ False
eq-class.eq (Pls k) 0 ⟷ False
eq-class.eq (Pls k) (Pls l) ⟷ eq-class.eq k l
eq-class.eq (Pls k) (Mns l) ⟷ False
eq-class.eq (Mns k) 0 ⟷ False
eq-class.eq (Mns k) (Pls l) ⟷ False
eq-class.eq (Mns k) (Mns l) ⟷ eq-class.eq k l
⟨*proof*⟩

lemma *less-eq-int-code* [*code*]:
0 ≤ (0::int) ⟷ True
0 ≤ Pls l ⟷ True
0 ≤ Mns l ⟷ False
Pls k ≤ 0 ⟷ False
Pls k ≤ Pls l ⟷ k ≤ l
Pls k ≤ Mns l ⟷ False
Mns k ≤ 0 ⟷ True
Mns k ≤ Pls l ⟷ True
Mns k ≤ Mns l ⟷ l ≤ k
⟨*proof*⟩

lemma *less-int-code* [*code*]:

$0 < (0::int) \longleftrightarrow False$
$0 < Pls\ l \longleftrightarrow True$
$0 < Mns\ l \longleftrightarrow False$
$Pls\ k < 0 \longleftrightarrow False$
$Pls\ k < Pls\ l \longleftrightarrow k < l$
$Pls\ k < Mns\ l \longleftrightarrow False$
$Mns\ k < 0 \longleftrightarrow True$
$Mns\ k < Pls\ l \longleftrightarrow True$
$Mns\ k < Mns\ l \longleftrightarrow l < k$
⟨*proof*⟩

lemma [*code inline del*]: $(0::int) \equiv Numeral0$ ⟨*proof*⟩
lemma [*code inline del*]: $(1::int) \equiv Numeral1$ ⟨*proof*⟩
declare *zero-is-num-zero* [*code inline del*]
declare *one-is-num-one* [*code inline del*]

hide (**open**) *const sub dup*

1.6 Numeral equations as default simplification rules

TODO. Be more precise here with respect to subsumed facts. Or use named theorems anyway.

declare (**in** *semiring-numeral*) *numeral* [*simp*]
declare (**in** *semiring-1*) *numeral* [*simp*]
declare (**in** *semiring-char-0*) *numeral* [*simp*]
declare (**in** *ring-1*) *numeral* [*simp*]
thm *numeral*

Toy examples

definition $bar \longleftrightarrow \#4 * \#2 + \#7 = (\#8 :: nat) \wedge \#4 * \#2 + \#7 \geq (\#8 :: int) - \#3$
code-thms *bar*
export-code *bar* **in** *Haskell* **file** −
export-code *bar* **in** *OCaml* **module-name** *Foo* **file** −
⟨*ML*⟩

end

2 Foundations of HOL

theory *Higher-Order-Logic* **imports** *Pure* **begin**

The following theory development demonstrates Higher-Order Logic itself, represented directly within the Pure framework of Isabelle. The "HOL" logic given here is essentially that of Gordon [1], although we prefer to present basic concepts in a slightly more conventional manner oriented towards plain Natural Deduction.

2.1 Pure Logic

classes *type*
defaultsort *type*

typedecl *o*
arities
o :: *type*
fun :: (*type*, *type*) *type*

2.1.1 Basic logical connectives

judgment
Trueprop :: $o \Rightarrow prop$ (- *5*)

axiomatization
imp :: $o \Rightarrow o \Rightarrow o$ (**infixr** $\longrightarrow$ *25*) **and**
All :: $('a \Rightarrow o) \Rightarrow o$ (**binder** $\forall$ *10*)
where
impI [*intro*]: $(A \Longrightarrow B) \Longrightarrow A \longrightarrow B$ **and**
impE [*dest*, *trans*]: $A \longrightarrow B \Longrightarrow A \Longrightarrow B$ **and**
allI [*intro*]: $(\bigwedge x.\ P\ x) \Longrightarrow \forall x.\ P\ x$ **and**
allE [*dest*]: $\forall x.\ P\ x \Longrightarrow P\ a$

2.1.2 Extensional equality

axiomatization
equal :: $'a \Rightarrow 'a \Rightarrow o$ (**infixl** = *50*)
where
refl [*intro*]: $x = x$ **and**
subst: $x = y \Longrightarrow P\ x \Longrightarrow P\ y$

axiomatization where
ext [*intro*]: $(\bigwedge x.\ f\ x = g\ x) \Longrightarrow f = g$ **and**
iff [*intro*]: $(A \Longrightarrow B) \Longrightarrow (B \Longrightarrow A) \Longrightarrow A = B$

theorem *sym* [*sym*]: $x = y \Longrightarrow y = x$
⟨*proof*⟩

lemma [*trans*]: $x = y \Longrightarrow P\ y \Longrightarrow P\ x$
⟨*proof*⟩

lemma [*trans*]: $P\ x \Longrightarrow x = y \Longrightarrow P\ y$
⟨*proof*⟩

theorem *trans* [*trans*]: $x = y \Longrightarrow y = z \Longrightarrow x = z$
⟨*proof*⟩

theorem *iff1* [*elim*]: $A = B \Longrightarrow A \Longrightarrow B$
⟨*proof*⟩

theorem *iff2* [*elim*]: $A = B \Longrightarrow B \Longrightarrow A$
⟨*proof*⟩

2.1.3 Derived connectives

definition
$\mathit{false} :: o$ $(\bot)$ **where**
$\bot \equiv \forall A.\ A$

definition
$\mathit{true} :: o$ $(\top)$ **where**
$\top \equiv \bot \longrightarrow \bot$

definition
$\mathit{not} :: o \Rightarrow o$ $(\neg\ \text{-}\ [40]\ 40)$ **where**
$\mathit{not} \equiv \lambda A.\ A \longrightarrow \bot$

definition
$\mathit{conj} :: o \Rightarrow o \Rightarrow o$ (**infixr** $\wedge$ *35*) **where**
$\mathit{conj} \equiv \lambda A\ B.\ \forall C.\ (A \longrightarrow B \longrightarrow C) \longrightarrow C$

definition
$\mathit{disj} :: o \Rightarrow o \Rightarrow o$ (**infixr** $\vee$ *30*) **where**
$\mathit{disj} \equiv \lambda A\ B.\ \forall C.\ (A \longrightarrow C) \longrightarrow (B \longrightarrow C) \longrightarrow C$

definition
$\mathit{Ex} :: ('a \Rightarrow o) \Rightarrow o$ (**binder** $\exists$ *10*) **where**
$\exists x.\ P\ x \equiv \forall C.\ (\forall x.\ P\ x \longrightarrow C) \longrightarrow C$

abbreviation
$\mathit{not\text{-}equal} :: 'a \Rightarrow 'a \Rightarrow o$ (**infixl** $\neq$ *50*) **where**
$x \neq y \equiv \neg\ (x = y)$

theorem *falseE* [*elim*]: $\bot \Longrightarrow A$
⟨*proof*⟩

theorem *trueI* [*intro*]: $\top$
⟨*proof*⟩

theorem *notI* [*intro*]: $(A \Longrightarrow \bot) \Longrightarrow \neg\ A$
⟨*proof*⟩

theorem *notE* [*elim*]: $\neg\ A \Longrightarrow A \Longrightarrow B$
⟨*proof*⟩

lemma *notE'*: $A \Longrightarrow \neg\ A \Longrightarrow B$
⟨*proof*⟩

lemmas *contradiction* = *notE notE′* — proof by contradiction in any order

theorem *conjI* [*intro*]: $A \Longrightarrow B \Longrightarrow A \wedge B$
⟨*proof*⟩

theorem *conjE* [*elim*]: $A \wedge B \Longrightarrow (A \Longrightarrow B \Longrightarrow C) \Longrightarrow C$
⟨*proof*⟩

theorem *disjI1* [*intro*]: $A \Longrightarrow A \vee B$
⟨*proof*⟩

theorem *disjI2* [*intro*]: $B \Longrightarrow A \vee B$
⟨*proof*⟩

theorem *disjE* [*elim*]: $A \vee B \Longrightarrow (A \Longrightarrow C) \Longrightarrow (B \Longrightarrow C) \Longrightarrow C$
⟨*proof*⟩

theorem *exI* [*intro*]: $P\ a \Longrightarrow \exists x.\ P\ x$
⟨*proof*⟩

theorem *exE* [*elim*]: $\exists x.\ P\ x \Longrightarrow (\bigwedge x.\ P\ x \Longrightarrow C) \Longrightarrow C$
⟨*proof*⟩

2.2 Classical logic

locale *classical* =
  assumes *classical*: $(\neg\ A \Longrightarrow A) \Longrightarrow A$

theorem (**in** *classical*)
  Peirce′s-Law: $((A \longrightarrow B) \longrightarrow A) \longrightarrow A$
⟨*proof*⟩

theorem (**in** *classical*)
  double-negation: $\neg\ \neg\ A \Longrightarrow A$
⟨*proof*⟩

theorem (**in** *classical*)
  tertium-non-datur: $A \vee \neg\ A$
⟨*proof*⟩

theorem (**in** *classical*)
  classical-cases: $(A \Longrightarrow C) \Longrightarrow (\neg\ A \Longrightarrow C) \Longrightarrow C$
⟨*proof*⟩

lemma (**in** *classical*) $(\neg\ A \Longrightarrow A) \Longrightarrow A$
⟨*proof*⟩

end

3 Abstract Natural Numbers primitive recursion

theory *Abstract-NAT*
imports *Main*
begin

Axiomatic Natural Numbers (Peano) – a monomorphic theory.

locale *NAT* =
  fixes *zero* :: $'n$
    and *succ* :: $'n \Rightarrow 'n$
  assumes *succ-inject* [*simp*]: $(succ\ m = succ\ n) = (m = n)$
    and *succ-neq-zero* [*simp*]: $succ\ m \neq zero$
    and *induct* [*case-names zero succ*, *induct type*: $'n$]:
      $P\ zero \Longrightarrow (\bigwedge n.\ P\ n \Longrightarrow P\ (succ\ n)) \Longrightarrow P\ n$
begin

lemma *zero-neq-succ* [*simp*]: $zero \neq succ\ m$
  ⟨*proof*⟩

Primitive recursion as a (functional) relation – polymorphic!

inductive
  Rec :: $'a \Rightarrow ('n \Rightarrow 'a \Rightarrow 'a) \Rightarrow 'n \Rightarrow 'a \Rightarrow bool$
  for $e :: 'a$ **and** $r :: 'n \Rightarrow 'a \Rightarrow 'a$
where
    Rec-zero: *Rec e r zero e*
  | *Rec-succ*: $Rec\ e\ r\ m\ n \Longrightarrow Rec\ e\ r\ (succ\ m)\ (r\ m\ n)$

lemma *Rec-functional*:
  fixes $x :: 'n$
  shows $\exists!y{::}'a.\ Rec\ e\ r\ x\ y$
⟨*proof*⟩

The recursion operator – polymorphic!

definition
  rec :: $'a \Rightarrow ('n \Rightarrow 'a \Rightarrow 'a) \Rightarrow 'n \Rightarrow 'a$ **where**
  $rec\ e\ r\ x = (THE\ y.\ Rec\ e\ r\ x\ y)$

lemma *rec-eval*:
  assumes *Rec*: *Rec e r x y*
  shows $rec\ e\ r\ x = y$
  ⟨*proof*⟩

lemma *rec-zero* [*simp*]: $rec\ e\ r\ zero = e$
⟨*proof*⟩

lemma *rec-succ* [*simp*]: $rec\ e\ r\ (succ\ m) = r\ m\ (rec\ e\ r\ m)$
⟨*proof*⟩

Example: addition (monomorphic)

definition
add :: *'n* ⇒ *'n* ⇒ *'n* **where**
add m n = *rec n* (λ- *k*. *succ k*) *m*

lemma *add-zero* [*simp*]: *add zero n* = *n*
and *add-succ* [*simp*]: *add* (*succ m*) *n* = *succ* (*add m n*)
⟨*proof*⟩

lemma *add-assoc*: *add* (*add k m*) *n* = *add k* (*add m n*)
⟨*proof*⟩

lemma *add-zero-right*: *add m zero* = *m*
⟨*proof*⟩

lemma *add-succ-right*: *add m* (*succ n*) = *succ* (*add m n*)
⟨*proof*⟩

lemma *add* (*succ* (*succ* (*succ zero*))) (*succ* (*succ zero*)) =
succ (*succ* (*succ* (*succ* (*succ zero*))))
⟨*proof*⟩

Example: replication (polymorphic)

definition
repl :: *'n* ⇒ *'a* ⇒ *'a list* **where**
repl n x = *rec* [] (λ- *xs*. *x* # *xs*) *n*

lemma *repl-zero* [*simp*]: *repl zero x* = []
and *repl-succ* [*simp*]: *repl* (*succ n*) *x* = *x* # *repl n x*
⟨*proof*⟩

lemma *repl* (*succ* (*succ* (*succ zero*))) *True* = [*True*, *True*, *True*]
⟨*proof*⟩

end

Just see that our abstract specification makes sense ...

interpretation *NAT 0 Suc*
⟨*proof*⟩

end

4 Proof by guessing

theory *Guess*
imports *Main*

begin

lemma *True*
⟨*proof*⟩

end

5 Simple and efficient binary numerals

theory *Binary*
imports *Main*
begin

5.1 Binary representation of natural numbers

definition
  bit :: *nat* ⇒ *bool* ⇒ *nat* **where**
  bit n b = (*if b then 2* ∗ *n* + *1 else 2* ∗ *n*)

lemma *bit-simps*:
    bit n False = *2* ∗ *n*
    bit n True = *2* ∗ *n* + *1*
  ⟨*proof*⟩

⟨*ML*⟩

5.2 Direct operations – plain normalization

lemma *binary-norm*:
    bit 0 False = *0*
    bit 0 True = *1*
  ⟨*proof*⟩

lemma *binary-add*:
    n + *0* = *n*
    0 + *n* = *n*
    1 + *1* = *bit 1 False*
    bit n False + *1* = *bit n True*
    bit n True + *1* = *bit* (*n* + *1*) *False*
    1 + *bit n False* = *bit n True*
    1 + *bit n True* = *bit* (*n* + *1*) *False*
    bit m False + *bit n False* = *bit* (*m* + *n*) *False*
    bit m False + *bit n True* = *bit* (*m* + *n*) *True*
    bit m True + *bit n False* = *bit* (*m* + *n*) *True*
    bit m True + *bit n True* = *bit* ((*m* + *n*) + *1*) *False*
  ⟨*proof*⟩

lemma *binary-mult*:

$n * 0 = 0$
$0 * n = 0$
$n * 1 = n$
$1 * n = n$
$bit\ m\ True * n = bit\ (m * n)\ False + n$
$bit\ m\ False * n = bit\ (m * n)\ False$
$n * bit\ m\ True = bit\ (m * n)\ False + n$
$n * bit\ m\ False = bit\ (m * n)\ False$
⟨*proof*⟩

lemmas *binary-simps* = *binary-norm binary-add binary-mult*

5.3 Indirect operations – ML will produce witnesses

lemma *binary-less-eq*:
fixes $n :: nat$
shows $n \equiv m + k \Longrightarrow (m \leq n) \equiv True$
and $m \equiv n + k + 1 \Longrightarrow (m \leq n) \equiv False$
⟨*proof*⟩

lemma *binary-less*:
fixes $n :: nat$
shows $m \equiv n + k \Longrightarrow (m < n) \equiv False$
and $n \equiv m + k + 1 \Longrightarrow (m < n) \equiv True$
⟨*proof*⟩

lemma *binary-diff*:
fixes $n :: nat$
shows $m \equiv n + k \Longrightarrow m - n \equiv k$
and $n \equiv m + k \Longrightarrow m - n \equiv 0$
⟨*proof*⟩

lemma *binary-divmod*:
fixes $n :: nat$
assumes $m \equiv n * k + l$ **and** $0 < n$ **and** $l < n$
shows $m\ div\ n \equiv k$
and $m\ mod\ n \equiv l$
⟨*proof*⟩

⟨*ML*⟩

5.4 Concrete syntax

syntax
-Binary :: *num-const* ⇒ *'a* ($-)

⟨*ML*⟩

5.5 Examples

lemma $\$6 = 6$
⟨*proof*⟩

lemma $bit\ (bit\ (bit\ 0\ False)\ False)\ True = 1$
⟨*proof*⟩

lemma $bit\ (bit\ (bit\ 0\ False)\ False)\ True = bit\ 0\ True$
⟨*proof*⟩

lemma $\$5 + \$3 = \$8$
⟨*proof*⟩

lemma $\$5 * \$3 = \$15$
⟨*proof*⟩

lemma $\$5 - \$3 = \$2$
⟨*proof*⟩

lemma $\$3 - \$5 = 0$
⟨*proof*⟩

lemma $\$123456789 - \$123 = \$123456666$
⟨*proof*⟩

lemma $\$11111111112222222222333333333344444444444 - \$998877665544332211$
=
$\$11111111112222222222323344556689001122 33$
⟨*proof*⟩

lemma $(11111111112222222222333333333344444444444::nat) - 998877665544332211$
=
$11111111112222222222323344556689001122 33$
⟨*proof*⟩

lemma $(11111111112222222222333333333344444444444::int) - 998877665544332211$
=
$11111111112222222222323344556689001122 33$
⟨*proof*⟩

lemma $\$11111111112222222222333333333344444444444 * \$998877665544332211$
=
$\$1109864072938022197293802219729380221972383090160869185684$
⟨*proof*⟩

lemma $\$11111111112222222222333333333344444444444 * \$998877665544332211$
−
$\$55555555556666666666777777777788888888888 =$
$\$1109864072938022191738246664062713555294605312381980296796$

⟨*proof*⟩

lemma $*42* < $*4* = *False*
⟨*proof*⟩

lemma $*4* < $*42* = *True*
⟨*proof*⟩

lemma $*42* <= $*4* = *False*
⟨*proof*⟩

lemma $*4* <= $*42* = *True*
⟨*proof*⟩

lemma $*1111111111222222222233333333334444444444* < $*9988776655443322 11*
= *False*
⟨*proof*⟩

lemma $*9988776655443322 11* < $*1111111111222222222233333333334444444444*
= *True*
⟨*proof*⟩

lemma $*1111111111222222222233333333334444444444* <= $*9988776655443322 11*
= *False*
⟨*proof*⟩

lemma $*9988776655443322 11* <= $*1111111111222222222233333333334444444444*
= *True*
⟨*proof*⟩

lemma $*1234* *div* $*23* = $*53*
⟨*proof*⟩

lemma $*1234* *mod* $*23* = $*15*
⟨*proof*⟩

lemma $*1111111111222222222233333333334444444444* *div* $*9988776655443322 11*
=
$*11123595506730337078 75*
⟨*proof*⟩

lemma $*1111111111222222222233333333334444444444* *mod* $*9988776655443322 11*
=
$*422451743175828 19*
⟨*proof*⟩

lemma (*1111111111222222222233333333334444444444*::*int*) *div* *9988776655443322 11*
=
11123595506730337078 75

⟨*proof*⟩

lemma (*11111111112222222222333333333344444444444*::*int*) *mod 9988776655443322111*
=
422451743175828l9
⟨*proof*⟩

end

6 Examples of recdef definitions

theory *Recdefs* **imports** *Main* **begin**

consts *fact* :: *nat* => *nat*
recdef *fact* *less-than*
fact x = (*if x* = *0 then 1 else x* ∗ *fact* (*x* − *1*))

consts *Fact* :: *nat* => *nat*
recdef *Fact* *less-than*
Fact 0 = *1*
Fact (*Suc x*) = *Fact x* ∗ *Suc x*

consts *fib* :: *int* => *int*
recdef *fib* *measure nat*
eqn: *fib n* = (*if n* < *1 then 0*
else if n=*1 then 1*
else fib(*n* − *2*) + *fib*(*n* − *1*))

lemma *fib 7* = *13*
⟨*proof*⟩

consts *map2* :: ('*a* => '*b* => '*c*) ∗ '*a list* ∗ '*b list* => '*c list*
recdef *map2* *measure*(λ(*f*, *l1*, *l2*). *size l1*)
map2 (*f*, [], []) = []
map2 (*f*, *h* # *t*, []) = []
map2 (*f*, *h1* # *t1*, *h2* # *t2*) = *f h1 h2* # *map2* (*f*, *t1*, *t2*)

consts *finiteRchain* :: ('*a* => '*a* => *bool*) ∗ '*a list* => *bool*
recdef *finiteRchain* *measure* (λ(*R*, *l*). *size l*)
finiteRchain(*R*, []) = *True*
finiteRchain(*R*, [*x*]) = *True*
finiteRchain(*R*, *x* # *y* # *rst*) = (*R x y* ∧ *finiteRchain* (*R*, *y* # *rst*))

Not handled automatically: too complicated.

consts *variant* :: *nat* ∗ *nat list* => *nat*
recdef (**permissive**) *variant measure* (λ(*n*,*ns*). *size* (*filter* (λ*y*. *n* ≤ *y*) *ns*))
variant (*x*, *L*) = (*if x mem L then variant* (*Suc x*, *L*) *else x*)

consts *gcd* :: *nat* ∗ *nat* => *nat*
recdef *gcd* *measure* (λ(*x*, *y*). *x* + *y*)
gcd (*0*, *y*) = *y*
gcd (*Suc x*, *0*) = *Suc x*
gcd (*Suc x*, *Suc y*) =
(*if y* ≤ *x then gcd* (*x* − *y*, *Suc y*) *else gcd* (*Suc x*, *y* − *x*))

The silly *g* function: example of nested recursion. Not handled automatically. In fact, *g* is the zero constant function.

consts *g* :: *nat* => *nat*
recdef (**permissive**) *g* *less-than*
g 0 = *0*
g (*Suc x*) = *g* (*g x*)

lemma *g-terminates*: *g x* < *Suc x*
⟨*proof*⟩

lemma *g-zero*: *g x* = *0*
⟨*proof*⟩

consts *Div* :: *nat* ∗ *nat* => *nat* ∗ *nat*
recdef *Div* *measure fst*
Div (*0*, *x*) = (*0*, *0*)
Div (*Suc x*, *y*) =
(*let* (*q*, *r*) = *Div* (*x*, *y*)
in if y ≤ *Suc r then* (*Suc q*, *0*) *else* (*q*, *Suc r*))

Not handled automatically. Should be the predecessor function, but there is an unnecessary "looping" recursive call in *k 1*.

consts *k* :: *nat* => *nat*

recdef (**permissive**) *k* *less-than*
k 0 = *0*
k (*Suc n*) =
(*let x* = *k 1*
in if False then k (*Suc 1*) *else n*)

consts *part* :: ('*a* => *bool*) ∗ '*a list* ∗ '*a list* ∗ '*a list* => '*a list* ∗ '*a list*
recdef *part* *measure* (λ(*P*, *l*, *l1*, *l2*). *size l*)
part (*P*, [], *l1*, *l2*) = (*l1*, *l2*)
part (*P*, *h* # *rst*, *l1*, *l2*) =
(*if P h then part* (*P*, *rst*, *h* # *l1*, *l2*)
else part (*P*, *rst*, *l1*, *h* # *l2*))

consts *fqsort* :: ('*a* => '*a* => *bool*) ∗ '*a list* => '*a list*
recdef (**permissive**) *fqsort* *measure* (*size o snd*)

fqsort (*ord*, []) = []
fqsort (*ord*, *x* # *rst*) =
(*let* (*less*, *more*) = *part* ((λ*y*. *ord y x*), *rst*, ([], []))
in fqsort (*ord*, *less*) @ [*x*] @ *fqsort* (*ord*, *more*))

Silly example which demonstrates the occasional need for additional congruence rules (here: *map-cong*). If the congruence rule is removed, an unprovable termination condition is generated! Termination not proved automatically. TFL requires λ*x*. *mapf x* instead of *mapf*.

consts *mapf* :: *nat* => *nat list*
recdef (**permissive**) *mapf measure* (λ*m*. *m*)
mapf 0 = []
mapf (*Suc n*) = *concat* (*map* (λ*x*. *mapf x*) (*replicate n n*))
(**hints** *cong*: *map-cong*)

recdef-tc *mapf-tc*: *mapf*
⟨*proof*⟩

Removing the termination condition from the generated thms:

lemma *mapf* (*Suc n*) = *concat* (*map mapf* (*replicate n n*))
⟨*proof*⟩

lemmas *mapf-induct* = *mapf.induct* [*OF mapf-tc*]

end

7 Examples of function definitions

theory *Fundefs*
imports *Main*
begin

7.1 Very basic

fun *fib* :: *nat* ⇒ *nat*
where
fib 0 = *1*
| *fib* (*Suc 0*) = *1*
| *fib* (*Suc* (*Suc n*)) = *fib n* + *fib* (*Suc n*)

partial simp and induction rules:

thm *fib.psimps*
thm *fib.pinduct*

There is also a cases rule to distinguish cases along the definition

thm *fib.cases*

total simp and induction rules:

thm *fib.simps*
thm *fib.induct*

7.2 Currying

fun *add*
where
  add 0 y = y
| *add (Suc x) y = Suc (add x y)*

thm *add.simps*
thm *add.induct* — Note the curried induction predicate

7.3 Nested recursion

function *nz*
where
  nz 0 = 0
| *nz (Suc x) = nz (nz x)*
⟨*proof*⟩

lemma *nz-is-zero*: — A lemma we need to prove termination
  assumes *trm*: *nz-dom x*
  shows *nz x = 0*
⟨*proof*⟩

termination *nz*
  ⟨*proof*⟩

thm *nz.simps*
thm *nz.induct*

Here comes McCarthy's 91-function

function *f91* :: *nat* => *nat*
where
  f91 n = (if 100 < n then n − 10 else f91 (f91 (n + 11)))
⟨*proof*⟩

lemma *f91-estimate*:
  assumes *trm*: *f91-dom n*
  shows *n < f91 n + 11*
⟨*proof*⟩

termination
⟨*proof*⟩

Now trivial (even though it does not belong here):

lemma *f91 n = (if 100 < n then n − 10 else 91)*
⟨*proof*⟩

7.4 More general patterns

7.4.1 Overlapping patterns

Currently, patterns must always be compatible with each other, since no automatic splitting takes place. But the following definition of gcd is ok, although patterns overlap:

fun *gcd2* :: *nat* ⇒ *nat* ⇒ *nat*
where
  gcd2 x 0 = x
| *gcd2 0 y = y*
| *gcd2 (Suc x) (Suc y) = (if x < y then gcd2 (Suc x) (y − x)*
else gcd2 (x − y) (Suc y))

thm *gcd2.simps*
thm *gcd2.induct*

7.4.2 Guards

We can reformulate the above example using guarded patterns

function *gcd3* :: *nat* ⇒ *nat* ⇒ *nat*
where
  gcd3 x 0 = x
| *gcd3 0 y = y*
| *x < y ⟹ gcd3 (Suc x) (Suc y) = gcd3 (Suc x) (y − x)*
| *¬ x < y ⟹ gcd3 (Suc x) (Suc y) = gcd3 (x − y) (Suc y)*
  ⟨*proof*⟩
termination ⟨*proof*⟩

thm *gcd3.simps*
thm *gcd3.induct*

General patterns allow even strange definitions:

function *ev* :: *nat* ⇒ *bool*
where
  ev (2 ∗ n) = True
| *ev (2 ∗ n + 1) = False*
⟨*proof*⟩
termination ⟨*proof*⟩

thm *ev.simps*
thm *ev.induct*
thm *ev.cases*

7.5 Mutual Recursion

fun *evn od* :: *nat* ⇒ *bool*
where
  evn 0 = *True*
| *od 0* = *False*
| *evn* (*Suc n*) = *od n*
| *od* (*Suc n*) = *evn n*

thm *evn.simps*
thm *od.simps*

thm *evn-od.induct*
thm *evn-od.termination*

7.6 Definitions in local contexts

locale *my-monoid* =
fixes *opr* :: *'a* ⇒ *'a* ⇒ *'a*
  and *un* :: *'a*
assumes *assoc*: *opr* (*opr x y*) *z* = *opr x* (*opr y z*)
  and *lunit*: *opr un x* = *x*
  and *runit*: *opr x un* = *x*
begin

fun *foldR* :: *'a list* ⇒ *'a*
where
  foldR [] = *un*
| *foldR* (*x*#*xs*) = *opr x* (*foldR xs*)

fun *foldL* :: *'a list* ⇒ *'a*
where
  foldL [] = *un*
| *foldL* [*x*] = *x*
| *foldL* (*x*#*y*#*ys*) = *foldL* (*opr x y* # *ys*)

thm *foldL.simps*

lemma *foldR-foldL*: *foldR xs* = *foldL xs*
⟨*proof*⟩

thm *foldR-foldL*

end

thm *my-monoid.foldL.simps*
thm *my-monoid.foldR-foldL*

7.7 Regression tests

The following examples mainly serve as tests for the function package

fun *listlen* :: *'a list* ⇒ *nat*
where
  listlen [] = *0*
| *listlen* (*x*#*xs*) = *Suc* (*listlen xs*)

fun *f* :: *nat* ⇒ *nat*
where
  zero: *f 0* = *0*
| *succ*: *f* (*Suc n*) = (*if f n* = *0 then 0 else f n*)

function *h* :: *nat* ⇒ *nat*
where
  h 0 = *0*
| *h* (*Suc n*) = (*if h n* = *0 then h* (*h n*) *else h n*)
  ⟨*proof*⟩

fun *i* :: *nat* ⇒ *nat*
where
  i 0 = *0*
| *i* (*Suc n*) = (*if n* = *0 then 0 else i n*)

fun *fa* :: *nat* ⇒ *nat* ⇒ *nat*
where
  fa 0 y = *0*
| *fa* (*Suc n*) *y* = (*if fa n y* = *0 then 0 else fa n y*)

fun *j* :: *nat* ⇒ *nat*
where
  j 0 = *0*
| *j* (*Suc n*) = (*let u* = *n in Suc* (*j u*))

function *k* :: *nat* ⇒ *nat*
where
  k x = (*let a* = *x*; *b* = *x in k x*)

⟨*proof*⟩

function *f2* :: (*nat* × *nat*) ⇒ (*nat* × *nat*)
where
f2 p = (*let* (*x*,*y*) = *p* *in* *f2* (*y*,*x*))
⟨*proof*⟩

fun *f3* :: *'a set* ⇒ *bool*
where
f3 x = *finite x*

datatype *'a tree* =
Leaf 'a
| *Branch 'a tree list*

lemma *lem*:*x* ∈ *set l* ⟹ *size x* < *Suc* (*list-size size l*)
⟨*proof*⟩

function *treemap* :: (*'a* ⇒ *'a*) ⇒ *'a tree* ⇒ *'a tree*
where
treemap fn (*Leaf n*) = (*Leaf* (*fn n*))
| *treemap fn* (*Branch l*) = (*Branch* (*map* (*treemap fn*) *l*))
⟨*proof*⟩
termination ⟨*proof*⟩

declare *lem*[*simp*]

fun *tinc* :: *nat tree* ⇒ *nat tree*
where
tinc (*Leaf n*) = *Leaf* (*Suc n*)
| *tinc* (*Branch l*) = *Branch* (*map tinc l*)

record *point* =
Xcoord :: *int*
Ycoord :: *int*

function *swp* :: *point* ⇒ *point*
where
swp (| *Xcoord* = *x*, *Ycoord* = *y* |) = (| *Xcoord* = *y*, *Ycoord* = *x* |)
⟨*proof*⟩
termination ⟨*proof*⟩

```
fun diag :: bool ⇒ bool ⇒ bool ⇒ nat
where
  diag x True False = 1
| diag False y True = 2
| diag True False z = 3
| diag True True True = 4
| diag False False False = 5



datatype DT =
  A | B | C | D | E | F | G | H | I | J | K | L | M | N | P
| Q | R | S | T | U | V

fun big :: DT ⇒ nat
where
  big A = 0
| big B = 0
| big C = 0
| big D = 0
| big E = 0
| big F = 0
| big G = 0
| big H = 0
| big I = 0
| big J = 0
| big K = 0
| big L = 0
| big M = 0
| big N = 0
| big P = 0
| big Q = 0
| big R = 0
| big S = 0
| big T = 0
| big U = 0
| big V = 0



fun
  f4 :: nat ⇒ nat ⇒ bool
where
  f4 0 0 = True
| f4 - - = False
```

end

8 Examples of automatically derived induction rules

theory *Induction-Scheme*
imports *Main*
begin

8.1 Some simple induction principles on nat

lemma *nat-standard-induct*:
$[\![P\ 0;\ \bigwedge n.\ P\ n \Longrightarrow P\ (Suc\ n)]\!] \Longrightarrow P\ x$
⟨*proof*⟩

lemma *nat-induct2*:
$[\![\ P\ 0;\ P\ (Suc\ 0);\ \bigwedge k.\ P\ k \Longrightarrow P\ (Suc\ k) \Longrightarrow P\ (Suc\ (Suc\ k))\ ]\!]$
$\Longrightarrow P\ n$
⟨*proof*⟩

lemma *minus-one-induct*:
$[\![\bigwedge n{::}nat.\ (n \neq 0 \Longrightarrow P\ (n - 1)) \Longrightarrow P\ n]\!] \Longrightarrow P\ x$
⟨*proof*⟩

theorem *diff-induct*:
$(!!x.\ P\ x\ 0) \Longrightarrow (!!y.\ P\ 0\ (Suc\ y)) \Longrightarrow$
$(!!x\ y.\ P\ x\ y \Longrightarrow P\ (Suc\ x)\ (Suc\ y)) \Longrightarrow P\ m\ n$
⟨*proof*⟩

lemma *list-induct2'*:
$[\![\ P\ []\ []$;
$\bigwedge x\ xs.\ P\ (x\#xs)\ []$;
$\bigwedge y\ ys.\ P\ []\ (y\#ys)$;
$\bigwedge x\ xs\ y\ ys.\ P\ xs\ ys \Longrightarrow P\ (x\#xs)\ (y\#ys)\ ]\!]$
$\Longrightarrow P\ xs\ ys$
⟨*proof*⟩

theorem *even-odd-induct*:
assumes $R\ 0$
assumes $Q\ 0$
assumes $\bigwedge n.\ Q\ n \Longrightarrow R\ (Suc\ n)$
assumes $\bigwedge n.\ R\ n \Longrightarrow Q\ (Suc\ n)$
shows $R\ n\ Q\ n$
⟨*proof*⟩

end

9 Some of the results in Inductive Invariants for Nested Recursion

theory *InductiveInvariant* **imports** *Main* **begin**

A formalization of some of the results in *Inductive Invariants for Nested Recursion*, by Sava Krstić and John Matthews. Appears in the proceedings of TPHOLs 2003, LNCS vol. 2758, pp. 253-269.

S is an inductive invariant of the functional F with respect to the wellfounded relation r.

definition
indinv :: $('a * 'a)$ *set* => $('a$ => $'b$ => *bool*) => (($'a$ => $'b$) => ($'a$ => $'b$)) => *bool* **where**
indinv r S F = $(\forall f\ x.\ (\forall y.\ (y,x) : r$ −−> *S y* $(f\ y))$ −−> *S x* $(F\ f\ x))$

S is an inductive invariant of the functional F on set D with respect to the wellfounded relation r.

definition
indinv-on :: $('a * 'a)$ *set* => $'a$ *set* => $('a$ => $'b$ => *bool*) => (($'a$ => $'b$) => ($'a$ => $'b$)) => *bool* **where**
indinv-on r D S F = $(\forall f.\ \forall x{\in}D.\ (\forall y{\in}D.\ (y,x) \in r$ −−> *S y* $(f\ y))$ −−> *S x* $(F\ f\ x))$

The key theorem, corresponding to theorem 1 of the paper. All other results in this theory are proved using instances of this theorem, and theorems derived from this theorem.

theorem *indinv-wfrec*:
assumes *wf*: *wf r* **and**
inv: *indinv r S F*
shows *S x* (*wfrec r F x*)
⟨*proof*⟩

theorem *indinv-on-wfrec*:
assumes *WF*: *wf r* **and**
INV: *indinv-on r D S F* **and**
D: $x{\in}D$
shows *S x* (*wfrec r F x*)
⟨*proof*⟩

theorem *ind-fixpoint-on-lemma*:
assumes *WF*: *wf r* **and**
INV: $\forall f.\ \forall x{\in}D.\ (\forall y{\in}D.\ (y,x) \in r$ −−> *S y* (*wfrec r F y*) & $f\ y$ = *wfrec r F y*)
−−> *S x* (*wfrec r F x*) & *F f x* = *wfrec r F x* **and**
D: $x{\in}D$
shows *F* (*wfrec r F*) *x* = *wfrec r F x* & *S x* (*wfrec r F x*)
⟨*proof*⟩

theorem *ind-fixpoint-lemma*:
assumes *WF*: *wf r* **and**
INV: $\forall f\ x.\ (\forall y.\ (y,x) \in r \longrightarrow S\ y\ (wfrec\ r\ F\ y)\ \&\ f\ y = wfrec\ r\ F\ y) \longrightarrow S\ x\ (wfrec\ r\ F\ x)\ \&\ F\ f\ x = wfrec\ r\ F\ x$
shows $F\ (wfrec\ r\ F)\ x = wfrec\ r\ F\ x\ \&\ S\ x\ (wfrec\ r\ F\ x)$
⟨*proof*⟩

theorem *tfl-indinv-wfrec*:
$[|\ f == wfrec\ r\ F;\ wf\ r;\ indinv\ r\ S\ F\ |] ==> S\ x\ (f\ x)$
⟨*proof*⟩

theorem *tfl-indinv-on-wfrec*:
$[|\ f == wfrec\ r\ F;\ wf\ r;\ indinv\text{-}on\ r\ D\ S\ F;\ x \in D\ |] ==> S\ x\ (f\ x)$
⟨*proof*⟩

end

10 Example use if an inductive invariant to solve termination conditions

theory *InductiveInvariant-examples* **imports** *InductiveInvariant* **begin**

A simple example showing how to use an inductive invariant to solve termination conditions generated by recdef on nested recursive function definitions.

consts $g :: nat => nat$

recdef (**permissive**) *g less-than*
$g\ 0 = 0$
$g\ (Suc\ n) = g\ (g\ n)$

We can prove the unsolved termination condition for g by showing it is an inductive invariant.

recdef-tc *g-tc*[*simp*]: *g*
⟨*proof*⟩

This declaration invokes Isabelle's simplifier to remove any termination conditions before adding g's rules to the simpset.

declare *g.simps* [*simplified*, *simp*]

This is an example where the termination condition generated by recdef is not itself an inductive invariant.

consts $g' :: nat => nat$

recdef (**permissive**) *g′ less-than*
g′ 0 = 0
g′ (Suc n) = g′ n + g′ (g′ n)

thm *g′.simps*

The strengthened inductive invariant is as follows (this invariant also works for the first example above):

lemma *g′-inv*: *g′ n = 0*
thm *tfl-indinv-wfrec* [*OF g′-def*]
⟨*proof*⟩

recdef-tc *g′-tc*[*simp*]: *g′*
⟨*proof*⟩

Now we can remove the termination condition from the rules for g' .

thm *g′.simps* [*simplified*]

Sometimes a recursive definition is partial, that is, it is only meant to be invoked on "good" inputs. As a contrived example, we will define a new version of g that is only well defined for even inputs greater than zero.

consts *g-even* :: *nat => nat*
recdef (**permissive**) *g-even less-than*
g-even (Suc (Suc 0)) = 3
g-even n = g-even (g-even (n − 2) − 1)

We can prove a conditional version of the unsolved termination condition for *g-even* by proving a stronger inductive invariant.

lemma *g-even-indinv*: ∃ *k*. *n = Suc (Suc (2∗k)) ==> g-even n = 3*
⟨*proof*⟩

Now we can prove that the second recursion equation for *g-even* holds, provided that n is an even number greater than two.

theorem *g-even-n*: ∃ *k*. *n = 2∗k + 4 ==> g-even n = g-even (g-even (n − 2) − 1)*
⟨*proof*⟩

McCarthy's ninety-one function. This function requires a non-standard measure to prove termination.

consts *ninety-one* :: *nat => nat*
recdef (**permissive**) *ninety-one measure* (%*n*. *101 − n*)
ninety-one x = (if 100 < x
then x − 10
else (ninety-one (ninety-one (x+11))))

To discharge the termination condition, we will prove a strengthened inductive invariant: S x y == x ¡ y + 11

lemma *ninety-one-inv*: $n < \mathit{ninety\text{-}one}\ n + 11$
⟨*proof*⟩

Proving the termination condition using the strengthened inductive invariant.

recdef-tc *ninety-one-tc*[*rule-format*]: *ninety-one*
⟨*proof*⟩

Now we can remove the termination condition from the simplification rule for *ninety-one*.

theorem *def-ninety-one*:
$\mathit{ninety\text{-}one}\ x = (\mathit{if}\ 100 < x$
$\quad \mathit{then}\ x - 10$
$\quad \mathit{else}\ \mathit{ninety\text{-}one}\ (\mathit{ninety\text{-}one}\ (x{+}11)))$
⟨*proof*⟩

end

11 Test of Locale Interpretation

theory *LocaleTest2*
imports *Main GCD*
begin

12 Interpretation of Defined Concepts

Naming convention for global objects: prefixes D and d

12.1 Lattices

Much of the lattice proofs are from HOL/Lattice.

12.1.1 Definitions

locale *dpo* =
  fixes *le* :: [$'a$, $'a$] => *bool* (**infixl** $\sqsubseteq$ *50*)
  assumes *refl* [*intro*, *simp*]: $x \sqsubseteq x$
    and *anti-sym* [*intro*]: [| $x \sqsubseteq y$; $y \sqsubseteq x$ |] ==> $x = y$
    and *trans* [*trans*]: [| $x \sqsubseteq y$; $y \sqsubseteq z$ |] ==> $x \sqsubseteq z$

begin

theorem *circular*:
  [| $x \sqsubseteq y$; $y \sqsubseteq z$; $z \sqsubseteq x$ |] ==> $x = y$ & $y = z$
  ⟨*proof*⟩

definition
less :: [*'a*, *'a*] => *bool* (**infixl** $\sqsubset$ *50*)
where $(x \sqsubset y) = (x \sqsubseteq y \;\&\; x \mathrel{\sim=} y)$

theorem *abs-test*:
$op \sqsubset = (\%x\ y.\ x \sqsubset y)$
⟨*proof*⟩

definition
is-inf :: [*'a*, *'a*, *'a*] => *bool*
where *is-inf x y i* $= (i \sqsubseteq x \wedge i \sqsubseteq y \wedge (\forall z.\ z \sqsubseteq x \wedge z \sqsubseteq y \longrightarrow z \sqsubseteq i))$

definition
is-sup :: [*'a*, *'a*, *'a*] => *bool*
where *is-sup x y s* $= (x \sqsubseteq s \wedge y \sqsubseteq s \wedge (\forall z.\ x \sqsubseteq z \wedge y \sqsubseteq z \longrightarrow s \sqsubseteq z))$

end

locale *dlat* = *dpo* +
assumes *ex-inf*: *EX inf*. *dpo.is-inf le x y inf*
and *ex-sup*: *EX sup*. *dpo.is-sup le x y sup*

begin

definition
meet :: [*'a*, *'a*] => *'a* (**infixl** $\sqcap$ *70*)
where $x \sqcap y = (THE\ inf.\ is\text{-}inf\ x\ y\ inf)$

definition
join :: [*'a*, *'a*] => *'a* (**infixl** $\sqcup$ *65*)
where $x \sqcup y = (THE\ sup.\ is\text{-}sup\ x\ y\ sup)$

lemma *is-infI* [*intro?*]: $i \sqsubseteq x \Longrightarrow i \sqsubseteq y \Longrightarrow$
$(\bigwedge z.\ z \sqsubseteq x \Longrightarrow z \sqsubseteq y \Longrightarrow z \sqsubseteq i) \Longrightarrow is\text{-}inf\ x\ y\ i$
⟨*proof*⟩

lemma *is-inf-lower* [*elim?*]:
$is\text{-}inf\ x\ y\ i \Longrightarrow (i \sqsubseteq x \Longrightarrow i \sqsubseteq y \Longrightarrow C) \Longrightarrow C$
⟨*proof*⟩

lemma *is-inf-greatest* [*elim?*]:
$is\text{-}inf\ x\ y\ i \Longrightarrow z \sqsubseteq x \Longrightarrow z \sqsubseteq y \Longrightarrow z \sqsubseteq i$
⟨*proof*⟩

theorem *is-inf-uniq*: $is\text{-}inf\ x\ y\ i \Longrightarrow is\text{-}inf\ x\ y\ i' \Longrightarrow i = i'$
⟨*proof*⟩

theorem *is-inf-related* [*elim?*]: $x \sqsubseteq y \Longrightarrow is\text{-}inf\ x\ y\ x$

⟨*proof*⟩

lemma *meet-equality* [*elim?*]: *is-inf x y i* ⟹ $x \sqcap y = i$
⟨*proof*⟩

lemma *meetI* [*intro?*]:
$i \sqsubseteq x \Longrightarrow i \sqsubseteq y \Longrightarrow (\bigwedge z.\ z \sqsubseteq x \Longrightarrow z \sqsubseteq y \Longrightarrow z \sqsubseteq i) \Longrightarrow x \sqcap y = i$
⟨*proof*⟩

lemma *is-inf-meet* [*intro?*]: *is-inf x y* $(x \sqcap y)$
⟨*proof*⟩

lemma *meet-left* [*intro?*]:
$x \sqcap y \sqsubseteq x$
⟨*proof*⟩

lemma *meet-right* [*intro?*]:
$x \sqcap y \sqsubseteq y$
⟨*proof*⟩

lemma *meet-le* [*intro?*]:
$[\!|\ z \sqsubseteq x;\ z \sqsubseteq y\ |\!] \Longrightarrow z \sqsubseteq x \sqcap y$
⟨*proof*⟩

lemma *is-supI* [*intro?*]: $x \sqsubseteq s \Longrightarrow y \sqsubseteq s \Longrightarrow$
$(\bigwedge z.\ x \sqsubseteq z \Longrightarrow y \sqsubseteq z \Longrightarrow s \sqsubseteq z) \Longrightarrow$ *is-sup x y s*
⟨*proof*⟩

lemma *is-sup-least* [*elim?*]:
is-sup x y s ⟹ $x \sqsubseteq z \Longrightarrow y \sqsubseteq z \Longrightarrow s \sqsubseteq z$
⟨*proof*⟩

lemma *is-sup-upper* [*elim?*]:
is-sup x y s ⟹ $(x \sqsubseteq s \Longrightarrow y \sqsubseteq s \Longrightarrow C) \Longrightarrow C$
⟨*proof*⟩

theorem *is-sup-uniq*: *is-sup x y s* ⟹ *is-sup x y s′* ⟹ $s = s'$
⟨*proof*⟩

theorem *is-sup-related* [*elim?*]: $x \sqsubseteq y \Longrightarrow$ *is-sup x y y*
⟨*proof*⟩

lemma *join-equality* [*elim?*]: *is-sup x y s* ⟹ $x \sqcup y = s$
⟨*proof*⟩

lemma *joinI* [*intro?*]: $x \sqsubseteq s \Longrightarrow y \sqsubseteq s \Longrightarrow$
$(\bigwedge z.\ x \sqsubseteq z \Longrightarrow y \sqsubseteq z \Longrightarrow s \sqsubseteq z) \Longrightarrow x \sqcup y = s$
⟨*proof*⟩

lemma *is-sup-join* [*intro?*]: *is-sup x y* $(x \sqcup y)$
⟨*proof*⟩

lemma *join-left* [*intro?*]:
$x \sqsubseteq x \sqcup y$
⟨*proof*⟩

lemma *join-right* [*intro?*]:
$y \sqsubseteq x \sqcup y$
⟨*proof*⟩

lemma *join-le* [*intro?*]:
$[\!| x \sqsubseteq z; y \sqsubseteq z |\!] ==> x \sqcup y \sqsubseteq z$
⟨*proof*⟩

theorem *meet-assoc*: $(x \sqcap y) \sqcap z = x \sqcap (y \sqcap z)$
⟨*proof*⟩

theorem *meet-commute*: $x \sqcap y = y \sqcap x$
⟨*proof*⟩

theorem *meet-join-absorb*: $x \sqcap (x \sqcup y) = x$
⟨*proof*⟩

theorem *join-assoc*: $(x \sqcup y) \sqcup z = x \sqcup (y \sqcup z)$
⟨*proof*⟩

theorem *join-commute*: $x \sqcup y = y \sqcup x$
⟨*proof*⟩

theorem *join-meet-absorb*: $x \sqcup (x \sqcap y) = x$
⟨*proof*⟩

theorem *meet-idem*: $x \sqcap x = x$
⟨*proof*⟩

theorem *meet-related* [*elim?*]: $x \sqsubseteq y \Longrightarrow x \sqcap y = x$
⟨*proof*⟩

theorem *meet-related2* [*elim?*]: $y \sqsubseteq x \Longrightarrow x \sqcap y = y$
⟨*proof*⟩

theorem *join-related* [*elim?*]: $x \sqsubseteq y \Longrightarrow x \sqcup y = y$
⟨*proof*⟩

theorem *join-related2* [*elim?*]: $y \sqsubseteq x \Longrightarrow x \sqcup y = x$
⟨*proof*⟩

Additional theorems

theorem *meet-connection*: $(x \sqsubseteq y) = (x \sqcap y = x)$
⟨*proof*⟩

theorem *meet-connection2*: $(x \sqsubseteq y) = (y \sqcap x = x)$
⟨*proof*⟩

theorem *join-connection*: $(x \sqsubseteq y) = (x \sqcup y = y)$
⟨*proof*⟩

theorem *join-connection2*: $(x \sqsubseteq y) = (x \sqcup y = y)$
⟨*proof*⟩

Naming according to Jacobson I, p. 459.

lemmas *L1* = *join-commute meet-commute*
lemmas *L2* = *join-assoc meet-assoc*

lemmas *L4* = *join-meet-absorb meet-join-absorb*

end

locale *ddlat* = *dlat* +
assumes *meet-distr*:
dlat.meet le x (*dlat.join le y z*) =
dlat.join le (*dlat.meet le x y*) (*dlat.meet le x z*)

begin

lemma *join-distr*:
$x \sqcup (y \sqcap z) = (x \sqcup y) \sqcap (x \sqcup z)$⟨*proof*⟩

end

locale *dlo* = *dpo* +
assumes *total*: $x \sqsubseteq y \mid y \sqsubseteq x$

begin

lemma *less-total*: $x \sqsubset y \mid x = y \mid y \sqsubset x$
⟨*proof*⟩

end

sublocale *dlo* < *dlat*
⟨*proof*⟩

sublocale *dlo* < *ddlat*
⟨*proof*⟩

12.1.2 Total order <= on *int*

interpretation *int*: *dpo op* <= :: [*int*, *int*] => *bool*
  where (*dpo.less* (*op* <=) (*x*::*int*) *y*) = (*x* < *y*)⟨*proof*⟩

thm *int.circular*
lemma ⟦ (*x*::*int*) ≤ *y*; *y* ≤ *z*; *z* ≤ *x*⟧ ⟹ *x* = *y* ∧ *y* = *z*
  ⟨*proof*⟩
thm *int.abs-test*
lemma (*op* < :: [*int*, *int*] => *bool*) = *op* <
  ⟨*proof*⟩

interpretation *int*: *dlat op* <= :: [*int*, *int*] => *bool*
  where *meet-eq*: *dlat.meet* (*op* <=) (*x*::*int*) *y* = *min x y*
    and *join-eq*: *dlat.join* (*op* <=) (*x*::*int*) *y* = *max x y*
⟨*proof*⟩

interpretation *int*: *dlo op* <= :: [*int*, *int*] => *bool*
  ⟨*proof*⟩

Interpreted theorems from the locales, involving defined terms.

thm *int.less-def*

from dpo

thm *int.meet-left*

from dlat

thm *int.meet-distr*

from ddlat

thm *int.less-total*

from dlo

12.1.3 Total order <= on *nat*

interpretation *nat*: *dpo op* <= :: [*nat*, *nat*] => *bool*
  where *dpo.less* (*op* <=) (*x*::*nat*) *y* = (*x* < *y*)⟨*proof*⟩

interpretation *nat*: *dlat op* <= :: [*nat*, *nat*] => *bool*
  where *dlat.meet* (*op* <=) (*x*::*nat*) *y* = *min x y*
    and *dlat.join* (*op* <=) (*x*::*nat*) *y* = *max x y*
⟨*proof*⟩

interpretation *nat*: *dlo op* <= :: [*nat*, *nat*] => *bool*
  ⟨*proof*⟩

Interpreted theorems from the locales, involving defined terms.

thm *nat.less-def*

from dpo

thm *nat.meet-left*

from dlat

thm *nat.meet-distr*

from ddlat

thm *nat.less-total*

from ldo

12.1.4 Lattice *dvd* on *nat*

interpretation *nat-dvd*: *dpo op dvd* :: [*nat*, *nat*] => *bool*
  where *dpo.less* (*op dvd*) (*x*::*nat*) *y* = (*x dvd y* & *x* ~= *y*)⟨*proof*⟩

interpretation *nat-dvd*: *dlat op dvd* :: [*nat*, *nat*] => *bool*
  where *dlat.meet* (*op dvd*) (*x*::*nat*) *y* = *gcd x y*
    and *dlat.join* (*op dvd*) (*x*::*nat*) *y* = *lcm x y*
⟨*proof*⟩

Interpreted theorems from the locales, involving defined terms.

thm *nat-dvd.less-def*

from dpo

lemma ((*x*::*nat*) *dvd y* & *x* ~= *y*) = (*x dvd y* & *x* ~= *y*)
  ⟨*proof*⟩
thm *nat-dvd.meet-left*

from dlat

lemma *gcd x y dvd x*
  ⟨*proof*⟩

12.2 Group example with defined operations *inv* and *unit*

12.2.1 Locale declarations and lemmas

locale *Dsemi* =
  fixes *prod* (**infixl** ** 65*)
  assumes *assoc*: (*x* ** *y*) ** *z* = *x* ** (*y* ** *z*)

locale *Dmonoid* = *Dsemi* +
  fixes *one*
  assumes *l-one* [*simp*]: *one* ** *x* = *x*
    and *r-one* [*simp*]: *x* ** *one* = *x*

begin

definition
inv **where** $inv\ x = (THE\ y.\ x\ **\ y = one\ \&\ y\ **\ x = one)$
definition
unit **where** $unit\ x = (EX\ y.\ x\ **\ y = one\ \ \&\ y\ **\ x = one)$

lemma *inv-unique*:
assumes *eq*: $y\ **\ x = one\ x\ **\ y' = one$
shows $y = y'$
⟨*proof*⟩

lemma *unit-one* [*intro*, *simp*]:
$unit\ one$
⟨*proof*⟩

lemma *unit-l-inv-ex*:
$unit\ x\ ==>\ \exists\, y.\ y\ **\ x = one$
⟨*proof*⟩

lemma *unit-r-inv-ex*:
$unit\ x\ ==>\ \exists\, y.\ x\ **\ y = one$
⟨*proof*⟩

lemma *unit-l-inv*:
$unit\ x\ ==>\ inv\ x\ **\ x = one$
⟨*proof*⟩

lemma *unit-r-inv*:
$unit\ x\ ==>\ x\ **\ inv\ x = one$
⟨*proof*⟩

lemma *unit-inv-unit* [*intro*, *simp*]:
$unit\ x\ ==>\ unit\ (inv\ x)$
⟨*proof*⟩

lemma *unit-l-cancel* [*simp*]:
$unit\ x\ ==>\ (x\ **\ y = x\ **\ z) = (y = z)$
⟨*proof*⟩

lemma *unit-inv-inv* [*simp*]:
$unit\ x\ ==>\ inv\ (inv\ x) = x$
⟨*proof*⟩

lemma *inv-inj-on-unit*:
$inj\text{-}on\ inv\ \{x.\ unit\ x\}$
⟨*proof*⟩

lemma *unit-inv-comm*:
assumes *inv*: $x\ **\ y = one$
and *G*: $unit\ x\ \ unit\ y$

shows $y ** x = one$
⟨*proof*⟩

end

locale *Dgrp* = *Dmonoid* +
assumes *unit* [*intro*, *simp*]: *Dmonoid.unit* (*op* **) *one x*

begin

lemma *l-inv-ex* [*simp*]:
$\exists y.\ y ** x = one$
⟨*proof*⟩

lemma *r-inv-ex* [*simp*]:
$\exists y.\ x ** y = one$
⟨*proof*⟩

lemma *l-inv* [*simp*]:
$inv\ x ** x = one$
⟨*proof*⟩

lemma *l-cancel* [*simp*]:
$(x ** y = x ** z) = (y = z)$
⟨*proof*⟩

lemma *r-inv* [*simp*]:
$x ** inv\ x = one$
⟨*proof*⟩

lemma *r-cancel* [*simp*]:
$(y ** x = z ** x) = (y = z)$
⟨*proof*⟩

lemma *inv-one* [*simp*]:
$inv\ one = one$
⟨*proof*⟩

lemma *inv-inv* [*simp*]:
$inv\ (inv\ x) = x$
⟨*proof*⟩

lemma *inv-inj*:
inj-on inv UNIV
⟨*proof*⟩

lemma *inv-mult-group*:
$inv\ (x ** y) = inv\ y ** inv\ x$

⟨*proof*⟩

lemma *inv-comm*:
*x ** y = one ==> y ** x = one*
⟨*proof*⟩

lemma *inv-equality*:
*y ** x = one ==> inv x = y*
⟨*proof*⟩

end

locale *Dhom = prod*: *Dgrp prod one + sum*: *Dgrp sum zero*
for *prod* (**infixl** ** 65*) **and** *one* **and** *sum* (**infixl** *+++ 60*) **and** *zero* +
fixes *hom*
assumes *hom-mult* [*simp*]: *hom (x ** y) = hom x +++ hom y*

begin

lemma *hom-one* [*simp*]:
hom one = zero
⟨*proof*⟩

end

12.2.2 Interpretation of Functions

interpretation *Dfun*: *Dmonoid op o id* :: *'a => 'a*
where *Dmonoid.unit (op o) id f = bij (f*::*'a => 'a)*

⟨*proof*⟩

thm *Dmonoid.unit-def Dfun.unit-def*

thm *Dmonoid.inv-inj-on-unit Dfun.inv-inj-on-unit*

lemma *unit-id*:
(f :: unit => unit) = id
⟨*proof*⟩

interpretation *Dfun*: *Dgrp op o id* :: *unit => unit*
where *Dmonoid.inv (op o) id f = inv (f :: unit => unit)*
⟨*proof*⟩

thm *Dfun.unit-l-inv Dfun.l-inv*

thm *Dfun.inv-equality*
thm *Dfun.inv-equality*

end

13 Monoids and Groups as predicates over record schemes

theory *MonoidGroup* **imports** *Main* **begin**

record *'a monoid-sig =*
  times :: 'a => 'a => 'a
  one :: 'a

record *'a group-sig = 'a monoid-sig +*
  inv :: 'a => 'a

definition
  monoid :: (| times :: 'a => 'a => 'a, one :: 'a, ... :: 'b |) => bool **where**
  monoid M = (∀ x y z.
    times M (times M x y) z = times M x (times M y z) ∧
    times M (one M) x = x ∧ times M x (one M) = x)

definition
  group :: (| times :: 'a => 'a => 'a, one :: 'a, inv :: 'a => 'a, ... :: 'b |) => bool
where
  group G = (monoid G ∧ (∀ x. times G (inv G x) x = one G))

definition
  reverse :: (| times :: 'a => 'a => 'a, one :: 'a, ... :: 'b |) =>
    (| times :: 'a => 'a => 'a, one :: 'a, ... :: 'b |) **where**
  reverse M = M (| times := λx y. times M y x |)

end

14 Binary arithmetic examples

theory *BinEx*
imports *Complex-Main*
begin

14.1 Regression Testing for Cancellation Simprocs

lemma *l + 2 + 2 + 2 + (l + 2) + (oo + 2) = (uu::int)*
⟨*proof*⟩

lemma *2*u = (u::int)*
⟨*proof*⟩

lemma $(i + j + 12 + (k::int)) - 15 = y$
⟨*proof*⟩

lemma $(i + j + 12 + (k::int)) - 5 = y$
⟨*proof*⟩

lemma $y - b < (b::int)$
⟨*proof*⟩

lemma $y - (3*b + c) < (b::int) - 2*c$
⟨*proof*⟩

lemma $(2*x - (u*v) + y) - v*3*u = (w::int)$
⟨*proof*⟩

lemma $(2*x*u*v + (u*v)*4 + y) - v*u*4 = (w::int)$
⟨*proof*⟩

lemma $(2*x*u*v + (u*v)*4 + y) - v*u = (w::int)$
⟨*proof*⟩

lemma $u*v - (x*u*v + (u*v)*4 + y) = (w::int)$
⟨*proof*⟩

lemma $(i + j + 12 + (k::int)) = u + 15 + y$
⟨*proof*⟩

lemma $(i + j*2 + 12 + (k::int)) = j + 5 + y$
⟨*proof*⟩

lemma $2*y + 3*z + 6*w + 2*y + 3*z + 2*u = 2*y' + 3*z' + 6*w' + 2*y' + 3*z' + u + (vv::int)$
⟨*proof*⟩

lemma $a + -(b+c) + b = (d::int)$
⟨*proof*⟩

lemma $a + -(b+c) - b = (d::int)$
⟨*proof*⟩

lemma $(i + j + -2 + (k::int)) - (u + 5 + y) = zz$
⟨*proof*⟩

lemma $(i + j + -3 + (k::int)) < u + 5 + y$
⟨*proof*⟩

lemma $(i + j + 3 + (k::int)) < u + -6 + y$

⟨*proof*⟩

lemma $(i + j + -12 + (k::int)) - 15 = y$
⟨*proof*⟩

lemma $(i + j + 12 + (k::int)) - -15 = y$
⟨*proof*⟩

lemma $(i + j + -12 + (k::int)) - -15 = y$
⟨*proof*⟩

lemma $- (2*i) + 3 + (2*i + 4) = (0::int)$
⟨*proof*⟩

14.2 Arithmetic Method Tests

lemma $!!a::int.\ [|\ a <= b;\ c <= d;\ x+y<z\ |] ==> a+c <= b+d$
⟨*proof*⟩

lemma $!!a::int.\ [|\ a < b;\ c < d\ |] ==> a-d+ 2 <= b+(-c)$
⟨*proof*⟩

lemma $!!a::int.\ [|\ a < b;\ c < d\ |] ==> a+c+ 1 < b+d$
⟨*proof*⟩

lemma $!!a::int.\ [|\ a <= b;\ b+b <= c\ |] ==> a+a <= c$
⟨*proof*⟩

lemma $!!a::int.\ [|\ a+b <= i+j;\ a<=b;\ i<=j\ |] ==> a+a <= j+j$
⟨*proof*⟩

lemma $!!a::int.\ [|\ a+b < i+j;\ a<b;\ i<j\ |] ==> a+a - - -1 < j+j - 3$
⟨*proof*⟩

lemma $!!a::int.\ a+b+c <= i+j+k\ \&\ a<=b\ \&\ b<=c\ \&\ i<=j\ \&\ j<=k\ --> a+a+a <= k+k+k$
⟨*proof*⟩

lemma $!!a::int.\ [|\ a+b+c+d <= i+j+k+l;\ a<=b;\ b<=c;\ c<=d;\ i<=j;\ j<=k;\ k<=l\ |]$
$==> a <= l$
⟨*proof*⟩

lemma $!!a::int.\ [|\ a+b+c+d <= i+j+k+l;\ a<=b;\ b<=c;\ c<=d;\ i<=j;\ j<=k;\ k<=l\ |]$
$==> a+a+a+a <= l+l+l+l$
⟨*proof*⟩

lemma $!!a::int.\ [|\ a+b+c+d <= i+j+k+l;\ a<=b;\ b<=c;\ c<=d;\ i<=j;\ j<=k;$

$k<=l$ |]
    ==> $a+a+a+a+a <= l+l+l+l+i$
⟨*proof*⟩

lemma !!a::*int*. [| $a+b+c+d <= i+j+k+l$; $a<=b$; $b<=c$; $c<=d$; $i<=j$; $j<=k$; $k<=l$ |]
    ==> $a+a+a+a+a+a <= l+l+l+l+i+l$
⟨*proof*⟩

lemma !!a::*int*. [| $a+b+c+d <= i+j+k+l$; $a<=b$; $b<=c$; $c<=d$; $i<=j$; $j<=k$; $k<=l$ |]
    ==> $6*a <= 5*l+i$
⟨*proof*⟩

14.3 The Integers

Addition

lemma (*13*::*int*) + *19* = *32*
  ⟨*proof*⟩

lemma (*1234*::*int*) + *5678* = *6912*
  ⟨*proof*⟩

lemma (*1359*::*int*) + *−2468* = *−1109*
  ⟨*proof*⟩

lemma (*93746*::*int*) + *−46375* = *47371*
  ⟨*proof*⟩

Negation

lemma − (*65745*::*int*) = *−65745*
  ⟨*proof*⟩

lemma − (*−54321*::*int*) = *54321*
  ⟨*proof*⟩

Multiplication

lemma (*13*::*int*) * *19* = *247*
  ⟨*proof*⟩

lemma (*−84*::*int*) * *51* = *−4284*
  ⟨*proof*⟩

lemma (*255*::*int*) * *255* = *65025*
  ⟨*proof*⟩

lemma (*1359*::*int*) * *−2468* = *−3354012*

⟨*proof*⟩

lemma (*89*::*int*) ∗ *10* ≠ *889*
⟨*proof*⟩

lemma (*13*::*int*) < *18* − *4*
⟨*proof*⟩

lemma (−*345*::*int*) < −*242* + −*100*
⟨*proof*⟩

lemma (*13557456*::*int*) < *18678654*
⟨*proof*⟩

lemma (*999999*::*int*) ≤ (*1000001* + *1*) − *2*
⟨*proof*⟩

lemma (*1234567*::*int*) ≤ *1234567*
⟨*proof*⟩

No integer overflow!

lemma *1234567* ∗ (*1234567*::*int*) < *1234567*∗*1234567*∗*1234567*
⟨*proof*⟩

Quotient and Remainder

lemma (*10*::*int*) *div 3* = *3*
⟨*proof*⟩

lemma (*10*::*int*) *mod 3* = *1*
⟨*proof*⟩

A negative divisor

lemma (*10*::*int*) *div* −*3* = −*4*
⟨*proof*⟩

lemma (*10*::*int*) *mod* −*3* = −*2*
⟨*proof*⟩

A negative dividend[1]

lemma (−*10*::*int*) *div 3* = −*4*
⟨*proof*⟩

lemma (−*10*::*int*) *mod 3* = *2*
⟨*proof*⟩

A negative dividend *and* divisor

[1]The definition agrees with mathematical convention and with ML, but not with the hardware of most computers

lemma $(-10::int)\ div\ -3 = 3$
⟨*proof*⟩

lemma $(-10::int)\ mod\ -3 = -1$
⟨*proof*⟩

A few bigger examples

lemma $(8452::int)\ mod\ 3 = 1$
⟨*proof*⟩

lemma $(59485::int)\ div\ 434 = 137$
⟨*proof*⟩

lemma $(1000006::int)\ mod\ 10 = 6$
⟨*proof*⟩

Division by shifting

lemma $10000000\ div\ 2 = (5000000::int)$
⟨*proof*⟩

lemma $10000001\ mod\ 2 = (1::int)$
⟨*proof*⟩

lemma $10000055\ div\ 32 = (312501::int)$
⟨*proof*⟩

lemma $10000055\ mod\ 32 = (23::int)$
⟨*proof*⟩

lemma $100094\ div\ 144 = (695::int)$
⟨*proof*⟩

lemma $100094\ mod\ 144 = (14::int)$
⟨*proof*⟩

Powers

lemma $2\ \hat{}\ 10 = (1024::int)$
⟨*proof*⟩

lemma $-3\ \hat{}\ 7 = (-2187::int)$
⟨*proof*⟩

lemma $13\ \hat{}\ 7 = (62748517::int)$
⟨*proof*⟩

lemma $3\ \hat{}\ 15 = (14348907::int)$
⟨*proof*⟩

lemma $-5 \hat{} 11 = (-48828125::int)$
⟨*proof*⟩

14.4 The Natural Numbers

Successor

lemma $Suc\ 99999 = 100000$
⟨*proof*⟩

Addition

lemma $(13::nat) + 19 = 32$
⟨*proof*⟩

lemma $(1234::nat) + 5678 = 6912$
⟨*proof*⟩

lemma $(973646::nat) + 6475 = 980121$
⟨*proof*⟩

Subtraction

lemma $(32::nat) - 14 = 18$
⟨*proof*⟩

lemma $(14::nat) - 15 = 0$
⟨*proof*⟩

lemma $(14::nat) - 1576644 = 0$
⟨*proof*⟩

lemma $(48273776::nat) - 3873737 = 44400039$
⟨*proof*⟩

Multiplication

lemma $(12::nat) * 11 = 132$
⟨*proof*⟩

lemma $(647::nat) * 3643 = 2357021$
⟨*proof*⟩

Quotient and Remainder

lemma $(10::nat)\ div\ 3 = 3$
⟨*proof*⟩

lemma $(10::nat)\ mod\ 3 = 1$
⟨*proof*⟩

lemma $(10000::nat)\ div\ 9 = 1111$
⟨*proof*⟩

lemma $(10000::nat)\ mod\ 9 = 1$
⟨*proof*⟩

lemma $(10000::nat)\ div\ 16 = 625$
⟨*proof*⟩

lemma $(10000::nat)\ mod\ 16 = 0$
⟨*proof*⟩

Powers

lemma $2\ \hat{}\ 12 = (4096::nat)$
⟨*proof*⟩

lemma $3\ \hat{}\ 10 = (59049::nat)$
⟨*proof*⟩

lemma $12\ \hat{}\ 7 = (35831808::nat)$
⟨*proof*⟩

lemma $3\ \hat{}\ 14 = (4782969::nat)$
⟨*proof*⟩

lemma $5\ \hat{}\ 11 = (48828125::nat)$
⟨*proof*⟩

Testing the cancellation of complementary terms

lemma $y + (x + -x) = (0::int) + y$
⟨*proof*⟩

lemma $y + (-x + (-\ y + x)) = (0::int)$
⟨*proof*⟩

lemma $-x + (y + (-\ y + x)) = (0::int)$
⟨*proof*⟩

lemma $x + (x + (-\ x + (-\ x + (-\ y + -\ z)))) = (0::int) - y - z$
⟨*proof*⟩

lemma $x + x - x - x - y - z = (0::int) - y - z$
⟨*proof*⟩

lemma $x + y + z - (x + z) = y - (0::int)$
⟨*proof*⟩

lemma $x + (y + (y + (y + (-x + -x)))) = (0::int) + y - x + y + y$

⟨*proof*⟩

lemma $x + (y + (y + (y + (-y + -x)))) = y + (0::int) + y$
⟨*proof*⟩

lemma $x + y - x + z - x - y - z + x < (1::int)$
⟨*proof*⟩

14.5 Real Arithmetic

14.5.1 Addition

lemma $(1359::real) + -2468 = -1109$
⟨*proof*⟩

lemma $(93746::real) + -46375 = 47371$
⟨*proof*⟩

14.5.2 Negation

lemma $- (65745::real) = -65745$
⟨*proof*⟩

lemma $- (-54321::real) = 54321$
⟨*proof*⟩

14.5.3 Multiplication

lemma $(-84::real) * 51 = -4284$
⟨*proof*⟩

lemma $(255::real) * 255 = 65025$
⟨*proof*⟩

lemma $(1359::real) * -2468 = -3354012$
⟨*proof*⟩

14.5.4 Inequalities

lemma $(89::real) * 10 \neq 889$
⟨*proof*⟩

lemma $(13::real) < 18 - 4$
⟨*proof*⟩

lemma $(-345::real) < -242 + -100$
⟨*proof*⟩

lemma $(13557456::real) < 18678654$
⟨*proof*⟩

lemma $(999999::real) \leq (1000001 + 1) - 2$
⟨*proof*⟩

lemma $(1234567::real) \leq 1234567$
⟨*proof*⟩

14.5.5 Powers

lemma $2 \hat{} 15 = (32768::real)$
⟨*proof*⟩

lemma $-3 \hat{} 7 = (-2187::real)$
⟨*proof*⟩

lemma $13 \hat{} 7 = (62748517::real)$
⟨*proof*⟩

lemma $3 \hat{} 15 = (14348907::real)$
⟨*proof*⟩

lemma $-5 \hat{} 11 = (-48828125::real)$
⟨*proof*⟩

14.5.6 Tests

lemma $(x + y = x) = (y = (0::real))$
⟨*proof*⟩

lemma $(x + y = y) = (x = (0::real))$
⟨*proof*⟩

lemma $(x + y = (0::real)) = (x = -y)$
⟨*proof*⟩

lemma $(x + y = (0::real)) = (y = -x)$
⟨*proof*⟩

lemma $((x + y) < (x + z)) = (y < (z::real))$
⟨*proof*⟩

lemma $((x + z) < (y + z)) = (x < (y::real))$
⟨*proof*⟩

lemma $(\neg\ x < y) = (y \leq (x::real))$
⟨*proof*⟩

lemma $\neg\ (x < y \wedge y < (x::real))$
⟨*proof*⟩

lemma $(x::real) < y ==> \neg\ y < x$
⟨*proof*⟩

lemma $((x::real) \neq y) = (x < y \vee y < x)$
⟨*proof*⟩

lemma $(\neg\ x \leq y) = (y < (x::real))$
⟨*proof*⟩

lemma $x \leq y \vee y \leq (x::real)$
⟨*proof*⟩

lemma $x \leq y \vee y < (x::real)$
⟨*proof*⟩

lemma $x < y \vee y \leq (x::real)$
⟨*proof*⟩

lemma $x \leq (x::real)$
⟨*proof*⟩

lemma $((x::real) \leq y) = (x < y \vee x = y)$
⟨*proof*⟩

lemma $((x::real) \leq y \wedge y \leq x) = (x = y)$
⟨*proof*⟩

lemma $\neg(x < y \wedge y \leq (x::real))$
⟨*proof*⟩

lemma $\neg(x \leq y \wedge y < (x::real))$
⟨*proof*⟩

lemma $(-x < (0::real)) = (0 < x)$
⟨*proof*⟩

lemma $((0::real) < -x) = (x < 0)$
⟨*proof*⟩

lemma $(-x \leq (0::real)) = (0 \leq x)$
⟨*proof*⟩

lemma $((0::real) \leq -x) = (x \leq 0)$
⟨*proof*⟩

lemma $(x::real) = y \vee x < y \vee y < x$
⟨*proof*⟩

lemma $(x::real) = 0 \vee 0 < x \vee 0 < -x$

⟨*proof*⟩

lemma $(0::real) \leq x \vee 0 \leq -x$
⟨*proof*⟩

lemma $((x::real) + y \leq x + z) = (y \leq z)$
⟨*proof*⟩

lemma $((x::real) + z \leq y + z) = (x \leq y)$
⟨*proof*⟩

lemma $(w::real) < x \wedge y < z \Longrightarrow w + y < x + z$
⟨*proof*⟩

lemma $(w::real) \leq x \wedge y \leq z \Longrightarrow w + y \leq x + z$
⟨*proof*⟩

lemma $(0::real) \leq x \wedge 0 \leq y \Longrightarrow 0 \leq x + y$
⟨*proof*⟩

lemma $(0::real) < x \wedge 0 < y \Longrightarrow 0 < x + y$
⟨*proof*⟩

lemma $(-x < y) = (0 < x + (y::real))$
⟨*proof*⟩

lemma $(x < -y) = (x + y < (0::real))$
⟨*proof*⟩

lemma $(y < x + -z) = (y + z < (x::real))$
⟨*proof*⟩

lemma $(x + -y < z) = (x < z + (y::real))$
⟨*proof*⟩

lemma $x \leq y \Longrightarrow x < y + (1::real)$
⟨*proof*⟩

lemma $(x - y) + y = (x::real)$
⟨*proof*⟩

lemma $y + (x - y) = (x::real)$
⟨*proof*⟩

lemma $x - x = (0::real)$
⟨*proof*⟩

lemma $(x - y = 0) = (x = (y::real))$
⟨*proof*⟩

lemma $((0::real) \leq x + x) = (0 \leq x)$
⟨*proof*⟩

lemma $(-x \leq x) = ((0::real) \leq x)$
⟨*proof*⟩

lemma $(x \leq -x) = (x \leq (0::real))$
⟨*proof*⟩

lemma $(-x = (0::real)) = (x = 0)$
⟨*proof*⟩

lemma $-(x - y) = y - (x::real)$
⟨*proof*⟩

lemma $((0::real) < x - y) = (y < x)$
⟨*proof*⟩

lemma $((0::real) \leq x - y) = (y \leq x)$
⟨*proof*⟩

lemma $(x + y) - x = (y::real)$
⟨*proof*⟩

lemma $(-x = y) = (x = (-y::real))$
⟨*proof*⟩

lemma $x < (y::real) ==> \neg(x = y)$
⟨*proof*⟩

lemma $(x \leq x + y) = ((0::real) \leq y)$
⟨*proof*⟩

lemma $(y \leq x + y) = ((0::real) \leq x)$
⟨*proof*⟩

lemma $(x < x + y) = ((0::real) < y)$
⟨*proof*⟩

lemma $(y < x + y) = ((0::real) < x)$
⟨*proof*⟩

lemma $(x - y) - x = (-y::real)$
⟨*proof*⟩

lemma $(x + y < z) = (x < z - (y::real))$
⟨*proof*⟩

lemma $(x - y < z) = (x < z + (y::real))$
⟨*proof*⟩

lemma $(x < y - z) = (x + z < (y::real))$
⟨*proof*⟩

lemma $(x \leq y - z) = (x + z \leq (y::real))$
⟨*proof*⟩

lemma $(x - y \leq z) = (x \leq z + (y::real))$
⟨*proof*⟩

lemma $(-x < -y) = (y < (x::real))$
⟨*proof*⟩

lemma $(-x \leq -y) = (y \leq (x::real))$
⟨*proof*⟩

lemma $(a + b) - (c + d) = (a - c) + (b - (d::real))$
⟨*proof*⟩

lemma $(0::real) - x = -x$
⟨*proof*⟩

lemma $x - (0::real) = x$
⟨*proof*⟩

lemma $w \leq x \wedge y < z ==> w + y < x + (z::real)$
⟨*proof*⟩

lemma $w < x \wedge y \leq z ==> w + y < x + (z::real)$
⟨*proof*⟩

lemma $(0::real) \leq x \wedge 0 < y ==> 0 < x + (y::real)$
⟨*proof*⟩

lemma $(0::real) < x \wedge 0 \leq y ==> 0 < x + y$
⟨*proof*⟩

lemma $-x - y = -(x + (y::real))$
⟨*proof*⟩

lemma $x - (-y) = x + (y::real)$
⟨*proof*⟩

lemma $-x - -y = y - (x::real)$
⟨*proof*⟩

lemma $(a - b) + (b - c) = a - (c::real)$

⟨*proof*⟩

lemma $(x = y - z) = (x + z = (y::real))$
⟨*proof*⟩

lemma $(x - y = z) = (x = z + (y::real))$
⟨*proof*⟩

lemma $x - (x - y) = (y::real)$
⟨*proof*⟩

lemma $x - (x + y) = -(y::real)$
⟨*proof*⟩

lemma $x = y ==> x \leq (y::real)$
⟨*proof*⟩

lemma $(0::real) < x ==> \neg(x = 0)$
⟨*proof*⟩

lemma $(x + y) * (x - y) = (x * x) - (y * y)$
⟨*proof*⟩

lemma $(-x = -y) = (x = (y::real))$
⟨*proof*⟩

lemma $(-x < -y) = (y < (x::real))$
⟨*proof*⟩

lemma $!!a::real.\ a \leq b ==> c \leq d ==> x + y < z ==> a + c \leq b + d$
⟨*proof*⟩

lemma $!!a::real.\ a < b ==> c < d ==> a - d \leq b + (-c)$
⟨*proof*⟩

lemma $!!a::real.\ a \leq b ==> b + b \leq c ==> a + a \leq c$
⟨*proof*⟩

lemma $!!a::real.\ a + b \leq i + j ==> a \leq b ==> i \leq j ==> a + a \leq j + j$
⟨*proof*⟩

lemma $!!a::real.\ a + b < i + j ==> a < b ==> i < j ==> a + a < j + j$
⟨*proof*⟩

lemma $!!a::real.\ a + b + c \leq i + j + k \wedge a \leq b \wedge b \leq c \wedge i \leq j \wedge j \leq k --> a + a + a \leq k + k + k$
⟨*proof*⟩

lemma $!!a::real.\ a + b + c + d \leq i + j + k + l ==> a \leq b ==> b \leq c$

$==> c \leq d ==> i \leq j ==> j \leq k ==> k \leq l ==> a \leq l$
⟨*proof*⟩

lemma !!a::*real*. $a + b + c + d \leq i + j + k + l ==> a \leq b ==> b \leq c$
$==> c \leq d ==> i \leq j ==> j \leq k ==> k \leq l ==> a + a + a + a \leq l + l + l + l$
⟨*proof*⟩

lemma !!a::*real*. $a + b + c + d \leq i + j + k + l ==> a \leq b ==> b \leq c$
$==> c \leq d ==> i \leq j ==> j \leq k ==> k \leq l ==> a + a + a + a + a \leq l + l + l + l + i$
⟨*proof*⟩

lemma !!a::*real*. $a + b + c + d \leq i + j + k + l ==> a \leq b ==> b \leq c$
$==> c \leq d ==> i \leq j ==> j \leq k ==> k \leq l ==> a + a + a + a + a + a \leq l + l + l + l + i + l$
⟨*proof*⟩

14.6 Complex Arithmetic

lemma $(1359 + 93746*ii) - (2468 + 46375*ii) = -1109 + 47371*ii$
⟨*proof*⟩

lemma $-(65745 + -47371*ii) = -65745 + 47371*ii$
⟨*proof*⟩

Multiplication requires distributive laws. Perhaps versions instantiated to literal constants should be added to the simpset.

lemma $(1 + ii) * (1 - ii) = 2$
⟨*proof*⟩

lemma $(1 + 2*ii) * (1 + 3*ii) = -5 + 5*ii$
⟨*proof*⟩

lemma $(-84 + 255*ii) + (51 * 255*ii) = -84 + 13260 * ii$
⟨*proof*⟩

No inequalities or linear arithmetic: the complex numbers are unordered!

No powers (not supported yet)

end

15 Examples for hexadecimal and binary numerals

theory *Hex-Bin-Examples* **imports** *Main*
begin

Hex and bin numerals can be used like normal decimal numerals in input

lemma *0xFF = 255* ⟨*proof*⟩
lemma *0xF = 0b1111* ⟨*proof*⟩

Just like decimal numeral they are polymorphic, for arithmetic they need to be constrained

lemma *0x0A + 0x10 = (0x1A :: nat)* ⟨*proof*⟩

The number of leading zeros is irrelevant

lemma *0b00010000 = 0x10* ⟨*proof*⟩

Unary minus works as for decimal numerals

lemma *− 0x0A = − 10* ⟨*proof*⟩

Hex and bin numerals are printed as decimal: *2*::*'a*

term *0b10*
term *0x0A*

The numerals 0 and 1 are syntactically different from the constants 0 and 1. For the usual numeric types, their values are the same, though.

lemma *0x01 = 1* ⟨*proof*⟩
lemma *0x00 = 0* ⟨*proof*⟩

lemma *0x01 = (1*::*nat)* ⟨*proof*⟩
lemma *0b0000 = (0*::*int)* ⟨*proof*⟩

end

16 Antiquotations

theory *Antiquote* **imports** *Main* **begin**

A simple example on quote / antiquote in higher-order abstract syntax.

syntax
-Expr :: *'a => 'a* (*EXPR - [1000] 999*)

constdefs
var :: *'a => ('a => nat) => nat* (*VAR - [1000] 999*)
var x env == env x

Expr :: *(('a => nat) => nat) => ('a => nat) => nat*
Expr exp env == exp env

⟨*ML*⟩

term *EXPR* (*a* + *b* + *c*)
term *EXPR* (*a* + *b* + *c* + *VAR x* + *VAR y* + *1*)
term *EXPR* (*VAR* (*f w*) + *VAR x*)

term *Expr* (λ*env*. *env x*)
term *Expr* (λ*env*. *f env*)
term *Expr* (λ*env*. *f env* + *env x*)
term *Expr* (λ*env*. *f env y z*)
term *Expr* (λ*env*. *f env* + *g y env*)
term *Expr* (λ*env*. *f env* + *g env y* + *h a env z*)

end

17 Multiple nested quotations and anti-quotations

theory *Multiquote* **imports** *Main* **begin**

Multiple nested quotations and anti-quotations – basically a generalized version of de-Bruijn representation.

syntax
-quote :: *'b* => (*'a* => *'b*) (≪-≫ [*0*] *1000*)
-antiquote :: (*'a* => *'b*) => *'b* (´- [*1000*] *1000*)

⟨*ML*⟩

basic examples

term ≪*a* + *b* + *c*≫
term ≪*a* + *b* + *c* + ´*x* + ´*y* + *1*≫
term ≪´(*f w*) + ´*x*≫
term ≪*f* ´*x* ´*y z*≫

advanced examples

term ≪≪´´*x* + ´*y*≫≫
term ≪≪´´*x* + ´*y*≫ *o* ´*f*≫
term ≪´(*f o* ´*g*)≫
term ≪≪´´(*f o* ´*g*)≫≫

end

18 Partial equivalence relations

theory *PER* **imports** *Main* **begin**

Higher-order quotients are defined over partial equivalence relations (PERs) instead of total ones. We provide axiomatic type classes *equiv* < *partial-equiv*

and a type constructor *'a quot* with basic operations. This development is based on:

Oscar Slotosch: *Higher Order Quotients and their Implementation in Isabelle HOL.* Elsa L. Gunter and Amy Felty, editors, Theorem Proving in Higher Order Logics: TPHOLs '97, Springer LNCS 1275, 1997.

18.1 Partial equivalence

Type class *partial-equiv* models partial equivalence relations (PERs) using the polymorphic $\sim :: 'a \Rightarrow 'a \Rightarrow bool$ relation, which is required to be symmetric and transitive, but not necessarily reflexive.

consts
  eqv :: *'a => 'a => bool* (**infixl** $\sim$ *50*)

axclass *partial-equiv* < *type*
  partial-equiv-sym [*elim?*]: $x \sim y \Longrightarrow y \sim x$
  partial-equiv-trans [*trans*]: $x \sim y \Longrightarrow y \sim z \Longrightarrow x \sim z$

The domain of a partial equivalence relation is the set of reflexive elements. Due to symmetry and transitivity this characterizes exactly those elements that are connected with *any* other one.

definition
  domain :: *'a*::*partial-equiv set* **where**
  $domain = \{x.\ x \sim x\}$

lemma *domainI* [*intro*]: $x \sim x \Longrightarrow x \in domain$
  ⟨*proof*⟩

lemma *domainD* [*dest*]: $x \in domain \Longrightarrow x \sim x$
  ⟨*proof*⟩

theorem *domainI'* [*elim?*]: $x \sim y \Longrightarrow x \in domain$
⟨*proof*⟩

18.2 Equivalence on function spaces

The $\sim$ relation is lifted to function spaces. It is important to note that this is *not* the direct product, but a structural one corresponding to the congruence property.

defs (**overloaded**)
  eqv-fun-def: $f \sim g == \forall x \in domain.\ \forall y \in domain.\ x \sim y \longrightarrow f\ x \sim g\ y$

lemma *partial-equiv-funI* [*intro?*]:
  $(!!x\ y.\ x \in domain \Longrightarrow y \in domain \Longrightarrow x \sim y \Longrightarrow f\ x \sim g\ y) \Longrightarrow f \sim g$
  ⟨*proof*⟩

lemma *partial-equiv-funD* [*dest?*]:
$f \sim g \Longrightarrow x \in \mathit{domain} \Longrightarrow y \in \mathit{domain} \Longrightarrow x \sim y \Longrightarrow f\,x \sim g\,y$
⟨*proof*⟩

The class of partial equivalence relations is closed under function spaces (in *both* argument positions).

instance *fun* :: (*partial-equiv*, *partial-equiv*) *partial-equiv*
⟨*proof*⟩

18.3 Total equivalence

The class of total equivalence relations on top of PERs. It coincides with the standard notion of equivalence, i.e. $\sim :: {'a} \Rightarrow {'a} \Rightarrow \mathit{bool}$ is required to be reflexive, transitive and symmetric.

axclass *equiv* < *partial-equiv*
eqv-refl [*intro*]: $x \sim x$

On total equivalences all elements are reflexive, and congruence holds unconditionally.

theorem *equiv-domain* [*intro*]: $(x::'a::\mathit{equiv}) \in \mathit{domain}$
⟨*proof*⟩

theorem *equiv-cong* [*dest?*]: $f \sim g \Longrightarrow x \sim y \Longrightarrow f\,x \sim g\,(y::'a::\mathit{equiv})$
⟨*proof*⟩

18.4 Quotient types

The quotient type ${'a}\ \mathit{quot}$ consists of all *equivalence classes* over elements of the base type ${'a}$.

typedef ${'a}\ \mathit{quot} = \{\{x.\ a \sim x\} |\ a::'a.\ \mathit{True}\}$
⟨*proof*⟩

lemma *quotI* [*intro*]: $\{x.\ a \sim x\} \in \mathit{quot}$
⟨*proof*⟩

lemma *quotE* [*elim*]: $R \in \mathit{quot} \Longrightarrow (!!a.\ R = \{x.\ a \sim x\} \Longrightarrow C) \Longrightarrow C$
⟨*proof*⟩

Abstracted equivalence classes are the canonical representation of elements of a quotient type.

definition
eqv-class :: $('a::\mathit{partial\text{-}equiv}) \Rightarrow {'a}\ \mathit{quot}$ $(\lfloor\text{-}\rfloor)$ **where**
$\lfloor a \rfloor = \mathit{Abs\text{-}quot}\ \{x.\ a \sim x\}$

theorem *quot-rep*: $\exists\, a.\ A = \lfloor a \rfloor$

⟨*proof*⟩

lemma *quot-cases* [*cases type*: *quot*]:
obtains (*rep*) *a* **where** $A = \lfloor a \rfloor$
⟨*proof*⟩

18.5 Equality on quotients

Equality of canonical quotient elements corresponds to the original relation as follows.

theorem *eqv-class-eqI* [*intro*]: $a \sim b \Longrightarrow \lfloor a \rfloor = \lfloor b \rfloor$
⟨*proof*⟩

theorem *eqv-class-eqD'* [*dest?*]: $\lfloor a \rfloor = \lfloor b \rfloor \Longrightarrow a \in \mathit{domain} \Longrightarrow a \sim b$
⟨*proof*⟩

theorem *eqv-class-eqD* [*dest?*]: $\lfloor a \rfloor = \lfloor b \rfloor \Longrightarrow a \sim (b::'a::\mathit{equiv})$
⟨*proof*⟩

lemma *eqv-class-eq'* [*simp*]: $a \in \mathit{domain} \Longrightarrow (\lfloor a \rfloor = \lfloor b \rfloor) = (a \sim b)$
⟨*proof*⟩

lemma *eqv-class-eq* [*simp*]: $(\lfloor a \rfloor = \lfloor b \rfloor) = (a \sim (b::'a::\mathit{equiv}))$
⟨*proof*⟩

18.6 Picking representing elements

definition
pick :: $'a::$*partial-equiv quot* $\Rightarrow 'a$ **where**
$\mathit{pick}\ A = (\mathit{SOME}\ a.\ A = \lfloor a \rfloor)$

theorem *pick-eqv'* [*intro?*, *simp*]: $a \in \mathit{domain} \Longrightarrow \mathit{pick} \lfloor a \rfloor \sim a$
⟨*proof*⟩

theorem *pick-eqv* [*intro*, *simp*]: $\mathit{pick} \lfloor a \rfloor \sim (a::'a::\mathit{equiv})$
⟨*proof*⟩

theorem *pick-inverse*: $\lfloor \mathit{pick}\ A \rfloor = (A::'a::\mathit{equiv}\ \mathit{quot})$
⟨*proof*⟩

end

19 Summing natural numbers

theory *NatSum* **imports** *Main Parity* **begin**

Summing natural numbers, squares, cubes, etc.

Thanks to Sloane's On-Line Encyclopedia of Integer Sequences, http://www.research.att.com/~njas/sequences/.

lemmas [*simp*] =
ring-distribs
diff-mult-distrib diff-mult-distrib2 — for type nat

The sum of the first n odd numbers equals n squared.

lemma *sum-of-odds*: $(\sum i{=}0..{<}n.\ Suc\ (i + i)) = n * n$
⟨*proof*⟩

The sum of the first n odd squares.

lemma *sum-of-odd-squares*:
$3 * (\sum i{=}0..{<}n.\ Suc(2{*}i) * Suc(2{*}i)) = n * (4 * n * n - 1)$
⟨*proof*⟩

The sum of the first n odd cubes

lemma *sum-of-odd-cubes*:
$(\sum i{=}0..{<}n.\ Suc\ (2{*}i) * Suc\ (2{*}i) * Suc\ (2{*}i)) =$
$n * n * (2 * n * n - 1)$
⟨*proof*⟩

The sum of the first n positive integers equals $n\ (n + 1)\ /\ 2$.

lemma *sum-of-naturals*:
$2 * (\sum i{=}0..n.\ i) = n * Suc\ n$
⟨*proof*⟩

lemma *sum-of-squares*:
$6 * (\sum i{=}0..n.\ i * i) = n * Suc\ n * Suc\ (2 * n)$
⟨*proof*⟩

lemma *sum-of-cubes*:
$4 * (\sum i{=}0..n.\ i * i * i) = n * n * Suc\ n * Suc\ n$
⟨*proof*⟩

A cute identity:

lemma *sum-squared*: $(\sum i{=}0..n.\ i)\ \hat{}\ 2 = (\sum i{=}0..n{::}nat.\ i\hat{}3)$
⟨*proof*⟩

Sum of fourth powers: three versions.

lemma *sum-of-fourth-powers*:
$30 * (\sum i{=}0..n.\ i * i * i * i) =$
$n * Suc\ n * Suc\ (2 * n) * (3 * n * n + 3 * n - 1)$
⟨*proof*⟩

Two alternative proofs, with a change of variables and much more subtraction, performed using the integers.

lemma *int-sum-of-fourth-powers*:
*30 * int (∑ i=0..<m. i * i * i * i) =*
*int m * (int m − 1) * (int(2 * m) − 1) **
*(int(3 * m * m) − int(3 * m) − 1)*
⟨*proof*⟩

lemma *of-nat-sum-of-fourth-powers*:
*30 * of-nat (∑ i=0..<m. i * i * i * i) =*
*of-nat m * (of-nat m − 1) * (of-nat (2 * m) − 1) **
*(of-nat (3 * m * m) − of-nat (3 * m) − (1::int))*
⟨*proof*⟩

Sums of geometric series: *2*, *3* and the general case.

lemma *sum-of-2-powers*: *(∑ i=0..<n. 2^i) = 2^n − (1::nat)*
⟨*proof*⟩

lemma *sum-of-3-powers*: *2 * (∑ i=0..<n. 3^i) = 3^n − (1::nat)*
⟨*proof*⟩

lemma *sum-of-powers*: *0 < k ==> (k − 1) * (∑ i=0..<n. k^i) = k^n − (1::nat)*
⟨*proof*⟩

end

20 Three Divides Theorem

theory *ThreeDivides*
imports *Main LaTeXsugar*
begin

20.1 Abstract

The following document presents a proof of the Three Divides N theorem formalised in the Isabelle/Isar theorem proving system.

Theorem: 3 divides n if and only if 3 divides the sum of all digits in n.

Informal Proof: Take $n = \sum n_j * 10^j$ where n_j is the j'th least significant digit of the decimal denotation of the number n and the sum ranges over all digits. Then

$$(n - \sum n_j) = \sum n_j * (10^j - 1)$$

We know $\forall j\ 3|(10^j - 1)$ and hence $3|LHS$, therefore

$$\forall n\ 3|n \Longleftrightarrow 3|\sum n_j$$

□

20.2 Formal proof

20.2.1 Miscellaneous summation lemmas

If a divides $A\ x$ for all x then a divides any sum over terms of the form $(A\ x)*(P\ x)$ for arbitrary P.

lemma *div-sum*:
  fixes a::*nat* **and** n::*nat*
  shows $\forall x.\ a\ dvd\ A\ x \Longrightarrow a\ dvd\ (\sum x{<}n.\ A\ x * D\ x)$
⟨*proof*⟩

20.2.2 Generalised Three Divides

This section solves a generalised form of the three divides problem. Here we show that for any sequence of numbers the theorem holds. In the next section we specialise this theorem to apply directly to the decimal expansion of the natural numbers.

Here we show that the first statement in the informal proof is true for all natural numbers. Note we are using $D\ i$ to denote the i'th element in a sequence of numbers.

lemma *digit-diff-split*:
  fixes n::*nat* **and** nd::*nat* **and** x::*nat*
  shows $n = (\sum x{\in}\{..{<}nd\}.\ (D\ x){*}((10{::}nat)\ \hat{}\ x)) \Longrightarrow$
        $(n - (\sum x{<}nd.\ (D\ x))) = (\sum x{<}nd.\ (D\ x){*}(10\ \hat{}\ x - 1))$
⟨*proof*⟩

Now we prove that 3 always divides numbers of the form $10^x - 1$.

lemma *three-divs-0*:
  shows $(3{::}nat)\ dvd\ (10\ \hat{}\ x - 1)$
⟨*proof*⟩

Expanding on the previous lemma and lemma *div-sum*.

lemma *three-divs-1*:
  fixes $D :: nat \Rightarrow nat$
  shows $3\ dvd\ (\sum x{<}nd.\ D\ x * (10\ \hat{}\ x - 1))$
  ⟨*proof*⟩

Using lemmas *digit-diff-split* and *three-divs-1* we now prove the following lemma.

lemma *three-divs-2*:
fixes *nd*::*nat* **and** *D*::*nat*⇒*nat*
shows *3 dvd* ((∑ *x*<*nd*. (*D x*)∗(*10*^*x*)) − (∑ *x*<*nd*. (*D x*)))
⟨*proof*⟩

We now present the final theorem of this section. For any sequence of numbers (defined by a function D), we show that 3 divides the expansive sum $\sum (D\ x) * 10^x$ over x if and only if 3 divides the sum of the individual numbers $\sum D\ x$.

lemma *three-div-general*:
fixes *D* :: *nat* ⇒ *nat*
shows (*3 dvd* (∑ *x*<*nd*. *D x* ∗ *10*^*x*)) = (*3 dvd* (∑ *x*<*nd*. *D x*))
⟨*proof*⟩

20.2.3 Three Divides Natural

This section shows that for all natural numbers we can generate a sequence of digits less than ten that represent the decimal expansion of the number. We then use the lemma *three-div-general* to prove our final theorem.

Definitions of length and digit sum.

This section introduces some functions to calculate the required properties of natural numbers. We then proceed to prove some properties of these functions.

The function *nlen* returns the number of digits in a natural number n.

consts *nlen* :: *nat* ⇒ *nat*
recdef *nlen measure id*
nlen 0 = *0*
nlen x = *1* + *nlen* (*x div 10*)

The function *sumdig* returns the sum of all digits in some number n.

definition
sumdig :: *nat* ⇒ *nat* **where**
sumdig n = (∑ *x* < *nlen n*. *n div 10*^*x mod 10*)

Some properties of these functions follow.

lemma *nlen-zero*:
0 = *nlen x* ⟹ *x* = *0*
⟨*proof*⟩

lemma *nlen-suc*:
Suc m = *nlen n* ⟹ *m* = *nlen* (*n div 10*)
⟨*proof*⟩

The following lemma is the principle lemma required to prove our theorem. It states that an expansion of some natural number n into a sequence of its individual digits is always possible.

lemma *exp-exists*:
  $m = (\sum x{<}nlen\ m.\ (m\ div\ (10{::}nat)\ \hat{}x\ mod\ 10) * 10\ \hat{}x)$
⟨*proof*⟩

Final theorem.

We now combine the general theorem *three-div-general* and existence result of *exp-exists* to prove our final theorem.

theorem *three-divides-nat*:
  shows (*3 dvd n*) = (*3 dvd sumdig n*)
⟨*proof*⟩

end

21 Higher-Order Logic: Intuitionistic predicate calculus problems

theory *Intuitionistic* **imports** *Main* **begin**

lemma $(\sim\sim(P\&Q)) = ((\sim\sim P)\ \&\ (\sim\sim Q))$
  ⟨*proof*⟩

lemma $\sim\sim\ ((\sim P\ {-}{-}{>}\ Q)\ {-}{-}{>}\ (\sim P\ {-}{-}{>}\ \sim Q)\ {-}{-}{>}\ P)$
  ⟨*proof*⟩

lemma $(\sim\sim(P{-}{-}{>}Q))\ = (\sim\sim P\ {-}{-}{>}\ \sim\sim Q)$
  ⟨*proof*⟩

lemma $(\sim\sim\sim P) = (\sim P)$
  ⟨*proof*⟩

lemma $\sim\sim((P\ {-}{-}{>}\ Q \mid R)\ \ {-}{-}{>}\ \ (P{-}{-}{>}Q) \mid (P{-}{-}{>}R))$
  ⟨*proof*⟩

lemma $(P{=}Q) = (Q{=}P)$
  ⟨*proof*⟩

lemma $((P\ {-}{-}{>}\ (Q \mid (Q{-}{-}{>}R)))\ {-}{-}{>}\ R)\ {-}{-}{>}\ R$
  ⟨*proof*⟩

lemma $(((G{-}{-}{>}A)\ {-}{-}{>}\ J)\ {-}{-}{>}\ D\ {-}{-}{>}\ E)\ {-}{-}{>}\ (((H{-}{-}{>}B){-}{-}{>}I){-}{-}{>}C{-}{-}{>}J)$
    $\ {-}{-}{>}\ (A{-}{-}{>}H)\ {-}{-}{>}\ F\ {-}{-}{>}\ G\ {-}{-}{>}\ (((C{-}{-}{>}B){-}{-}{>}I){-}{-}{>}D){-}{-}{>}(A{-}{-}{>}C)$

$--> (((F-->A)-->B) --> I) --> E$
⟨*proof*⟩

lemma $P --> \sim\sim P$
⟨*proof*⟩

lemma $\sim\sim(\sim\sim P --> P)$
⟨*proof*⟩

lemma $\sim\sim P \;\&\; \sim\sim(P --> Q) --> \sim\sim Q$
⟨*proof*⟩

lemma $((P=Q) --> P\&Q\&R) \;\&$
$((Q=R) --> P\&Q\&R) \;\&$
$((R=P) --> P\&Q\&R) --> P\&Q\&R$
⟨*proof*⟩

lemma $((P=Q) --> P\&Q\&R\&S\&T) \;\&$
$((Q=R) --> P\&Q\&R\&S\&T) \;\&$
$((R=S) --> P\&Q\&R\&S\&T) \;\&$
$((S=T) --> P\&Q\&R\&S\&T) \;\&$
$((T=P) --> P\&Q\&R\&S\&T) --> P\&Q\&R\&S\&T$
⟨*proof*⟩

lemma $(ALL\ x.\ EX\ y.\ ALL\ z.\ p(x) \;\&\; q(y) \;\&\; r(z)) =$
$(ALL\ z.\ EX\ y.\ ALL\ x.\ p(x) \;\&\; q(y) \;\&\; r(z))$
⟨*proof*⟩

lemma $\sim\ (EX\ x.\ ALL\ y.\ p\ y\ x = (\sim\ p\ x\ x))$
⟨*proof*⟩

lemma $\sim\sim((P-->Q) \;=\; (\sim Q --> \sim P))$

⟨*proof*⟩

lemma $\sim\sim(\sim\sim P \;=\; P)$
⟨*proof*⟩

lemma $\sim(P{-}{-}{>}Q) \;{-}{-}{>}\; (Q{-}{-}{>}P)$
⟨*proof*⟩

lemma $\sim\sim((\sim P{-}{-}{>}Q) \;=\; (\sim Q \;{-}{-}{>}\; P))$
⟨*proof*⟩

lemma $\sim\sim((P|Q{-}{-}{>}P|R) \;{-}{-}{>}\; P|(Q{-}{-}{>}R))$
⟨*proof*⟩

lemma $\sim\sim(P \mid \sim P)$
⟨*proof*⟩

lemma $\sim\sim(P \mid \sim\sim\sim P)$
⟨*proof*⟩

lemma $\sim\sim(((P{-}{-}{>}Q) \;{-}{-}{>}\; P) \;\; {-}{-}{>}\;\; P)$
⟨*proof*⟩

lemma $((P|Q) \;\&\; (\sim P|Q) \;\&\; (P|\sim Q)) \;{-}{-}{>}\; \sim\; (\sim P \mid \sim Q)$
⟨*proof*⟩

lemma $(Q{-}{-}{>}R) \;{-}{-}{>}\; (R{-}{-}{>}P\&Q) \;{-}{-}{>}\; (P{-}{-}{>}(Q|R)) \;{-}{-}{>}\; (P{=}Q)$
⟨*proof*⟩

lemma $P{=}P$
⟨*proof*⟩

lemma $\sim\sim(((P \;=\; Q) \;=\; R) \;\; = \;\; (P \;=\; (Q \;=\; R)))$
⟨*proof*⟩

lemma $((P \;=\; Q) \;=\; R) \;\; {-}{-}{>}\;\; \sim\sim(P \;=\; (Q \;=\; R))$
⟨*proof*⟩

lemma $(P \mid (Q \;\&\; R)) \;= ((P \mid Q) \;\&\; (P \mid R))$
⟨*proof*⟩

lemma $^{\sim}{}^{\sim}((P = Q) = ((Q \mid {}^{\sim}P) \;\&\; ({}^{\sim}Q|P)))$
⟨*proof*⟩

lemma $^{\sim}{}^{\sim}((P \dashrightarrow Q) = ({}^{\sim}P \mid Q))$
⟨*proof*⟩

lemma $^{\sim}{}^{\sim}((P \dashrightarrow Q) \mid (Q \dashrightarrow P))$
⟨*proof*⟩

lemma $^{\sim}{}^{\sim}(((P \;\&\; (Q \dashrightarrow R)) \dashrightarrow S) = (({}^{\sim}P \mid Q \mid S) \;\&\; ({}^{\sim}P \mid {}^{\sim}R \mid S)))$
⟨*proof*⟩

lemma $(P \& Q) = (P = (Q = (P|Q)))$
⟨*proof*⟩

lemma $(EX\ x.\ P(x) \dashrightarrow Q) \;\dashrightarrow\; (ALL\ x.\ P(x)) \dashrightarrow Q$
⟨*proof*⟩

lemma $((ALL\ x.\ P(x)) \dashrightarrow Q) \dashrightarrow {}^{\sim}\ (ALL\ x.\ P(x) \;\&\; {}^{\sim}Q)$
⟨*proof*⟩

lemma $((ALL\ x.\ {}^{\sim}P(x)) \dashrightarrow Q) \;\dashrightarrow\; {}^{\sim}\ (ALL\ x.\ {}^{\sim}\ (P(x)|Q))$
⟨*proof*⟩

lemma $(ALL\ x.\ P(x)) \mid Q \;\dashrightarrow\; (ALL\ x.\ P(x) \mid Q)$
⟨*proof*⟩

lemma $(EX\ x.\ P \dashrightarrow Q(x)) \dashrightarrow (P \dashrightarrow (EX\ x.\ Q(x)))$
⟨*proof*⟩

lemma $\sim\sim(EX\ x.\ ALL\ y\ z.\ (P(y)-->Q(z))\ -->\ (P(x)-->Q(x)))$
⟨*proof*⟩

lemma $(ALL\ x\ y.\ EX\ z.\ ALL\ w.\ (P(x)\&Q(y)-->R(z)\&S(w)))$
$-->\ (EX\ x\ y.\ P(x)\ \&\ Q(y))\ -->\ (EX\ z.\ R(z))$
⟨*proof*⟩

lemma $(EX\ x.\ P-->Q(x))\ \&\ (EX\ x.\ Q(x)-->P)\ -->\ \sim\sim(EX\ x.\ P=Q(x))$
⟨*proof*⟩

lemma $(ALL\ x.\ P\ =\ Q(x))\ \ -->\ \ (P\ =\ (ALL\ x.\ Q(x)))$
⟨*proof*⟩

lemma $\sim\sim\ ((ALL\ x.\ P\ |\ Q(x))\ \ =\ \ (P\ |\ (ALL\ x.\ Q(x))))$
⟨*proof*⟩

lemma $(EX\ x.\ P(x))\ \&$
$(ALL\ x.\ L(x)\ -->\ \sim\ (M(x)\ \&\ R(x)))\ \&$
$(ALL\ x.\ P(x)\ -->\ (M(x)\ \&\ L(x)))\ \&$
$((ALL\ x.\ P(x)-->Q(x))\ |\ (EX\ x.\ P(x)\&R(x)))$
$-->\ (EX\ x.\ Q(x)\&P(x))$
⟨*proof*⟩

lemma $(EX\ x.\ P(x)\ \&\ \sim Q(x))\ \&$
$(ALL\ x.\ P(x)\ -->\ R(x))\ \&$
$(ALL\ x.\ M(x)\ \&\ L(x)\ -->\ P(x))\ \&$
$((EX\ x.\ R(x)\ \&\ \sim\ Q(x))\ -->\ (ALL\ x.\ L(x)\ -->\ \sim\ R(x)))$
$-->\ (ALL\ x.\ M(x)\ -->\ \sim L(x))$
⟨*proof*⟩

lemma $(ALL\ x.\ P(x)\ -->\ (ALL\ x.\ Q(x)))\ \&$
$(\sim\sim(ALL\ x.\ Q(x)|R(x))\ -->\ (EX\ x.\ Q(x)\&S(x)))\ \&$
$(\sim\sim(EX\ x.\ S(x))\ -->\ (ALL\ x.\ L(x)\ -->\ M(x)))$
$-->\ (ALL\ x.\ P(x)\ \&\ L(x)\ -->\ M(x))$
⟨*proof*⟩

lemma $(((EX\ x.\ P(x))\ \&\ (EX\ y.\ Q(y)))\ -->$
$(((ALL\ x.\ (P(x)\ -->\ R(x)))\ \&\ (ALL\ y.\ (Q(y)\ -->\ S(y))))\ =$
$(ALL\ x\ y.\ ((P(x)\ \&\ Q(y))\ -->\ (R(x)\ \&\ S(y))))))$
⟨*proof*⟩

lemma $(ALL\ x.\ (P(x) \mid Q(x)) \dashrightarrow {\sim} R(x)) \ \&$
$(ALL\ x.\ (Q(x) \dashrightarrow {\sim} S(x)) \dashrightarrow P(x) \ \& \ R(x))$
$\dashrightarrow (ALL\ x.\ {\sim\sim} S(x))$
⟨*proof*⟩

lemma ${\sim}(EX\ x.\ P(x) \ \& \ (Q(x) \mid R(x))) \ \&$
$(EX\ x.\ L(x) \ \& \ P(x)) \ \&$
$(ALL\ x.\ {\sim} R(x) \dashrightarrow M(x))$
$\dashrightarrow (EX\ x.\ L(x) \ \& \ M(x))$
⟨*proof*⟩

lemma $(ALL\ x.\ P(x) \ \& \ (Q(x) \mid R(x)) \dashrightarrow S(x)) \ \&$
$(ALL\ x.\ S(x) \ \& \ R(x) \dashrightarrow L(x)) \ \&$
$(ALL\ x.\ M(x) \dashrightarrow R(x))$
$\dashrightarrow (ALL\ x.\ P(x) \ \& \ M(x) \dashrightarrow L(x))$
⟨*proof*⟩

lemma $(ALL\ x.\ {\sim\sim}(P(a) \ \& \ (P(x) \dashrightarrow P(b)) \dashrightarrow P(c))) \ =$
$(ALL\ x.\ {\sim\sim}(({\sim} P(a) \mid P(x) \mid P(c)) \ \& \ ({\sim} P(a) \mid {\sim} P(b) \mid P(c))))$
⟨*proof*⟩

lemma
$(ALL\ x.\ EX\ y.\ J\ x\ y) \ \&$
$(ALL\ x.\ EX\ y.\ G\ x\ y) \ \&$
$(ALL\ x\ y.\ J\ x\ y \mid G\ x\ y \dashrightarrow (ALL\ z.\ J\ y\ z \mid G\ y\ z \dashrightarrow H\ x\ z))$
$\dashrightarrow (ALL\ x.\ EX\ y.\ H\ x\ y)$
⟨*proof*⟩

lemma ${\sim}\ (EX\ x.\ ALL\ y.\ F\ y\ x = ({\sim} F\ y\ y))$
⟨*proof*⟩

lemma $(EX\ y.\ ALL\ x.\ F\ x\ y = F\ x\ x) \dashrightarrow$
${\sim}(ALL\ x.\ EX\ y.\ ALL\ z.\ F\ z\ y = ({\sim}\ F\ z\ x))$
⟨*proof*⟩

lemma $(ALL\ x.\ f(x) \dashrightarrow$
$(EX\ y.\ g(y) \ \& \ h\ x\ y \ \& \ (EX\ y.\ g(y) \ \& \ {\sim}\ h\ x\ y))) \ \&$
$(EX\ x.\ j(x) \ \& \ (ALL\ y.\ g(y) \dashrightarrow h\ x\ y))$
$\dashrightarrow (EX\ x.\ j(x) \ \& \ {\sim} f(x))$
⟨*proof*⟩

lemma $(a{=}b \mid c{=}d)$ & $(a{=}c \mid b{=}d)$ $--{>}$ $a{=}d \mid b{=}c$
⟨*proof*⟩

lemma $((EX\ z\ w.\ (ALL\ x\ y.\ (P\ x\ y = ((x = z)\ \&\ (y = w))))) --{>}$
$(EX\ z.\ (ALL\ x.\ (EX\ w.\ ((ALL\ y.\ (P\ x\ y = (y = w))) = (x = z))))))$
⟨*proof*⟩

lemma $((EX\ z\ w.\ (ALL\ x\ y.\ (P\ x\ y = ((x = z)\ \&\ (y = w))))) --{>}$
$(EX\ w.\ (ALL\ y.\ (EX\ z.\ ((ALL\ x.\ (P\ x\ y = (x = z))) = (y = w))))))$
⟨*proof*⟩

lemma $(ALL\ x.\ (EX\ y.\ P(y)\ \&\ x{=}f(y))\ --{>}\ P(x)) = (ALL\ x.\ P(x)\ --{>}\ P(f(x)))$
⟨*proof*⟩

lemma $P\ (f\ a\ b)\ (f\ b\ c)\ \&\ P\ (f\ b\ c)\ (f\ a\ c)\ \&$
$(ALL\ x\ y\ z.\ P\ x\ y\ \&\ P\ y\ z\ --{>}\ P\ x\ z)\ --{>}\ P\ (f\ a\ b)\ (f\ a\ c)$
⟨*proof*⟩

lemma $ALL\ x.\ P\ x\ (f\ x) = (EX\ y.\ (ALL\ z.\ P\ z\ y\ --{>}\ P\ z\ (f\ x))\ \&\ P\ x\ y)$
⟨*proof*⟩

end

22 CTL formulae

theory *CTL* **imports** *Main* **begin**

We formalize basic concepts of Computational Tree Logic (CTL) [3, 2] within the simply-typed set theory of HOL.

By using the common technique of "shallow embedding", a CTL formula is identified with the corresponding set of states where it holds. Consequently, CTL operations such as negation, conjunction, disjunction simply become complement, intersection, union of sets. We only require a separate operation for implication, as point-wise inclusion is usually not encountered in plain set-theory.

lemmas [*intro*!] = *Int-greatest Un-upper2 Un-upper1 Int-lower1 Int-lower2*

types $'a\ ctl = 'a\ set$

definition
imp :: *'a ctl* ⇒ *'a ctl* ⇒ *'a ctl* (**infixr** → *75*) **where**
p → *q* = − *p* ∪ *q*

lemma [*intro*!]: *p* ∩ *p* → *q* ⊆ *q* ⟨*proof*⟩
lemma [*intro*!]: *p* ⊆ (*q* → *p*) ⟨*proof*⟩

The CTL path operators are more interesting; they are based on an arbitrary, but fixed model $\mathcal{M}$, which is simply a transition relation over states *'a*.

axiomatization $\mathcal{M}$ :: (*'a* × *'a*) *set*

The operators EX, EF, EG are taken as primitives, while AX, AF, AG are defined as derived ones. The formula EX p holds in a state s, iff there is a successor state s' (with respect to the model $\mathcal{M}$), such that p holds in s'. The formula EF p holds in a state s, iff there is a path in $\mathcal{M}$, starting from s, such that there exists a state s' on the path, such that p holds in s'. The formula EG p holds in a state s, iff there is a path, starting from s, such that for all states s' on the path, p holds in s'. It is easy to see that EF p and EG p may be expressed using least and greatest fixed points [3].

definition
EX (EX - [*80*] *90*) **where** EX *p* = {*s*. ∃ *s'*. (*s*, *s'*) ∈ $\mathcal{M}$ ∧ *s'* ∈ *p*}
definition
EF (EF - [*80*] *90*) **where** EF *p* = *lfp* (λ*s*. *p* ∪ EX *s*)
definition
EG (EG - [*80*] *90*) **where** EG *p* = *gfp* (λ*s*. *p* ∩ EX *s*)

AX, AF and AG are now defined dually in terms of EX, EF and EG.

definition
AX (AX - [*80*] *90*) **where** AX *p* = − EX − *p*
definition
AF (AF - [*80*] *90*) **where** AF *p* = − EG − *p*
definition
AG (AG - [*80*] *90*) **where** AG *p* = − EF − *p*

lemmas [*simp*] = *EX-def EG-def AX-def EF-def AF-def AG-def*

22.1 Basic fixed point properties

First of all, we use the de-Morgan property of fixed points

lemma *lfp-gfp*: *lfp f* = − *gfp* (λ*s*::*'a set*. − (*f* (− *s*)))
⟨*proof*⟩

lemma *lfp-gfp'*: − *lfp f* = *gfp* (λ*s*::*'a set*. − (*f* (− *s*)))
⟨*proof*⟩

lemma *gfp-lfp'*: $-$ *gfp* f = *lfp* ($\lambda s::'a$ *set*. $-$ (f ($-$ s)))
⟨*proof*⟩

in order to give dual fixed point representations of AF p and AG p:

lemma *AF-lfp*: AF p = *lfp* (λs. $p \cup$ AX s) ⟨*proof*⟩
lemma *AG-gfp*: AG p = *gfp* (λs. $p \cap$ AX s) ⟨*proof*⟩

lemma *EF-fp*: EF $p = p \cup$ EX EF p
⟨*proof*⟩

lemma *AF-fp*: AF $p = p \cup$ AX AF p
⟨*proof*⟩

lemma *EG-fp*: EG $p = p \cap$ EX EG p
⟨*proof*⟩

From the greatest fixed point definition of AG p, we derive as a consequence of the Knaster-Tarski theorem on the one hand that AG p is a fixed point of the monotonic function λs. $p \cap$ AX s.

lemma *AG-fp*: AG $p = p \cap$ AX AG p
⟨*proof*⟩

This fact may be split up into two inequalities (merely using transitivity of $\subseteq$, which is an instance of the overloaded $\leq$ in Isabelle/HOL).

lemma *AG-fp-1*: AG $p \subseteq p$
⟨*proof*⟩

lemma *AG-fp-2*: AG $p \subseteq$ AX AG p
⟨*proof*⟩

On the other hand, we have from the Knaster-Tarski fixed point theorem that any other post-fixed point of λs. $p \cap$ AX s is smaller than AG p. A post-fixed point is a set of states q such that $q \subseteq p \cap$ AX q. This leads to the following co-induction principle for AG p.

lemma *AG-I*: $q \subseteq p \cap$ AX $q \Longrightarrow q \subseteq$ AG p
⟨*proof*⟩

22.2 The tree induction principle

With the most basic facts available, we are now able to establish a few more interesting results, leading to the *tree induction* principle for AG (see below). We will use some elementary monotonicity and distributivity rules.

lemma *AX-int*: AX ($p \cap q$) = AX $p \cap$ AX q ⟨*proof*⟩
lemma *AX-mono*: $p \subseteq q \Longrightarrow$ AX $p \subseteq$ AX q ⟨*proof*⟩
lemma *AG-mono*: $p \subseteq q \Longrightarrow$ AG $p \subseteq$ AG q
⟨*proof*⟩

The formula AG p implies AX p (we use substitution of $\subseteq$ with monotonicity).

lemma *AG-AX*: AG $p \subseteq$ AX p
⟨*proof*⟩

Furthermore we show idempotency of the AG operator. The proof is a good example of how accumulated facts may get used to feed a single rule step.

lemma *AG-AG*: AG AG $p =$ AG p
⟨*proof*⟩

We now give an alternative characterization of the AG operator, which describes the AG operator in an "operational" way by tree induction: In a state holds AG p iff in that state holds p, and in all reachable states s follows from the fact that p holds in s, that p also holds in all successor states of s. We use the co-induction principle *AG-I* to establish this in a purely algebraic manner.

theorem *AG-induct*: $p \cap$ AG $(p \rightarrow$ AX $p) =$ AG p
⟨*proof*⟩

22.3 An application of tree induction

Further interesting properties of CTL expressions may be demonstrated with the help of tree induction; here we show that AX and AG commute.

theorem *AG-AX-commute*: AG AX $p =$ AX AG p
⟨*proof*⟩

end

23 Arithmetic

theory *Arith-Examples* **imports** *Main* **begin**

The *arith* method is used frequently throughout the Isabelle distribution. This file merely contains some additional tests and special corner cases. Some rather technical remarks:

`fast_arith_tac` is a very basic version of the tactic. It performs no meta-to-object-logic conversion, and only some splitting of operators. `linear_arith_tac` performs meta-to-object-logic conversion, full splitting of operators, and NNF normalization of the goal. The *arith* method combines them both, and tries other methods (e.g. *presburger*) as well. This is the one that you should use in your proofs!

An *arith*-based simproc is available as well (see `Lin_Arith.lin_arith_simproc`), which—for performance reasons—however does even less splitting than `fast_arith_tac`

at the moment (namely inequalities only). (On the other hand, it does take apart conjunctions, which `fast_arith_tac` currently does not do.)

23.1 Splitting of Operators: *max*, *min*, *abs*, *op −*, *nat*, *op mod*, *op div*

lemma $(i::nat) <= max\ i\ j$
⟨*proof*⟩

lemma $(i::int) <= max\ i\ j$
⟨*proof*⟩

lemma $min\ i\ j <= (i::nat)$
⟨*proof*⟩

lemma $min\ i\ j <= (i::int)$
⟨*proof*⟩

lemma $min\ (i::nat)\ j <= max\ i\ j$
⟨*proof*⟩

lemma $min\ (i::int)\ j <= max\ i\ j$
⟨*proof*⟩

lemma $min\ (i::nat)\ j + max\ i\ j = i + j$
⟨*proof*⟩

lemma $min\ (i::int)\ j + max\ i\ j = i + j$
⟨*proof*⟩

lemma $(i::nat) < j ==> min\ i\ j < max\ i\ j$
⟨*proof*⟩

lemma $(i::int) < j ==> min\ i\ j < max\ i\ j$
⟨*proof*⟩

lemma $(0::int) <= abs\ i$
⟨*proof*⟩

lemma $(i::int) <= abs\ i$
⟨*proof*⟩

lemma $abs\ (abs\ (i::int)) = abs\ i$
⟨*proof*⟩

Also testing subgoals with bound variables.

lemma $!!x.\ (x::nat) <= y ==> x - y = 0$
⟨*proof*⟩

lemma !!x. $(x::nat) - y = 0 ==> x <= y$
⟨*proof*⟩

lemma !!x. $((x::nat) <= y) = (x - y = 0)$
⟨*proof*⟩

lemma [| $(x::nat) < y$; $d < 1$ |] ==> $x - y = d$
⟨*proof*⟩

lemma [| $(x::nat) < y$; $d < 1$ |] ==> $x - y - x = d - x$
⟨*proof*⟩

lemma $(x::int) < y ==> x - y < 0$
⟨*proof*⟩

lemma $nat\ (i + j) <= nat\ i + nat\ j$
⟨*proof*⟩

lemma $i < j ==> nat\ (i - j) = 0$
⟨*proof*⟩

lemma $(i::nat)\ mod\ 0 = i$

⟨*proof*⟩

lemma $(i::nat)\ mod\ 1 = 0$

⟨*proof*⟩

lemma $(i::nat)\ mod\ 42 <= 41$
⟨*proof*⟩

lemma $(i::int)\ mod\ 0 = i$

⟨*proof*⟩

lemma $(i::int)\ mod\ 1 = 0$

⟨*proof*⟩

lemma $(i::int)\ mod\ 42 <= 41$

⟨*proof*⟩

lemma $-(i::int) * 1 = 0 ==> i = 0$
⟨*proof*⟩

lemma [| $(0::int) < abs\ i$; $abs\ i * 1 < abs\ i * j$ |] ==> $1 < abs\ i * j$
⟨*proof*⟩

23.2 Meta-Logic

lemma $x < Suc\ y == x <= y$
⟨*proof*⟩

lemma $((x::nat) == z ==> x \sim= y) ==> x \sim= y \mid z \sim= y$
⟨*proof*⟩

23.3 Various Other Examples

lemma $(x < Suc\ y) = (x <= y)$
⟨*proof*⟩

lemma $[|\ (x::nat) < y;\ y < z\ |] ==> x < z$
⟨*proof*⟩

lemma $(x::nat) < y\ \&\ y < z ==> x < z$
⟨*proof*⟩

This example involves no arithmetic at all, but is solved by preprocessing (i.e. NNF normalization) alone.

lemma $(P::bool) = Q ==> Q = P$
⟨*proof*⟩

lemma $[|\ P = (x = 0);\ (\sim P) = (y = 0)\ |] ==> min\ (x::nat)\ y = 0$
⟨*proof*⟩

lemma $[|\ P = (x = 0);\ (\sim P) = (y = 0)\ |] ==> max\ (x::nat)\ y = x + y$
⟨*proof*⟩

lemma $[|\ (x::nat) \sim= y;\ a + 2 = b;\ a < y;\ y < b;\ a < x;\ x < b\ |] ==> False$
⟨*proof*⟩

lemma $[|\ (x::nat) > y;\ y > z;\ z > x\ |] ==> False$
⟨*proof*⟩

lemma $(x::nat) - 5 > y ==> y < x$
⟨*proof*⟩

lemma $(x::nat) \sim= 0 ==> 0 < x$
⟨*proof*⟩

lemma $[|\ (x::nat) \sim= y;\ x <= y\ |] ==> x < y$
⟨*proof*⟩

lemma $[|\ (x::nat) < y;\ P\ (x - y)\ |] ==> P\ 0$
⟨*proof*⟩

lemma $(x - y) - (x::nat) = (x - x) - y$
⟨*proof*⟩

lemma [| (a::*nat*) < b; c < d |] ==> (a − b) = (c − d)
⟨*proof*⟩

lemma ((a::*nat*) − (b − (c − (d − e)))) = (a − (b − (c − (d − e))))
⟨*proof*⟩

lemma (n < m & m < n′) | (n < m & m = n′) | (n < n′ & n′ < m) |
(n = n′ & n′ < m) | (n = m & m < n′) |
(n′ < m & m < n) | (n′ < m & m = n) |
(n′ < n & n < m) | (n′ = n & n < m) | (n′ = m & m < n) |
(m < n & n < n′) | (m < n & n′ = n) | (m < n′ & n′ < n) |
(m = n & n < n′) | (m = n′ & n′ < n) |
(n′ = m & m = (n::*nat*))

⟨*proof*⟩

lemma *2* ∗ (x::*nat*) ~= *1*

⟨*proof*⟩

Constants.

lemma (*0*::*nat*) < *1*
⟨*proof*⟩

lemma (*0*::*int*) < *1*
⟨*proof*⟩

lemma (*47*::*nat*) + *11* < *08* ∗ *15*
⟨*proof*⟩

lemma (*47*::*int*) + *11* < *08* ∗ *15*
⟨*proof*⟩

Splitting of inequalities of different type.

lemma [| (a::*nat*) ~= b; (i::*int*) ~= j; a < *2*; b < *2* |] ==>
a + b <= *nat* (*max* (*abs* i) (*abs* j))
⟨*proof*⟩

Again, but different order.

lemma [| (i::*int*) ~= j; (a::*nat*) ~= b; a < *2*; b < *2* |] ==>
a + b <= *nat* (*max* (*abs* i) (*abs* j))
⟨*proof*⟩

end

24 Binary trees

theory *BT* **imports** *Main* **begin**

datatype *'a bt* =
  Lf
| *Br 'a 'a bt 'a bt*

consts
  n-nodes :: *'a bt => nat*
  n-leaves :: *'a bt => nat*
  depth :: *'a bt => nat*
  reflect :: *'a bt => 'a bt*
  bt-map :: *('a => 'b) => ('a bt => 'b bt)*
  preorder :: *'a bt => 'a list*
  inorder :: *'a bt => 'a list*
  postorder :: *'a bt => 'a list*
  append :: *'a bt => 'a bt => 'a bt*

primrec
  n-nodes Lf = 0
  n-nodes (Br a t1 t2) = Suc (n-nodes t1 + n-nodes t2)

primrec
  n-leaves Lf = Suc 0
  n-leaves (Br a t1 t2) = n-leaves t1 + n-leaves t2

primrec
  depth Lf = 0
  depth (Br a t1 t2) = Suc (max (depth t1) (depth t2))

primrec
  reflect Lf = Lf
  reflect (Br a t1 t2) = Br a (reflect t2) (reflect t1)

primrec
  bt-map f Lf = Lf
  bt-map f (Br a t1 t2) = Br (f a) (bt-map f t1) (bt-map f t2)

primrec
  preorder Lf = []
  preorder (Br a t1 t2) = [a] @ (preorder t1) @ (preorder t2)

primrec
inorder Lf = []
inorder (*Br a t1 t2*) = (*inorder t1*) @ [*a*] @ (*inorder t2*)

primrec
postorder Lf = []
postorder (*Br a t1 t2*) = (*postorder t1*) @ (*postorder t2*) @ [*a*]

primrec
append Lf t = *t*
append (*Br a t1 t2*) *t* = *Br a* (*append t1 t*) (*append t2 t*)

BT simplification

lemma *n-leaves-reflect*: *n-leaves* (*reflect t*) = *n-leaves t*
⟨*proof*⟩

lemma *n-nodes-reflect*: *n-nodes* (*reflect t*) = *n-nodes t*
⟨*proof*⟩

lemma *depth-reflect*: *depth* (*reflect t*) = *depth t*
⟨*proof*⟩

The famous relationship between the numbers of leaves and nodes.

lemma *n-leaves-nodes*: *n-leaves t* = *Suc* (*n-nodes t*)
⟨*proof*⟩

lemma *reflect-reflect-ident*: *reflect* (*reflect t*) = *t*
⟨*proof*⟩

lemma *bt-map-reflect*: *bt-map f* (*reflect t*) = *reflect* (*bt-map f t*)
⟨*proof*⟩

lemma *preorder-bt-map*: *preorder* (*bt-map f t*) = *map f* (*preorder t*)
⟨*proof*⟩

lemma *inorder-bt-map*: *inorder* (*bt-map f t*) = *map f* (*inorder t*)
⟨*proof*⟩

lemma *postorder-bt-map*: *postorder* (*bt-map f t*) = *map f* (*postorder t*)
⟨*proof*⟩

lemma *depth-bt-map* [*simp*]: *depth* (*bt-map f t*) = *depth t*
⟨*proof*⟩

lemma *n-leaves-bt-map* [*simp*]: *n-leaves* (*bt-map f t*) = *n-leaves t*
⟨*proof*⟩

lemma *preorder-reflect*: *preorder* (*reflect t*) = *rev* (*postorder t*)
⟨*proof*⟩

lemma *inorder-reflect*: *inorder* (*reflect t*) = *rev* (*inorder t*)
⟨*proof*⟩

lemma *postorder-reflect*: *postorder* (*reflect t*) = *rev* (*preorder t*)
⟨*proof*⟩

Analogues of the standard properties of the append function for lists.

lemma *append-assoc* [*simp*]:
append (*append t1 t2*) *t3* = *append t1* (*append t2 t3*)
⟨*proof*⟩

lemma *append-Lf2* [*simp*]: *append t Lf* = *t*
⟨*proof*⟩

lemma *depth-append* [*simp*]: *depth* (*append t1 t2*) = *depth t1* + *depth t2*
⟨*proof*⟩

lemma *n-leaves-append* [*simp*]:
n-leaves (*append t1 t2*) = *n-leaves t1* ∗ *n-leaves t2*
⟨*proof*⟩

lemma *bt-map-append*:
bt-map f (*append t1 t2*) = *append* (*bt-map f t1*) (*bt-map f t2*)
⟨*proof*⟩

end

25 Sorting: Basic Theory

theory *Sorting*
imports *Main Multiset*
begin

consts
sorted1 :: ('*a* ⇒ '*a* ⇒ *bool*) ⇒ '*a list* ⇒ *bool*
sorted :: ('*a* ⇒ '*a* ⇒ *bool*) ⇒ '*a list* ⇒ *bool*

primrec
sorted1 le [] = *True*
sorted1 le (*x*#*xs*) = ((*case xs of* [] => *True* | *y*#*ys* => *le x y*) &
sorted1 le xs)

primrec
sorted le [] = *True*
sorted le (*x*#*xs*) = ((∀ *y* ∈ *set xs*. *le x y*) & *sorted le xs*)

definition
total :: ('*a* ⇒ '*a* ⇒ *bool*) => *bool* **where**
total r = (∀ *x y*. *r x y* | *r y x*)

definition
transf :: ('*a* ⇒ '*a* ⇒ *bool*) => *bool* **where**
transf f = (∀ *x y z*. *f x y* & *f y z* −−> *f x z*)

lemma *sorted1-is-sorted*: *transf*(*le*) ==> *sorted1 le xs* = *sorted le xs*
⟨*proof*⟩

lemma *sorted-append* [*simp*]:
sorted le (*xs*@*ys*) =
(*sorted le xs* & *sorted le ys* & (∀ *x* ∈ *set xs*. ∀ *y* ∈ *set ys*. *le x y*))
⟨*proof*⟩

end

26 Merge Sort

theory *MergeSort*
imports *Sorting*
begin

context *linorder*
begin

fun *merge* :: '*a list* ⇒ '*a list* ⇒ '*a list*
where
merge (*x*#*xs*) (*y*#*ys*) =
(*if x* ≤ *y then x* # *merge xs* (*y*#*ys*) *else y* # *merge* (*x*#*xs*) *ys*)
| *merge xs* [] = *xs*
| *merge* [] *ys* = *ys*

lemma *multiset-of-merge*[*simp*]:
multiset-of (*merge xs ys*) = *multiset-of xs* + *multiset-of ys*
⟨*proof*⟩

lemma *set-merge*[*simp*]: *set* (*merge xs ys*) = *set xs* ∪ *set ys*
⟨*proof*⟩

lemma *sorted-merge*[*simp*]:
sorted (*op* ≤) (*merge xs ys*) = (*sorted* (*op* ≤) *xs* & *sorted* (*op* ≤) *ys*)
⟨*proof*⟩

fun *msort* :: *'a list* ⇒ *'a list*
where
  msort [] = []
| *msort* [*x*] = [*x*]
| *msort xs* = *merge* (*msort* (*take* (*size xs div 2*) *xs*))
                       (*msort* (*drop* (*size xs div 2*) *xs*))

theorem *sorted-msort*: *sorted* (*op* ≤) (*msort xs*)
⟨*proof*⟩

theorem *multiset-of-msort*: *multiset-of* (*msort xs*) = *multiset-of xs*
⟨*proof*⟩

end

end

27 A lemma for Lagrange's theorem

theory *Lagrange* **imports** *Main* **begin**

This theory only contains a single theorem, which is a lemma in Lagrange's proof that every natural number is the sum of 4 squares. Its sole purpose is to demonstrate ordered rewriting for commutative rings.

The enterprising reader might consider proving all of Lagrange's theorem.

definition *sq* :: *'a*::*times* => *'a* **where** *sq x* == *x*∗*x*

The following lemma essentially shows that every natural number is the sum of four squares, provided all prime numbers are. However, this is an abstract theorem about commutative rings. It has, a priori, nothing to do with nat.

⟨*ML*⟩

lemma *Lagrange-lemma*: **fixes** *x1* :: *'a*::*comm-ring* **shows**
  (*sq x1* + *sq x2* + *sq x3* + *sq x4*) ∗ (*sq y1* + *sq y2* + *sq y3* + *sq y4*) =
   sq (*x1*∗*y1* − *x2*∗*y2* − *x3*∗*y3* − *x4*∗*y4*) +
   sq (*x1*∗*y2* + *x2*∗*y1* + *x3*∗*y4* − *x4*∗*y3*) +
   sq (*x1*∗*y3* − *x2*∗*y4* + *x3*∗*y1* + *x4*∗*y2*) +
   sq (*x1*∗*y4* + *x2*∗*y3* − *x3*∗*y2* + *x4*∗*y1*)
⟨*proof*⟩

A challenge by John Harrison. Takes about 12s on a 1.6GHz machine.

lemma fixes *p1* :: *'a*::*comm-ring* **shows**
  (*sq p1* + *sq q1* + *sq r1* + *sq s1* + *sq t1* + *sq u1* + *sq v1* + *sq w1*) ∗
   (*sq p2* + *sq q2* + *sq r2* + *sq s2* + *sq t2* + *sq u2* + *sq v2* + *sq w2*)

$= sq\ (p1*p2 - q1*q2 - r1*r2 - s1*s2 - t1*t2 - u1*u2 - v1*v2 - w1*w2)$
+
$sq\ (p1*q2 + q1*p2 + r1*s2 - s1*r2 + t1*u2 - u1*t2 - v1*w2 + w1*v2)$
+
$sq\ (p1*r2 - q1*s2 + r1*p2 + s1*q2 + t1*v2 + u1*w2 - v1*t2 - w1*u2)$
+
$sq\ (p1*s2 + q1*r2 - r1*q2 + s1*p2 + t1*w2 - u1*v2 + v1*u2 - w1*t2)$
+
$sq\ (p1*t2 - q1*u2 - r1*v2 - s1*w2 + t1*p2 + u1*q2 + v1*r2 + w1*s2)$
+
$sq\ (p1*u2 + q1*t2 - r1*w2 + s1*v2 - t1*q2 + u1*p2 - v1*s2 + w1*r2)$
+
$sq\ (p1*v2 + q1*w2 + r1*t2 - s1*u2 - t1*r2 + u1*s2 + v1*p2 - w1*q2)$
+
$sq\ (p1*w2 - q1*v2 + r1*u2 + s1*t2 - t1*s2 - u1*r2 + v1*q2 + w1*p2)$
⟨*proof*⟩

end

28 Groebner Basis Examples

theory *Groebner-Examples*
imports *Groebner-Basis*
begin

28.1 Basic examples

lemma $3\ \hat{}\ 3 == (?X::'a::\{number\text{-}ring,recpower\})$
⟨*proof*⟩

lemma $(x - (-2))\hat{}5 == ?X::int$
⟨*proof*⟩

lemma $(x - (-2))\hat{}5 * (y - 78)\ \hat{}\ 8 == ?X::int$
⟨*proof*⟩

lemma $((-3)\ \hat{}\ (Suc\ (Suc\ (Suc\ 0)))) == (X::'a::\{number\text{-}ring,recpower\})$
⟨*proof*⟩

lemma $((x::int) + y)\hat{}3 - 1 = (x - z)\hat{}2 - 10 \Longrightarrow x = z + 3 \Longrightarrow x = -y$
⟨*proof*⟩

lemma $(4::nat) + 4 = 3 + 5$
⟨*proof*⟩

lemma $(4::int) + 0 = 4$
⟨*proof*⟩

lemma
assumes $a * x\hat{}2 + b * x + c = (0::int)$ **and** $d * x\hat{}2 + e * x + f = 0$
shows $d\hat{}2*c\hat{}2 - 2*d*c*a*f + a\hat{}2*f\hat{}2 - e*d*b*c - e*b*a*f + a*e\hat{}2*c + f*d*b\hat{}2 = 0$
⟨*proof*⟩

lemma $(x::int)\hat{}3 - x\hat{}2 - 5*x - 3 = 0 \longleftrightarrow (x = 3 \vee x = -1)$
⟨*proof*⟩

theorem $x*(x^2 - x - 5) - 3 = (0::int) \longleftrightarrow (x = 3 \vee x = -1)$
⟨*proof*⟩

lemma
fixes $x::'a::\{idom,recpower,number\text{-}ring\}$
shows $x\hat{}2*y = x\hat{}2 \ \&\ x*y\hat{}2 = y\hat{}2 \longleftrightarrow x=1 \ \&\ y=1 \mid x=0 \ \&\ y=0$
⟨*proof*⟩

28.2 Lemmas for Lagrange's theorem

definition
$sq :: 'a::times => 'a$ **where**
$sq\ x == x*x$

lemma
fixes $x1 :: 'a::\{idom,recpower,number\text{-}ring\}$
shows
$(sq\ x1 + sq\ x2 + sq\ x3 + sq\ x4) * (sq\ y1 + sq\ y2 + sq\ y3 + sq\ y4) =$
$sq\ (x1*y1 - x2*y2 - x3*y3 - x4*y4) +$
$sq\ (x1*y2 + x2*y1 + x3*y4 - x4*y3) +$
$sq\ (x1*y3 - x2*y4 + x3*y1 + x4*y2) +$
$sq\ (x1*y4 + x2*y3 - x3*y2 + x4*y1)$
⟨*proof*⟩

lemma
fixes $p1 :: 'a::\{idom,recpower,number\text{-}ring\}$
shows
$(sq\ p1 + sq\ q1 + sq\ r1 + sq\ s1 + sq\ t1 + sq\ u1 + sq\ v1 + sq\ w1) *$
$(sq\ p2 + sq\ q2 + sq\ r2 + sq\ s2 + sq\ t2 + sq\ u2 + sq\ v2 + sq\ w2)$
$= sq\ (p1*p2 - q1*q2 - r1*r2 - s1*s2 - t1*t2 - u1*u2 - v1*v2 - w1*w2)$
$+$
$sq\ (p1*q2 + q1*p2 + r1*s2 - s1*r2 + t1*u2 - u1*t2 - v1*w2 + w1*v2)$
$+$
$sq\ (p1*r2 - q1*s2 + r1*p2 + s1*q2 + t1*v2 + u1*w2 - v1*t2 - w1*u2)$
$+$
$sq\ (p1*s2 + q1*r2 - r1*q2 + s1*p2 + t1*w2 - u1*v2 + v1*u2 - w1*t2)$
$+$
$sq\ (p1*t2 - q1*u2 - r1*v2 - s1*w2 + t1*p2 + u1*q2 + v1*r2 + w1*s2)$
$+$
$sq\ (p1*u2 + q1*t2 - r1*w2 + s1*v2 - t1*q2 + u1*p2 - v1*s2 + w1*r2)$

+
    $sq\ (p1*v2 + q1*w2 + r1*t2 - s1*u2 - t1*r2 + u1*s2 + v1*p2 - w1*q2)$
+
    $sq\ (p1*w2 - q1*v2 + r1*u2 + s1*t2 - t1*s2 - u1*r2 + v1*q2 + w1*p2)$
  ⟨*proof*⟩

28.3 Colinearity is invariant by rotation

types *point* = *int* × *int*

definition *collinear* ::*point* ⇒ *point* ⇒ *point* ⇒ *bool* **where**
  collinear ≡ λ(*Ax*,*Ay*) (*Bx*,*By*) (*Cx*,*Cy*).
    ((*Ax* − *Bx*) ∗ (*By* − *Cy*) = (*Ay* − *By*) ∗ (*Bx* − *Cx*))

lemma *collinear-inv-rotation*:
  assumes *collinear* (*Ax*, *Ay*) (*Bx*, *By*) (*Cx*, *Cy*) **and** $c^2 + s^2 = 1$
  shows *collinear* (*Ax* ∗ *c* − *Ay* ∗ *s*, *Ay* ∗ *c* + *Ax* ∗ *s*)
    (*Bx* ∗ *c* − *By* ∗ *s*, *By* ∗ *c* + *Bx* ∗ *s*) (*Cx* ∗ *c* − *Cy* ∗ *s*, *Cy* ∗ *c* + *Cx* ∗ *s*)
  ⟨*proof*⟩

lemma *EX* (*d*::*int*). *a*∗*y* − *a*∗*x* = *n*∗*d* ⟹ *EX u v*. *a*∗*u* + *n*∗*v* = *1* ⟹ *EX e*. *y* − *x* = *n*∗*e*
  ⟨*proof*⟩

end

29 Milner-Tofte: Co-induction in Relational Semantics

theory *MT*
imports *Main*
begin

typedecl *Const*

typedecl *ExVar*
typedecl *Ex*

typedecl *TyConst*
typedecl *Ty*

typedecl *Clos*
typedecl *Val*

typedecl *ValEnv*
typedecl *TyEnv*

consts

c-app :: $[Const,\ Const] \Rightarrow Const$

e-const :: $Const \Rightarrow Ex$
e-var :: $ExVar \Rightarrow Ex$
e-fn :: $[ExVar,\ Ex] \Rightarrow Ex$ (*fn - => - [0,51] 1000*)
e-fix :: $[ExVar,\ ExVar,\ Ex] \Rightarrow Ex$ (*fix - (-) = - [0,51,51] 1000*)
e-app :: $[Ex,\ Ex] \Rightarrow Ex$ (*- @@ - [51,51] 1000*)
e-const-fst :: $Ex \Rightarrow Const$

t-const :: $TyConst \Rightarrow Ty$
t-fun :: $[Ty,\ Ty] \Rightarrow Ty$ (*- -> - [51,51] 1000*)

v-const :: $Const \Rightarrow Val$
v-clos :: $Clos \Rightarrow Val$

ve-emp :: $ValEnv$
ve-owr :: $[ValEnv,\ ExVar,\ Val] \Rightarrow ValEnv$ (*- + { - |-> - } [36,0,0] 50*)
ve-dom :: $ValEnv \Rightarrow ExVar\ set$
ve-app :: $[ValEnv,\ ExVar] \Rightarrow Val$

clos-mk :: $[ExVar,\ Ex,\ ValEnv] \Rightarrow Clos$ (*<| - , - , - |> [0,0,0] 1000*)

te-emp :: $TyEnv$
te-owr :: $[TyEnv,\ ExVar,\ Ty] \Rightarrow TyEnv$ (*- + { - |=> - } [36,0,0] 50*)
te-app :: $[TyEnv,\ ExVar] \Rightarrow Ty$
te-dom :: $TyEnv \Rightarrow ExVar\ set$

eval-fun :: $((ValEnv * Ex) * Val)\ set \Rightarrow ((ValEnv * Ex) * Val)\ set$
eval-rel :: $((ValEnv * Ex) * Val)\ set$
eval :: $[ValEnv,\ Ex,\ Val] \Rightarrow bool$ (*- |- - ---> - [36,0,36] 50*)

elab-fun :: $((TyEnv * Ex) * Ty)\ set \Rightarrow ((TyEnv * Ex) * Ty)\ set$
elab-rel :: $((TyEnv * Ex) * Ty)\ set$
elab :: $[TyEnv,\ Ex,\ Ty] \Rightarrow bool$ (*- |- - ===> - [36,0,36] 50*)

isof :: $[Const,\ Ty] \Rightarrow bool$ (*- isof - [36,36] 50*)
isof-env :: $[ValEnv,TyEnv] \Rightarrow bool$ (*- isofenv -*)

hasty-fun :: $(Val * Ty)\ set \Rightarrow (Val * Ty)\ set$
hasty-rel :: $(Val * Ty)\ set$
hasty :: $[Val,\ Ty] \Rightarrow bool$ (*- hasty - [36,36] 50*)
hasty-env :: $[ValEnv,TyEnv] \Rightarrow bool$ (*- hastyenv - [36,36] 35*)

axioms

e-const-inj: *e-const*(*c1*) = *e-const*(*c2*) ==> *c1* = *c2*
e-var-inj: *e-var*(*ev1*) = *e-var*(*ev2*) ==> *ev1* = *ev2*
e-fn-inj: *fn ev1 => e1* = *fn ev2 => e2* ==> *ev1* = *ev2* & *e1* = *e2*
e-fix-inj:
  fix ev11e(*v12*) = *e1* = *fix ev21*(*ev22*) = *e2* ==>
  ev11 = *ev21* & *ev12* = *ev22* & *e1* = *e2*

e-app-inj: *e11* @@ *e12* = *e21* @@ *e22* ==> *e11* = *e21* & *e12* = *e22*

e-disj-const-var: ~*e-const*(*c*) = *e-var*(*ev*)
e-disj-const-fn: ~*e-const*(*c*) = *fn ev => e*
e-disj-const-fix: ~*e-const*(*c*) = *fix ev1*(*ev2*) = *e*
e-disj-const-app: ~*e-const*(*c*) = *e1* @@ *e2*
e-disj-var-fn: ~*e-var*(*ev1*) = *fn ev2 => e*
e-disj-var-fix: ~*e-var*(*ev*) = *fix ev1*(*ev2*) = *e*
e-disj-var-app: ~*e-var*(*ev*) = *e1* @@ *e2*
e-disj-fn-fix: ~*fn ev1 => e1* = *fix ev21*(*ev22*) = *e2*
e-disj-fn-app: ~*fn ev1 => e1* = *e21* @@ *e22*
e-disj-fix-app: ~*fix ev11*(*ev12*) = *e1* = *e21* @@ *e22*

e-ind:
  [| !!*ev*. *P*(*e-var*(*ev*));
     !!*c*. *P*(*e-const*(*c*));
     !!*ev e*. *P*(*e*) ==> *P*(*fn ev => e*);
     !!*ev1 ev2 e*. *P*(*e*) ==> *P*(*fix ev1*(*ev2*) = *e*);
     !!*e1 e2*. *P*(*e1*) ==> *P*(*e2*) ==> *P*(*e1* @@ *e2*)
  |] ==>
 P(*e*)

t-const-inj: *t-const*(*c1*) = *t-const*(*c2*) ==> *c1* = *c2*
t-fun-inj: *t11* -> *t12* = *t21* -> *t22* ==> *t11* = *t21* & *t12* = *t22*

t-ind:
  [| !!*p*. *P*(*t-const p*); !!*t1 t2*. *P*(*t1*) ==> *P*(*t2*) ==> *P*(*t-fun t1 t2*) |]
  ==> *P*(*t*)

v-const-inj: *v-const*(*c1*) = *v-const*(*c2*) ==> *c1* = *c2*
v-clos-inj:
  v-clos(<|*ev1*,*e1*,*ve1*|>) = *v-clos*(<|*ev2*,*e2*,*ve2*|>) ==>
  ev1 = *ev2* & *e1* = *e2* & *ve1* = *ve2*

v-disj-const-clos: ~*v-const*(*c*) = *v-clos*(*cl*)

ve-dom-owr: *ve-dom*(*ve* + {*ev* |-> *v*}) = *ve-dom*(*ve*) *Un* {*ev*}

ve-app-owr1: *ve-app* (*ve* + {*ev* |-> *v*}) *ev*=*v*
ve-app-owr2: ~*ev1*=*ev2* ==> *ve-app* (*ve*+{*ev1* |-> *v*}) *ev2*=*ve-app ve ev2*

te-dom-owr: *te-dom*(*te* + {*ev* |=> *t*}) = *te-dom*(*te*) *Un* {*ev*}

te-app-owr1: *te-app* (*te* + {*ev* |=> *t*}) *ev*=*t*
te-app-owr2: ~*ev1*=*ev2* ==> *te-app* (*te*+{*ev1* |=> *t*}) *ev2*=*te-app te ev2*

defs
eval-fun-def:
  eval-fun(*s*) ==
  { *pp*.
    (*? ve c*. *pp*=((*ve*,*e-const*(*c*)),*v-const*(*c*))) |
    (*? ve x*. *pp*=((*ve*,*e-var*(*x*)),*ve-app ve x*) & *x*:*ve-dom*(*ve*)) |
    (*? ve e x*. *pp*=((*ve*,*fn x* => *e*),*v-clos*(<|*x*,*e*,*ve*|>)))|
    (*? ve e x f cl*.
        pp=((*ve*,*fix f*(*x*) = *e*),*v-clos*(*cl*)) &
        cl=<|*x*, *e*, *ve*+{*f* |-> *v-clos*(*cl*)} |>
) |
    (*? ve e1 e2 c1 c2*.
        pp=((*ve*,*e1* @@ *e2*),*v-const*(*c-app c1 c2*)) &

```
        ((ve,e1),v-const(c1)):s & ((ve,e2),v-const(c2)):s
    ) |
    ( ? ve vem e1 e2 em xm v v2.
        pp=((ve,e1 @@ e2),v) &
        ((ve,e1),v-clos(<|xm,em,vem|>)):s &
        ((ve,e2),v2):s &
        ((vem+{xm |-> v2},em),v):s
    )
  }

eval-rel-def: eval-rel == lfp(eval-fun)
eval-def: ve |- e ---> v == ((ve,e),v):eval-rel



elab-fun-def:
elab-fun(s) ==
{ pp.
  (? te c t. pp=((te,e-const(c)),t) & c isof t) |
  (? te x. pp=((te,e-var(x)),te-app te x) & x:te-dom(te)) |
  (? te x e t1 t2. pp=((te,fn x => e),t1->t2) & ((te+{x |=> t1},e),t2):s) |
  (? te f x e t1 t2.
     pp=((te,fix f(x)=e),t1->t2) & ((te+{f |=> t1->t2}+{x |=> t1},e),t2):s
  ) |
  (? te e1 e2 t1 t2.
     pp=((te,e1 @@ e2),t2) & ((te,e1),t1->t2):s & ((te,e2),t1):s
  )
}

elab-rel-def: elab-rel == lfp(elab-fun)
elab-def: te |- e ===> t == ((te,e),t):elab-rel



isof-env-def:
   ve isofenv te ==
   ve-dom(ve) = te-dom(te) &
   ( ! x.
       x:ve-dom(ve) -->
       (? c. ve-app ve x = v-const(c) & c isof te-app te x)
   )
```

axioms

```
  isof-app: [| c1 isof t1->t2; c2 isof t1 |] ==> c-app c1 c2 isof t2
```

defs

hasty-fun-def:
  hasty-fun(*r*) ==
  { *p*.
    (*?* *c* *t*. *p* = (*v-const*(*c*),*t*) & *c* *isof* *t*) |
    (*?* *ev* *e* *ve* *t* *te*.
        p = (*v-clos*(<|*ev*,*e*,*ve*|>),*t*) &
        te |− *fn* *ev* => *e* ===> *t* &
        ve-dom(*ve*) = *te-dom*(*te*) &
        (! *ev1*. *ev1*:*ve-dom*(*ve*) −−> (*ve-app* *ve* *ev1*,*te-app* *te* *ev1*) : *r*)
)
  }

hasty-rel-def: *hasty-rel* == *gfp*(*hasty-fun*)
hasty-def: *v* *hasty* *t* == (*v*,*t*) : *hasty-rel*
hasty-env-def:
  ve *hastyenv* *te* ==
  ve-dom(*ve*) = *te-dom*(*te*) &
  (! *x*. *x*: *ve-dom*(*ve*) −−> *ve-app* *ve* *x* *hasty* *te-app* *te* *x*)

⟨*ML*⟩

lemma *infsys-p1*: *P* *a* *b* ==> *P* (*fst* (*a*,*b*)) (*snd* (*a*,*b*))
  ⟨*proof*⟩

lemma *infsys-p2*: *P* (*fst* (*a*,*b*)) (*snd* (*a*,*b*)) ==> *P* *a* *b*
  ⟨*proof*⟩

lemma *infsys-pp1*: *P* *a* *b* *c* ==> *P* (*fst*(*fst*((*a*,*b*),*c*))) (*snd*(*fst* ((*a*,*b*),*c*))) (*snd* ((*a*,*b*),*c*))
  ⟨*proof*⟩

lemma *infsys-pp2*: *P* (*fst*(*fst*((*a*,*b*),*c*))) (*snd*(*fst*((*a*,*b*),*c*))) (*snd*((*a*,*b*),*c*)) ==> *P* *a* *b* *c*
  ⟨*proof*⟩

lemma *lfp-intro2*: [| *mono*(*f*); *x*:*f*(*lfp*(*f*)) |] ==> *x*:*lfp*(*f*)

⟨*proof*⟩

lemma *lfp-elim2*:
  assumes *lfp*: x:*lfp*(f)
    and *mono*: *mono*(f)
    and r: !!y. y:f(*lfp*(f)) ==> P(y)
  shows P(x)
⟨*proof*⟩

lemma *lfp-ind2*:
  assumes *lfp*: x:*lfp*(f)
    and *mono*: *mono*(f)
    and r: !!y. y:f(*lfp*(f) *Int* {x. P(x)}) ==> P(y)
  shows P(x)
⟨*proof*⟩

lemma *gfp-coind2*:
  assumes *cih*: x:f({x} *Un* *gfp*(f))
    and *monoh*: *mono*(f)
  shows x:*gfp*(f)
⟨*proof*⟩

lemma *gfp-elim2*:
  assumes *gfph*: x:*gfp*(f)
    and *monoh*: *mono*(f)
    and *caseh*: !!y. y:f(*gfp*(f)) ==> P(y)
  shows P(x)
⟨*proof*⟩

lemmas *e-injs* = *e-const-inj e-var-inj e-fn-inj e-fix-inj e-app-inj*

lemmas *e-disjs* =
  e-disj-const-var
  e-disj-const-fn
  e-disj-const-fix
  e-disj-const-app
  e-disj-var-fn
  e-disj-var-fix
  e-disj-var-app
  e-disj-fn-fix
  e-disj-fn-app

e-disj-fix-app

lemmas *e-disj-si* = *e-disjs* *e-disjs* [*symmetric*]

lemmas *e-disj-se* = *e-disj-si* [*THEN notE*]

lemmas *v-disjs* = *v-disj-const-clos*
lemmas *v-disj-si* = *v-disjs* *v-disjs* [*symmetric*]
lemmas *v-disj-se* = *v-disj-si* [*THEN notE*]

lemmas *v-injs* = *v-const-inj v-clos-inj*

lemma *eval-fun-mono*: *mono*(*eval-fun*)
⟨*proof*⟩

lemma *eval-const*: *ve* |− *e-const*(*c*) −−−> *v-const*(*c*)
⟨*proof*⟩

lemma *eval-var2*:
ev:*ve-dom*(*ve*) ==> *ve* |− *e-var*(*ev*) −−−> *ve-app ve ev*
⟨*proof*⟩

lemma *eval-fn*:
ve |− *fn ev* => *e* −−−> *v-clos*(<|*ev*,*e*,*ve*|>)
⟨*proof*⟩

lemma *eval-fix*:
cl = <| *ev1*, *e*, *ve* + {*ev2* |−> *v-clos*(*cl*)} |> ==>
ve |− *fix ev2*(*ev1*) = *e* −−−> *v-clos*(*cl*)
⟨*proof*⟩

lemma *eval-app1*:
[| *ve* |− *e1* −−−> *v-const*(*c1*); *ve* |− *e2* −−−> *v-const*(*c2*) |] ==>
ve |− *e1* @@ *e2* −−−> *v-const*(*c-app c1 c2*)
⟨*proof*⟩

lemma *eval-app2*:

```
  [|  ve |- e1 ---> v-clos(<|xm,em,vem|>);
      ve |- e2 ---> v2;
      vem + {xm |-> v2} |- em ---> v
   |] ==>
   ve |- e1 @@ e2 ---> v
⟨proof⟩


lemma eval-ind0:
  [| ve |- e ---> v;
     !!ve c. P(((ve,e-const(c)),v-const(c)));
     !!ev ve. ev:ve-dom(ve) ==> P(((ve,e-var(ev)),ve-app ve ev));
     !!ev ve e. P(((ve,fn ev => e),v-clos(<|ev,e,ve|>)));
     !!ev1 ev2 ve cl e.
       cl = <| ev1, e, ve + {ev2 |-> v-clos(cl)} |> ==>
       P(((ve,fix ev2(ev1) = e),v-clos(cl)));
     !!ve c1 c2 e1 e2.
       [| P(((ve,e1),v-const(c1))); P(((ve,e2),v-const(c2))) |] ==>
       P(((ve,e1 @@ e2),v-const(c-app c1 c2)));
     !!ve vem xm e1 e2 em v v2.
       [|  P(((ve,e1),v-clos(<|xm,em,vem|>)));
           P(((ve,e2),v2));
           P(((vem + {xm |-> v2},em),v))
       |] ==>
       P(((ve,e1 @@ e2),v))
   |] ==>
   P(((ve,e),v))
⟨proof⟩

lemma eval-ind:
  [| ve |- e ---> v;
     !!ve c. P ve (e-const c) (v-const c);
     !!ev ve. ev:ve-dom(ve) ==> P ve (e-var ev) (ve-app ve ev);
     !!ev ve e. P ve (fn ev => e) (v-clos <|ev,e,ve|>);
     !!ev1 ev2 ve cl e.
       cl = <| ev1, e, ve + {ev2 |-> v-clos(cl)} |> ==>
       P ve (fix ev2(ev1) = e) (v-clos cl);
     !!ve c1 c2 e1 e2.
       [| P ve e1 (v-const c1); P ve e2 (v-const c2) |] ==>
       P ve (e1 @@ e2) (v-const(c-app c1 c2));
     !!ve vem evm e1 e2 em v v2.
       [|  P ve e1 (v-clos <|evm,em,vem|>);
           P ve e2 v2;
           P (vem + {evm |-> v2}) em v
       |] ==> P ve (e1 @@ e2) v
   |] ==> P ve e v
⟨proof⟩
```

lemma *elab-fun-mono*: *mono*(*elab-fun*)
⟨*proof*⟩

lemma *elab-const*:
c isof ty ==> te |- e-const(*c*) *===> ty*
⟨*proof*⟩

lemma *elab-var*:
x:*te-dom*(*te*) *==> te |- e-var*(*x*) *===> te-app te x*
⟨*proof*⟩

lemma *elab-fn*:
te + {*x* |=> *ty1*} *|- e ===> ty2 ==> te |- fn x => e ===> ty1->ty2*
⟨*proof*⟩

lemma *elab-fix*:
te + {*f* |=> *ty1->ty2*} + {*x* |=> *ty1*} *|- e ===> ty2 ==>*
te |- fix f(*x*) = *e ===> ty1->ty2*
⟨*proof*⟩

lemma *elab-app*:
[| *te |- e1 ===> ty1->ty2*; *te |- e2 ===> ty1* |] *==>*
te |- e1 @@ e2 ===> ty2
⟨*proof*⟩

lemma *elab-ind0*:
assumes *1*: *te |- e ===> t*
and *2*: !!*te c t*. *c isof t ==> P*(((*te*,*e-const*(*c*)),*t*))
and *3*: !!*te x*. *x*:*te-dom*(*te*) *==> P*(((*te*,*e-var*(*x*)),*te-app te x*))
and *4*: !!*te x e t1 t2*.
[| *te* + {*x* |=> *t1*} *|- e ===> t2*; *P*(((*te* + {*x* |=> *t1*},*e*),*t2*)) |] *==>*
P(((*te*,*fn x => e*),*t1->t2*))
and *5*: !!*te f x e t1 t2*.
[| *te* + {*f* |=> *t1->t2*} + {*x* |=> *t1*} *|- e ===> t2*;
P(((*te* + {*f* |=> *t1->t2*} + {*x* |=> *t1*},*e*),*t2*))
|] *==>*
P(((*te*,*fix f*(*x*) = *e*),*t1->t2*))
and *6*: !!*te e1 e2 t1 t2*.
[| *te |- e1 ===> t1->t2*; *P*(((*te*,*e1*),*t1->t2*));
te |- e2 ===> t1; *P*(((*te*,*e2*),*t1*))
|] *==>*

```
      P(((te,e1 @@ e2),t2))
  shows P(((te,e),t))
⟨proof⟩

lemma elab-ind:
   [| te |- e ===> t;
       !!te c t. c isof t ==> P te (e-const c) t;
      !!te x. x:te-dom(te) ==> P te (e-var x) (te-app te x);
      !!te x e t1 t2.
        [| te + {x |=> t1} |- e ===> t2; P (te + {x |=> t1}) e t2 |] ==>
        P te (fn x => e) (t1->t2);
      !!te f x e t1 t2.
        [| te + {f |=> t1->t2} + {x |=> t1} |- e ===> t2;
           P (te + {f |=> t1->t2} + {x |=> t1}) e t2
        |] ==>
        P te (fix f(x) = e) (t1->t2);
      !!te e1 e2 t1 t2.
        [| te |- e1 ===> t1->t2; P te e1 (t1->t2);
           te |- e2 ===> t1; P te e2 t1
        |] ==>
        P te (e1 @@ e2) t2
    |] ==>
    P te e t
⟨proof⟩



lemma elab-elim0:
  assumes 1: te |- e ===> t
    and 2: !!te c t. c isof t ==> P(((te,e-const(c)),t))
    and 3: !!te x. x:te-dom(te) ==> P(((te,e-var(x)),te-app te x))
    and 4: !!te x e t1 t2.
         te + {x |=> t1} |- e ===> t2 ==> P(((te,fn x => e),t1->t2))
    and 5: !!te f x e t1 t2.
         te + {f |=> t1->t2} + {x |=> t1} |- e ===> t2 ==>
         P(((te,fix f(x) = e),t1->t2))
    and 6: !!te e1 e2 t1 t2.
         [| te |- e1 ===> t1->t2; te |- e2 ===> t1 |] ==>
         P(((te,e1 @@ e2),t2))
  shows P(((te,e),t))
⟨proof⟩

lemma elab-elim:
   [| te |- e ===> t;
       !!te c t. c isof t ==> P te (e-const c) t;
      !!te x. x:te-dom(te) ==> P te (e-var x) (te-app te x);
      !!te x e t1 t2.
        te + {x |=> t1} |- e ===> t2 ==> P te (fn x => e) (t1->t2);
      !!te f x e t1 t2.
```

$te + \{f \mid\Rightarrow t1 \rightarrow t2\} + \{x \mid\Rightarrow t1\} \mid- e \Longrightarrow t2 \Longrightarrow$
$P\ te\ (\mathit{fix}\ f(x) = e)\ (t1 \rightarrow t2);$
$!!te\ e1\ e2\ t1\ t2.$
$[\mid te \mid- e1 \Longrightarrow t1 \rightarrow t2;\ te \mid- e2 \Longrightarrow t1 \mid] \Longrightarrow$
$P\ te\ (e1\ @@\ e2)\ t2$
$|] \Longrightarrow$
$P\ te\ e\ t$
⟨*proof*⟩

lemma *elab-const-elim-lem*:
$te \mid- e \Longrightarrow t \Longrightarrow (e = \textit{e-const}(c) \longrightarrow c\ \textit{isof}\ t)$
⟨*proof*⟩

lemma *elab-const-elim*: $te \mid- \textit{e-const}(c) \Longrightarrow t \Longrightarrow c\ \textit{isof}\ t$
⟨*proof*⟩

lemma *elab-var-elim-lem*:
$te \mid- e \Longrightarrow t \Longrightarrow (e = \textit{e-var}(x) \longrightarrow t{=}\textit{te-app}\ te\ x\ \&\ x{:}\textit{te-dom}(te))$
⟨*proof*⟩

lemma *elab-var-elim*: $te \mid- \textit{e-var}(ev) \Longrightarrow t \Longrightarrow t{=}\textit{te-app}\ te\ ev\ \&\ ev : \textit{te-dom}(te)$
⟨*proof*⟩

lemma *elab-fn-elim-lem*:
$te \mid- e \Longrightarrow t \Longrightarrow$
$(\ e = \mathit{fn}\ x1 \Rightarrow e1 \longrightarrow$
$(?\ t1\ t2.\ t{=}\textit{t-fun}\ t1\ t2\ \&\ te + \{x1 \mid\Rightarrow t1\} \mid- e1 \Longrightarrow t2)$
$)$
⟨*proof*⟩

lemma *elab-fn-elim*: $te \mid- \mathit{fn}\ x1 \Rightarrow e1 \Longrightarrow t \Longrightarrow$
$(?\ t1\ t2.\ t{=}t1 \rightarrow t2\ \&\ te + \{x1 \mid\Rightarrow t1\} \mid- e1 \Longrightarrow t2)$
⟨*proof*⟩

lemma *elab-fix-elim-lem*:
$te \mid- e \Longrightarrow t \Longrightarrow$
$(e = \mathit{fix}\ f(x) = e1 \longrightarrow$
$(?\ t1\ t2.\ t{=}t1 \rightarrow t2\ \&\ te + \{f \mid\Rightarrow t1 \rightarrow t2\} + \{x \mid\Rightarrow t1\} \mid- e1 \Longrightarrow t2))$
⟨*proof*⟩

lemma *elab-fix-elim*: $te \mid- \mathit{fix}\ ev1(ev2) = e1 \Longrightarrow t \Longrightarrow$
$(?\ t1\ t2.\ t{=}t1 \rightarrow t2\ \&\ te + \{ev1 \mid\Rightarrow t1 \rightarrow t2\} + \{ev2 \mid\Rightarrow t1\} \mid- e1 \Longrightarrow t2)$
⟨*proof*⟩

lemma *elab-app-elim-lem*:

te |− *e* ===> *t2* ==>
(*e* = *e1* @@ *e2* −−> (*? t1* . *te* |− *e1* ===> *t1*−>*t2* & *te* |− *e2* ===> *t1*))
⟨*proof*⟩

lemma *elab-app-elim*: *te* |− *e1* @@ *e2* ===> *t2* ==> (*? t1* . *te* |− *e1* ===> *t1*−>*t2* & *te* |− *e2* ===> *t1*)
⟨*proof*⟩

lemma *mono-hasty-fun*: *mono*(*hasty-fun*)
⟨*proof*⟩

lemma *hasty-rel-const-coind*: *c isof t* ==> (*v-const*(*c*),*t*) : *hasty-rel*
⟨*proof*⟩

lemma *hasty-rel-clos-coind*:
[| *te* |− *fn ev* => *e* ===> *t*;
ve-dom(*ve*) = *te-dom*(*te*);
! *ev1*.
ev1:*ve-dom*(*ve*) −−>
(*ve-app ve ev1*,*te-app te ev1*) : {(*v-clos*(<|*ev*,*e*,*ve*|>),*t*)} *Un hasty-rel*
|] ==>
(*v-clos*(<|*ev*,*e*,*ve*|>),*t*) : *hasty-rel*
⟨*proof*⟩

lemma *hasty-rel-elim0*:
[| !! *c t*. *c isof t* ==> *P*((*v-const*(*c*),*t*));
!! *te ev e t ve*.
[| *te* |− *fn ev* => *e* ===> *t*;
ve-dom(*ve*) = *te-dom*(*te*);
!*ev1*. *ev1*:*ve-dom*(*ve*) −−> (*ve-app ve ev1*,*te-app te ev1*) : *hasty-rel*
|] ==> *P*((*v-clos*(<|*ev*,*e*,*ve*|>),*t*));
(*v*,*t*) : *hasty-rel*
|] ==> *P*(*v*,*t*)
⟨*proof*⟩

lemma *hasty-rel-elim*:
[| (*v*,*t*) : *hasty-rel*;
!! *c t*. *c isof t* ==> *P* (*v-const c*) *t*;
!! *te ev e t ve*.
[| *te* |- *fn ev* => *e* ===> *t*;
ve-dom(*ve*) = *te-dom*(*te*);
!*ev1*. *ev1*:*ve-dom*(*ve*) --> (*ve-app ve ev1*,*te-app te ev1*) : *hasty-rel*
|] ==> *P* (*v-clos* <|*ev*,*e*,*ve*|>) *t*
|] ==> *P v t*
⟨*proof*⟩

lemma *hasty-const*: *c isof t* ==> *v-const*(*c*) *hasty t*
⟨*proof*⟩

lemma *hasty-clos*:
te |- *fn ev* => *e* ===> *t* & *ve hastyenv te* ==> *v-clos*(<|*ev*,*e*,*ve*|>) *hasty t*
⟨*proof*⟩

lemma *hasty-elim-const-lem*:
v hasty t ==> (!*c*.(*v* = *v-const*(*c*) --> *c isof t*))
⟨*proof*⟩

lemma *hasty-elim-const*: *v-const*(*c*) *hasty t* ==> *c isof t*
⟨*proof*⟩

lemma *hasty-elim-clos-lem*:
v hasty t ==>
! *x e ve*.
v=*v-clos*(<|*x*,*e*,*ve*|>) --> (*? te*. *te* |- *fn x* => *e* ===> *t* & *ve hastyenv te*)
⟨*proof*⟩

lemma *hasty-elim-clos*: *v-clos*(<|*ev*,*e*,*ve*|>) *hasty t* ==>
? te. *te* |- *fn ev* => *e* ===> *t* & *ve hastyenv te*
⟨*proof*⟩

lemma *hasty-env1*: [| *ve hastyenv te*; *v hasty t* |] ==>
ve + {*ev* |-> *v*} *hastyenv te* + {*ev* |=> *t*}
⟨*proof*⟩

lemma *consistency-const*: [| *ve hastyenv te* ; *te* |− *e-const*(*c*) ===> *t* |] ==> *v-const*(*c*) *hasty t*
⟨*proof*⟩

lemma *consistency-var*:
[| *ev* : *ve-dom*(*ve*); *ve hastyenv te* ; *te* |− *e-var*(*ev*) ===> *t* |] ==>
ve-app ve ev hasty t
⟨*proof*⟩

lemma *consistency-fn*: [| *ve hastyenv te* ; *te* |− *fn ev* => *e* ===> *t* |] ==>
v-clos(<| *ev*, *e*, *ve* |>) *hasty t*
⟨*proof*⟩

lemma *consistency-fix*:
[| *cl* = <| *ev1*, *e*, *ve* + { *ev2* |−> *v-clos*(*cl*) } |>;
ve hastyenv te ;
te |− *fix ev2 ev1* = *e* ===> *t*
|] ==>
v-clos(*cl*) *hasty t*
⟨*proof*⟩

lemma *consistency-app1*: [| ! *t te*. *ve hastyenv te* −−> *te* |− *e1* ===> *t* −−> *v-const*(*c1*) *hasty t*;
! *t te*. *ve hastyenv te* −−> *te* |− *e2* ===> *t* −−> *v-const*(*c2*) *hasty t*;
ve hastyenv te ; *te* |− *e1* @@ *e2* ===> *t*
|] ==>
v-const(*c-app c1 c2*) *hasty t*
⟨*proof*⟩

lemma *consistency-app2*: [| ! *t te*.
ve hastyenv te −−>
te |− *e1* ===> *t* −−> *v-clos*(<|*evm*, *em*, *vem*|>) *hasty t*;
! *t te*. *ve hastyenv te* −−> *te* |− *e2* ===> *t* −−> *v2 hasty t*;
! *t te*.
vem + { *evm* |−> *v2* } *hastyenv te* −−> *te* |− *em* ===> *t* −−> *v hasty t*;
ve hastyenv te ;
te |− *e1* @@ *e2* ===> *t*
|] ==>
v hasty t
⟨*proof*⟩

lemma *consistency*: *ve* |− *e* −−−> *v* ==>
(! *t te*. *ve hastyenv te* −−> *te* |− *e* ===> *t* −−> *v hasty t*)

⟨*proof*⟩

lemma *basic-consistency-lem*:
ve isofenv te ==> ve hastyenv te
⟨*proof*⟩

lemma *basic-consistency*:
[| ve isofenv te; ve |- e ---> v-const(c); te |- e ===> t |] ==> c isof t
⟨*proof*⟩

end

30 Case study: Unification Algorithm

theory *Unification*
imports *Main*
begin

This is a formalization of a first-order unification algorithm. It uses the new "function" package to define recursive functions, which allows a better treatment of nested recursion.

This is basically a modernized version of a previous formalization by Konrad Slind (see: HOL/Subst/Unify.thy), which itself builds on previous work by Paulson and Manna & Waldinger (for details, see there).

Unlike that formalization, where the proofs of termination and some partial correctness properties are intertwined, we can prove partial correctness and termination separately.

30.1 Basic definitions

datatype *'a trm* =
Var 'a
| *Const 'a*
| *App 'a trm 'a trm* (**infix** · *60*)

types
'a subst = (*'a* × *'a trm*) *list*

Applying a substitution to a variable:

fun *assoc* :: $'a \Rightarrow 'b \Rightarrow ('a \times 'b)\ list \Rightarrow 'b$
where
  $assoc\ x\ d\ [] = d$
| $assoc\ x\ d\ ((p,q)\#t) = (if\ x = p\ then\ q\ else\ assoc\ x\ d\ t)$

Applying a substitution to a term:

fun *apply-subst* :: $'a\ trm \Rightarrow 'a\ subst \Rightarrow 'a\ trm$ (**infixl** $\lhd$ *60*)
where
  $(Var\ v) \lhd s = assoc\ v\ (Var\ v)\ s$
| $(Const\ c) \lhd s = (Const\ c)$
| $(M \cdot N) \lhd s = (M \lhd s) \cdot (N \lhd s)$

Composition of substitutions:

fun
  compose :: $'a\ subst \Rightarrow 'a\ subst \Rightarrow 'a\ subst$ (**infixl** $\cdot$ *80*)
where
  $[] \cdot bl = bl$
| $((a,b)\ \#\ al) \cdot bl = (a,\ b \lhd bl)\ \#\ (al \cdot bl)$

Equivalence of substitutions:

definition *eqv* (**infix** $=_s$ *50*)
where
  $s1 =_s s2 \equiv \forall t.\ t \lhd s1 = t \lhd s2$

30.2 Basic lemmas

lemma *apply-empty*[*simp*]: $t \lhd [] = t$
⟨*proof*⟩

lemma *compose-empty*[*simp*]: $\sigma \cdot [] = \sigma$
⟨*proof*⟩

lemma *apply-compose*[*simp*]: $t \lhd (s1 \cdot s2) = t \lhd s1 \lhd s2$
⟨*proof*⟩

lemma *eqv-refl*[*intro*]: $s =_s s$
  ⟨*proof*⟩

lemma *eqv-trans*[*trans*]: $[\![s1 =_s s2;\ s2 =_s s3]\!] \Longrightarrow s1 =_s s3$
  ⟨*proof*⟩

lemma *eqv-sym*[*sym*]: $[\![s1 =_s s2]\!] \Longrightarrow s2 =_s s1$
  ⟨*proof*⟩

lemma *eqv-intro*[*intro*]: $(\bigwedge t.\ t \lhd \sigma = t \lhd \vartheta) \Longrightarrow \sigma =_s \vartheta$
  ⟨*proof*⟩

lemma *eqv-dest*[*dest*]: $s1 =_s s2 \Longrightarrow t \lhd s1 = t \lhd s2$
  ⟨*proof*⟩

lemma *compose-eqv*: $[\![\sigma =_s \sigma'; \vartheta =_s \vartheta']\!] \Longrightarrow (\sigma \cdot \vartheta) =_s (\sigma' \cdot \vartheta')$
⟨*proof*⟩

lemma *compose-assoc*: $(a \cdot b) \cdot c =_s a \cdot (b \cdot c)$
⟨*proof*⟩

30.3 Specification: Most general unifiers

definition
$Unifier\ \sigma\ t\ u \equiv (t \lhd \sigma = u \lhd \sigma)$

definition
$MGU\ \sigma\ t\ u \equiv Unifier\ \sigma\ t\ u \wedge (\forall \vartheta.\ Unifier\ \vartheta\ t\ u \longrightarrow (\exists \gamma.\ \vartheta =_s \sigma \cdot \gamma))$

lemma *MGUI*[*intro*]:
$[\![t \lhd \sigma = u \lhd \sigma; \bigwedge \vartheta.\ t \lhd \vartheta = u \lhd \vartheta \Longrightarrow \exists \gamma.\ \vartheta =_s \sigma \cdot \gamma]\!] \Longrightarrow MGU\ \sigma\ t\ u$
⟨*proof*⟩

lemma *MGU-sym*[*sym*]:
$MGU\ \sigma\ s\ t \Longrightarrow MGU\ \sigma\ t\ s$
⟨*proof*⟩

30.4 The unification algorithm

Occurs check: Proper subterm relation

fun *occ* :: *'a trm* ⇒ *'a trm* ⇒ *bool*
where
occ u (*Var v*) = *False*
| *occ u* (*Const c*) = *False*
| *occ u* (*M* · *N*) = (*u* = *M* ∨ *u* = *N* ∨ *occ u M* ∨ *occ u N*)

The unification algorithm:

function *unify* :: *'a trm* ⇒ *'a trm* ⇒ *'a subst option*
where
unify (*Const c*) (*M* · *N*) = *None*
| *unify* (*M* · *N*) (*Const c*) = *None*
| *unify* (*Const c*) (*Var v*) = *Some* [(*v*, *Const c*)]
| *unify* (*M* · *N*) (*Var v*) = (*if* (*occ* (*Var v*) (*M* · *N*))
then None
else Some [(*v*, *M* · *N*)])
| *unify* (*Var v*) *M* = (*if* (*occ* (*Var v*) *M*)
then None
else Some [(*v*, *M*)])
| *unify* (*Const c*) (*Const d*) = (*if c*=*d then Some* [] *else None*)
| *unify* (*M* · *N*) (*M'* · *N'*) = (*case unify M M' of*
None ⇒ *None* |

Some $\vartheta \Rightarrow$ (*case unify* ($N \lhd \vartheta$) ($N' \lhd \vartheta$)
of None $\Rightarrow$ *None* |
Some $\sigma \Rightarrow$ *Some* ($\vartheta \cdot \sigma$)))

⟨*proof*⟩

30.5 Partial correctness

Some lemmas about occ and MGU:

lemma *subst-no-occ*: $\neg$*occ* (*Var v*) *t* $\Longrightarrow$ *Var v* $\neq$ *t*
$\Longrightarrow$ *t* $\lhd$ [(*v*,*s*)] = *t*
⟨*proof*⟩

lemma *MGU-Var*[*intro*]:
assumes *no-occ*: $\neg$*occ* (*Var v*) *t*
shows *MGU* [(*v*,*t*)] (*Var v*) *t*
⟨*proof*⟩

declare *MGU-Var*[*symmetric*, *intro*]

lemma *MGU-Const*[*simp*]: *MGU* [] (*Const c*) (*Const d*) = (*c* = *d*)
⟨*proof*⟩

If unification terminates, then it computes most general unifiers:

lemma *unify-partial-correctness*:
assumes *unify-dom* (*M*, *N*)
assumes *unify M N* = *Some* σ
shows *MGU* σ *M N*
⟨*proof*⟩

30.6 Properties used in termination proof

The variables of a term:

fun *vars-of*:: *'a trm* $\Rightarrow$ *'a set*
where
vars-of (*Var v*) = { *v* }
| *vars-of* (*Const c*) = {}
| *vars-of* (*M* · *N*) = *vars-of M* $\cup$ *vars-of N*

lemma *vars-of-finite*[*intro*]: *finite* (*vars-of t*)
⟨*proof*⟩

Elimination of variables by a substitution:

definition
elim σ *v* $\equiv \forall$ *t*. *v* $\notin$ *vars-of* (*t* $\lhd$ σ)

lemma *elim-intro*[*intro*]: ($\bigwedge$*t*. *v* $\notin$ *vars-of* (*t* $\lhd$ σ)) $\Longrightarrow$ *elim* σ *v*
⟨*proof*⟩

lemma *elim-dest*[*dest*]: *elim* σ $v \Longrightarrow v \notin$ *vars-of* $(t \lhd \sigma)$
⟨*proof*⟩

lemma *elim-eqv*: $\sigma =_s \vartheta \Longrightarrow$ *elim* σ $x =$ *elim* ϑ x
⟨*proof*⟩

Replacing a variable by itself yields an identity subtitution:

lemma *var-self*[*intro*]: $[(v,\ Var\ v)] =_s []$
⟨*proof*⟩

lemma *var-same*: $(t = Var\ v) = ([(v,\ t)] =_s [])$
⟨*proof*⟩

A lemma about occ and elim

lemma *remove-var*:
assumes [*simp*]: $v \notin$ *vars-of* s
shows $v \notin$ *vars-of* $(t \lhd [(v,\ s)])$
⟨*proof*⟩

lemma *occ-elim*: $\neg$*occ* $(Var\ v)\ t$
$\Longrightarrow$ *elim* $[(v,t)]\ v \vee [(v,t)] =_s []$
⟨*proof*⟩

The result of a unification never introduces new variables:

lemma *unify-vars*:
assumes *unify-dom* $(M,\ N)$
assumes *unify* $M\ N =$ *Some* σ
shows *vars-of* $(t \lhd \sigma) \subseteq$ *vars-of* $M \cup$ *vars-of* $N \cup$ *vars-of* t
(**is** *?P* $M\ N\ \sigma\ t$)
⟨*proof*⟩

The result of a unification is either the identity substitution or it eliminates a variable from one of the terms:

lemma *unify-eliminates*:
assumes *unify-dom* $(M,\ N)$
assumes *unify* $M\ N =$ *Some* σ
shows $(\exists\, v{\in}$*vars-of* $M \cup$ *vars-of* $N.$ *elim* $\sigma\ v) \vee \sigma =_s []$
(**is** *?P* $M\ N\ \sigma$)
⟨*proof*⟩

30.7 Termination proof

termination *unify*
⟨*proof*⟩

end

31 Some examples demonstrating the comm-ring method

theory *Commutative-RingEx*
imports *Commutative-Ring*
begin

lemma $4*(x::int)\hat{}5*y\hat{}3*x\hat{}2*3 + x*z + 3\hat{}5 = 12*x\hat{}7*y\hat{}3 + z*x + 243$
⟨*proof*⟩

lemma $((x::int) + y)\hat{}2 = x\hat{}2 + y\hat{}2 + 2*x*y$
⟨*proof*⟩

lemma $((x::int) + y)\hat{}3 = x\hat{}3 + y\hat{}3 + 3*x\hat{}2*y + 3*y\hat{}2*x$
⟨*proof*⟩

lemma $((x::int) - y)\hat{}3 = x\hat{}3 + 3*x*y\hat{}2 + (-3)*y*x\hat{}2 - y\hat{}3$
⟨*proof*⟩

lemma $((x::int) - y)\hat{}2 = x\hat{}2 + y\hat{}2 - 2*x*y$
⟨*proof*⟩

lemma $((a::int) + b + c)\hat{}2 = a\hat{}2 + b\hat{}2 + c\hat{}2 + 2*a*b + 2*b*c + 2*a*c$
⟨*proof*⟩

lemma $((a::int) - b - c)\hat{}2 = a\hat{}2 + b\hat{}2 + c\hat{}2 - 2*a*b + 2*b*c - 2*a*c$
⟨*proof*⟩

lemma $(a::int)*b + a*c = a*(b+c)$
⟨*proof*⟩

lemma $(a::int)\hat{}2 - b\hat{}2 = (a - b) * (a + b)$
⟨*proof*⟩

lemma $(a::int)\hat{}3 - b\hat{}3 = (a - b) * (a\hat{}2 + a*b + b\hat{}2)$
⟨*proof*⟩

lemma $(a::int)\hat{}3 + b\hat{}3 = (a + b) * (a\hat{}2 - a*b + b\hat{}2)$
⟨*proof*⟩

lemma $(a::int)\hat{}4 - b\hat{}4 = (a - b) * (a + b)*(a\hat{}2 + b\hat{}2)$
⟨*proof*⟩

lemma $(a::int)\hat{}10 - b\hat{}10 = (a - b) * (a\hat{}9 + a\hat{}8*b + a\hat{}7*b\hat{}2 + a\hat{}6*b\hat{}3 + a\hat{}5*b\hat{}4 + a\hat{}4*b\hat{}5 + a\hat{}3*b\hat{}6 + a\hat{}2*b\hat{}7 + a*b\hat{}8 + b\hat{}9\ )$
⟨*proof*⟩

end

32 Primitive Recursive Functions

theory *Primrec* **imports** *Main* **begin**

Proof adopted from

Nora Szasz, A Machine Checked Proof that Ackermann's Function is not Primitive Recursive, In: Huet & Plotkin, eds., Logical Environments (CUP, 1993), 317-338.

See also E. Mendelson, Introduction to Mathematical Logic. (Van Nostrand, 1964), page 250, exercise 11.

32.1 Ackermann's Function

fun *ack* :: *nat* => *nat* => *nat* **where**
ack 0 n = *Suc n* |
ack (*Suc m*) *0* = *ack m 1* |
ack (*Suc m*) (*Suc n*) = *ack m* (*ack* (*Suc m*) *n*)

PROPERTY A 4

lemma *less-ack2* [*iff*]: *j* < *ack i j*
⟨*proof*⟩

PROPERTY A 5-, the single-step lemma

lemma *ack-less-ack-Suc2* [*iff*]: *ack i j* < *ack i* (*Suc j*)
⟨*proof*⟩

PROPERTY A 5, monotonicity for <

lemma *ack-less-mono2*: *j* < *k* ==> *ack i j* < *ack i k*
⟨*proof*⟩

PROPERTY A 5', monotonicity for ≤

lemma *ack-le-mono2*: *j* ≤ *k* ==> *ack i j* ≤ *ack i k*
⟨*proof*⟩

PROPERTY A 6

lemma *ack2-le-ack1* [*iff*]: *ack i* (*Suc j*) ≤ *ack* (*Suc i*) *j*
⟨*proof*⟩

PROPERTY A 7-, the single-step lemma

lemma *ack-less-ack-Suc1* [*iff*]: *ack i j* < *ack* (*Suc i*) *j*
⟨*proof*⟩

PROPERTY A 4'? Extra lemma needed for *CONSTANT* case, constant functions

lemma *less-ack1* [*iff*]: *i* < *ack i j*
⟨*proof*⟩

PROPERTY A 8

lemma *ack-1* [*simp*]: *ack* (*Suc 0*) *j* = *j* + *2*
⟨*proof*⟩

PROPERTY A 9. The unary *1* and *2* in *ack* is essential for the rewriting.

lemma *ack-2* [*simp*]: *ack* (*Suc* (*Suc 0*)) *j* = *2* ∗ *j* + *3*
⟨*proof*⟩

PROPERTY A 7, monotonicity for < [not clear why *ack-1* is now needed first!]

lemma *ack-less-mono1-aux*: *ack i k* < *ack* (*Suc* (*i* + *i'*)) *k*
⟨*proof*⟩

lemma *ack-less-mono1*: *i* < *j* ==> *ack i k* < *ack j k*
⟨*proof*⟩

PROPERTY A 7', monotonicity for ≤

lemma *ack-le-mono1*: *i* ≤ *j* ==> *ack i k* ≤ *ack j k*
⟨*proof*⟩

PROPERTY A 10

lemma *ack-nest-bound*: *ack i1* (*ack i2 j*) < *ack* (*2* + (*i1* + *i2*)) *j*
⟨*proof*⟩

PROPERTY A 11

lemma *ack-add-bound*: *ack i1 j* + *ack i2 j* < *ack* (*4* + (*i1* + *i2*)) *j*
⟨*proof*⟩

PROPERTY A 12. Article uses existential quantifier but the ALF proof used *k* + *4*. Quantified version must be nested ∃ *k'*. ∀ *i j*. ...

lemma *ack-add-bound2*: *i* < *ack k j* ==> *i* + *j* < *ack* (*4* + *k*) *j*
⟨*proof*⟩

32.2 Primitive Recursive Functions

primrec *hd0* :: *nat list* => *nat* **where**
hd0 [] = *0* |
hd0 (*m* # *ms*) = *m*

Inductive definition of the set of primitive recursive functions of type *nat list* ⇒ *nat*.

definition *SC* :: *nat list* => *nat* **where**
SC l = *Suc* (*hd0 l*)

definition *CONSTANT* :: *nat* => *nat list* => *nat* **where**
CONSTANT k l = *k*

definition *PROJ* :: *nat* => *nat list* => *nat* **where**
PROJ i l = *hd0* (*drop i l*)

definition
COMP :: (*nat list* => *nat*) => (*nat list* => *nat*) *list* => *nat list* => *nat*
where *COMP g fs l* = *g* (*map* (λ*f*. *f l*) *fs*)

definition *PREC* :: (*nat list* => *nat*) => (*nat list* => *nat*) => *nat list* => *nat*
where
PREC f g l =
(*case l of*
[] => *0*
| *x* # *l'* => *nat-rec* (*f l'*) (λ*y r*. *g* (*r* # *y* # *l'*)) *x*)
— Note that *g* is applied first to *PREC f g y* and then to *y*!

inductive *PRIMREC* :: (*nat list* => *nat*) => *bool* **where**
SC: *PRIMREC SC* |
CONSTANT: *PRIMREC* (*CONSTANT k*) |
PROJ: *PRIMREC* (*PROJ i*) |
COMP: *PRIMREC g* ==> ∀*f* ∈ *set fs*. *PRIMREC f* ==> *PRIMREC* (*COMP g fs*) |
PREC: *PRIMREC f* ==> *PRIMREC g* ==> *PRIMREC* (*PREC f g*)

Useful special cases of evaluation

lemma *SC* [*simp*]: *SC* (*x* # *l*) = *Suc x*
⟨*proof*⟩

lemma *CONSTANT* [*simp*]: *CONSTANT k l* = *k*
⟨*proof*⟩

lemma *PROJ-0* [*simp*]: *PROJ 0* (*x* # *l*) = *x*
⟨*proof*⟩

lemma *COMP-1* [*simp*]: *COMP g* [*f*] *l* = *g* [*f l*]
⟨*proof*⟩

lemma *PREC-0* [*simp*]: *PREC f g* (*0* # *l*) = *f l*
⟨*proof*⟩

lemma *PREC-Suc* [*simp*]: *PREC f g* (*Suc x* # *l*) = *g* (*PREC f g* (*x* # *l*) # *x* # *l*)
⟨*proof*⟩

MAIN RESULT

lemma *SC-case*: *SC l* < *ack 1* (*listsum l*)
⟨*proof*⟩

lemma *CONSTANT-case*: *CONSTANT k l* < *ack k* (*listsum l*)
⟨*proof*⟩

lemma *PROJ-case*: *PROJ i l* < *ack 0* (*listsum l*)
⟨*proof*⟩

COMP case

lemma *COMP-map-aux*: ∀ *f* ∈ *set fs*. *PRIMREC f* ∧ (∃ *kf*. ∀ *l*. *f l* < *ack kf* (*listsum l*))
==> ∃ *k*. ∀ *l*. *listsum* (*map* (λ*f*. *f l*) *fs*) < *ack k* (*listsum l*)
⟨*proof*⟩

lemma *COMP-case*:
∀ *l*. *g l* < *ack kg* (*listsum l*) ==>
∀ *f* ∈ *set fs*. *PRIMREC f* ∧ (∃ *kf*. ∀ *l*. *f l* < *ack kf* (*listsum l*))
==> ∃ *k*. ∀ *l*. *COMP g fs l* < *ack k* (*listsum l*)
⟨*proof*⟩

PREC case

lemma *PREC-case-aux*:
∀ *l*. *f l* + *listsum l* < *ack kf* (*listsum l*) ==>
∀ *l*. *g l* + *listsum l* < *ack kg* (*listsum l*) ==>
PREC f g l + *listsum l* < *ack* (*Suc* (*kf* + *kg*)) (*listsum l*)
⟨*proof*⟩

lemma *PREC-case*:
∀ *l*. *f l* < *ack kf* (*listsum l*) ==>
∀ *l*. *g l* < *ack kg* (*listsum l*) ==>
∃ *k*. ∀ *l*. *PREC f g l* < *ack k* (*listsum l*)
⟨*proof*⟩

lemma *ack-bounds-PRIMREC*: *PRIMREC f* ==> ∃ *k*. ∀ *l*. *f l* < *ack k* (*listsum l*)
⟨*proof*⟩

theorem *ack-not-PRIMREC*:
¬ *PRIMREC* (λ*l*. *case l of* [] => *0* | *x* # *l'* => *ack x x*)
⟨*proof*⟩

end

33 The Full Theorem of Tarski

theory *Tarski*
imports *Main FuncSet*
begin

Minimal version of lattice theory plus the full theorem of Tarski: The fixed-points of a complete lattice themselves form a complete lattice.

Illustrates first-class theories, using the Sigma representation of structures. Tidied and converted to Isar by lcp.

record *'a potype* =
  pset :: *'a set*
  order :: *('a * 'a) set*

definition
  monotone :: *['a => 'a, 'a set, ('a *'a)set] => bool* **where**
  monotone f A r = *(∀x∈A. ∀y∈A. (x, y): r --> ((f x), (f y)) : r)*

definition
  least :: *['a => bool, 'a potype] => 'a* **where**
  least P po = *(SOME x. x: pset po & P x &*
                (∀y ∈ pset po. P y --> (x,y): order po))

definition
  greatest :: *['a => bool, 'a potype] => 'a* **where**
  greatest P po = *(SOME x. x: pset po & P x &*
                  (∀y ∈ pset po. P y --> (y,x): order po))

definition
  lub :: *['a set, 'a potype] => 'a* **where**
  lub S po = *least (%x. ∀y∈S. (y,x): order po) po*

definition
  glb :: *['a set, 'a potype] => 'a* **where**
  glb S po = *greatest (%x. ∀y∈S. (x,y): order po) po*

definition
  isLub :: *['a set, 'a potype, 'a] => bool* **where**
  isLub S po = *(%L. (L: pset po & (∀y∈S. (y,L): order po) &*
                (∀z∈pset po. (∀y∈S. (y,z): order po) --> (L,z): order po)))

definition
  isGlb :: *['a set, 'a potype, 'a] => bool* **where**
  isGlb S po = *(%G. (G: pset po & (∀y∈S. (G,y): order po) &*
                (∀z ∈ pset po. (∀y∈S. (z,y): order po) --> (z,G): order po)))

definition
  fix :: *[('a => 'a), 'a set] => 'a set* **where**
  fix f A = *{x. x: A & f x = x}*

definition
  interval :: *[('a*'a) set,'a, 'a] => 'a set* **where**
  interval r a b = *{x. (a,x): r & (x,b): r}*

definition
  Bot :: *'a potype => 'a* **where**

```
  Bot po = least (%x. True) po

definition
  Top :: 'a potype => 'a where
  Top po = greatest (%x. True) po

definition
  PartialOrder :: ('a potype) set where
  PartialOrder = {P. refl-on (pset P) (order P) & antisym (order P) &
                     trans (order P)}

definition
  CompleteLattice :: ('a potype) set where
  CompleteLattice = {cl. cl: PartialOrder &
                       (∀S. S ⊆ pset cl --> (∃L. isLub S cl L)) &
                       (∀S. S ⊆ pset cl --> (∃G. isGlb S cl G))}

definition
  CLF-set :: ('a potype * ('a => 'a)) set where
  CLF-set = (SIGMA cl: CompleteLattice.
           {f. f: pset cl -> pset cl & monotone f (pset cl) (order cl)})

definition
  induced :: ['a set, ('a * 'a) set] => ('a *'a)set where
  induced A r = {(a,b). a : A & b: A & (a,b): r}


definition
  sublattice :: ('a potype * 'a set)set where
  sublattice =
      (SIGMA cl: CompleteLattice.
          {S. S ⊆ pset cl &
           (| pset = S, order = induced S (order cl) |): CompleteLattice})

abbreviation
  sublat :: ['a set, 'a potype] => bool  (- <<= - [51,50]50) where
  S <<= cl == S : sublattice `` {cl}

definition
  dual :: 'a potype => 'a potype where
  dual po = (| pset = pset po, order = converse (order po) |)

locale S =
  fixes cl :: 'a potype
    and A  :: 'a set
    and r  :: ('a * 'a) set
  defines A-def: A == pset cl
     and  r-def: r == order cl
```

locale *PO* = *S* +
  assumes *cl-po*: *cl* : *PartialOrder*

locale *CL* = *S* +
  assumes *cl-co*: *cl* : *CompleteLattice*

sublocale *CL* < *PO*
⟨*proof*⟩

locale *CLF* = *S* +
  fixes *f* :: *'a* => *'a*
    and *P* :: *'a set*
  assumes *f-cl*: (*cl*,*f*) : *CLF-set*
  defines *P-def*: *P* == *fix f A*

sublocale *CLF* < *CL*
⟨*proof*⟩

locale *Tarski* = *CLF* +
  fixes *Y* :: *'a set*
    and *intY1* :: *'a set*
    and *v* :: *'a*
  assumes
    Y-ss: *Y* ⊆ *P*
  defines
    intY1-def: *intY1* == *interval r* (*lub Y cl*) (*Top cl*)
    and *v-def*: *v* == *glb* {*x*. ((%*x*: *intY1*. *f x*) *x*, *x*): *induced intY1 r* &
                    x: *intY1*}
                (| *pset*=*intY1*, *order*=*induced intY1 r*|)

33.1 Partial Order

lemma (**in** *PO*) *dual*:
  PO (*dual cl*)
⟨*proof*⟩

lemma (**in** *PO*) *PO-imp-refl-on* [*simp*]: *refl-on A r*
⟨*proof*⟩

lemma (**in** *PO*) *PO-imp-sym* [*simp*]: *antisym r*
⟨*proof*⟩

lemma (**in** *PO*) *PO-imp-trans* [*simp*]: *trans r*
⟨*proof*⟩

lemma (**in** *PO*) *reflE*: *x* ∈ *A* ==> (*x*, *x*) ∈ *r*
⟨*proof*⟩

lemma (**in** *PO*) *antisymE*: [| (*a*, *b*) ∈ *r*; (*b*, *a*) ∈ *r* |] ==> *a* = *b*

⟨*proof*⟩

lemma (**in** *PO*) *transE*: [| (*a*, *b*) ∈ *r*; (*b*, *c*) ∈ *r*|] ==> (*a*,*c*) ∈ *r*
⟨*proof*⟩

lemma (**in** *PO*) *monotoneE*:
[| *monotone f A r*; *x* ∈ *A*; *y* ∈ *A*; (*x*, *y*) ∈ *r* |] ==> (*f x*, *f y*) ∈ *r*
⟨*proof*⟩

lemma (**in** *PO*) *po-subset-po*:
S ⊆ *A* ==> (| *pset* = *S*, *order* = *induced S r* |) ∈ *PartialOrder*
⟨*proof*⟩

lemma (**in** *PO*) *indE*: [| (*x*, *y*) ∈ *induced S r*; *S* ⊆ *A* |] ==> (*x*, *y*) ∈ *r*
⟨*proof*⟩

lemma (**in** *PO*) *indI*: [| (*x*, *y*) ∈ *r*; *x* ∈ *S*; *y* ∈ *S* |] ==> (*x*, *y*) ∈ *induced S r*
⟨*proof*⟩

lemma (**in** *CL*) *CL-imp-ex-isLub*: *S* ⊆ *A* ==> ∃ *L*. *isLub S cl L*
⟨*proof*⟩

declare (**in** *CL*) *cl-co* [*simp*]

lemma *isLub-lub*: (∃ *L*. *isLub S cl L*) = *isLub S cl* (*lub S cl*)
⟨*proof*⟩

lemma *isGlb-glb*: (∃ *G*. *isGlb S cl G*) = *isGlb S cl* (*glb S cl*)
⟨*proof*⟩

lemma *isGlb-dual-isLub*: *isGlb S cl* = *isLub S* (*dual cl*)
⟨*proof*⟩

lemma *isLub-dual-isGlb*: *isLub S cl* = *isGlb S* (*dual cl*)
⟨*proof*⟩

lemma (**in** *PO*) *dualPO*: *dual cl* ∈ *PartialOrder*
⟨*proof*⟩

lemma *Rdual*:
∀ *S*. (*S* ⊆ *A* −−>(∃ *L*. *isLub S* (| *pset* = *A*, *order* = *r*|) *L*))
==> ∀ *S*. (*S* ⊆ *A* −−> (∃ *G*. *isGlb S* (| *pset* = *A*, *order* = *r*|) *G*))
⟨*proof*⟩

lemma *lub-dual-glb*: *lub S cl* = *glb S* (*dual cl*)
⟨*proof*⟩

lemma *glb-dual-lub*: *glb S cl* = *lub S* (*dual cl*)
⟨*proof*⟩

lemma *CL-subset-PO*: $CompleteLattice \subseteq PartialOrder$
⟨*proof*⟩

lemmas *CL-imp-PO* = *CL-subset-PO* [*THEN subsetD*]

lemma (**in** *CL*) *CO-refl-on*: *refl-on A r*
⟨*proof*⟩

lemma (**in** *CL*) *CO-antisym*: *antisym r*
⟨*proof*⟩

lemma (**in** *CL*) *CO-trans*: *trans r*
⟨*proof*⟩

lemma *CompleteLatticeI*:
[| *po* ∈ *PartialOrder*; (∀ *S*. *S* ⊆ *pset po* −−> (∃ *L*. *isLub S po L*));
(∀ *S*. *S* ⊆ *pset po* −−> (∃ *G*. *isGlb S po G*))|]
==> *po* ∈ *CompleteLattice*
⟨*proof*⟩

lemma (**in** *CL*) *CL-dualCL*: *dual cl* ∈ *CompleteLattice*
⟨*proof*⟩

lemma (**in** *PO*) *dualA-iff*: *pset* (*dual cl*) = *pset cl*
⟨*proof*⟩

lemma (**in** *PO*) *dualr-iff*: ((*x*, *y*) ∈ (*order*(*dual cl*))) = ((*y*, *x*) ∈ *order cl*)
⟨*proof*⟩

lemma (**in** *PO*) *monotone-dual*:
monotone f (*pset cl*) (*order cl*)
==> *monotone f* (*pset* (*dual cl*)) (*order*(*dual cl*))
⟨*proof*⟩

lemma (**in** *PO*) *interval-dual*:
[| *x* ∈ *A*; *y* ∈ *A*|] ==> *interval r x y* = *interval* (*order*(*dual cl*)) *y x*
⟨*proof*⟩

lemma (**in** *PO*) *trans*:
(*x*, *y*) ∈ *r* ⟹ (*y*, *z*) ∈ *r* ⟹ (*x*, *z*) ∈ *r*
⟨*proof*⟩

lemma (**in** *PO*) *interval-not-empty*:
interval r a b ≠ {} ==> (*a*, *b*) ∈ *r*
⟨*proof*⟩

lemma (**in** *PO*) *interval-imp-mem*: $x \in \mathit{interval}\ r\ a\ b \Longrightarrow (a, x) \in r$
⟨*proof*⟩

lemma (**in** *PO*) *left-in-interval*:
$[\!|\ a \in A;\ b \in A;\ \mathit{interval}\ r\ a\ b \neq \{\}\ |\!] \Longrightarrow a \in \mathit{interval}\ r\ a\ b$
⟨*proof*⟩

lemma (**in** *PO*) *right-in-interval*:
$[\!|\ a \in A;\ b \in A;\ \mathit{interval}\ r\ a\ b \neq \{\}\ |\!] \Longrightarrow b \in \mathit{interval}\ r\ a\ b$
⟨*proof*⟩

33.2 sublattice

lemma (**in** *PO*) *sublattice-imp-CL*:
$S <<= cl \Longrightarrow (|\ \mathit{pset} = S,\ \mathit{order} = \mathit{induced}\ S\ r\ |) \in \mathit{CompleteLattice}$
⟨*proof*⟩

lemma (**in** *CL*) *sublatticeI*:
$[\!|\ S \subseteq A;\ (|\ \mathit{pset} = S,\ \mathit{order} = \mathit{induced}\ S\ r\ |) \in \mathit{CompleteLattice}\ |\!]$
$\Longrightarrow S <<= cl$
⟨*proof*⟩

lemma (**in** *CL*) *dual*:
$CL\ (\mathit{dual}\ cl)$
⟨*proof*⟩

33.3 lub

lemma (**in** *CL*) *lub-unique*: $[\!|\ S \subseteq A;\ \mathit{isLub}\ S\ cl\ x;\ \mathit{isLub}\ S\ cl\ L|\!] \Longrightarrow x = L$
⟨*proof*⟩

lemma (**in** *CL*) *lub-upper*: $[|S \subseteq A;\ x \in S|\!] \Longrightarrow (x,\ \mathit{lub}\ S\ cl) \in r$
⟨*proof*⟩

lemma (**in** *CL*) *lub-least*:
$[\!|\ S \subseteq A;\ L \in A;\ \forall x \in S.\ (x,L) \in r\ |\!] \Longrightarrow (\mathit{lub}\ S\ cl,\ L) \in r$
⟨*proof*⟩

lemma (**in** *CL*) *lub-in-lattice*: $S \subseteq A \Longrightarrow \mathit{lub}\ S\ cl \in A$
⟨*proof*⟩

lemma (**in** *CL*) *lubI*:
$[\!|\ S \subseteq A;\ L \in A;\ \forall x \in S.\ (x,L) \in r;$
$\forall z \in A.\ (\forall y \in S.\ (y,z) \in r) \longrightarrow (L,z) \in r\ |\!] \Longrightarrow L = \mathit{lub}\ S\ cl$
⟨*proof*⟩

lemma (**in** *CL*) *lubIa*: $[\!|\ S \subseteq A;\ \mathit{isLub}\ S\ cl\ L\ |\!] \Longrightarrow L = \mathit{lub}\ S\ cl$
⟨*proof*⟩

lemma (**in** *CL*) *isLub-in-lattice*: $\mathit{isLub}\ S\ cl\ L \Longrightarrow L \in A$

⟨*proof*⟩

lemma (**in** *CL*) *isLub-upper*: [|*isLub S cl L*; $y \in S$|] ==> $(y, L) \in r$
⟨*proof*⟩

lemma (**in** *CL*) *isLub-least*:
[| *isLub S cl L*; $z \in A$; $\forall y \in S.\ (y, z) \in r$|] ==> $(L, z) \in r$
⟨*proof*⟩

lemma (**in** *CL*) *isLubI*:
[| $L \in A$; $\forall y \in S.\ (y, L) \in r$;
$(\forall z \in A.\ (\forall y \in S.\ (y, z){:}r) \dashrightarrow (L, z) \in r)$|] ==> *isLub S cl L*
⟨*proof*⟩

33.4 glb

lemma (**in** *CL*) *glb-in-lattice*: $S \subseteq A$ ==> *glb S cl* $\in A$
⟨*proof*⟩

lemma (**in** *CL*) *glb-lower*: [|$S \subseteq A$; $x \in S$|] ==> $(glb\ S\ cl, x) \in r$
⟨*proof*⟩

Reduce the sublattice property by using substructural properties; abandoned see *Tarski-4.ML*.

lemma (**in** *CLF*) [*simp*]:
f: *pset cl* -> *pset cl* & *monotone f* (*pset cl*) (*order cl*)
⟨*proof*⟩

declare (**in** *CLF*) *f-cl* [*simp*]

lemma (**in** *CLF*) *f-in-funcset*: $f \in A$ -> A
⟨*proof*⟩

lemma (**in** *CLF*) *monotone-f*: *monotone f A r*
⟨*proof*⟩

lemma (**in** *CLF*) *CLF-dual*: $(dual\ cl, f) \in$ *CLF-set*
⟨*proof*⟩

lemma (**in** *CLF*) *dual*:
CLF (*dual cl*) *f*
⟨*proof*⟩

33.5 fixed points

lemma *fix-subset*: *fix f A* $\subseteq A$
⟨*proof*⟩

lemma *fix-imp-eq*: $x \in \mathit{fix}\ f\ A \Longrightarrow f\ x = x$
⟨*proof*⟩

lemma *fixf-subset*:
$[\!|\ A \subseteq B;\ x \in \mathit{fix}\ (\%y\colon A.\ f\ y)\ A\ |\!] \Longrightarrow x \in \mathit{fix}\ f\ B$
⟨*proof*⟩

33.6 lemmas for Tarski, lub

lemma (**in** *CLF*) *lubH-le-flubH*:
$H = \{x.\ (x, f\ x) \in r\ \&\ x \in A\} \Longrightarrow (\mathit{lub}\ H\ cl, f\ (\mathit{lub}\ H\ cl)) \in r$
⟨*proof*⟩

lemma (**in** *CLF*) *flubH-le-lubH*:
$[\!|\ H = \{x.\ (x, f\ x) \in r\ \&\ x \in A\}\ |\!] \Longrightarrow (f\ (\mathit{lub}\ H\ cl), \mathit{lub}\ H\ cl) \in r$
⟨*proof*⟩

lemma (**in** *CLF*) *lubH-is-fixp*:
$H = \{x.\ (x, f\ x) \in r\ \&\ x \in A\} \Longrightarrow \mathit{lub}\ H\ cl \in \mathit{fix}\ f\ A$
⟨*proof*⟩

lemma (**in** *CLF*) *fix-in-H*:
$[\!|\ H = \{x.\ (x, f\ x) \in r\ \&\ x \in A\};\ \ x \in P\ |\!] \Longrightarrow x \in H$
⟨*proof*⟩

lemma (**in** *CLF*) *fixf-le-lubH*:
$H = \{x.\ (x, f\ x) \in r\ \&\ x \in A\} \Longrightarrow \forall x \in \mathit{fix}\ f\ A.\ (x, \mathit{lub}\ H\ cl) \in r$
⟨*proof*⟩

lemma (**in** *CLF*) *lubH-least-fixf*:
$H = \{x.\ (x, f\ x) \in r\ \&\ x \in A\}$
$\Longrightarrow \forall L.\ (\forall y \in \mathit{fix}\ f\ A.\ (y,L) \in r) \longrightarrow (\mathit{lub}\ H\ cl,\ L) \in r$
⟨*proof*⟩

33.7 Tarski fixpoint theorem 1, first part

lemma (**in** *CLF*) *T-thm-1-lub*: $\mathit{lub}\ P\ cl = \mathit{lub}\ \{x.\ (x, f\ x) \in r\ \&\ x \in A\}\ cl$
⟨*proof*⟩

lemma (**in** *CLF*) *glbH-is-fixp*: $H = \{x.\ (f\ x, x) \in r\ \&\ x \in A\} \Longrightarrow \mathit{glb}\ H\ cl \in P$
— Tarski for glb
⟨*proof*⟩

lemma (**in** *CLF*) *T-thm-1-glb*: $\mathit{glb}\ P\ cl = \mathit{glb}\ \{x.\ (f\ x, x) \in r\ \&\ x \in A\}\ cl$
⟨*proof*⟩

33.8 interval

lemma (**in** *CLF*) *rel-imp-elem*: $(x, y) \in r \Longrightarrow x \in A$
⟨*proof*⟩

lemma (**in** *CLF*) *interval-subset*: [| $a \in A$; $b \in A$ |] ==> *interval* $r\ a\ b \subseteq A$
⟨*proof*⟩

lemma (**in** *CLF*) *intervalI*:
[| $(a, x) \in r$; $(x, b) \in r$ |] ==> $x \in$ *interval* $r\ a\ b$
⟨*proof*⟩

lemma (**in** *CLF*) *interval-lemma1*:
[| $S \subseteq$ *interval* $r\ a\ b$; $x \in S$ |] ==> $(a, x) \in r$
⟨*proof*⟩

lemma (**in** *CLF*) *interval-lemma2*:
[| $S \subseteq$ *interval* $r\ a\ b$; $x \in S$ |] ==> $(x, b) \in r$
⟨*proof*⟩

lemma (**in** *CLF*) *a-less-lub*:
[| $S \subseteq A$; $S \neq \{\}$;
$\forall x \in S.\ (a,x) \in r$; $\forall y \in S.\ (y, L) \in r$ |] ==> $(a,L) \in r$
⟨*proof*⟩

lemma (**in** *CLF*) *glb-less-b*:
[| $S \subseteq A$; $S \neq \{\}$;
$\forall x \in S.\ (x,b) \in r$; $\forall y \in S.\ (G, y) \in r$ |] ==> $(G,b) \in r$
⟨*proof*⟩

lemma (**in** *CLF*) *S-intv-cl*:
[| $a \in A$; $b \in A$; $S \subseteq$ *interval* $r\ a\ b$ |]==> $S \subseteq A$
⟨*proof*⟩

lemma (**in** *CLF*) *L-in-interval*:
[| $a \in A$; $b \in A$; $S \subseteq$ *interval* $r\ a\ b$;
$S \neq \{\}$; *isLub* $S\ cl\ L$; *interval* $r\ a\ b \neq \{\}$ |] ==> $L \in$ *interval* $r\ a\ b$
⟨*proof*⟩

lemma (**in** *CLF*) *G-in-interval*:
[| $a \in A$; $b \in A$; *interval* $r\ a\ b \neq \{\}$; $S \subseteq$ *interval* $r\ a\ b$; *isGlb* $S\ cl\ G$;
$S \neq \{\}$ |] ==> $G \in$ *interval* $r\ a\ b$
⟨*proof*⟩

lemma (**in** *CLF*) *intervalPO*:
[| $a \in A$; $b \in A$; *interval* $r\ a\ b \neq \{\}$ |]
==> (| *pset* = *interval* $r\ a\ b$, *order* = *induced* (*interval* $r\ a\ b$) r |)
$\in$ *PartialOrder*
⟨*proof*⟩

lemma (**in** *CLF*) *intv-CL-lub*:
[| $a \in A$; $b \in A$; *interval* $r\ a\ b \neq \{\}$ |]
==> $\forall S.\ S \subseteq$ *interval* $r\ a\ b$ −−>

$(\exists L.\ isLub\ S\ (|\ pset = interval\ r\ a\ b,$
$order = induced\ (interval\ r\ a\ b)\ r\ |)\ \ L)$
⟨*proof*⟩

lemmas (**in** *CLF*) *intv-CL-glb* = *intv-CL-lub* [*THEN Rdual*]

lemma (**in** *CLF*) *interval-is-sublattice*:
$[|\ a \in A;\ b \in A;\ interval\ r\ a\ b \neq \{\}\ |]$
$==>\ interval\ r\ a\ b <<=\ cl$
⟨*proof*⟩

lemmas (**in** *CLF*) *interv-is-compl-latt* =
interval-is-sublattice [*THEN sublattice-imp-CL*]

33.9 Top and Bottom

lemma (**in** *CLF*) *Top-dual-Bot*: $Top\ cl = Bot\ (dual\ cl)$
⟨*proof*⟩

lemma (**in** *CLF*) *Bot-dual-Top*: $Bot\ cl = Top\ (dual\ cl)$
⟨*proof*⟩

lemma (**in** *CLF*) *Bot-in-lattice*: $Bot\ cl \in A$
⟨*proof*⟩

lemma (**in** *CLF*) *Top-in-lattice*: $Top\ cl \in A$
⟨*proof*⟩

lemma (**in** *CLF*) *Top-prop*: $x \in A ==> (x,\ Top\ cl) \in r$
⟨*proof*⟩

lemma (**in** *CLF*) *Bot-prop*: $x \in A ==> (Bot\ cl,\ x) \in r$
⟨*proof*⟩

lemma (**in** *CLF*) *Top-intv-not-empty*: $x \in A\ \ ==>\ interval\ r\ x\ (Top\ cl) \neq \{\}$
⟨*proof*⟩

lemma (**in** *CLF*) *Bot-intv-not-empty*: $x \in A ==>\ interval\ r\ (Bot\ cl)\ x \neq \{\}$
⟨*proof*⟩

33.10 fixed points form a partial order

lemma (**in** *CLF*) *fixf-po*: $(|\ pset = P,\ order = induced\ P\ r|) \in PartialOrder$
⟨*proof*⟩

lemma (**in** *Tarski*) *Y-subset-A*: $Y \subseteq A$
⟨*proof*⟩

lemma (**in** *Tarski*) *lubY-in-A*: $lub\ Y\ cl \in A$
⟨*proof*⟩

lemma (**in** *Tarski*) *lubY-le-flubY*: $(lub\ Y\ cl,\ f\ (lub\ Y\ cl)) \in r$
⟨*proof*⟩

lemma (**in** *Tarski*) *intY1-subset*: $intY1 \subseteq A$
⟨*proof*⟩

lemmas (**in** *Tarski*) *intY1-elem* = *intY1-subset* [*THEN subsetD*]

lemma (**in** *Tarski*) *intY1-f-closed*: $x \in intY1 \Longrightarrow f\ x \in intY1$
⟨*proof*⟩

lemma (**in** *Tarski*) *intY1-func*: $(\%x{:}\ intY1.\ f\ x) \in intY1 \to intY1$
⟨*proof*⟩

lemma (**in** *Tarski*) *intY1-mono*:
$monotone\ (\%x{:}\ intY1.\ f\ x)\ intY1\ (induced\ intY1\ r)$
⟨*proof*⟩

lemma (**in** *Tarski*) *intY1-is-cl*:
$(|\ pset = intY1,\ order = induced\ intY1\ r\ |) \in CompleteLattice$
⟨*proof*⟩

lemma (**in** *Tarski*) *v-in-P*: $v \in P$
⟨*proof*⟩

lemma (**in** *Tarski*) *z-in-interval*:
$[|\ z \in P;\ \forall\, y{\in}Y.\ (y,\ z) \in induced\ P\ r\ |] \Longrightarrow z \in intY1$
⟨*proof*⟩

lemma (**in** *Tarski*) *f'z-in-int-rel*: $[|\ z \in P;\ \forall\, y{\in}Y.\ (y,\ z) \in induced\ P\ r\ |]$
$\Longrightarrow ((\%x{:}\ intY1.\ f\ x)\ z,\ z) \in induced\ intY1\ r$
⟨*proof*⟩

lemma (**in** *Tarski*) *tarski-full-lemma*:
$\exists\, L.\ isLub\ Y\ (|\ pset = P,\ order = induced\ P\ r\ |)\ L$
⟨*proof*⟩

lemma *CompleteLatticeI-simp*:
$[|\ (|\ pset = A,\ order = r\ |) \in PartialOrder;$
$\forall\, S.\ S \subseteq A \to (\exists\, L.\ isLub\ S\ (|\ pset = A,\ order = r\ |)\ \ L)\ |]$
$\Longrightarrow (|\ pset = A,\ order = r\ |) \in CompleteLattice$
⟨*proof*⟩

theorem (**in** *CLF*) *Tarski-full*:
$(|\ pset = P,\ order = induced\ P\ r|) \in CompleteLattice$
⟨*proof*⟩

end

34 Implementation of carry chain incrementor and adder

theory *Adder* **imports** *Main Word* **begin**

lemma [*simp*]: *bv-to-nat* [*b*] = *bitval b*
⟨*proof*⟩

lemma *bv-to-nat-helper'*:
bv ≠ [] ==> *bv-to-nat bv* = *bitval* (*hd bv*) ∗ *2* ^ (*length bv* − *1*) + *bv-to-nat* (*tl bv*)
⟨*proof*⟩

definition
half-adder :: [*bit*, *bit*] => *bit list* **where**
half-adder a b = [*a bitand b*, *a bitxor b*]

lemma *half-adder-correct*: *bv-to-nat* (*half-adder a b*) = *bitval a* + *bitval b*
⟨*proof*⟩

lemma [*simp*]: *length* (*half-adder a b*) = *2*
⟨*proof*⟩

definition
full-adder :: [*bit*, *bit*, *bit*] => *bit list* **where**
full-adder a b c =
(*let x* = *a bitxor b in* [*a bitand b bitor c bitand x*, *x bitxor c*])

lemma *full-adder-correct*:
bv-to-nat (*full-adder a b c*) = *bitval a* + *bitval b* + *bitval c*
⟨*proof*⟩

lemma [*simp*]: *length* (*full-adder a b c*) = *2*
⟨*proof*⟩

34.1 Carry chain incrementor

consts
carry-chain-inc :: [*bit list*, *bit*] => *bit list*
primrec
carry-chain-inc [] *c* = [*c*]
carry-chain-inc (*a*#*as*) *c* =
(*let chain* = *carry-chain-inc as c*
in half-adder a (*hd chain*) @ *tl chain*)

lemma *cci-nonnull*: *carry-chain-inc as c* ≠ []

⟨*proof*⟩

lemma *cci-length* [*simp*]: *length* (*carry-chain-inc as c*) = *length as* + *1*
⟨*proof*⟩

lemma *cci-correct*: *bv-to-nat* (*carry-chain-inc as c*) = *bv-to-nat as* + *bitval c*
⟨*proof*⟩

consts
carry-chain-adder :: [*bit list*, *bit list*, *bit*] => *bit list*
primrec
carry-chain-adder [] *bs c* = [*c*]
carry-chain-adder (*a* # *as*) *bs c* =
(*let chain* = *carry-chain-adder as* (*tl bs*) *c*
in full-adder a (*hd bs*) (*hd chain*) @ *tl chain*)

lemma *cca-nonnull*: *carry-chain-adder as bs c* ≠ []
⟨*proof*⟩

lemma *cca-length*: *length as* = *length bs* ⟹
length (*carry-chain-adder as bs c*) = *Suc* (*length bs*)
⟨*proof*⟩

theorem *cca-correct*:
length as = *length bs* ⟹
bv-to-nat (*carry-chain-adder as bs c*) =
bv-to-nat as + *bv-to-nat bs* + *bitval c*
⟨*proof*⟩

end

35 Classical Predicate Calculus Problems

theory *Classical* **imports** *Main* **begin**

35.1 Traditional Classical Reasoner

The machine "griffon" mentioned below is a 2.5GHz Power Mac G5.

Taken from *FOL/Classical.thy*. When porting examples from first-order logic, beware of the precedence of = versus ↔.

lemma (*P* −−> *Q* | *R*) −−> (*P*−−>*Q*) | (*P*−−>*R*)
⟨*proof*⟩

If and only if

lemma (*P*=*Q*) = (*Q* = (*P*::*bool*))
⟨*proof*⟩

lemma $\sim (P = (\sim P))$
⟨*proof*⟩

Sample problems from F. J. Pelletier, Seventy-Five Problems for Testing Automatic Theorem Provers, J. Automated Reasoning 2 (1986), 191-216. Errata, JAR 4 (1988), 236-236.

The hardest problems – judging by experience with several theorem provers, including matrix ones – are 34 and 43.

35.1.1 Pelletier's examples

1

lemma $(P --> Q) \;=\; (\sim Q \;--> \sim P)$
⟨*proof*⟩

2

lemma $(\sim \sim P) = \; P$
⟨*proof*⟩

3

lemma $\sim (P --> Q) \;--> (Q --> P)$
⟨*proof*⟩

4

lemma $(\sim P --> Q) \;=\; (\sim Q \;--> P)$
⟨*proof*⟩

5

lemma $((P|Q) --> (P|R)) \;--> (P|(Q --> R))$
⟨*proof*⟩

6

lemma $P \mid \sim P$
⟨*proof*⟩

7

lemma $P \mid \sim \sim \sim P$
⟨*proof*⟩

8. Peirce's law

lemma $((P --> Q) \;--> P) \;\;--> \;\; P$
⟨*proof*⟩

9

lemma $((P|Q) \;\&\; (\sim P|Q) \;\&\; (P| \sim Q)) \;--> \sim (\sim P \mid \sim Q)$

⟨*proof*⟩

10

lemma $(Q-->R)$ & $(R-->P\&Q)$ & $(P-->Q|R)$ --> $(P=Q)$
⟨*proof*⟩

11. Proved in each direction (incorrectly, says Pelletier!!)

lemma $P=(P::bool)$
⟨*proof*⟩

12. "Dijkstra's law"

lemma $((P = Q) = R) = (P = (Q = R))$
⟨*proof*⟩

13. Distributive law

lemma $(P \mid (Q \ \& \ R)) = ((P \mid Q) \ \& \ (P \mid R))$
⟨*proof*⟩

14

lemma $(P = Q) = ((Q \mid \sim P) \ \& \ (\sim Q|P))$
⟨*proof*⟩

15

lemma $(P \ --> \ Q) = (\sim P \mid Q)$
⟨*proof*⟩

16

lemma $(P-->Q) \mid (Q-->P)$
⟨*proof*⟩

17

lemma $((P \ \& \ (Q-->R))-->S) \ = \ ((\sim P \mid Q \mid S) \ \& \ (\sim P \mid \sim R \mid S))$
⟨*proof*⟩

35.1.2 Classical Logic: examples with quantifiers

lemma $(\forall x. \ P(x) \ \& \ Q(x)) = ((\forall x. \ P(x)) \ \& \ (\forall x. \ Q(x)))$
⟨*proof*⟩

lemma $(\exists x. \ P-->Q(x)) \ = \ (P \ --> \ (\exists x. \ Q(x)))$
⟨*proof*⟩

lemma $(\exists x. \ P(x)-->Q) = ((\forall x. \ P(x)) \ --> \ Q)$
⟨*proof*⟩

lemma $((\forall x. \ P(x)) \mid Q) \ = \ (\forall x. \ P(x) \mid Q)$
⟨*proof*⟩

From Wishnu Prasetya

lemma $(\forall s.\ q(s) \longrightarrow r(s)) \ \&\ \sim r(s) \ \&\ (\forall s.\ \sim r(s) \ \&\ \sim q(s) \longrightarrow p(t) \mid q(t)) \longrightarrow p(t) \mid r(t)$
⟨*proof*⟩

35.1.3 Problems requiring quantifier duplication

Theorem B of Peter Andrews, Theorem Proving via General Matings, JACM 28 (1981).

lemma $(\exists x.\ \forall y.\ P(x) = P(y)) \longrightarrow ((\exists x.\ P(x)) = (\forall y.\ P(y)))$
⟨*proof*⟩

Needs multiple instantiation of the quantifier.

lemma $(\forall x.\ P(x) \longrightarrow P(f(x))) \ \ \&\ \ P(d) \longrightarrow P(f(f(f(d))))$
⟨*proof*⟩

Needs double instantiation of the quantifier

lemma $\exists x.\ P(x) \longrightarrow P(a) \ \&\ P(b)$
⟨*proof*⟩

lemma $\exists z.\ P(z) \longrightarrow (\forall x.\ P(x))$
⟨*proof*⟩

lemma $\exists x.\ (\exists y.\ P(y)) \longrightarrow P(x)$
⟨*proof*⟩

35.1.4 Hard examples with quantifiers

Problem 18

lemma $\exists y.\ \forall x.\ P(y) \longrightarrow P(x)$
⟨*proof*⟩

Problem 19

lemma $\exists x.\ \forall y\ z.\ (P(y) \longrightarrow Q(z)) \longrightarrow (P(x) \longrightarrow Q(x))$
⟨*proof*⟩

Problem 20

lemma $(\forall x\ y.\ \exists z.\ \forall w.\ (P(x) \& Q(y) \longrightarrow R(z) \& S(w))) \longrightarrow (\exists x\ y.\ P(x) \ \&\ Q(y)) \longrightarrow (\exists z.\ R(z))$
⟨*proof*⟩

Problem 21

lemma $(\exists x.\ P \longrightarrow Q(x)) \ \&\ (\exists x.\ Q(x) \longrightarrow P) \longrightarrow (\exists x.\ P = Q(x))$
⟨*proof*⟩

Problem 22

lemma $(\forall x.\ P = Q(x)) \ \text{--->}\ (P = (\forall x.\ Q(x)))$
⟨*proof*⟩

Problem 23

lemma $(\forall x.\ P \mid Q(x)) \ = \ (P \mid (\forall x.\ Q(x)))$
⟨*proof*⟩

Problem 24

lemma $\sim(\exists x.\ S(x)\&Q(x)) \ \&\ (\forall x.\ P(x) \text{--->}\ Q(x)|R(x)) \ \&$
$\quad (\sim(\exists x.\ P(x)) \text{--->}\ (\exists x.\ Q(x))) \ \&\ (\forall x.\ Q(x)|R(x) \text{--->}\ S(x))$
$\quad \text{--->}\ (\exists x.\ P(x)\&R(x))$
⟨*proof*⟩

Problem 25

lemma $(\exists x.\ P(x)) \ \&$
$\quad (\forall x.\ L(x) \text{--->} \sim (M(x) \ \&\ R(x))) \ \&$
$\quad (\forall x.\ P(x) \text{--->}\ (M(x) \ \&\ L(x))) \ \&$
$\quad ((\forall x.\ P(x)\text{--->}Q(x)) \mid (\exists x.\ P(x)\&R(x)))$
$\text{--->}\ (\exists x.\ Q(x)\&P(x))$
⟨*proof*⟩

Problem 26

lemma $((\exists x.\ p(x)) = (\exists x.\ q(x))) \ \&$
$\quad (\forall x.\ \forall y.\ p(x) \ \&\ q(y) \text{--->}\ (r(x) = s(y)))$
$\text{--->}\ ((\forall x.\ p(x)\text{--->}r(x)) = (\forall x.\ q(x)\text{--->}s(x)))$
⟨*proof*⟩

Problem 27

lemma $(\exists x.\ P(x) \ \&\ \sim Q(x)) \ \&$
$\quad (\forall x.\ P(x) \text{--->}\ R(x)) \ \&$
$\quad (\forall x.\ M(x) \ \&\ L(x) \text{--->}\ P(x)) \ \&$
$\quad ((\exists x.\ R(x) \ \&\ \sim\ Q(x)) \text{--->}\ (\forall x.\ L(x) \text{--->} \sim\ R(x)))$
$\text{--->}\ (\forall x.\ M(x) \text{--->} \sim L(x))$
⟨*proof*⟩

Problem 28. AMENDED

lemma $(\forall x.\ P(x) \text{--->}\ (\forall x.\ Q(x))) \ \&$
$\quad ((\forall x.\ Q(x)|R(x)) \text{--->}\ (\exists x.\ Q(x)\&S(x))) \ \&$
$\quad ((\exists x.\ S(x)) \text{--->}\ (\forall x.\ L(x) \text{--->}\ M(x)))$
$\text{--->}\ (\forall x.\ P(x) \ \&\ L(x) \text{--->}\ M(x))$
⟨*proof*⟩

Problem 29. Essentially the same as Principia Mathematica *11.71

lemma $(\exists x.\ F(x)) \ \&\ (\exists y.\ G(y))$
$\text{--->}\ (\ ((\forall x.\ F(x)\text{--->}H(x)) \ \&\ (\forall y.\ G(y)\text{--->}J(y))) \ =$
$\quad (\forall x\ y.\ F(x) \ \&\ G(y) \text{--->}\ H(x) \ \&\ J(y)))$
⟨*proof*⟩

Problem 30

lemma $(\forall x.\ P(x) \mid Q(x) \dashrightarrow \sim R(x)) \ \&$
$\quad (\forall x.\ (Q(x) \dashrightarrow \sim S(x)) \dashrightarrow P(x) \ \& \ R(x))$
$\quad \dashrightarrow (\forall x.\ S(x))$
⟨*proof*⟩

Problem 31

lemma $\sim(\exists x.\ P(x) \ \& \ (Q(x) \mid R(x))) \ \&$
$\quad (\exists x.\ L(x) \ \& \ P(x)) \ \&$
$\quad (\forall x.\ \sim R(x) \dashrightarrow M(x))$
$\quad \dashrightarrow (\exists x.\ L(x) \ \& \ M(x))$
⟨*proof*⟩

Problem 32

lemma $(\forall x.\ P(x) \ \& \ (Q(x)|R(x)) \dashrightarrow S(x)) \ \&$
$\quad (\forall x.\ S(x) \ \& \ R(x) \dashrightarrow L(x)) \ \&$
$\quad (\forall x.\ M(x) \dashrightarrow R(x))$
$\quad \dashrightarrow (\forall x.\ P(x) \ \& \ M(x) \dashrightarrow L(x))$
⟨*proof*⟩

Problem 33

lemma $(\forall x.\ P(a) \ \& \ (P(x) \dashrightarrow P(b)) \dashrightarrow P(c)) \ =$
$\quad (\forall x.\ (\sim P(a) \mid P(x) \mid P(c)) \ \& \ (\sim P(a) \mid \sim P(b) \mid P(c)))$
⟨*proof*⟩

Problem 34 AMENDED (TWICE!!)

Andrews's challenge

lemma $((\exists x.\ \forall y.\ p(x) = p(y)) \ =$
$\quad ((\exists x.\ q(x)) = (\forall y.\ p(y)))) \ =$
$\quad ((\exists x.\ \forall y.\ q(x) = q(y)) \ =$
$\quad ((\exists x.\ p(x)) = (\forall y.\ q(y))))$
⟨*proof*⟩

Problem 35

lemma $\exists x\ y.\ P\ x\ y \dashrightarrow \ (\forall u\ v.\ P\ u\ v)$
⟨*proof*⟩

Problem 36

lemma $(\forall x.\ \exists y.\ J\ x\ y) \ \&$
$\quad (\forall x.\ \exists y.\ G\ x\ y) \ \&$
$\quad (\forall x\ y.\ J\ x\ y \mid G\ x\ y \dashrightarrow$
$\quad (\forall z.\ J\ y\ z \mid G\ y\ z \dashrightarrow H\ x\ z))$
$\quad \dashrightarrow (\forall x.\ \exists y.\ H\ x\ y)$
⟨*proof*⟩

Problem 37

lemma $(\forall z.\ \exists w.\ \forall x.\ \exists y.$
$\quad (P\ x\ z\ \text{--->} P\ y\ w)\ \&\ P\ y\ z\ \&\ (P\ y\ w\ \text{--->}\ (\exists u.\ Q\ u\ w)))\ \&$
$\quad (\forall x\ z.\ {\sim}(P\ x\ z)\ \text{--->}\ (\exists y.\ Q\ y\ z))\ \&$
$\quad ((\exists x\ y.\ Q\ x\ y)\ \text{--->}\ (\forall x.\ R\ x\ x))$
$\text{--->}\ (\forall x.\ \exists y.\ R\ x\ y)$
⟨*proof*⟩

Problem 38

lemma $(\forall x.\ p(a)\ \&\ (p(x)\ \text{--->}\ (\exists y.\ p(y)\ \&\ r\ x\ y))\ \text{--->}$
$\quad (\exists z.\ \exists w.\ p(z)\ \&\ r\ x\ w\ \&\ r\ w\ z))\ =$
$\quad (\forall x.\ ({\sim}p(a)\ |\ p(x)\ |\ (\exists z.\ \exists w.\ p(z)\ \&\ r\ x\ w\ \&\ r\ w\ z))\ \&$
$\quad ({\sim}p(a)\ |\ {\sim}(\exists y.\ p(y)\ \&\ r\ x\ y)\ |$
$\quad (\exists z.\ \exists w.\ p(z)\ \&\ r\ x\ w\ \&\ r\ w\ z)))$
⟨*proof*⟩

Problem 39

lemma $\sim\ (\exists x.\ \forall y.\ F\ y\ x = (\sim\ F\ y\ y))$
⟨*proof*⟩

Problem 40. AMENDED

lemma $(\exists y.\ \forall x.\ F\ x\ y = F\ x\ x)$
$\quad \text{--->}\ \sim\ (\forall x.\ \exists y.\ \forall z.\ F\ z\ y = (\sim\ F\ z\ x))$
⟨*proof*⟩

Problem 41

lemma $(\forall z.\ \exists y.\ \forall x.\ f\ x\ y = (f\ x\ z\ \&\ \sim\ f\ x\ x))$
$\quad \text{--->}\ \sim\ (\exists z.\ \forall x.\ f\ x\ z)$
⟨*proof*⟩

Problem 42

lemma $\sim\ (\exists y.\ \forall x.\ p\ x\ y = (\sim\ (\exists z.\ p\ x\ z\ \&\ p\ z\ x)))$
⟨*proof*⟩

Problem 43!!

lemma $(\forall x{::}'a.\ \forall y{::}'a.\ q\ x\ y = (\forall z.\ p\ z\ x = (p\ z\ y{::}bool)))$
$\quad \text{--->}\ (\forall x.\ (\forall y.\ q\ x\ y = (q\ y\ x{::}bool)))$
⟨*proof*⟩

Problem 44

lemma $(\forall x.\ f(x)\ \text{--->}$
$\quad (\exists y.\ g(y)\ \&\ h\ x\ y\ \&\ (\exists y.\ g(y)\ \&\ \sim\ h\ x\ y)))\ \&$
$\quad (\exists x.\ j(x)\ \&\ (\forall y.\ g(y)\ \text{--->}\ h\ x\ y))$
$\quad \text{--->}\ (\exists x.\ j(x)\ \&\ {\sim}f(x))$
⟨*proof*⟩

Problem 45

lemma $(\forall x.\ f(x)\ \&\ (\forall y.\ g(y)\ \&\ h\ x\ y\ \text{--->}\ j\ x\ y)$

$$\longrightarrow (\forall y.\ g(y)\ \&\ h\ x\ y \longrightarrow k(y)))\ \&$$
$$\sim (\exists y.\ l(y)\ \&\ k(y))\ \&$$
$$(\exists x.\ f(x)\ \&\ (\forall y.\ h\ x\ y \longrightarrow l(y))$$
$$\&\ (\forall y.\ g(y)\ \&\ h\ x\ y \longrightarrow j\ x\ y))$$
$$\longrightarrow (\exists x.\ f(x)\ \&\ \sim (\exists y.\ g(y)\ \&\ h\ x\ y))$$

⟨*proof*⟩

35.1.5 Problems (mainly) involving equality or functions

Problem 48

lemma $(a{=}b \mid c{=}d)\ \&\ (a{=}c \mid b{=}d) \longrightarrow a{=}d \mid b{=}c$
⟨*proof*⟩

Problem 49 NOT PROVED AUTOMATICALLY. Hard because it involves substitution for Vars the type constraint ensures that x,y,z have the same type as a,b,u.

lemma $(\exists x\ y{::}'a.\ \forall z.\ z{=}x \mid z{=}y)\ \&\ P(a)\ \&\ P(b)\ \&\ (\sim a{=}b)$
$$\longrightarrow (\forall u{::}'a.\ P(u))$$
⟨*proof*⟩

Problem 50. (What has this to do with equality?)

lemma $(\forall x.\ P\ a\ x \mid (\forall y.\ P\ x\ y)) \longrightarrow (\exists x.\ \forall y.\ P\ x\ y)$
⟨*proof*⟩

Problem 51

lemma $(\exists z\ w.\ \forall x\ y.\ P\ x\ y = (x{=}z\ \&\ y{=}w)) \longrightarrow$
$(\exists z.\ \forall x.\ \exists w.\ (\forall y.\ P\ x\ y = (y{=}w)) = (x{=}z))$
⟨*proof*⟩

Problem 52. Almost the same as 51.

lemma $(\exists z\ w.\ \forall x\ y.\ P\ x\ y = (x{=}z\ \&\ y{=}w)) \longrightarrow$
$(\exists w.\ \forall y.\ \exists z.\ (\forall x.\ P\ x\ y = (x{=}z)) = (y{=}w))$
⟨*proof*⟩

Problem 55

Non-equational version, from Manthey and Bry, CADE-9 (Springer, 1988). fast DISCOVERS who killed Agatha.

lemma $lives(agatha)\ \&\ lives(butler)\ \&\ lives(charles)\ \&$
$(killed\ agatha\ agatha \mid killed\ butler\ agatha \mid killed\ charles\ agatha)\ \&$
$(\forall x\ y.\ killed\ x\ y \longrightarrow hates\ x\ y\ \&\ \sim richer\ x\ y)\ \&$
$(\forall x.\ hates\ agatha\ x \longrightarrow \sim hates\ charles\ x)\ \&$
$(hates\ agatha\ agatha\ \&\ hates\ agatha\ charles)\ \&$
$(\forall x.\ lives(x)\ \&\ \sim richer\ x\ agatha \longrightarrow hates\ butler\ x)\ \&$
$(\forall x.\ hates\ agatha\ x \longrightarrow hates\ butler\ x)\ \&$
$(\forall x.\ \sim hates\ x\ agatha \mid \sim hates\ x\ butler \mid \sim hates\ x\ charles) \longrightarrow$
$killed\ ?who\ agatha$

⟨*proof*⟩

Problem 56

lemma $(\forall x.\ (\exists y.\ P(y)\ \&\ x{=}f(y)) \longrightarrow P(x)) = (\forall x.\ P(x) \longrightarrow P(f(x)))$
⟨*proof*⟩

Problem 57

lemma $P\ (f\ a\ b)\ (f\ b\ c)\ \&\ P\ (f\ b\ c)\ (f\ a\ c)\ \&$
$\quad (\forall x\ y\ z.\ P\ x\ y\ \&\ P\ y\ z \longrightarrow P\ x\ z) \quad \longrightarrow \quad P\ (f\ a\ b)\ (f\ a\ c)$
⟨*proof*⟩

Problem 58 NOT PROVED AUTOMATICALLY

lemma $(\forall x\ y.\ f(x){=}g(y)) \longrightarrow (\forall x\ y.\ f(f(x)){=}f(g(y)))$
⟨*proof*⟩

Problem 59

lemma $(\forall x.\ P(x) = (\sim P(f(x)))) \longrightarrow (\exists x.\ P(x)\ \&\ \sim P(f(x)))$
⟨*proof*⟩

Problem 60

lemma $\forall x.\ P\ x\ (f\ x) = (\exists y.\ (\forall z.\ P\ z\ y \longrightarrow P\ z\ (f\ x))\ \&\ P\ x\ y)$
⟨*proof*⟩

Problem 62 as corrected in JAR 18 (1997), page 135

lemma $(\forall x.\ p\ a\ \&\ (p\ x \longrightarrow p(f\ x)) \longrightarrow p(f(f\ x))) =$
$\quad (\forall x.\ (\sim\ p\ a \mid p\ x \mid p(f(f\ x)))\ \&$
$\qquad (\sim\ p\ a \mid \sim\ p(f\ x) \mid p(f(f\ x))))$
⟨*proof*⟩

From Davis, Obvious Logical Inferences, IJCAI-81, 530-531 fast indeed copes!

lemma $(\forall x.\ F(x)\ \&\ \sim G(x) \longrightarrow (\exists y.\ H(x,y)\ \&\ J(y)))\ \&$
$\quad (\exists x.\ K(x)\ \&\ F(x)\ \&\ (\forall y.\ H(x,y) \longrightarrow K(y)))\ \&$
$\quad (\forall x.\ K(x) \longrightarrow \sim G(x)) \longrightarrow (\exists x.\ K(x)\ \&\ J(x))$
⟨*proof*⟩

From Rudnicki, Obvious Inferences, JAR 3 (1987), 383-393. It does seem obvious!

lemma $(\forall x.\ F(x)\ \&\ \sim G(x) \longrightarrow (\exists y.\ H(x,y)\ \&\ J(y)))\ \&$
$\quad (\exists x.\ K(x)\ \&\ F(x)\ \&\ (\forall y.\ H(x,y) \longrightarrow K(y)))\ \&$
$\quad (\forall x.\ K(x) \longrightarrow \sim G(x)) \quad \longrightarrow \quad (\exists x.\ K(x) \longrightarrow \sim G(x))$
⟨*proof*⟩

Attributed to Lewis Carroll by S. G. Pulman. The first or last assumption can be deleted.

lemma $(\forall x.\ \mathit{honest}(x)\ \&\ \mathit{industrious}(x) \longrightarrow \mathit{healthy}(x))\ \&$

$\sim$ ($\exists\, x.\ grocer(x)$ & $healthy(x)$) &
($\forall\, x.\ industrious(x)$ & $grocer(x)$ $-->$ $honest(x)$) &
($\forall\, x.\ cyclist(x)$ $-->$ $industrious(x)$) &
($\forall\, x.\ \sim healthy(x)$ & $cyclist(x)$ $-->$ $\sim honest(x)$)
$-->$ ($\forall\, x.\ grocer(x)$ $-->$ $\sim cyclist(x)$)
⟨*proof*⟩

lemma ($\forall\, x\ y.\ R(x,y)$ | $R(y,x)$) &
($\forall\, x\ y.\ S(x,y)$ & $S(y,x)$ $-->$ $x=y$) &
($\forall\, x\ y.\ R(x,y)$ $-->$ $S(x,y)$) $-->$ ($\forall\, x\ y.\ S(x,y)$ $-->$ $R(x,y)$)
⟨*proof*⟩

35.2 Model Elimination Prover

Trying out meson with arguments

lemma $x < y$ & $y < z$ $-->$ $\sim$ ($z < (x::nat)$)
⟨*proof*⟩

The "small example" from Bezem, Hendriks and de Nivelle, Automatic Proof Construction in Type Theory Using Resolution, JAR 29: 3-4 (2002), pages 253-275

lemma ($\forall\, x\ y\ z.\ R(x,y)$ & $R(y,z)$ $-->$ $R(x,z)$) &
($\forall\, x.\ \exists\, y.\ R(x,y)$) $-->$
$\sim$ ($\forall\, x.\ P\ x = (\forall\, y.\ R(x,y)$ $-->$ $\sim$ $P\ y)$)
⟨*proof*⟩

35.2.1 Pelletier's examples

1

lemma (P $-->$ Q) = ($\sim Q$ $-->$ $\sim P$)
⟨*proof*⟩

2

lemma ($\sim$ $\sim$ P) = P
⟨*proof*⟩

3

lemma $\sim(P-->Q)$ $-->$ $(Q-->P)$
⟨*proof*⟩

4

lemma ($\sim P-->Q$) = ($\sim Q$ $-->$ P)
⟨*proof*⟩

5

lemma $((P|Q)-->(P|R))$ $-->$ $(P|(Q-->R))$

⟨*proof*⟩

6

lemma $P \mid \sim P$
⟨*proof*⟩

7

lemma $P \mid \sim \sim \sim P$
⟨*proof*⟩

8. Peirce's law

lemma $((P-->Q) \dashrightarrow P) \dashrightarrow P$
⟨*proof*⟩

9

lemma $((P|Q) \,\&\, (\sim P|Q) \,\&\, (P|\sim Q)) \dashrightarrow \sim (\sim P \mid \sim Q)$
⟨*proof*⟩

10

lemma $(Q-->R) \,\&\, (R-->P\&Q) \,\&\, (P-->Q|R) \dashrightarrow (P=Q)$
⟨*proof*⟩

11. Proved in each direction (incorrectly, says Pelletier!!)

lemma $P=(P::bool)$
⟨*proof*⟩

12. "Dijkstra's law"

lemma $((P = Q) = R) = (P = (Q = R))$
⟨*proof*⟩

13. Distributive law

lemma $(P \mid (Q \,\&\, R)) = ((P \mid Q) \,\&\, (P \mid R))$
⟨*proof*⟩

14

lemma $(P = Q) = ((Q \mid \sim P) \,\&\, (\sim Q|P))$
⟨*proof*⟩

15

lemma $(P \dashrightarrow Q) = (\sim P \mid Q)$
⟨*proof*⟩

16

lemma $(P-->Q) \mid (Q-->P)$
⟨*proof*⟩

17

lemma $((P \,\&\, (Q-->R))-->S) \;=\; ((\sim P \mid Q \mid S) \,\&\, (\sim P \mid \sim R \mid S))$
⟨*proof*⟩

35.2.2 Classical Logic: examples with quantifiers

lemma $(\forall x.\ P\ x\ \&\ Q\ x) = ((\forall x.\ P\ x)\ \&\ (\forall x.\ Q\ x))$
⟨*proof*⟩

lemma $(\exists x.\ P\ \longrightarrow\ Q\ x)\ =\ (P\ \longrightarrow\ (\exists x.\ Q\ x))$
⟨*proof*⟩

lemma $(\exists x.\ P\ x\ \longrightarrow\ Q) = ((\forall x.\ P\ x)\ \longrightarrow\ Q)$
⟨*proof*⟩

lemma $((\forall x.\ P\ x)\ |\ Q)\ =\ (\forall x.\ P\ x\ |\ Q)$
⟨*proof*⟩

lemma $(\forall x.\ P\ x\ \longrightarrow\ P(f\ x))\ \ \&\ \ P\ d\ \longrightarrow\ P(f(f(f\ d)))$
⟨*proof*⟩

Needs double instantiation of EXISTS

lemma $\exists x.\ P\ x\ \longrightarrow\ P\ a\ \&\ P\ b$
⟨*proof*⟩

lemma $\exists z.\ P\ z\ \longrightarrow\ (\forall x.\ P\ x)$
⟨*proof*⟩

From a paper by Claire Quigley

lemma $\exists y.\ ((P\ c\ \&\ Q\ y)\ |\ (\exists z.\ \sim\ Q\ z))\ |\ (\exists x.\ \sim\ P\ x\ \&\ Q\ d)$
⟨*proof*⟩

35.2.3 Hard examples with quantifiers

Problem 18

lemma $\exists y.\ \forall x.\ P\ y\ \longrightarrow\ P\ x$
⟨*proof*⟩

Problem 19

lemma $\exists x.\ \forall y\ z.\ (P\ y\ \longrightarrow\ Q\ z)\ \longrightarrow\ (P\ x\ \longrightarrow\ Q\ x)$
⟨*proof*⟩

Problem 20

lemma $(\forall x\ y.\ \exists z.\ \forall w.\ (P\ x\ \&\ Q\ y\ \longrightarrow\ R\ z\ \&\ S\ w))$
$\quad\longrightarrow\ (\exists x\ y.\ P\ x\ \&\ Q\ y)\ \longrightarrow\ (\exists z.\ R\ z)$
⟨*proof*⟩

Problem 21

lemma $(\exists x.\ P\ \longrightarrow\ Q\ x)\ \&\ (\exists x.\ Q\ x\ \longrightarrow\ P)\ \longrightarrow\ (\exists x.\ P{=}Q\ x)$
⟨*proof*⟩

Problem 22

lemma $(\forall x.\ P = Q\ x)\ \ --> \ \ (P = (\forall x.\ Q\ x))$
⟨*proof*⟩

Problem 23

lemma $(\forall x.\ P \mid Q\ x)\ \ =\ \ (P \mid (\forall x.\ Q\ x))$
⟨*proof*⟩

Problem 24

lemma $\sim(\exists x.\ S\ x\ \&\ Q\ x)\ \&\ (\forall x.\ P\ x\ --> Q\ x \mid R\ x)\ \&$
$\quad (\sim(\exists x.\ P\ x)\ --> (\exists x.\ Q\ x))\ \&\ (\forall x.\ Q\ x \mid R\ x\ --> S\ x)$
$\quad --> (\exists x.\ P\ x\ \&\ R\ x)$
⟨*proof*⟩

Problem 25

lemma $(\exists x.\ P\ x)\ \&$
$\quad (\forall x.\ L\ x\ --> \sim (M\ x\ \&\ R\ x))\ \&$
$\quad (\forall x.\ P\ x\ --> (M\ x\ \&\ L\ x))\ \&$
$\quad ((\forall x.\ P\ x\ --> Q\ x) \mid (\exists x.\ P\ x\ \&\ R\ x))$
$\quad --> (\exists x.\ Q\ x\ \&\ P\ x)$
⟨*proof*⟩

Problem 26; has 24 Horn clauses

lemma $((\exists x.\ p\ x) = (\exists x.\ q\ x))\ \&$
$\quad (\forall x.\ \forall y.\ p\ x\ \&\ q\ y\ --> (r\ x = s\ y))$
$\ --> ((\forall x.\ p\ x\ --> r\ x) = (\forall x.\ q\ x\ --> s\ x))$
⟨*proof*⟩

Problem 27; has 13 Horn clauses

lemma $(\exists x.\ P\ x\ \&\ \sim Q\ x)\ \&$
$\quad (\forall x.\ P\ x\ --> R\ x)\ \&$
$\quad (\forall x.\ M\ x\ \&\ L\ x\ --> P\ x)\ \&$
$\quad ((\exists x.\ R\ x\ \&\ \sim Q\ x)\ --> (\forall x.\ L\ x\ --> \sim R\ x))$
$\quad --> (\forall x.\ M\ x\ --> \sim L\ x)$
⟨*proof*⟩

Problem 28. AMENDED; has 14 Horn clauses

lemma $(\forall x.\ P\ x\ --> (\forall x.\ Q\ x))\ \&$
$\quad ((\forall x.\ Q\ x \mid R\ x)\ --> (\exists x.\ Q\ x\ \&\ S\ x))\ \&$
$\quad ((\exists x.\ S\ x)\ --> (\forall x.\ L\ x\ --> M\ x))$
$\quad --> (\forall x.\ P\ x\ \&\ L\ x\ --> M\ x)$
⟨*proof*⟩

Problem 29. Essentially the same as Principia Mathematica *11.71. 62 Horn clauses

lemma $(\exists x.\ F\ x)\ \&\ (\exists y.\ G\ y)$
$\quad --> (\ ((\forall x.\ F\ x\ --> H\ x)\ \&\ (\forall y.\ G\ y\ --> J\ y))\ \ =$
$\qquad (\forall x\ y.\ F\ x\ \&\ G\ y\ --> H\ x\ \&\ J\ y))$

⟨*proof*⟩

Problem 30

lemma $(\forall x.\ P\ x \mid Q\ x \longrightarrow {\sim} R\ x)\ \&\ (\forall x.\ (Q\ x \longrightarrow {\sim} S\ x) \longrightarrow P\ x\ \&\ R\ x)$
$\longrightarrow (\forall x.\ S\ x)$
⟨*proof*⟩

Problem 31; has 10 Horn clauses; first negative clauses is useless

lemma ${\sim}(\exists x.\ P\ x\ \&\ (Q\ x \mid R\ x))\ \&$
$(\exists x.\ L\ x\ \&\ P\ x)\ \&$
$(\forall x.\ {\sim} R\ x \longrightarrow M\ x)$
$\longrightarrow (\exists x.\ L\ x\ \&\ M\ x)$
⟨*proof*⟩

Problem 32

lemma $(\forall x.\ P\ x\ \&\ (Q\ x \mid R\ x) \longrightarrow S\ x)\ \&$
$(\forall x.\ S\ x\ \&\ R\ x \longrightarrow L\ x)\ \&$
$(\forall x.\ M\ x \longrightarrow R\ x)$
$\longrightarrow (\forall x.\ P\ x\ \&\ M\ x \longrightarrow L\ x)$
⟨*proof*⟩

Problem 33; has 55 Horn clauses

lemma $(\forall x.\ P\ a\ \&\ (P\ x \longrightarrow P\ b) \longrightarrow P\ c)\ =$
$(\forall x.\ ({\sim}P\ a \mid P\ x \mid P\ c)\ \&\ ({\sim}P\ a \mid {\sim}P\ b \mid P\ c))$
⟨*proof*⟩

Problem 34: Andrews's challenge has 924 Horn clauses

lemma $((\exists x.\ \forall y.\ p\ x = p\ y)\ = ((\exists x.\ q\ x) = (\forall y.\ p\ y)))\ \ =$
$((\exists x.\ \forall y.\ q\ x = q\ y)\ = ((\exists x.\ p\ x) = (\forall y.\ q\ y)))$
⟨*proof*⟩

Problem 35

lemma $\exists x\ y.\ P\ x\ y \longrightarrow\ (\forall u\ v.\ P\ u\ v)$
⟨*proof*⟩

Problem 36; has 15 Horn clauses

lemma $(\forall x.\ \exists y.\ J\ x\ y)\ \&\ (\forall x.\ \exists y.\ G\ x\ y)\ \&$
$(\forall x\ y.\ J\ x\ y \mid G\ x\ y \longrightarrow (\forall z.\ J\ y\ z \mid G\ y\ z \longrightarrow H\ x\ z))$
$\longrightarrow (\forall x.\ \exists y.\ H\ x\ y)$
⟨*proof*⟩

Problem 37; has 10 Horn clauses

lemma $(\forall z.\ \exists w.\ \forall x.\ \exists y.$
$(P\ x\ z \longrightarrow P\ y\ w)\ \&\ P\ y\ z\ \&\ (P\ y\ w \longrightarrow (\exists u.\ Q\ u\ w)))\ \&$
$(\forall x\ z.\ {\sim}P\ x\ z \longrightarrow (\exists y.\ Q\ y\ z))\ \&$
$((\exists x\ y.\ Q\ x\ y) \longrightarrow (\forall x.\ R\ x\ x))$
$\longrightarrow (\forall x.\ \exists y.\ R\ x\ y)$

⟨*proof*⟩

Problem 38

Quite hard: 422 Horn clauses!!

lemma $(\forall x.\ p\ a\ \&\ (p\ x\ \text{--->}\ (\exists y.\ p\ y\ \&\ r\ x\ y))\ \text{--->}$
$(\exists z.\ \exists w.\ p\ z\ \&\ r\ x\ w\ \&\ r\ w\ z))\ =$
$(\forall x.\ (\sim p\ a \mid p\ x \mid (\exists z.\ \exists w.\ p\ z\ \&\ r\ x\ w\ \&\ r\ w\ z))\ \&$
$(\sim p\ a \mid \sim(\exists y.\ p\ y\ \&\ r\ x\ y) \mid$
$(\exists z.\ \exists w.\ p\ z\ \&\ r\ x\ w\ \&\ r\ w\ z)))$
⟨*proof*⟩

Problem 39

lemma $\sim\ (\exists x.\ \forall y.\ F\ y\ x = (\sim F\ y\ y))$
⟨*proof*⟩

Problem 40. AMENDED

lemma $(\exists y.\ \forall x.\ F\ x\ y = F\ x\ x)$
$\text{--->}\ \sim\ (\forall x.\ \exists y.\ \forall z.\ F\ z\ y = (\sim F\ z\ x))$
⟨*proof*⟩

Problem 41

lemma $(\forall z.\ (\exists y.\ (\forall x.\ f\ x\ y = (f\ x\ z\ \&\ \sim\ f\ x\ x))))$
$\text{--->}\ \sim\ (\exists z.\ \forall x.\ f\ x\ z)$
⟨*proof*⟩

Problem 42

lemma $\sim\ (\exists y.\ \forall x.\ p\ x\ y = (\sim\ (\exists z.\ p\ x\ z\ \&\ p\ z\ x)))$
⟨*proof*⟩

Problem 43 NOW PROVED AUTOMATICALLY!!

lemma $(\forall x.\ \forall y.\ q\ x\ y = (\forall z.\ p\ z\ x = (p\ z\ y{::}bool)))$
$\text{--->}\ (\forall x.\ (\forall y.\ q\ x\ y = (q\ y\ x{::}bool)))$
⟨*proof*⟩

Problem 44: 13 Horn clauses; 7-step proof

lemma $(\forall x.\ f\ x\ \text{--->}\ (\exists y.\ g\ y\ \&\ h\ x\ y\ \&\ (\exists y.\ g\ y\ \&\ \sim\ h\ x\ y)))\ \ \&$
$(\exists x.\ j\ x\ \&\ (\forall y.\ g\ y\ \text{--->}\ h\ x\ y))$
$\text{--->}\ (\exists x.\ j\ x\ \&\ \sim f\ x)$
⟨*proof*⟩

Problem 45; has 27 Horn clauses; 54-step proof

lemma $(\forall x.\ f\ x\ \&\ (\forall y.\ g\ y\ \&\ h\ x\ y\ \text{--->}\ j\ x\ y)$
$\text{--->}\ (\forall y.\ g\ y\ \&\ h\ x\ y\ \text{--->}\ k\ y))\ \&$
$\sim\ (\exists y.\ l\ y\ \&\ k\ y)\ \&$
$(\exists x.\ f\ x\ \&\ (\forall y.\ h\ x\ y\ \text{--->}\ l\ y)$
$\&\ (\forall y.\ g\ y\ \&\ h\ x\ y\ \text{--->}\ j\ x\ y))$

$--> (\exists\, x.\ f\ x\ \&\ \sim\ (\exists\, y.\ g\ y\ \&\ h\ x\ y))$
⟨*proof*⟩

Problem 46; has 26 Horn clauses; 21-step proof

lemma $(\forall\, x.\ f\ x\ \&\ (\forall\, y.\ f\ y\ \&\ h\ y\ x\ --> g\ y)\ --> g\ x)\ \&$
$((\exists\, x.\ f\ x\ \&\ \sim g\ x)\ -->$
$(\exists\, x.\ f\ x\ \&\ \sim g\ x\ \&\ (\forall\, y.\ f\ y\ \&\ \sim g\ y\ --> j\ x\ y)))\ \&$
$(\forall\, x\ y.\ f\ x\ \&\ f\ y\ \&\ h\ x\ y\ --> \sim j\ y\ x)$
$--> (\forall\, x.\ f\ x\ --> g\ x)$
⟨*proof*⟩

Problem 47. Schubert's Steamroller. 26 clauses; 63 Horn clauses. 87094 inferences so far. Searching to depth 36

lemma $(\forall\, x.\ wolf\ x \longrightarrow animal\ x)\ \&\ (\exists\, x.\ wolf\ x)\ \&$
$(\forall\, x.\ fox\ x \longrightarrow animal\ x)\ \&\ (\exists\, x.\ fox\ x)\ \&$
$(\forall\, x.\ bird\ x \longrightarrow animal\ x)\ \&\ (\exists\, x.\ bird\ x)\ \&$
$(\forall\, x.\ caterpillar\ x \longrightarrow animal\ x)\ \&\ (\exists\, x.\ caterpillar\ x)\ \&$
$(\forall\, x.\ snail\ x \longrightarrow animal\ x)\ \&\ (\exists\, x.\ snail\ x)\ \&$
$(\forall\, x.\ grain\ x \longrightarrow plant\ x)\ \&\ (\exists\, x.\ grain\ x)\ \&$
$(\forall\, x.\ animal\ x \longrightarrow$
$((\forall\, y.\ plant\ y \longrightarrow eats\ x\ y)\ \vee$
$(\forall\, y.\ animal\ y\ \&\ smaller\text{-}than\ y\ x\ \&$
$(\exists\, z.\ plant\ z\ \&\ eats\ y\ z) \longrightarrow eats\ x\ y)))\ \&$
$(\forall\, x\ y.\ bird\ y\ \&\ (snail\ x \vee caterpillar\ x) \longrightarrow smaller\text{-}than\ x\ y)\ \&$
$(\forall\, x\ y.\ bird\ x\ \&\ fox\ y \longrightarrow smaller\text{-}than\ x\ y)\ \&$
$(\forall\, x\ y.\ fox\ x\ \&\ wolf\ y \longrightarrow smaller\text{-}than\ x\ y)\ \&$
$(\forall\, x\ y.\ wolf\ x\ \&\ (fox\ y \vee grain\ y) \longrightarrow \sim eats\ x\ y)\ \&$
$(\forall\, x\ y.\ bird\ x\ \&\ caterpillar\ y \longrightarrow eats\ x\ y)\ \&$
$(\forall\, x\ y.\ bird\ x\ \&\ snail\ y \longrightarrow \sim eats\ x\ y)\ \&$
$(\forall\, x.\ (caterpillar\ x \vee snail\ x) \longrightarrow (\exists\, y.\ plant\ y\ \&\ eats\ x\ y))$
$\longrightarrow (\exists\, x\ y.\ animal\ x\ \&\ animal\ y\ \&\ (\exists\, z.\ grain\ z\ \&\ eats\ y\ z\ \&\ eats\ x\ y))$
⟨*proof*⟩

The Los problem. Circulated by John Harrison

lemma $(\forall\, x\ y\ z.\ P\ x\ y\ \&\ P\ y\ z\ --> P\ x\ z)\ \&$
$(\forall\, x\ y\ z.\ Q\ x\ y\ \&\ Q\ y\ z\ --> Q\ x\ z)\ \&$
$(\forall\, x\ y.\ P\ x\ y\ --> P\ y\ x)\ \&$
$(\forall\, x\ y.\ P\ x\ y \mid Q\ x\ y)$
$--> (\forall\, x\ y.\ P\ x\ y) \mid (\forall\, x\ y.\ Q\ x\ y)$
⟨*proof*⟩

A similar example, suggested by Johannes Schumann and credited to Pelletier

lemma $(\forall\, x\ y\ z.\ P\ x\ y\ --> P\ y\ z\ --> P\ x\ z)\ -->$
$(\forall\, x\ y\ z.\ Q\ x\ y\ --> Q\ y\ z\ --> Q\ x\ z)\ -->$
$(\forall\, x\ y.\ Q\ x\ y\ --> Q\ y\ x)\ -->\ (\forall\, x\ y.\ P\ x\ y \mid Q\ x\ y)\ -->$
$(\forall\, x\ y.\ P\ x\ y) \mid (\forall\, x\ y.\ Q\ x\ y)$
⟨*proof*⟩

Problem 50. What has this to do with equality?

lemma $(\forall x.\ P\ a\ x \mid (\forall y.\ P\ x\ y)) \longrightarrow (\exists x.\ \forall y.\ P\ x\ y)$
⟨*proof*⟩

Problem 54: NOT PROVED

lemma $(\forall y::'a.\ \exists z.\ \forall x.\ F\ x\ z = (x=y)) \longrightarrow$
$\sim (\exists w.\ \forall x.\ F\ x\ w = (\forall u.\ F\ x\ u \longrightarrow (\exists y.\ F\ y\ u\ \&\ \sim (\exists z.\ F\ z\ u\ \&\ F\ z\ y))))$
⟨*proof*⟩

Problem 55

Non-equational version, from Manthey and Bry, CADE-9 (Springer, 1988). *meson* cannot report who killed Agatha.

lemma *lives agatha & lives butler & lives charles &*
(killed agatha agatha | killed butler agatha | killed charles agatha) &
(∀ x y. killed x y −−> hates x y & ~richer x y) &
(∀ x. hates agatha x −−> ~hates charles x) &
(hates agatha agatha & hates agatha charles) &
(∀ x. lives x & ~richer x agatha −−> hates butler x) &
(∀ x. hates agatha x −−> hates butler x) &
(∀ x. ~hates x agatha | ~hates x butler | ~hates x charles) −−>
(∃ x. killed x agatha)
⟨*proof*⟩

Problem 57

lemma $P\ (f\ a\ b)\ (f\ b\ c)\ \&\ P\ (f\ b\ c)\ (f\ a\ c)\ \&$
$(\forall x\ y\ z.\ P\ x\ y\ \&\ P\ y\ z \longrightarrow P\ x\ z) \longrightarrow P\ (f\ a\ b)\ (f\ a\ c)$
⟨*proof*⟩

Problem 58: Challenge found on info-hol

lemma $\forall P\ Q\ R\ x.\ \exists v\ w.\ \forall y\ z.\ P\ x\ \&\ Q\ y \longrightarrow (P\ v \mid R\ w)\ \&\ (R\ z \longrightarrow Q\ v)$
⟨*proof*⟩

Problem 59

lemma $(\forall x.\ P\ x = (\sim P(f\ x))) \longrightarrow (\exists x.\ P\ x\ \&\ \sim P(f\ x))$
⟨*proof*⟩

Problem 60

lemma $\forall x.\ P\ x\ (f\ x) = (\exists y.\ (\forall z.\ P\ z\ y \longrightarrow P\ z\ (f\ x))\ \&\ P\ x\ y)$
⟨*proof*⟩

Problem 62 as corrected in JAR 18 (1997), page 135

lemma $(\forall x.\ p\ a\ \&\ (p\ x \longrightarrow p(f\ x)) \longrightarrow p(f(f\ x))) =$
$(\forall x.\ (\sim\ p\ a \mid p\ x \mid p(f(f\ x)))\ \&$
$(\sim\ p\ a \mid \sim\ p(f\ x) \mid p(f(f\ x))))$
⟨*proof*⟩

* Charles Morgan's problems *

lemma
assumes a: $\forall x\ y.\ \ T(i\ x(i\ y\ x))$
and b: $\forall x\ y\ z.\ T(i\ (i\ x\ (i\ y\ z))\ (i\ (i\ x\ y)\ (i\ x\ z)))$
and c: $\forall x\ y.\ \ T(i\ (i\ (n\ x)\ (n\ y))\ (i\ y\ x))$
and c': $\forall x\ y.\ \ T(i\ (i\ y\ x)\ (i\ (n\ x)\ (n\ y)))$
and d: $\forall x\ y.\ \ T(i\ x\ y)\ \&\ T\ x\ --{>}\ T\ y$
shows *True*
⟨*proof*⟩

Problem 71, as found in TPTP (SYN007+1.005)

lemma $p1 = (p2 = (p3 = (p4 = (p5 = (p1 = (p2 = (p3 = (p4 = p5))))))))$
⟨*proof*⟩

end

36 Set Theory examples: Cantor's Theorem, Schröder-Bernstein Theorem, etc.

theory *set* **imports** *Main* **begin**

These two are cited in Benzmueller and Kohlhase's system description of LEO, CADE-15, 1998 (pages 139-143) as theorems LEO could not prove.

lemma $(X = Y \cup Z) =$
$(Y \subseteq X \wedge Z \subseteq X \wedge (\forall V.\ Y \subseteq V \wedge Z \subseteq V \longrightarrow X \subseteq V))$
⟨*proof*⟩

lemma $(X = Y \cap Z) =$
$(X \subseteq Y \wedge X \subseteq Z \wedge (\forall V.\ V \subseteq Y \wedge V \subseteq Z \longrightarrow V \subseteq X))$
⟨*proof*⟩

Trivial example of term synthesis: apparently hard for some provers!

lemma $a \neq b \Longrightarrow a \in ?X \wedge b \notin ?X$
⟨*proof*⟩

36.1 Examples for the *blast* paper

lemma $(\bigcup x \in C.\ f\ x \cup g\ x) = \bigcup(f\ `\ C)\ \cup\ \bigcup(g\ `\ C)$
— Union-image, called *Un-Union-image* in Main HOL
⟨*proof*⟩

lemma $(\bigcap x \in C.\ f\ x \cap g\ x) = \bigcap(f\ `\ C) \cap \bigcap(g\ `\ C)$
— Inter-image, called *Int-Inter-image* in Main HOL
⟨*proof*⟩

lemma *singleton-example-1*:
$\bigwedge S::'a\ set\ set.\ \forall x \in S.\ \forall y \in S.\ x \subseteq y \Longrightarrow \exists z.\ S \subseteq \{z\}$
⟨*proof*⟩

lemma *singleton-example-2*:
$\forall x \in S. \bigcup S \subseteq x \Longrightarrow \exists z.\ S \subseteq \{z\}$
— Variant of the problem above.
⟨*proof*⟩

lemma $\exists! x.\ f\ (g\ x) = x \Longrightarrow \exists! y.\ g\ (f\ y) = y$
— A unique fixpoint theorem — *fast*/*best*/*meson* all fail.
⟨*proof*⟩

36.2 Cantor's Theorem: There is no surjection from a set to its powerset

lemma *cantor1*: $\neg\ (\exists f{::}\ 'a \Rightarrow 'a\ set.\ \forall S.\ \exists x.\ f\ x = S)$
— Requires best-first search because it is undirectional.
⟨*proof*⟩

lemma $\forall f{::}\ 'a \Rightarrow 'a\ set.\ \forall x.\ f\ x \neq ?S\ f$
— This form displays the diagonal term.
⟨*proof*⟩

lemma *?S* $\notin$ *range* $(f :: 'a \Rightarrow 'a\ set)$
— This form exploits the set constructs.
⟨*proof*⟩

lemma *?S* $\notin$ *range* $(f :: 'a \Rightarrow 'a\ set)$
— Or just this!
⟨*proof*⟩

36.3 The Schröder-Berstein Theorem

lemma *disj-lemma*: $-\ (f\ `\ X) = g\ `\ (-X) \Longrightarrow f\ a = g\ b \Longrightarrow a \in X \Longrightarrow b \in X$
⟨*proof*⟩

lemma *surj-if-then-else*:
$-(f\ `\ X) = g\ `\ (-X) \Longrightarrow surj\ (\lambda z.\ if\ z \in X\ then\ f\ z\ else\ g\ z)$
⟨*proof*⟩

lemma *bij-if-then-else*:
inj-on f X $\Longrightarrow$ *inj-on g* $(-X) \Longrightarrow -(f\ `\ X) = g\ `\ (-X) \Longrightarrow$
$h = (\lambda z.\ if\ z \in X\ then\ f\ z\ else\ g\ z) \Longrightarrow inj\ h \wedge surj\ h$
⟨*proof*⟩

lemma *decomposition*: $\exists X.\ X = -\ (g\ `\ (-\ (f\ `\ X)))$
⟨*proof*⟩

theorem *Schroeder-Bernstein*:
$inj\ (f :: 'a \Rightarrow 'b) \Longrightarrow inj\ (g :: 'b \Rightarrow 'a)$
$\Longrightarrow \exists h{::}\ 'a \Rightarrow 'b.\ inj\ h \wedge surj\ h$

⟨*proof*⟩

36.4 A simple party theorem

At any party there are two people who know the same number of people. Provided the party consists of at least two people and the knows relation is symmetric. Knowing yourself does not count — otherwise knows needs to be reflexive. (From Freek Wiedijk's talk at TPHOLs 2007.)

lemma *equal-number-of-acquaintances*:
assumes *Domain R* <= *A* **and** *sym R* **and** *card A* ≥ *2*
shows ¬ *inj-on* (%*a*. *card*(*R* `` {*a*} − {*a*})) *A*
⟨*proof*⟩

From W. W. Bledsoe and Guohui Feng, SET-VAR. JAR 11 (3), 1993, pages 293-314.

Isabelle can prove the easy examples without any special mechanisms, but it can't prove the hard ones.

lemma ∃ *A*. (∀ *x* ∈ *A*. *x* ≤ (*0*::*int*))
— Example 1, page 295.
⟨*proof*⟩

lemma *D* ∈ *F* ⟹ ∃ *G*. ∀ *A* ∈ *G*. ∃ *B* ∈ *F*. *A* ⊆ *B*
— Example 2.
⟨*proof*⟩

lemma *P a* ⟹ ∃ *A*. (∀ *x* ∈ *A*. *P x*) ∧ (∃ *y*. *y* ∈ *A*)
— Example 3.
⟨*proof*⟩

lemma *a* < *b* ∧ *b* < (*c*::*int*) ⟹ ∃ *A*. *a* ∉ *A* ∧ *b* ∈ *A* ∧ *c* ∉ *A*
— Example 4.
⟨*proof*⟩

lemma *P* (*f b*) ⟹ ∃ *s A*. (∀ *x* ∈ *A*. *P x*) ∧ *f s* ∈ *A*
— Example 5, page 298.
⟨*proof*⟩

lemma *P* (*f b*) ⟹ ∃ *s A*. (∀ *x* ∈ *A*. *P x*) ∧ *f s* ∈ *A*
— Example 6.
⟨*proof*⟩

lemma ∃ *A*. *a* ∉ *A*
— Example 7.
⟨*proof*⟩

lemma (∀ *u v*. *u* < (*0*::*int*) ⟶ *u* ≠ *abs v*)
⟶ (∃ *A*::*int set*. (∀ *y*. *abs y* ∉ *A*) ∧ −*2* ∈ *A*)
— Example 8 now needs a small hint.

⟨*proof*⟩

Example 9 omitted (requires the reals).

The paper has no Example 10!

lemma $(\forall A.\ 0 \in A \wedge (\forall x \in A.\ Suc\ x \in A) \longrightarrow n \in A) \wedge$
$P\ 0 \wedge (\forall x.\ P\ x \longrightarrow P\ (Suc\ x)) \longrightarrow P\ n$
— Example 11: needs a hint.
⟨*proof*⟩

lemma
$(\forall A.\ (0,\ 0) \in A \wedge (\forall x\ y.\ (x,\ y) \in A \longrightarrow (Suc\ x,\ Suc\ y) \in A) \longrightarrow (n,\ m) \in A)$
$\wedge\ P\ n \longrightarrow P\ m$
— Example 12.
⟨*proof*⟩

lemma
$(\forall x.\ (\exists u.\ x = 2 * u) = (\neg\ (\exists v.\ Suc\ x = 2 * v))) \longrightarrow$
$(\exists A.\ \forall x.\ (x \in A) = (Suc\ x \notin A))$
— Example EO1: typo in article, and with the obvious fix it seems to require arithmetic reasoning.
⟨*proof*⟩

end

37 Meson test cases

theory *Meson-Test*
imports *Main*
begin

WARNING: there are many potential conflicts between variables used below and constants declared in HOL!

hide *const subset member quotient*

Test data for the MESON proof procedure (Excludes the equality problems 51, 52, 56, 58)

37.1 Interactive examples

⟨*ML*⟩
MORE and MUCH HARDER test data for the MESON proof procedure (courtesy John Harrison).

abbreviation *EQU001-0-ax equal* $\equiv (\forall X.\ equal(X::'a,X))\ \&$
$(\forall Y\ X.\ equal(X::'a,Y)\ --{>}\ equal(Y::'a,X))\ \&$
$(\forall Y\ X\ Z.\ equal(X::'a,Y)\ \&\ equal(Y::'a,Z)\ --{>}\ equal(X::'a,Z))$

abbreviation *BOO002-0-ax equal INVERSE multiplicative-identity additive-identity multiply product add sum* ≡
(∀ *X Y*. *sum*(*X*::'*a*,*Y*,*add*(*X*::'*a*,*Y*))) &
(∀ *X Y*. *product*(*X*::'*a*,*Y*,*multiply*(*X*::'*a*,*Y*))) &
(∀ *Y X Z*. *sum*(*X*::'*a*,*Y*,*Z*) −−> *sum*(*Y*::'*a*,*X*,*Z*)) &
(∀ *Y X Z*. *product*(*X*::'*a*,*Y*,*Z*) −−> *product*(*Y*::'*a*,*X*,*Z*)) &
(∀ *X*. *sum*(*additive-identity*::'*a*,*X*,*X*)) &
(∀ *X*. *sum*(*X*::'*a*,*additive-identity*,*X*)) &
(∀ *X*. *product*(*multiplicative-identity*::'*a*,*X*,*X*)) &
(∀ *X*. *product*(*X*::'*a*,*multiplicative-identity*,*X*)) &
(∀ *Y Z X V3 V1 V2 V4*. *product*(*X*::'*a*,*Y*,*V1*) & *product*(*X*::'*a*,*Z*,*V2*) & *sum*(*Y*::'*a*,*Z*,*V3*) & *product*(*X*::'*a*,*V3*,*V4*) −−> *sum*(*V1*::'*a*,*V2*,*V4*)) &
(∀ *Y Z V1 V2 X V3 V4*. *product*(*X*::'*a*,*Y*,*V1*) & *product*(*X*::'*a*,*Z*,*V2*) & *sum*(*Y*::'*a*,*Z*,*V3*) & *sum*(*V1*::'*a*,*V2*,*V4*) −−> *product*(*X*::'*a*,*V3*,*V4*)) &
(∀ *Y Z V3 X V1 V2 V4*. *product*(*Y*::'*a*,*X*,*V1*) & *product*(*Z*::'*a*,*X*,*V2*) & *sum*(*Y*::'*a*,*Z*,*V3*) & *product*(*V3*::'*a*,*X*,*V4*) −−> *sum*(*V1*::'*a*,*V2*,*V4*)) &
(∀ *Y Z V1 V2 V3 X V4*. *product*(*Y*::'*a*,*X*,*V1*) & *product*(*Z*::'*a*,*X*,*V2*) & *sum*(*Y*::'*a*,*Z*,*V3*) & *sum*(*V1*::'*a*,*V2*,*V4*) −−> *product*(*V3*::'*a*,*X*,*V4*)) &
(∀ *Y Z X V3 V1 V2 V4*. *sum*(*X*::'*a*,*Y*,*V1*) & *sum*(*X*::'*a*,*Z*,*V2*) & *product*(*Y*::'*a*,*Z*,*V3*) & *sum*(*X*::'*a*,*V3*,*V4*) −−> *product*(*V1*::'*a*,*V2*,*V4*)) &
(∀ *Y Z V1 V2 X V3 V4*. *sum*(*X*::'*a*,*Y*,*V1*) & *sum*(*X*::'*a*,*Z*,*V2*) & *product*(*Y*::'*a*,*Z*,*V3*) & *product*(*V1*::'*a*,*V2*,*V4*) −−> *sum*(*X*::'*a*,*V3*,*V4*)) &
(∀ *Y Z V3 X V1 V2 V4*. *sum*(*Y*::'*a*,*X*,*V1*) & *sum*(*Z*::'*a*,*X*,*V2*) & *product*(*Y*::'*a*,*Z*,*V3*) & *sum*(*V3*::'*a*,*X*,*V4*) −−> *product*(*V1*::'*a*,*V2*,*V4*)) &
(∀ *Y Z V1 V2 V3 X V4*. *sum*(*Y*::'*a*,*X*,*V1*) & *sum*(*Z*::'*a*,*X*,*V2*) & *product*(*Y*::'*a*,*Z*,*V3*) & *product*(*V1*::'*a*,*V2*,*V4*) −−> *sum*(*V3*::'*a*,*X*,*V4*)) &
(∀ *X*. *sum*(*INVERSE*(*X*),*X*,*multiplicative-identity*)) &
(∀ *X*. *sum*(*X*::'*a*,*INVERSE*(*X*),*multiplicative-identity*)) &
(∀ *X*. *product*(*INVERSE*(*X*),*X*,*additive-identity*)) &
(∀ *X*. *product*(*X*::'*a*,*INVERSE*(*X*),*additive-identity*)) &
(∀ *X Y U V*. *sum*(*X*::'*a*,*Y*,*U*) & *sum*(*X*::'*a*,*Y*,*V*) −−> *equal*(*U*::'*a*,*V*)) &
(∀ *X Y U V*. *product*(*X*::'*a*,*Y*,*U*) & *product*(*X*::'*a*,*Y*,*V*) −−> *equal*(*U*::'*a*,*V*))

abbreviation *BOO002-0-eq INVERSE multiply add product sum equal* ≡
(∀ *X Y W Z*. *equal*(*X*::'*a*,*Y*) & *sum*(*X*::'*a*,*W*,*Z*) −−> *sum*(*Y*::'*a*,*W*,*Z*)) &
(∀ *X W Y Z*. *equal*(*X*::'*a*,*Y*) & *sum*(*W*::'*a*,*X*,*Z*) −−> *sum*(*W*::'*a*,*Y*,*Z*)) &
(∀ *X W Z Y*. *equal*(*X*::'*a*,*Y*) & *sum*(*W*::'*a*,*Z*,*X*) −−> *sum*(*W*::'*a*,*Z*,*Y*)) &
(∀ *X Y W Z*. *equal*(*X*::'*a*,*Y*) & *product*(*X*::'*a*,*W*,*Z*) −−> *product*(*Y*::'*a*,*W*,*Z*)) &
(∀ *X W Y Z*. *equal*(*X*::'*a*,*Y*) & *product*(*W*::'*a*,*X*,*Z*) −−> *product*(*W*::'*a*,*Y*,*Z*)) &
(∀ *X W Z Y*. *equal*(*X*::'*a*,*Y*) & *product*(*W*::'*a*,*Z*,*X*) −−> *product*(*W*::'*a*,*Z*,*Y*)) &
(∀ *X Y W*. *equal*(*X*::'*a*,*Y*) −−> *equal*(*add*(*X*::'*a*,*W*),*add*(*Y*::'*a*,*W*))) &
(∀ *X W Y*. *equal*(*X*::'*a*,*Y*) −−> *equal*(*add*(*W*::'*a*,*X*),*add*(*W*::'*a*,*Y*))) &
(∀ *X Y W*. *equal*(*X*::'*a*,*Y*) −−> *equal*(*multiply*(*X*::'*a*,*W*),*multiply*(*Y*::'*a*,*W*))) &
(∀ *X W Y*. *equal*(*X*::'*a*,*Y*) −−> *equal*(*multiply*(*W*::'*a*,*X*),*multiply*(*W*::'*a*,*Y*)))

&
(∀ X Y. equal(X::'a,Y) −−> equal(INVERSE(X),INVERSE(Y)))

lemma *BOO003-1*:
EQU001-0-ax equal &
BOO002-0-ax equal INVERSE multiplicative-identity additive-identity multiply product add sum &
BOO002-0-eq INVERSE multiply add product sum equal &
(~product(x::'a,x,x)) −−> False
⟨*proof*⟩

lemma *BOO004-1*:
EQU001-0-ax equal &
BOO002-0-ax equal INVERSE multiplicative-identity additive-identity multiply product add sum &
BOO002-0-eq INVERSE multiply add product sum equal &
(~sum(x::'a,x,x)) −−> False
⟨*proof*⟩

lemma *BOO005-1*:
EQU001-0-ax equal &
BOO002-0-ax equal INVERSE multiplicative-identity additive-identity multiply product add sum &
BOO002-0-eq INVERSE multiply add product sum equal &
(~sum(x::'a,multiplicative-identity,multiplicative-identity)) −−> False
⟨*proof*⟩

lemma *BOO006-1*:
EQU001-0-ax equal &
BOO002-0-ax equal INVERSE multiplicative-identity additive-identity multiply product add sum &
BOO002-0-eq INVERSE multiply add product sum equal &
(~product(x::'a,additive-identity,additive-identity)) −−> False
⟨*proof*⟩

lemma *BOO011-1*:
EQU001-0-ax equal &
BOO002-0-ax equal INVERSE multiplicative-identity additive-identity multiply product add sum &
BOO002-0-eq INVERSE multiply add product sum equal &
(~equal(INVERSE(additive-identity),multiplicative-identity)) −−> False
⟨*proof*⟩

abbreviation *CAT003-0-ax f1 compos codomain domain equal there-exists equiv-*

$alent \equiv$
$(\forall\ Y\ X.\ equivalent(X::'a,Y)\ --{>}\ there\text{-}exists(X))\ \&$
$(\forall\ X\ Y.\ equivalent(X::'a,Y)\ --{>}\ equal(X::'a,Y))\ \&$
$(\forall\ X\ Y.\ there\text{-}exists(X)\ \&\ equal(X::'a,Y)\ --{>}\ equivalent(X::'a,Y))\ \&$
$(\forall\ X.\ there\text{-}exists(domain(X))\ --{>}\ there\text{-}exists(X))\ \&$
$(\forall\ X.\ there\text{-}exists(codomain(X))\ --{>}\ there\text{-}exists(X))\ \&$
$(\forall\ Y\ X.\ there\text{-}exists(compos(X::'a,Y))\ --{>}\ there\text{-}exists(domain(X)))\ \&$
$(\forall\ X\ Y.\ there\text{-}exists(compos(X::'a,Y))\ --{>}\ equal(domain(X),codomain(Y)))$
$\&$
$(\forall\ X\ Y.\ there\text{-}exists(domain(X))\ \&\ equal(domain(X),codomain(Y))\ --{>}\ there\text{-}exists(compos(X::'a,Y)))$
$\&$
$(\forall\ X\ Y\ Z.\ equal(compos(X::'a,compos(Y::'a,Z)),compos(compos(X::'a,Y),Z)))$
$\&$
$(\forall\ X.\ equal(compos(X::'a,domain(X)),X))\ \&$
$(\forall\ X.\ equal(compos(codomain(X),X),X))\ \&$
$(\forall\ X\ Y.\ equivalent(X::'a,Y)\ --{>}\ there\text{-}exists(Y))\ \&$
$(\forall\ X\ Y.\ there\text{-}exists(X)\ \&\ there\text{-}exists(Y)\ \&\ equal(X::'a,Y)\ --{>}\ equivalent(X::'a,Y))$
$\&$
$(\forall\ Y\ X.\ there\text{-}exists(compos(X::'a,Y))\ --{>}\ there\text{-}exists(codomain(X)))\ \&$
$(\forall\ X\ Y.\ there\text{-}exists(f1(X::'a,Y))\ |\ equal(X::'a,Y))\ \&$
$(\forall\ X\ Y.\ equal(X::'a,f1(X::'a,Y))\ |\ equal(Y::'a,f1(X::'a,Y))\ |\ equal(X::'a,Y))$
$\&$
$(\forall\ X\ Y.\ equal(X::'a,f1(X::'a,Y))\ \&\ equal(Y::'a,f1(X::'a,Y))\ --{>}\ equal(X::'a,Y))$

abbreviation *CAT003-0-eq f1 compos codomain domain equivalent there-exists equal* $\equiv$
$(\forall\ X\ Y.\ equal(X::'a,Y)\ \&\ there\text{-}exists(X)\ --{>}\ there\text{-}exists(Y))\ \&$
$(\forall\ X\ Y\ Z.\ equal(X::'a,Y)\ \&\ equivalent(X::'a,Z)\ --{>}\ equivalent(Y::'a,Z))\ \&$
$(\forall\ X\ Z\ Y.\ equal(X::'a,Y)\ \&\ equivalent(Z::'a,X)\ --{>}\ equivalent(Z::'a,Y))\ \&$
$(\forall\ X\ Y.\ equal(X::'a,Y)\ --{>}\ equal(domain(X),domain(Y)))\ \&$
$(\forall\ X\ Y.\ equal(X::'a,Y)\ --{>}\ equal(codomain(X),codomain(Y)))\ \&$
$(\forall\ X\ Y\ Z.\ equal(X::'a,Y)\ --{>}\ equal(compos(X::'a,Z),compos(Y::'a,Z)))\ \&$
$(\forall\ X\ Z\ Y.\ equal(X::'a,Y)\ --{>}\ equal(compos(Z::'a,X),compos(Z::'a,Y)))\ \&$
$(\forall\ A\ B\ C.\ equal(A::'a,B)\ --{>}\ equal(f1(A::'a,C),f1(B::'a,C)))\ \&$
$(\forall\ D\ F'\ E.\ equal(D::'a,E)\ --{>}\ equal(f1(F'::'a,D),f1(F'::'a,E)))$

lemma *CAT001-3*:
EQU001-0-ax equal $\&$
CAT003-0-ax f1 compos codomain domain equal there-exists equivalent $\&$
CAT003-0-eq f1 compos codomain domain equivalent there-exists equal $\&$
$(there\text{-}exists(compos(a::'a,b)))\ \&$
$(\forall\ Y\ X\ Z.\ equal(compos(compos(a::'a,b),X),Y)\ \&\ equal(compos(compos(a::'a,b),Z),Y)$
$--{>}\ equal(X::'a,Z))\ \&$
$(there\text{-}exists(compos(b::'a,h)))\ \&$
$(equal(compos(b::'a,h),compos(b::'a,g)))\ \&$
$(\sim equal(h::'a,g))\ --{>}\ False$
$\langle proof \rangle$

lemma *CAT003-3*:
EQU001-0-ax equal &
CAT003-0-ax f1 compos codomain domain equal there-exists equivalent &
CAT003-0-eq f1 compos codomain domain equivalent there-exists equal &
(*there-exists*(*compos*(*a*::'*a*,*b*))) &
(∀ *Y X Z*. *equal*(*compos*(*X*::'*a*,*compos*(*a*::'*a*,*b*)),*Y*) & *equal*(*compos*(*Z*::'*a*,*compos*(*a*::'*a*,*b*)),*Y*) −−> *equal*(*X*::'*a*,*Z*)) &
(*there-exists*(*h*)) &
(*equal*(*compos*(*h*::'*a*,*a*),*compos*(*g*::'*a*,*a*))) &
(~*equal*(*g*::'*a*,*h*)) −−> *False*
⟨*proof*⟩

abbreviation *CAT001-0-ax equal codomain domain identity-map compos product defined* ≡
(∀ *X Y*. *defined*(*X*::'*a*,*Y*) −−> *product*(*X*::'*a*,*Y*,*compos*(*X*::'*a*,*Y*))) &
(∀ *Z X Y*. *product*(*X*::'*a*,*Y*,*Z*) −−> *defined*(*X*::'*a*,*Y*)) &
(∀ *X Xy Y Z*. *product*(*X*::'*a*,*Y*,*Xy*) & *defined*(*Xy*::'*a*,*Z*) −−> *defined*(*Y*::'*a*,*Z*)) &
(∀ *Y Xy Z X Yz*. *product*(*X*::'*a*,*Y*,*Xy*) & *product*(*Y*::'*a*,*Z*,*Yz*) & *defined*(*Xy*::'*a*,*Z*) −−> *defined*(*X*::'*a*,*Yz*)) &
(∀ *Xy Y Z X Yz Xyz*. *product*(*X*::'*a*,*Y*,*Xy*) & *product*(*Xy*::'*a*,*Z*,*Xyz*) & *product*(*Y*::'*a*,*Z*,*Yz*) −−> *product*(*X*::'*a*,*Yz*,*Xyz*)) &
(∀ *Z Yz X Y*. *product*(*Y*::'*a*,*Z*,*Yz*) & *defined*(*X*::'*a*,*Yz*) −−> *defined*(*X*::'*a*,*Y*)) &
(∀ *Y X Yz Xy Z*. *product*(*Y*::'*a*,*Z*,*Yz*) & *product*(*X*::'*a*,*Y*,*Xy*) & *defined*(*X*::'*a*,*Yz*) −−> *defined*(*Xy*::'*a*,*Z*)) &
(∀ *Yz X Y Xy Z Xyz*. *product*(*Y*::'*a*,*Z*,*Yz*) & *product*(*X*::'*a*,*Yz*,*Xyz*) & *product*(*X*::'*a*,*Y*,*Xy*) −−> *product*(*Xy*::'*a*,*Z*,*Xyz*)) &
(∀ *Y X Z*. *defined*(*X*::'*a*,*Y*) & *defined*(*Y*::'*a*,*Z*) & *identity-map*(*Y*) −−> *defined*(*X*::'*a*,*Z*)) &
(∀ *X*. *identity-map*(*domain*(*X*))) &
(∀ *X*. *identity-map*(*codomain*(*X*))) &
(∀ *X*. *defined*(*X*::'*a*,*domain*(*X*))) &
(∀ *X*. *defined*(*codomain*(*X*),*X*)) &
(∀ *X*. *product*(*X*::'*a*,*domain*(*X*),*X*)) &
(∀ *X*. *product*(*codomain*(*X*),*X*,*X*)) &
(∀ *X Y*. *defined*(*X*::'*a*,*Y*) & *identity-map*(*X*) −−> *product*(*X*::'*a*,*Y*,*Y*)) &
(∀ *Y X*. *defined*(*X*::'*a*,*Y*) & *identity-map*(*Y*) −−> *product*(*X*::'*a*,*Y*,*X*)) &
(∀ *X Y Z W*. *product*(*X*::'*a*,*Y*,*Z*) & *product*(*X*::'*a*,*Y*,*W*) −−> *equal*(*Z*::'*a*,*W*))

abbreviation *CAT001-0-eq compos defined identity-map codomain domain product equal* ≡
(∀ *X Y Z W*. *equal*(*X*::'*a*,*Y*) & *product*(*X*::'*a*,*Z*,*W*) −−> *product*(*Y*::'*a*,*Z*,*W*)) &
(∀ *X Z Y W*. *equal*(*X*::'*a*,*Y*) & *product*(*Z*::'*a*,*X*,*W*) −−> *product*(*Z*::'*a*,*Y*,*W*)) &
(∀ *X Z W Y*. *equal*(*X*::'*a*,*Y*) & *product*(*Z*::'*a*,*W*,*X*) −−> *product*(*Z*::'*a*,*W*,*Y*)) &

$(\forall\ X\ Y.\ equal(X::'a,Y)\ --{>}\ equal(domain(X),domain(Y)))\ \&$
$(\forall\ X\ Y.\ equal(X::'a,Y)\ --{>}\ equal(codomain(X),codomain(Y)))\ \&$
$(\forall\ X\ Y.\ equal(X::'a,Y)\ \&\ identity\text{-}map(X)\ --{>}\ identity\text{-}map(Y))\ \&$
$(\forall\ X\ Y\ Z.\ equal(X::'a,Y)\ \&\ defined(X::'a,Z)\ --{>}\ defined(Y::'a,Z))\ \&$
$(\forall\ X\ Z\ Y.\ equal(X::'a,Y)\ \&\ defined(Z::'a,X)\ --{>}\ defined(Z::'a,Y))\ \&$
$(\forall\ X\ Z\ Y.\ equal(X::'a,Y)\ --{>}\ equal(compos(Z::'a,X),compos(Z::'a,Y)))\ \&$
$(\forall\ X\ Y\ Z.\ equal(X::'a,Y)\ --{>}\ equal(compos(X::'a,Z),compos(Y::'a,Z)))$

lemma *CAT005-1*:
EQU001-0-ax equal &
CAT001-0-ax equal codomain domain identity-map compos product defined &
CAT001-0-eq compos defined identity-map codomain domain product equal &
$(defined(a::'a,d))\ \&$
$(identity\text{-}map(d))\ \&$
$(\sim equal(domain(a),d))\ --{>}\ False$
⟨*proof*⟩

lemma *CAT007-1*:
EQU001-0-ax equal &
CAT001-0-ax equal codomain domain identity-map compos product defined &
CAT001-0-eq compos defined identity-map codomain domain product equal &
$(equal(domain(a),codomain(b)))\ \&$
$(\sim defined(a::'a,b))\ --{>}\ False$
⟨*proof*⟩

lemma *CAT018-1*:
EQU001-0-ax equal &
CAT001-0-ax equal codomain domain identity-map compos product defined &
CAT001-0-eq compos defined identity-map codomain domain product equal &
$(defined(a::'a,b))\ \&$
$(defined(b::'a,c))\ \&$
$(\sim defined(a::'a,compos(b::'a,c)))\ --{>}\ False$
⟨*proof*⟩

lemma *COL001-2*:
EQU001-0-ax equal &
$(\forall\ X\ Y\ Z.\ equal(apply(apply(apply(s::'a,X),Y),Z),apply(apply(X::'a,Z),apply(Y::'a,Z))))$
&
$(\forall\ Y\ X.\ equal(apply(apply(k::'a,X),Y),X))\ \&$
$(\forall\ X\ Y\ Z.\ equal(apply(apply(apply(b::'a,X),Y),Z),apply(X::'a,apply(Y::'a,Z))))$
&
$(\forall\ X.\ equal(apply(i::'a,X),X))\ \&$
$(\forall\ A\ B\ C.\ equal(A::'a,B)\ --{>}\ equal(apply(A::'a,C),apply(B::'a,C)))\ \&$
$(\forall\ D\ F'\ E.\ equal(D::'a,E)\ --{>}\ equal(apply(F'::'a,D),apply(F'::'a,E)))\ \&$
$(\forall\ X.\ equal(apply(apply(apply(s::'a,apply(b::'a,X)),i),apply(apply(s::'a,apply(b::'a,X)),i)),apply(x::'a,apply($

&
(∀ Y. ~equal(Y::'a,apply(combinator::'a,Y))) −−> False
⟨proof⟩

lemma *COL023-1*:
EQU001-0-ax equal &
(∀ X Y Z. equal(apply(apply(apply(b::'a,X),Y),Z),apply(X::'a,apply(Y::'a,Z))))
&
(∀ X Y Z. equal(apply(apply(apply(n::'a,X),Y),Z),apply(apply(apply(X::'a,Z),Y),Z)))
&
(∀ A B C. equal(A::'a,B) −−> equal(apply(A::'a,C),apply(B::'a,C))) &
(∀ D F' E. equal(D::'a,E) −−> equal(apply(F'::'a,D),apply(F'::'a,E))) &
(∀ Y. ~equal(Y::'a,apply(combinator::'a,Y))) −−> False
⟨proof⟩

lemma *COL032-1*:
EQU001-0-ax equal &
(∀ X. equal(apply(m::'a,X),apply(X::'a,X))) &
(∀ Y X Z. equal(apply(apply(apply(q::'a,X),Y),Z),apply(Y::'a,apply(X::'a,Z))))
&
(∀ A B C. equal(A::'a,B) −−> equal(apply(A::'a,C),apply(B::'a,C))) &
(∀ D F' E. equal(D::'a,E) −−> equal(apply(F'::'a,D),apply(F'::'a,E))) &
(∀ G H. equal(G::'a,H) −−> equal(f(G),f(H))) &
(∀ Y. ~equal(apply(Y::'a,f(Y)),apply(f(Y),apply(Y::'a,f(Y))))) −−> False
⟨proof⟩

lemma *COL052-2*:
EQU001-0-ax equal &
(∀ X Y W. equal(response(compos(X::'a,Y),W),response(X::'a,response(Y::'a,W))))
&
(∀ X Y. agreeable(X) −−> equal(response(X::'a,common-bird(Y)),response(Y::'a,common-bird(Y))))
&
(∀ Z X. equal(response(X::'a,Z),response(compatible(X),Z)) −−> agreeable(X))
&
(∀ A B. equal(A::'a,B) −−> equal(common-bird(A),common-bird(B))) &
(∀ C D. equal(C::'a,D) −−> equal(compatible(C),compatible(D))) &
(∀ Q R. equal(Q::'a,R) & agreeable(Q) −−> agreeable(R)) &
(∀ A B C. equal(A::'a,B) −−> equal(compos(A::'a,C),compos(B::'a,C))) &
(∀ D F' E. equal(D::'a,E) −−> equal(compos(F'::'a,D),compos(F'::'a,E))) &
(∀ G H I'. equal(G::'a,H) −−> equal(response(G::'a,I'),response(H::'a,I'))) &
(∀ J L K'. equal(J::'a,K') −−> equal(response(L::'a,J),response(L::'a,K'))) &
(agreeable(c)) &
(~agreeable(a)) &
(equal(c::'a,compos(a::'a,b))) −−> False
⟨proof⟩

lemma *COL075-2*:
EQU001-0-ax equal &
(∀ *Y X*. *equal*(*apply*(*apply*(*k*::'*a*,*X*),*Y*),*X*)) &
(∀ *X Y Z*. *equal*(*apply*(*apply*(*apply*(*abstraction*::'*a*,*X*),*Y*),*Z*),*apply*(*apply*(*X*::'*a*,*apply*(*k*::'*a*,*Z*)),*apply*(*Y*::'*a*,*Z*
&
(∀ *D E F'*. *equal*(*D*::'*a*,*E*) −−> *equal*(*apply*(*D*::'*a*,*F'*),*apply*(*E*::'*a*,*F'*))) &
(∀ *G I' H*. *equal*(*G*::'*a*,*H*) −−> *equal*(*apply*(*I'*::'*a*,*G*),*apply*(*I'*::'*a*,*H*))) &
(∀ *A B*. *equal*(*A*::'*a*,*B*) −−> *equal*(*b*(*A*),*b*(*B*))) &
(∀ *C D*. *equal*(*C*::'*a*,*D*) −−> *equal*(*c*(*C*),*c*(*D*))) &
(∀ *Y*. ~*equal*(*apply*(*apply*(*Y*::'*a*,*b*(*Y*)),*c*(*Y*)),*apply*(*b*(*Y*),*b*(*Y*)))) −−> *False*
⟨*proof*⟩

lemma *COM001-1*:
(∀ *Goal-state Start-state*. *follows*(*Goal-state*::'*a*,*Start-state*) −−> *succeeds*(*Goal-state*::'*a*,*Start-state*))
&
(∀ *Goal-state Intermediate-state Start-state*. *succeeds*(*Goal-state*::'*a*,*Intermediate-state*)
& *succeeds*(*Intermediate-state*::'*a*,*Start-state*) −−> *succeeds*(*Goal-state*::'*a*,*Start-state*))
&
(∀ *Start-state Label Goal-state*. *has*(*Start-state*::'*a*,*goto*(*Label*)) & *labels*(*Label*::'*a*,*Goal-state*)
−−> *succeeds*(*Goal-state*::'*a*,*Start-state*)) &
(∀ *Start-state Condition Goal-state*. *has*(*Start-state*::'*a*,*ifthen*(*Condition*::'*a*,*Goal-state*))
−−> *succeeds*(*Goal-state*::'*a*,*Start-state*)) &
(*labels*(*loop*::'*a*,*p3*)) &
(*has*(*p3*::'*a*,*ifthen*(*equal*(*register-j*::'*a*,*n*),*p4*))) &
(*has*(*p4*::'*a*,*goto*(*out*))) &
(*follows*(*p5*::'*a*,*p4*)) &
(*follows*(*p8*::'*a*,*p3*)) &
(*has*(*p8*::'*a*,*goto*(*loop*))) &
(~*succeeds*(*p3*::'*a*,*p3*)) −−> *False*
⟨*proof*⟩

lemma *COM002-1*:
(∀ *Goal-state Start-state*. *follows*(*Goal-state*::'*a*,*Start-state*) −−> *succeeds*(*Goal-state*::'*a*,*Start-state*))
&
(∀ *Goal-state Intermediate-state Start-state*. *succeeds*(*Goal-state*::'*a*,*Intermediate-state*)
& *succeeds*(*Intermediate-state*::'*a*,*Start-state*) −−> *succeeds*(*Goal-state*::'*a*,*Start-state*))
&
(∀ *Start-state Label Goal-state*. *has*(*Start-state*::'*a*,*goto*(*Label*)) & *labels*(*Label*::'*a*,*Goal-state*)
−−> *succeeds*(*Goal-state*::'*a*,*Start-state*)) &
(∀ *Start-state Condition Goal-state*. *has*(*Start-state*::'*a*,*ifthen*(*Condition*::'*a*,*Goal-state*))
−−> *succeeds*(*Goal-state*::'*a*,*Start-state*)) &
(*has*(*p1*::'*a*,*assign*(*register-j*::'*a*,*num0*))) &
(*follows*(*p2*::'*a*,*p1*)) &
(*has*(*p2*::'*a*,*assign*(*register-k*::'*a*,*num1*))) &
(*labels*(*loop*::'*a*,*p3*)) &
(*follows*(*p3*::'*a*,*p2*)) &

(*has*(*p3*::'*a*,*ifthen*(*equal*(*register-j*::'*a*,*n*),*p4*))) &
(*has*(*p4*::'*a*,*goto*(*out*))) &
(*follows*(*p5*::'*a*,*p4*)) &
(*follows*(*p6*::'*a*,*p3*)) &
(*has*(*p6*::'*a*,*assign*(*register-k*::'*a*,*mtimes*(*num2*::'*a*,*register-k*)))) &
(*follows*(*p7*::'*a*,*p6*)) &
(*has*(*p7*::'*a*,*assign*(*register-j*::'*a*,*mplus*(*register-j*::'*a*,*num1*)))) &
(*follows*(*p8*::'*a*,*p7*)) &
(*has*(*p8*::'*a*,*goto*(*loop*))) &
(~*succeeds*(*p3*::'*a*,*p3*)) −−> *False*
⟨*proof*⟩

lemma *COM002-2*:
(∀ *Goal-state Start-state*. ~(*fails*(*Goal-state*::'*a*,*Start-state*) & *follows*(*Goal-state*::'*a*,*Start-state*)))
&
(∀ *Goal-state Intermediate-state Start-state*. *fails*(*Goal-state*::'*a*,*Start-state*) −−>
fails(*Goal-state*::'*a*,*Intermediate-state*) | *fails*(*Intermediate-state*::'*a*,*Start-state*)) &
(∀ *Start-state Label Goal-state*. ~(*fails*(*Goal-state*::'*a*,*Start-state*) & *has*(*Start-state*::'*a*,*goto*(*Label*))
& *labels*(*Label*::'*a*,*Goal-state*))) &
(∀ *Start-state Condition Goal-state*. ~(*fails*(*Goal-state*::'*a*,*Start-state*) & *has*(*Start-state*::'*a*,*ifthen*(*Condition*::
&
(*has*(*p1*::'*a*,*assign*(*register-j*::'*a*,*num0*))) &
(*follows*(*p2*::'*a*,*p1*)) &
(*has*(*p2*::'*a*,*assign*(*register-k*::'*a*,*num1*))) &
(*labels*(*loop*::'*a*,*p3*)) &
(*follows*(*p3*::'*a*,*p2*)) &
(*has*(*p3*::'*a*,*ifthen*(*equal*(*register-j*::'*a*,*n*),*p4*))) &
(*has*(*p4*::'*a*,*goto*(*out*))) &
(*follows*(*p5*::'*a*,*p4*)) &
(*follows*(*p6*::'*a*,*p3*)) &
(*has*(*p6*::'*a*,*assign*(*register-k*::'*a*,*mtimes*(*num2*::'*a*,*register-k*)))) &
(*follows*(*p7*::'*a*,*p6*)) &
(*has*(*p7*::'*a*,*assign*(*register-j*::'*a*,*mplus*(*register-j*::'*a*,*num1*)))) &
(*follows*(*p8*::'*a*,*p7*)) &
(*has*(*p8*::'*a*,*goto*(*loop*))) &
(*fails*(*p3*::'*a*,*p3*)) −−> *False*
⟨*proof*⟩

lemma *COM003-2*:
(∀ *X Y Z*. *program-decides*(*X*) & *program*(*Y*) −−> *decides*(*X*::'*a*,*Y*,*Z*)) &
(∀ *X*. *program-decides*(*X*) | *program*(*f2*(*X*))) &
(∀ *X*. *decides*(*X*::'*a*,*f2*(*X*),*f1*(*X*)) −−> *program-decides*(*X*)) &
(∀ *X*. *program-program-decides*(*X*) −−> *program*(*X*)) &
(∀ *X*. *program-program-decides*(*X*) −−> *program-decides*(*X*)) &
(∀ *X*. *program*(*X*) & *program-decides*(*X*) −−> *program-program-decides*(*X*)) &
(∀ *X*. *algorithm-program-decides*(*X*) −−> *algorithm*(*X*)) &
(∀ *X*. *algorithm-program-decides*(*X*) −−> *program-decides*(*X*)) &

$$
\begin{array}{l}
\quad (\forall\, X.\ \mathit{algorithm}(X)\ \&\ \mathit{program\text{-}decides}(X)\ \text{--}\!>\ \mathit{algorithm\text{-}program\text{-}decides}(X)) \\
\& \\
\quad (\forall\, Y\ X.\ \mathit{program\text{-}halts2}(X{::}'a, Y)\ \text{--}\!>\ \mathit{program}(X))\ \& \\
\quad (\forall\, X\ Y.\ \mathit{program\text{-}halts2}(X{::}'a, Y)\ \text{--}\!>\ \mathit{halts2}(X{::}'a, Y))\ \& \\
\quad (\forall\, X\ Y.\ \mathit{program}(X)\ \&\ \mathit{halts2}(X{::}'a, Y)\ \text{--}\!>\ \mathit{program\text{-}halts2}(X{::}'a, Y))\ \& \\
\quad (\forall\, W\ X\ Y\ Z.\ \mathit{halts3\text{-}outputs}(X{::}'a, Y, Z, W)\ \text{--}\!>\ \mathit{halts3}(X{::}'a, Y, Z))\ \& \\
\quad (\forall\, Y\ Z\ X\ W.\ \mathit{halts3\text{-}outputs}(X{::}'a, Y, Z, W)\ \text{--}\!>\ \mathit{outputs}(X{::}'a, W))\ \& \\
\quad (\forall\, Y\ Z\ X\ W.\ \mathit{halts3}(X{::}'a, Y, Z)\ \&\ \mathit{outputs}(X{::}'a, W)\ \text{--}\!>\ \mathit{halts3\text{-}outputs}(X{::}'a, Y, Z, W)) \\
\& \\
\quad (\forall\, Y\ X.\ \mathit{program\text{-}not\text{-}halts2}(X{::}'a, Y)\ \text{--}\!>\ \mathit{program}(X))\ \& \\
\quad (\forall\, X\ Y.\ \sim(\mathit{program\text{-}not\text{-}halts2}(X{::}'a, Y)\ \&\ \mathit{halts2}(X{::}'a, Y)))\ \& \\
\quad (\forall\, X\ Y.\ \mathit{program}(X)\ \text{--}\!>\ \mathit{program\text{-}not\text{-}halts2}(X{::}'a, Y)\ |\ \mathit{halts2}(X{::}'a, Y))\ \& \\
\quad (\forall\, W\ X\ Y.\ \mathit{halts2\text{-}outputs}(X{::}'a, Y, W)\ \text{--}\!>\ \mathit{halts2}(X{::}'a, Y))\ \& \\
\quad (\forall\, Y\ X\ W.\ \mathit{halts2\text{-}outputs}(X{::}'a, Y, W)\ \text{--}\!>\ \mathit{outputs}(X{::}'a, W))\ \& \\
\quad (\forall\, Y\ X\ W.\ \mathit{halts2}(X{::}'a, Y)\ \&\ \mathit{outputs}(X{::}'a, W)\ \text{--}\!>\ \mathit{halts2\text{-}outputs}(X{::}'a, Y, W)) \\
\& \\
\quad (\forall\, X\ W\ Y\ Z.\ \mathit{program\text{-}halts2\text{-}halts3\text{-}outputs}(X{::}'a, Y, Z, W)\ \text{--}\!>\ \mathit{program\text{-}halts2}(Y{::}'a, Z)) \\
\& \\
\quad (\forall\, X\ Y\ Z\ W.\ \mathit{program\text{-}halts2\text{-}halts3\text{-}outputs}(X{::}'a, Y, Z, W)\ \text{--}\!>\ \mathit{halts3\text{-}outputs}(X{::}'a, Y, Z, W)) \\
\& \\
\quad (\forall\, X\ \ Y\ \ Z\ \ W.\ \ \mathit{program\text{-}halts2}(Y{::}'a, Z)\ \&\ \ \mathit{halts3\text{-}outputs}(X{::}'a, Y, Z, W)\ \text{--}\!> \\
\mathit{program\text{-}halts2\text{-}halts3\text{-}outputs}(X{::}'a, Y, Z, W))\ \& \\
\quad (\forall\, X\ W\ Y\ Z.\ \mathit{program\text{-}not\text{-}halts2\text{-}halts3\text{-}outputs}(X{::}'a, Y, Z, W)\ \text{--}\!>\ \mathit{program\text{-}not\text{-}halts2}(Y{::}'a, Z)) \\
\& \\
\quad (\forall\, X\ Y\ Z\ W.\ \mathit{program\text{-}not\text{-}halts2\text{-}halts3\text{-}outputs}(X{::}'a, Y, Z, W)\ \text{--}\!>\ \mathit{halts3\text{-}outputs}(X{::}'a, Y, Z, W)) \\
\& \\
\quad (\forall\, X\ Y\ Z\ W.\ \mathit{program\text{-}not\text{-}halts2}(Y{::}'a, Z)\ \&\ \mathit{halts3\text{-}outputs}(X{::}'a, Y, Z, W)\ \text{--}\!> \\
\mathit{program\text{-}not\text{-}halts2\text{-}halts3\text{-}outputs}(X{::}'a, Y, Z, W))\ \& \\
\quad (\forall\, X\ W\ Y.\ \mathit{program\text{-}halts2\text{-}halts2\text{-}outputs}(X{::}'a, Y, W)\ \text{--}\!>\ \mathit{program\text{-}halts2}(Y{::}'a, Y)) \\
\& \\
\quad (\forall\, X\ Y\ W.\ \mathit{program\text{-}halts2\text{-}halts2\text{-}outputs}(X{::}'a, Y, W)\ \text{--}\!>\ \mathit{halts2\text{-}outputs}(X{::}'a, Y, W)) \\
\& \\
\quad (\forall\, X\ Y\ W.\ \mathit{program\text{-}halts2}(Y{::}'a, Y)\ \&\ \mathit{halts2\text{-}outputs}(X{::}'a, Y, W)\ \text{--}\!>\ \mathit{program\text{-}halts2\text{-}halts2\text{-}outputs}(X{::}'a \\
\& \\
\quad (\forall\, X\ W\ Y.\ \mathit{program\text{-}not\text{-}halts2\text{-}halts2\text{-}outputs}(X{::}'a, Y, W)\ \text{--}\!>\ \mathit{program\text{-}not\text{-}halts2}(Y{::}'a, Y)) \\
\& \\
\quad (\forall\, X\ Y\ W.\ \mathit{program\text{-}not\text{-}halts2\text{-}halts2\text{-}outputs}(X{::}'a, Y, W)\ \text{--}\!>\ \mathit{halts2\text{-}outputs}(X{::}'a, Y, W)) \\
\& \\
\quad (\forall\, X\ \ Y\ \ W.\ \ \mathit{program\text{-}not\text{-}halts2}(Y{::}'a, Y)\ \ \&\ \ \mathit{halts2\text{-}outputs}(X{::}'a, Y, W)\ \ \text{--}\!> \\
\mathit{program\text{-}not\text{-}halts2\text{-}halts2\text{-}outputs}(X{::}'a, Y, W))\ \& \\
\quad (\forall\, X.\ \mathit{algorithm\text{-}program\text{-}decides}(X)\ \text{--}\!>\ \mathit{program\text{-}program\text{-}decides}(c1))\ \& \\
\quad (\forall\, W\ Y\ Z.\ \mathit{program\text{-}program\text{-}decides}(W)\ \text{--}\!>\ \mathit{program\text{-}halts2\text{-}halts3\text{-}outputs}(W{::}'a, Y, Z, \mathit{good})) \\
\& \\
\quad (\forall\, W\ Y\ Z.\ \mathit{program\text{-}program\text{-}decides}(W)\ \text{--}\!>\ \mathit{program\text{-}not\text{-}halts2\text{-}halts3\text{-}outputs}(W{::}'a, Y, Z, \mathit{bad})) \\
\& \\
\quad (\forall\, W.\ \mathit{program}(W)\ \&\ \mathit{program\text{-}halts2\text{-}halts3\text{-}outputs}(W{::}'a, \mathit{f3}(W), \mathit{f3}(W), \mathit{good}) \\
\&\ \mathit{program\text{-}not\text{-}halts2\text{-}halts3\text{-}outputs}(W{::}'a, \mathit{f3}(W), \mathit{f3}(W), \mathit{bad})\ \text{--}\!>\ \mathit{program}(c2)) \\
\& \\
\quad (\forall\, W\ Y.\ \mathit{program}(W)\ \&\ \mathit{program\text{-}halts2\text{-}halts3\text{-}outputs}(W{::}'a, \mathit{f3}(W), \mathit{f3}(W), \mathit{good})
\end{array}
$$

& *program-not-halts2-halts3-outputs*(*W*::'*a*,*f3*(*W*),*f3*(*W*),*bad*) −−> *program-halts2-halts2-outputs*(*c2*::'*a*,*Y*,*g*
&
(∀ *W Y*. *program*(*W*) & *program-halts2-halts3-outputs*(*W*::'*a*,*f3*(*W*),*f3*(*W*),*good*)
& *program-not-halts2-halts3-outputs*(*W*::'*a*,*f3*(*W*),*f3*(*W*),*bad*) −−> *program-not-halts2-halts2-outputs*(*c2*::'*a*,
&
(∀ *V*. *program*(*V*) & *program-halts2-halts2-outputs*(*V*::'*a*,*f4*(*V*),*good*) & *program-not-halts2-halts2-outputs*(*V*
−−> *program*(*c3*)) &
(∀ *V Y*. *program*(*V*) & *program-halts2-halts2-outputs*(*V*::'*a*,*f4*(*V*),*good*) & *program-not-halts2-halts2-outputs*
& *program-halts2*(*Y*::'*a*,*Y*) −−> *halts2*(*c3*::'*a*,*Y*)) &
(∀ *V Y*. *program*(*V*) & *program-halts2-halts2-outputs*(*V*::'*a*,*f4*(*V*),*good*) & *program-not-halts2-halts2-outputs*
−−> *program-not-halts2-halts2-outputs*(*c3*::'*a*,*Y*,*bad*)) &
(*algorithm-program-decides*(*c4*)) −−> *False*
⟨*proof*⟩

lemma *COM004-1*:
EQU001-0-ax equal &
(∀ *C D P Q X Y*. *failure-node*(*X*::'*a*,*or*(*C*::'*a*,*P*)) & *failure-node*(*Y*::'*a*,*or*(*D*::'*a*,*Q*))
& *contradictory*(*P*::'*a*,*Q*) & *siblings*(*X*::'*a*,*Y*) −−> *failure-node*(*parent-of*(*X*::'*a*,*Y*),*or*(*C*::'*a*,*D*)))
&
(∀ *X*. *contradictory*(*negate*(*X*),*X*)) &
(∀ *X*. *contradictory*(*X*::'*a*,*negate*(*X*))) &
(∀ *X*. *siblings*(*left-child-of*(*X*),*right-child-of*(*X*))) &
(∀ *D E*. *equal*(*D*::'*a*,*E*) −−> *equal*(*left-child-of*(*D*),*left-child-of*(*E*))) &
(∀ *F′ G*. *equal*(*F′*::'*a*,*G*) −−> *equal*(*negate*(*F′*),*negate*(*G*))) &
(∀ *H I′ J*. *equal*(*H*::'*a*,*I′*) −−> *equal*(*or*(*H*::'*a*,*J*),*or*(*I′*::'*a*,*J*))) &
(∀ *K′ M L*. *equal*(*K′*::'*a*,*L*) −−> *equal*(*or*(*M*::'*a*,*K′*),*or*(*M*::'*a*,*L*))) &
(∀ *N O′ P*. *equal*(*N*::'*a*,*O′*) −−> *equal*(*parent-of*(*N*::'*a*,*P*),*parent-of*(*O′*::'*a*,*P*)))
&
(∀ *Q S′ R*. *equal*(*Q*::'*a*,*R*) −−> *equal*(*parent-of*(*S′*::'*a*,*Q*),*parent-of*(*S′*::'*a*,*R*)))
&
(∀ *T′ U*. *equal*(*T′*::'*a*,*U*) −−> *equal*(*right-child-of*(*T′*),*right-child-of*(*U*))) &
(∀ *V W X*. *equal*(*V*::'*a*,*W*) & *contradictory*(*V*::'*a*,*X*) −−> *contradictory*(*W*::'*a*,*X*))
&
(∀ *Y A1 Z*. *equal*(*Y*::'*a*,*Z*) & *contradictory*(*A1*::'*a*,*Y*) −−> *contradictory*(*A1*::'*a*,*Z*))
&
(∀ *B1 C1 D1*. *equal*(*B1*::'*a*,*C1*) & *failure-node*(*B1*::'*a*,*D1*) −−> *failure-node*(*C1*::'*a*,*D1*))
&
(∀ *E1 G1 F1*. *equal*(*E1*::'*a*,*F1*) & *failure-node*(*G1*::'*a*,*E1*) −−> *failure-node*(*G1*::'*a*,*F1*))
&
(∀ *H1 I1 J1*. *equal*(*H1*::'*a*,*I1*) & *siblings*(*H1*::'*a*,*J1*) −−> *siblings*(*I1*::'*a*,*J1*)) &
(∀ *K1 M1 L1*. *equal*(*K1*::'*a*,*L1*) & *siblings*(*M1*::'*a*,*K1*) −−> *siblings*(*M1*::'*a*,*L1*))
&
(*failure-node*(*n-left*::'*a*,*or*(*EMPTY*::'*a*,*atom*))) &
(*failure-node*(*n-right*::'*a*,*or*(*EMPTY*::'*a*,*negate*(*atom*)))) &
(*equal*(*n-left*::'*a*,*left-child-of*(*n*))) &
(*equal*(*n-right*::'*a*,*right-child-of*(*n*))) &
(∀ *Z*. ~*failure-node*(*Z*::'*a*,*or*(*EMPTY*::'*a*,*EMPTY*))) −−> *False*
⟨*proof*⟩

abbreviation *GEO001-0-ax continuous lower-dimension-point-3 lower-dimension-point-2 lower-dimension-point-1 extension euclid2 euclid1 outer-pasch equidistant equal between* ≡
(∀ *X Y*. *between*(*X*::′*a*,*Y*,*X*) −−> *equal*(*X*::′*a*,*Y*)) &
(∀ *V X Y Z*. *between*(*X*::′*a*,*Y*,*V*) & *between*(*Y*::′*a*,*Z*,*V*) −−> *between*(*X*::′*a*,*Y*,*Z*))
&
(∀ *Y X V Z*. *between*(*X*::′*a*,*Y*,*Z*) & *between*(*X*::′*a*,*Y*,*V*) −−> *equal*(*X*::′*a*,*Y*) |
between(*X*::′*a*,*Z*,*V*) | *between*(*X*::′*a*,*V*,*Z*)) &
(∀ *Y X*. *equidistant*(*X*::′*a*,*Y*,*Y*,*X*)) &
(∀ *Z X Y*. *equidistant*(*X*::′*a*,*Y*,*Z*,*Z*) −−> *equal*(*X*::′*a*,*Y*)) &
(∀ *X Y Z V V2 W*. *equidistant*(*X*::′*a*,*Y*,*Z*,*V*) & *equidistant*(*X*::′*a*,*Y*,*V2*,*W*)
−−> *equidistant*(*Z*::′*a*,*V*,*V2*,*W*)) &
(∀ *W X Z V Y*. *between*(*X*::′*a*,*W*,*V*) & *between*(*Y*::′*a*,*V*,*Z*) −−> *between*(*X*::′*a*,*outer-pasch*(*W*::′*a*,*X*,*Y*,*Z*,*W*
&
(∀ *W X Y Z V*. *between*(*X*::′*a*,*W*,*V*) & *between*(*Y*::′*a*,*V*,*Z*) −−> *between*(*Z*::′*a*,*W*,*outer-pasch*(*W*::′*a*,*X*,*Y*,*Z*
&
(∀ *W X Y Z V*. *between*(*X*::′*a*,*V*,*W*) & *between*(*Y*::′*a*,*V*,*Z*) −−> *equal*(*X*::′*a*,*V*)
| *between*(*X*::′*a*,*Z*,*euclid1*(*W*::′*a*,*X*,*Y*,*Z*,*V*))) &
(∀ *W X Y Z V*. *between*(*X*::′*a*,*V*,*W*) & *between*(*Y*::′*a*,*V*,*Z*) −−> *equal*(*X*::′*a*,*V*)
| *between*(*X*::′*a*,*Y*,*euclid2*(*W*::′*a*,*X*,*Y*,*Z*,*V*))) &
(∀ *W X Y Z V*. *between*(*X*::′*a*,*V*,*W*) & *between*(*Y*::′*a*,*V*,*Z*) −−> *equal*(*X*::′*a*,*V*)
| *between*(*euclid1*(*W*::′*a*,*X*,*Y*,*Z*,*V*),*W*,*euclid2*(*W*::′*a*,*X*,*Y*,*Z*,*V*))) &
(∀ *X1 Y1 X Y Z V Z1 V1*. *equidistant*(*X*::′*a*,*Y*,*X1*,*Y1*) & *equidistant*(*Y*::′*a*,*Z*,*Y1*,*Z1*)
& *equidistant*(*X*::′*a*,*V*,*X1*,*V1*) & *equidistant*(*Y*::′*a*,*V*,*Y1*,*V1*) & *between*(*X*::′*a*,*Y*,*Z*)
& *between*(*X1*::′*a*,*Y1*,*Z1*) −−> *equal*(*X*::′*a*,*Y*) | *equidistant*(*Z*::′*a*,*V*,*Z1*,*V1*)) &
(∀ *X Y W V*. *between*(*X*::′*a*,*Y*,*extension*(*X*::′*a*,*Y*,*W*,*V*))) &
(∀ *X Y W V*. *equidistant*(*Y*::′*a*,*extension*(*X*::′*a*,*Y*,*W*,*V*),*W*,*V*)) &
(~ *between*(*lower-dimension-point-1*::′*a*,*lower-dimension-point-2*,*lower-dimension-point-3*))
&
(~ *between*(*lower-dimension-point-2*::′*a*,*lower-dimension-point-3*,*lower-dimension-point-1*))
&
(~ *between*(*lower-dimension-point-3*::′*a*,*lower-dimension-point-1*,*lower-dimension-point-2*))
&
(∀ *Z X Y W V*. *equidistant*(*X*::′*a*,*W*,*X*,*V*) & *equidistant*(*Y*::′*a*,*W*,*Y*,*V*) & *equidis-*
tant(*Z*::′*a*,*W*,*Z*,*V*) −−> *between*(*X*::′*a*,*Y*,*Z*) | *between*(*Y*::′*a*,*Z*,*X*) | *between*(*Z*::′*a*,*X*,*Y*)
| *equal*(*W*::′*a*,*V*)) &
(∀ *X Y Z X1 Z1 V*. *equidistant*(*V*::′*a*,*X*,*V*,*X1*) & *equidistant*(*V*::′*a*,*Z*,*V*,*Z1*) &
between(*V*::′*a*,*X*,*Z*) & *between*(*X*::′*a*,*Y*,*Z*) −−> *equidistant*(*V*::′*a*,*Y*,*Z*,*continuous*(*X*::′*a*,*Y*,*Z*,*X1*,*Z1*,*V*)))
&
(∀ *X Y Z X1 V Z1*. *equidistant*(*V*::′*a*,*X*,*V*,*X1*) & *equidistant*(*V*::′*a*,*Z*,*V*,*Z1*) &
between(*V*::′*a*,*X*,*Z*) & *between*(*X*::′*a*,*Y*,*Z*) −−> *between*(*X1*::′*a*,*continuous*(*X*::′*a*,*Y*,*Z*,*X1*,*Z1*,*V*),*Z1*))

abbreviation *GEO001-0-eq continuous extension euclid2 euclid1 outer-pasch equidistant between equal* ≡
(∀ *X Y W Z*. *equal*(*X*::′*a*,*Y*) & *between*(*X*::′*a*,*W*,*Z*) −−> *between*(*Y*::′*a*,*W*,*Z*))
&
(∀ *X W Y Z*. *equal*(*X*::′*a*,*Y*) & *between*(*W*::′*a*,*X*,*Z*) −−> *between*(*W*::′*a*,*Y*,*Z*))
&

$(\forall\ X\ W\ Z\ Y.\ equal(X::'a,Y)\ \&\ between(W::'a,Z,X)\ -->\ between(W::'a,Z,Y))$
&
$(\forall\ X\ Y\ V\ W\ Z.\ equal(X::'a,Y)\ \&\ equidistant(X::'a,V,W,Z)\ -->\ equidistant(Y::'a,V,W,Z))$ &
$(\forall\ X\ V\ Y\ W\ Z.\ equal(X::'a,Y)\ \&\ equidistant(V::'a,X,W,Z)\ -->\ equidistant(V::'a,Y,W,Z))$ &
$(\forall\ X\ V\ W\ Y\ Z.\ equal(X::'a,Y)\ \&\ equidistant(V::'a,W,X,Z)\ -->\ equidistant(V::'a,W,Y,Z))$ &
$(\forall\ X\ V\ W\ Z\ Y.\ equal(X::'a,Y)\ \&\ equidistant(V::'a,W,Z,X)\ -->\ equidistant(V::'a,W,Z,Y))$ &
$(\forall\ X\ Y\ V1\ V2\ V3\ V4.\ equal(X::'a,Y)\ -->\ equal(outer\text{-}pasch(X::'a,V1,V2,V3,V4),outer\text{-}pasch(Y::'a,V1,V$
&
$(\forall\ X\ V1\ Y\ V2\ V3\ V4.\ equal(X::'a,Y)\ -->\ equal(outer\text{-}pasch(V1::'a,X,V2,V3,V4),outer\text{-}pasch(V1::'a,Y,V$
&
$(\forall\ X\ V1\ V2\ Y\ V3\ V4.\ equal(X::'a,Y)\ -->\ equal(outer\text{-}pasch(V1::'a,V2,X,V3,V4),outer\text{-}pasch(V1::'a,V2,$
&
$(\forall\ X\ V1\ V2\ V3\ Y\ V4.\ equal(X::'a,Y)\ -->\ equal(outer\text{-}pasch(V1::'a,V2,V3,X,V4),outer\text{-}pasch(V1::'a,V2,$
&
$(\forall\ X\ V1\ V2\ V3\ V4\ Y.\ equal(X::'a,Y)\ -->\ equal(outer\text{-}pasch(V1::'a,V2,V3,V4,X),outer\text{-}pasch(V1::'a,V2,$
&
$(\forall\ A\ B\ C\ D\ E\ F'.\ equal(A::'a,B)\ -->\ equal(euclid1(A::'a,C,D,E,F'),euclid1(B::'a,C,D,E,F')))$
&
$(\forall\ G\ I'\ H\ J\ K'\ L.\ equal(G::'a,H)\ -->\ equal(euclid1(I'::'a,G,J,K',L),euclid1(I'::'a,H,J,K',L)))$
&
$(\forall\ M\ O'\ P\ N\ Q\ R.\ equal(M::'a,N)\ -->\ equal(euclid1(O'::'a,P,M,Q,R),euclid1(O'::'a,P,N,Q,R)))$
&
$(\forall\ S'\ U\ V\ W\ T'\ X.\ equal(S'::'a,T')\ -->\ equal(euclid1(U::'a,V,W,S',X),euclid1(U::'a,V,W,T',X)))$
&
$(\forall\ Y\ A1\ B1\ C1\ D1\ Z.\ equal(Y::'a,Z)\ -->\ equal(euclid1(A1::'a,B1,C1,D1,Y),euclid1(A1::'a,B1,C1,D1,Z)$
&
$(\forall\ E1\ F1\ G1\ H1\ I1\ J1.\ equal(E1::'a,F1)\ -->\ equal(euclid2(E1::'a,G1,H1,I1,J1),euclid2(F1::'a,G1,H1,I1,$
&
$(\forall\ K1\ M1\ L1\ N1\ O1\ P1.\ equal(K1::'a,L1)\ -->\ equal(euclid2(M1::'a,K1,N1,O1,P1),euclid2(M1::'a,L1,N1,$
&
$(\forall\ Q1\ S1\ T1\ R1\ U1\ V1.\ equal(Q1::'a,R1)\ -->\ equal(euclid2(S1::'a,T1,Q1,U1,V1),euclid2(S1::'a,T1,R1,U$
&
$(\forall\ W1\ Y1\ Z1\ A2\ X1\ B2.\ equal(W1::'a,X1)\ -->\ equal(euclid2(Y1::'a,Z1,A2,W1,B2),euclid2(Y1::'a,Z1,A2$
&
$(\forall\ C2\ E2\ F2\ G2\ H2\ D2.\ equal(C2::'a,D2)\ -->\ equal(euclid2(E2::'a,F2,G2,H2,C2),euclid2(E2::'a,F2,G2,H$
&
$(\forall\ X\ Y\ V1\ V2\ V3.\ equal(X::'a,Y)\ -->\ equal(extension(X::'a,V1,V2,V3),extension(Y::'a,V1,V2,V3)))$
&
$(\forall\ X\ V1\ Y\ V2\ V3.\ equal(X::'a,Y)\ -->\ equal(extension(V1::'a,X,V2,V3),extension(V1::'a,Y,V2,V3)))$
&
$(\forall\ X\ V1\ V2\ Y\ V3.\ equal(X::'a,Y)\ -->\ equal(extension(V1::'a,V2,X,V3),extension(V1::'a,V2,Y,V3)))$
&
$(\forall\ X\ V1\ V2\ V3\ Y.\ equal(X::'a,Y)\ -->\ equal(extension(V1::'a,V2,V3,X),extension(V1::'a,V2,V3,Y)))$
&
$(\forall\ X\ Y\ V1\ V2\ V3\ V4\ V5.\ equal(X::'a,Y)\ -->\ equal(continuous(X::'a,V1,V2,V3,V4,V5),continuous(Y::'a,$

&
(∀ X V1 Y V2 V3 V4 V5. equal(X::'a,Y) −−> equal(continuous(V1::'a,X,V2,V3,V4,V5),continuous(V1::'a
&
(∀ X V1 V2 Y V3 V4 V5. equal(X::'a,Y) −−> equal(continuous(V1::'a,V2,X,V3,V4,V5),continuous(V1::'a
&
(∀ X V1 V2 V3 Y V4 V5. equal(X::'a,Y) −−> equal(continuous(V1::'a,V2,V3,X,V4,V5),continuous(V1::'a
&
(∀ X V1 V2 V3 V4 Y V5. equal(X::'a,Y) −−> equal(continuous(V1::'a,V2,V3,V4,X,V5),continuous(V1::'a
&
(∀ X V1 V2 V3 V4 V5 Y. equal(X::'a,Y) −−> equal(continuous(V1::'a,V2,V3,V4,V5,X),continuous(V1::'a

lemma *GEO003-1*:
EQU001-0-ax equal &
GEO001-0-ax continuous lower-dimension-point-3 lower-dimension-point-2 lower-dimension-point-1 extension euclid2 euclid1 outer-pasch equidistant equal between &
GEO001-0-eq continuous extension euclid2 euclid1 outer-pasch equidistant between equal &
(~*between*(a::'a,b,b)) −−> *False*
⟨*proof*⟩

abbreviation *GEO002-ax-eq continuous euclid2 euclid1 lower-dimension-point-3 lower-dimension-point-2 lower-dimension-point-1 inner-pasch extension between equal equidistant* ≡
(∀ Y X. *equidistant*(X::'a,Y,Y,X)) &
(∀ X Y Z V V2 W. *equidistant*(X::'a,Y,Z,V) & *equidistant*(X::'a,Y,V2,W) −−> *equidistant*(Z::'a,V,V2,W)) &
(∀ Z X Y. *equidistant*(X::'a,Y,Z,Z) −−> *equal*(X::'a,Y)) &
(∀ X Y W V. *between*(X::'a,Y,*extension*(X::'a,Y,W,V))) &
(∀ X Y W V. *equidistant*(Y::'a,*extension*(X::'a,Y,W,V),W,V)) &
(∀ X1 Y1 X Y Z V Z1 V1. *equidistant*(X::'a,Y,X1,Y1) & *equidistant*(Y::'a,Z,Y1,Z1) & *equidistant*(X::'a,V,X1,V1) & *equidistant*(Y::'a,V,Y1,V1) & *between*(X::'a,Y,Z) & *between*(X1::'a,Y1,Z1) −−> *equal*(X::'a,Y) | *equidistant*(Z::'a,V,Z1,V1)) &
(∀ X Y. *between*(X::'a,Y,X) −−> *equal*(X::'a,Y)) &
(∀ U V W X Y. *between*(U::'a,V,W) & *between*(Y::'a,X,W) −−> *between*(V::'a,*inner-pasch*(U::'a,V,W,X
&
(∀ V W X Y U. *between*(U::'a,V,W) & *between*(Y::'a,X,W) −−> *between*(X::'a,*inner-pasch*(U::'a,V,W,X,
&
(~*between*(*lower-dimension-point-1*::'a,*lower-dimension-point-2*,*lower-dimension-point-3*))
&
(~*between*(*lower-dimension-point-2*::'a,*lower-dimension-point-3*,*lower-dimension-point-1*))
&
(~*between*(*lower-dimension-point-3*::'a,*lower-dimension-point-1*,*lower-dimension-point-2*))
&
(∀ Z X Y W V. *equidistant*(X::'a,W,X,V) & *equidistant*(Y::'a,W,Y,V) & *equidistant*(Z::'a,W,Z,V) −−> *between*(X::'a,Y,Z) | *between*(Y::'a,Z,X) | *between*(Z::'a,X,Y) | *equal*(W::'a,V)) &

$(\forall\ U\ V\ W\ X\ Y.\ between(U::'a,W,Y)\ \&\ between(V::'a,W,X)\ -->\ equal(U::'a,W)$
$|\ between(U::'a,V,euclid1(U::'a,V,W,X,Y)))\ \&$
$(\forall\ U\ V\ W\ X\ Y.\ between(U::'a,W,Y)\ \&\ between(V::'a,W,X)\ -->\ equal(U::'a,W)$
$|\ between(U::'a,X,euclid2(U::'a,V,W,X,Y)))\ \&$
$(\forall\ U\ V\ W\ X\ Y.\ between(U::'a,W,Y)\ \&\ between(V::'a,W,X)\ -->\ equal(U::'a,W)$
$|\ between(euclid1(U::'a,V,W,X,Y),Y,euclid2(U::'a,V,W,X,Y)))\ \&$
$(\forall\ U\ V\ V1\ W\ X\ X1.\ equidistant(U::'a,V,U,V1)\ \&\ equidistant(U::'a,X,U,X1)\ \&$
$between(U::'a,V,X)\ \&\ between(V::'a,W,X)\ -->\ between(V1::'a,continuous(U::'a,V,V1,W,X,X1),X1))$
$\&$
$(\forall\ U\ V\ V1\ W\ X\ X1.\ equidistant(U::'a,V,U,V1)\ \&\ equidistant(U::'a,X,U,X1)\ \&$
$between(U::'a,V,X)\ \&\ between(V::'a,W,X)\ -->\ equidistant(U::'a,W,U,continuous(U::'a,V,V1,W,X,X1))$
$\&$
$(\forall\ X\ Y\ W\ Z.\ equal(X::'a,Y)\ \&\ between(X::'a,W,Z)\ -->\ between(Y::'a,W,Z))$
$\&$
$(\forall\ X\ W\ Y\ Z.\ equal(X::'a,Y)\ \&\ between(W::'a,X,Z)\ -->\ between(W::'a,Y,Z))$
$\&$
$(\forall\ X\ W\ Z\ Y.\ equal(X::'a,Y)\ \&\ between(W::'a,Z,X)\ -->\ between(W::'a,Z,Y))$
$\&$
$(\forall\ X\ Y\ V\ W\ Z.\ equal(X::'a,Y)\ \&\ equidistant(X::'a,V,W,Z)\ -->\ equidistant(Y::'a,V,W,Z))\ \&$
$(\forall\ X\ V\ Y\ W\ Z.\ equal(X::'a,Y)\ \&\ equidistant(V::'a,X,W,Z)\ -->\ equidistant(V::'a,Y,W,Z))\ \&$
$(\forall\ X\ V\ W\ Y\ Z.\ equal(X::'a,Y)\ \&\ equidistant(V::'a,W,X,Z)\ -->\ equidistant(V::'a,W,Y,Z))\ \&$
$(\forall\ X\ V\ W\ Z\ Y.\ equal(X::'a,Y)\ \&\ equidistant(V::'a,W,Z,X)\ -->\ equidistant(V::'a,W,Z,Y))\ \&$
$(\forall\ X\ Y\ V1\ V2\ V3\ V4.\ equal(X::'a,Y)\ -->\ equal(inner\text{-}pasch(X::'a,V1,V2,V3,V4),inner\text{-}pasch(Y::'a,V1,V$
$\&$
$(\forall\ X\ V1\ Y\ V2\ V3\ V4.\ equal(X::'a,Y)\ -->\ equal(inner\text{-}pasch(V1::'a,X,V2,V3,V4),inner\text{-}pasch(V1::'a,Y,V$
$\&$
$(\forall\ X\ V1\ V2\ Y\ V3\ V4.\ equal(X::'a,Y)\ -->\ equal(inner\text{-}pasch(V1::'a,V2,X,V3,V4),inner\text{-}pasch(V1::'a,V2,$
$\&$
$(\forall\ X\ V1\ V2\ V3\ Y\ V4.\ equal(X::'a,Y)\ -->\ equal(inner\text{-}pasch(V1::'a,V2,V3,X,V4),inner\text{-}pasch(V1::'a,V2,$
$\&$
$(\forall\ X\ V1\ V2\ V3\ V4\ Y.\ equal(X::'a,Y)\ -->\ equal(inner\text{-}pasch(V1::'a,V2,V3,V4,X),inner\text{-}pasch(V1::'a,V2,$
$\&$
$(\forall\ A\ B\ C\ D\ E\ F'.\ equal(A::'a,B)\ -->\ equal(euclid1(A::'a,C,D,E,F'),euclid1(B::'a,C,D,E,F')))$
$\&$
$(\forall\ G\ I'\ H\ J\ K'\ L.\ equal(G::'a,H)\ -->\ equal(euclid1(I'::'a,G,J,K',L),euclid1(I'::'a,H,J,K',L)))$
$\&$
$(\forall\ M\ O'\ P\ N\ Q\ R.\ equal(M::'a,N)\ -->\ equal(euclid1(O'::'a,P,M,Q,R),euclid1(O'::'a,P,N,Q,R)))$
$\&$
$(\forall\ S'\ U\ V\ W\ T'\ X.\ equal(S'::'a,T')\ -->\ equal(euclid1(U::'a,V,W,S',X),euclid1(U::'a,V,W,T',X)))$
$\&$
$(\forall\ Y\ A1\ B1\ C1\ D1\ Z.\ equal(Y::'a,Z)\ -->\ equal(euclid1(A1::'a,B1,C1,D1,Y),euclid1(A1::'a,B1,C1,D1,Z)$
$\&$
$(\forall\ E1\ F1\ G1\ H1\ I1\ J1.\ equal(E1::'a,F1)\ -->\ equal(euclid2(E1::'a,G1,H1,I1,J1),euclid2(F1::'a,G1,H1,I1,$
$\&$
$(\forall\ K1\ M1\ L1\ N1\ O1\ P1.\ equal(K1::'a,L1)\ -->\ equal(euclid2(M1::'a,K1,N1,O1,P1),euclid2(M1::'a,L1,N1,$

&
 (∀ Q1 S1 T1 R1 U1 V1. equal(Q1::'a,R1) −−> equal(euclid2(S1::'a,T1,Q1,U1,V1),euclid2(S1::'a,T1,R1,U
&
 (∀ W1 Y1 Z1 A2 X1 B2. equal(W1::'a,X1) −−> equal(euclid2(Y1::'a,Z1,A2,W1,B2),euclid2(Y1::'a,Z1,A2
&
 (∀ C2 E2 F2 G2 H2 D2. equal(C2::'a,D2) −−> equal(euclid2(E2::'a,F2,G2,H2,C2),euclid2(E2::'a,F2,G2,H
&
 (∀ X Y V1 V2 V3. equal(X::'a,Y) −−> equal(extension(X::'a,V1,V2,V3),extension(Y::'a,V1,V2,V3)))
&
 (∀ X V1 Y V2 V3. equal(X::'a,Y) −−> equal(extension(V1::'a,X,V2,V3),extension(V1::'a,Y,V2,V3)))
&
 (∀ X V1 V2 Y V3. equal(X::'a,Y) −−> equal(extension(V1::'a,V2,X,V3),extension(V1::'a,V2,Y,V3)))
&
 (∀ X V1 V2 V3 Y. equal(X::'a,Y) −−> equal(extension(V1::'a,V2,V3,X),extension(V1::'a,V2,V3,Y)))
&
 (∀ X Y V1 V2 V3 V4 V5. equal(X::'a,Y) −−> equal(continuous(X::'a,V1,V2,V3,V4,V5),continuous(Y::'a,
&
 (∀ X V1 Y V2 V3 V4 V5. equal(X::'a,Y) −−> equal(continuous(V1::'a,X,V2,V3,V4,V5),continuous(V1::'a
&
 (∀ X V1 V2 Y V3 V4 V5. equal(X::'a,Y) −−> equal(continuous(V1::'a,V2,X,V3,V4,V5),continuous(V1::'a
&
 (∀ X V1 V2 V3 Y V4 V5. equal(X::'a,Y) −−> equal(continuous(V1::'a,V2,V3,X,V4,V5),continuous(V1::'a
&
 (∀ X V1 V2 V3 V4 Y V5. equal(X::'a,Y) −−> equal(continuous(V1::'a,V2,V3,V4,X,V5),continuous(V1::'a
&
 (∀ X V1 V2 V3 V4 V5 Y. equal(X::'a,Y) −−> equal(continuous(V1::'a,V2,V3,V4,V5,X),continuous(V1::'a

lemma *GEO017-2*:
 EQU001-0-ax equal &
 GEO002-ax-eq continuous euclid2 euclid1 lower-dimension-point-3
  lower-dimension-point-2 lower-dimension-point-1 inner-pasch extension
  between equal equidistant &
 (*equidistant*(u::'a,v,w,x)) &
 (~*equidistant*(u::'a,v,x,w)) −−> *False*
 ⟨*proof*⟩

lemma *GEO027-3*:
 EQU001-0-ax equal &
 GEO002-ax-eq continuous euclid2 euclid1 lower-dimension-point-3
  lower-dimension-point-2 lower-dimension-point-1 inner-pasch extension
  between equal equidistant &
 (∀ U V. equal(reflection(U::'a,V),extension(U::'a,V,U,V))) &
 (∀ X Y Z. equal(X::'a,Y) −−> equal(reflection(X::'a,Z),reflection(Y::'a,Z))) &
 (∀ A1 C1 B1. equal(A1::'a,B1) −−> equal(reflection(C1::'a,A1),reflection(C1::'a,B1)))
&
 (∀ U V. equidistant(U::'a,V,U,V)) &
 (∀ W X U V. equidistant(U::'a,V,W,X) −−> equidistant(W::'a,X,U,V)) &

$(\forall\ V\ U\ W\ X.\ equidistant(U::'a,V,W,X)\ --{>}\ equidistant(V::'a,U,W,X))\ \&$
$(\forall\ U\ V\ X\ W.\ equidistant(U::'a,V,W,X)\ --{>}\ equidistant(U::'a,V,X,W))\ \&$
$(\forall\ V\ U\ X\ W.\ equidistant(U::'a,V,W,X)\ --{>}\ equidistant(V::'a,U,X,W))\ \&$
$(\forall\ W\ X\ V\ U.\ equidistant(U::'a,V,W,X)\ --{>}\ equidistant(W::'a,X,V,U))\ \&$
$(\forall\ X\ W\ U\ V.\ equidistant(U::'a,V,W,X)\ --{>}\ equidistant(X::'a,W,U,V))\ \&$
$(\forall\ X\ W\ V\ U.\ equidistant(U::'a,V,W,X)\ --{>}\ equidistant(X::'a,W,V,U))\ \&$
$(\forall\ W\ X\ U\ V\ Y\ Z.\ equidistant(U::'a,V,W,X)\ \&\ equidistant(W::'a,X,Y,Z)\ --{>}$
$equidistant(U::'a,V,Y,Z))\ \&$
$(\forall\ U\ V\ W.\ equal(V::'a,extension(U::'a,V,W,W)))\ \&$
$(\forall\ W\ X\ U\ V\ Y.\ equal(Y::'a,extension(U::'a,V,W,X))\ --{>}\ between(U::'a,V,Y))$
$\&$
$(\forall\ U\ V.\ between(U::'a,V,reflection(U::'a,V)))\ \&$
$(\forall\ U\ V.\ equidistant(V::'a,reflection(U::'a,V),U,V))\ \&$
$(\forall\ U\ V.\ equal(U::'a,V)\ --{>}\ equal(V::'a,reflection(U::'a,V)))\ \&$
$(\forall\ U.\ equal(U::'a,reflection(U::'a,U)))\ \&$
$(\forall\ U\ V.\ equal(V::'a,reflection(U::'a,V))\ --{>}\ equal(U::'a,V))\ \&$
$(\forall\ U\ V.\ equidistant(U::'a,U,V,V))\ \&$
$(\forall\ V\ V1\ U\ W\ U1\ W1.\ equidistant(U::'a,V,U1,V1)\ \&\ equidistant(V::'a,W,V1,W1)$
$\&\ between(U::'a,V,W)\ \&\ between(U1::'a,V1,W1)\ --{>}\ equidistant(U::'a,W,U1,W1))$
$\&$
$(\forall\ U\ V\ W\ X.\ between(U::'a,V,W)\ \&\ between(U::'a,V,X)\ \&\ equidistant(V::'a,W,V,X)$
$--{>}\ equal(U::'a,V)\ |\ equal(W::'a,X))\ \&$
$(between(u::'a,v,w))\ \&$
$(\sim equal(u::'a,v))\ \&$
$(\sim equal(w::'a,extension(u::'a,v,v,w)))\ --{>}\ False$
$\langle proof\rangle$

lemma *GEO058-2*:
EQU001-0-ax equal &
GEO002-ax-eq continuous euclid2 euclid1 lower-dimension-point-3
lower-dimension-point-2 lower-dimension-point-1 inner-pasch extension
between equal equidistant &
$(\forall\ U\ V.\ equal(reflection(U::'a,V),extension(U::'a,V,U,V)))\ \&$
$(\forall\ X\ Y\ Z.\ equal(X::'a,Y)\ --{>}\ equal(reflection(X::'a,Z),reflection(Y::'a,Z)))\ \&$
$(\forall\ A1\ C1\ B1.\ equal(A1::'a,B1)\ --{>}\ equal(reflection(C1::'a,A1),reflection(C1::'a,B1)))$
$\&$
$(equal(v::'a,reflection(u::'a,v)))\ \&$
$(\sim equal(u::'a,v))\ --{>}\ False$
$\langle proof\rangle$

lemma *GEO079-1*:
$(\forall\ U\ V\ W\ X\ Y\ Z.\ right\text{-}angle(U::'a,V,W)\ \&\ right\text{-}angle(X::'a,Y,Z)\ --{>}\ eq(U::'a,V,W,X,Y,Z))$
$\&$
$(\forall\ U\ V\ W\ X\ Y\ Z.\ CONGRUENT(U::'a,V,W,X,Y,Z)\ --{>}\ eq(U::'a,V,W,X,Y,Z))$
$\&$
$(\forall\ V\ W\ U\ X.\ trapezoid(U::'a,V,W,X)\ --{>}\ parallel(V::'a,W,U,X))\ \&$
$(\forall\ U\ V\ X\ Y.\ parallel(U::'a,V,X,Y)\ --{>}\ eq(X::'a,V,U,V,X,Y))\ \&$

(*trapezoid*(a::'a,b,c,d)) &
(~*eq*(a::'a,c,b,c,a,d)) --> *False*
⟨*proof*⟩

abbreviation *GRP003-0-ax equal multiply INVERSE identity product* ≡
(∀ X. *product*(*identity*::'a,X,X)) &
(∀ X. *product*(X::'a,*identity*,X)) &
(∀ X. *product*(*INVERSE*(X),X,*identity*)) &
(∀ X. *product*(X::'a,*INVERSE*(X),*identity*)) &
(∀ X Y. *product*(X::'a,Y,*multiply*(X::'a,Y))) &
(∀ X Y Z W. *product*(X::'a,Y,Z) & *product*(X::'a,Y,W) --> *equal*(Z::'a,W))
&
(∀ Y U Z X V W. *product*(X::'a,Y,U) & *product*(Y::'a,Z,V) & *product*(U::'a,Z,W)
--> *product*(X::'a,V,W)) &
(∀ Y X V U Z W. *product*(X::'a,Y,U) & *product*(Y::'a,Z,V) & *product*(X::'a,V,W)
--> *product*(U::'a,Z,W))

abbreviation *GRP003-0-eq product multiply INVERSE equal* ≡
(∀ X Y. *equal*(X::'a,Y) --> *equal*(*INVERSE*(X),*INVERSE*(Y))) &
(∀ X Y W. *equal*(X::'a,Y) --> *equal*(*multiply*(X::'a,W),*multiply*(Y::'a,W)))
&
(∀ X W Y. *equal*(X::'a,Y) --> *equal*(*multiply*(W::'a,X),*multiply*(W::'a,Y)))
&
(∀ X Y W Z. *equal*(X::'a,Y) & *product*(X::'a,W,Z) --> *product*(Y::'a,W,Z))
&
(∀ X W Y Z. *equal*(X::'a,Y) & *product*(W::'a,X,Z) --> *product*(W::'a,Y,Z))
&
(∀ X W Z Y. *equal*(X::'a,Y) & *product*(W::'a,Z,X) --> *product*(W::'a,Z,Y))

lemma *GRP001-1*:
EQU001-0-ax equal &
GRP003-0-ax equal multiply INVERSE identity product &
GRP003-0-eq product multiply INVERSE equal &
(∀ X. *product*(X::'a,X,*identity*)) &
(*product*(a::'a,b,c)) &
(~*product*(b::'a,a,c)) --> *False*
⟨*proof*⟩

lemma *GRP008-1*:
EQU001-0-ax equal &
GRP003-0-ax equal multiply INVERSE identity product &
GRP003-0-eq product multiply INVERSE equal &
(∀ A B. *equal*(A::'a,B) --> *equal*(*h*(A),*h*(B))) &
(∀ C D. *equal*(C::'a,D) --> *equal*(*j*(C),*j*(D))) &
(∀ A B. *equal*(A::'a,B) & *q*(A) --> *q*(B)) &
(∀ B A C. *q*(A) & *product*(A::'a,B,C) --> *product*(B::'a,A,C)) &
(∀ A. *product*(*j*(A),A,*h*(A)) | *product*(A::'a,*j*(A),*h*(A)) | *q*(A)) &

$(\forall A.\ product(j(A),A,h(A))\ \&\ product(A::'a,j(A),h(A))\ -->\ q(A))\ \&$
$(\sim q(identity))\ -->\ False$
$\langle proof \rangle$

lemma *GRP013-1*:
EQU001-0-ax equal &
GRP003-0-ax equal multiply INVERSE identity product &
GRP003-0-eq product multiply INVERSE equal &
$(\forall A.\ product(A::'a,A,identity))\ \&$
$(product(a::'a,b,c))\ \&$
$(product(INVERSE(a),INVERSE(b),d))\ \&$
$(\forall A\ C\ B.\ product(INVERSE(A),INVERSE(B),C)\ -->\ product(A::'a,C,B))\ \&$
$(\sim product(c::'a,d,identity))\ -->\ False$
$\langle proof \rangle$

lemma *GRP037-3*:
EQU001-0-ax equal &
GRP003-0-ax equal multiply INVERSE identity product &
GRP003-0-eq product multiply INVERSE equal &
$(\forall A\ B\ C.\ subgroup\text{-}member(A)\ \&\ subgroup\text{-}member(B)\ \&\ product(A::'a,INVERSE(B),C)\ -->\ subgroup\text{-}member(C))\ \&$
$(\forall A\ B.\ equal(A::'a,B)\ \&\ subgroup\text{-}member(A)\ -->\ subgroup\text{-}member(B))\ \&$
$(\forall A.\ subgroup\text{-}member(A)\ -->\ product(Gidentity::'a,A,A))\ \&$
$(\forall A.\ subgroup\text{-}member(A)\ -->\ product(A::'a,Gidentity,A))\ \&$
$(\forall A.\ subgroup\text{-}member(A)\ -->\ product(A::'a,Ginverse(A),Gidentity))\ \&$
$(\forall A.\ subgroup\text{-}member(A)\ -->\ product(Ginverse(A),A,Gidentity))\ \&$
$(\forall A.\ subgroup\text{-}member(A)\ -->\ subgroup\text{-}member(Ginverse(A)))\ \&$
$(\forall A\ B.\ equal(A::'a,B)\ -->\ equal(Ginverse(A),Ginverse(B)))\ \&$
$(\forall A\ C\ D\ B.\ product(A::'a,B,C)\ \&\ product(A::'a,D,C)\ -->\ equal(D::'a,B))\ \&$
$(\forall B\ C\ D\ A.\ product(A::'a,B,C)\ \&\ product(D::'a,B,C)\ -->\ equal(D::'a,A))\ \&$
$(subgroup\text{-}member(a))\ \&$
$(subgroup\text{-}member(Gidentity))\ \&$
$(\sim equal(INVERSE(a),Ginverse(a)))\ -->\ False$
$\langle proof \rangle$

lemma *GRP031-2*:
$(\forall X\ Y.\ product(X::'a,Y,multiply(X::'a,Y)))\ \&$
$(\forall X\ Y\ Z\ W.\ product(X::'a,Y,Z)\ \&\ product(X::'a,Y,W)\ -->\ equal(Z::'a,W))$
$\&$
$(\forall Y\ U\ Z\ X\ V\ W.\ product(X::'a,Y,U)\ \&\ product(Y::'a,Z,V)\ \&\ product(U::'a,Z,W)\ -->\ product(X::'a,V,W))\ \&$
$(\forall Y\ X\ V\ U\ Z\ W.\ product(X::'a,Y,U)\ \&\ product(Y::'a,Z,V)\ \&\ product(X::'a,V,W)\ -->\ product(U::'a,Z,W))\ \&$
$(\forall A.\ product(A::'a,INVERSE(A),identity))\ \&$
$(\forall A.\ product(A::'a,identity,A))\ \&$
$(\forall A.\ \sim product(A::'a,a,identity))\ -->\ False$

⟨*proof*⟩

lemma *GRP034-4*:
(∀ *X Y*. *product*(*X*::'*a*,*Y*,*multiply*(*X*::'*a*,*Y*))) &
(∀ *X*. *product*(*identity*::'*a*,*X*,*X*)) &
(∀ *X*. *product*(*X*::'*a*,*identity*,*X*)) &
(∀ *X*. *product*(*X*::'*a*,*INVERSE*(*X*),*identity*)) &
(∀ *Y U Z X V W*. *product*(*X*::'*a*,*Y*,*U*) & *product*(*Y*::'*a*,*Z*,*V*) & *product*(*U*::'*a*,*Z*,*W*)
−−> *product*(*X*::'*a*,*V*,*W*)) &
(∀ *Y X V U Z W*. *product*(*X*::'*a*,*Y*,*U*) & *product*(*Y*::'*a*,*Z*,*V*) & *product*(*X*::'*a*,*V*,*W*)
−−> *product*(*U*::'*a*,*Z*,*W*)) &
(∀ *B A C*. *subgroup-member*(*A*) & *subgroup-member*(*B*) & *product*(*B*::'*a*,*INVERSE*(*A*),*C*)
−−> *subgroup-member*(*C*)) &
(*subgroup-member*(*a*)) &
(~*subgroup-member*(*INVERSE*(*a*))) −−> *False*
⟨*proof*⟩

lemma *GRP047-2*:
(∀ *X*. *product*(*identity*::'*a*,*X*,*X*)) &
(∀ *X*. *product*(*INVERSE*(*X*),*X*,*identity*)) &
(∀ *X Y*. *product*(*X*::'*a*,*Y*,*multiply*(*X*::'*a*,*Y*))) &
(∀ *X Y Z W*. *product*(*X*::'*a*,*Y*,*Z*) & *product*(*X*::'*a*,*Y*,*W*) −−> *equal*(*Z*::'*a*,*W*))
&
(∀ *Y U Z X V W*. *product*(*X*::'*a*,*Y*,*U*) & *product*(*Y*::'*a*,*Z*,*V*) & *product*(*U*::'*a*,*Z*,*W*)
−−> *product*(*X*::'*a*,*V*,*W*)) &
(∀ *Y X V U Z W*. *product*(*X*::'*a*,*Y*,*U*) & *product*(*Y*::'*a*,*Z*,*V*) & *product*(*X*::'*a*,*V*,*W*)
−−> *product*(*U*::'*a*,*Z*,*W*)) &
(∀ *X W Z Y*. *equal*(*X*::'*a*,*Y*) & *product*(*W*::'*a*,*Z*,*X*) −−> *product*(*W*::'*a*,*Z*,*Y*))
&
(*equal*(*a*::'*a*,*b*)) &
(~*equal*(*multiply*(*c*::'*a*,*a*),*multiply*(*c*::'*a*,*b*))) −−> *False*
⟨*proof*⟩

lemma *GRP130-1-002*:
(*group-element*(*e-1*)) &
(*group-element*(*e-2*)) &
(~*equal*(*e-1*::'*a*,*e-2*)) &
(~*equal*(*e-2*::'*a*,*e-1*)) &
(∀ *X Y*. *group-element*(*X*) & *group-element*(*Y*) −−> *product*(*X*::'*a*,*Y*,*e-1*) |
product(*X*::'*a*,*Y*,*e-2*)) &
(∀ *X Y W Z*. *product*(*X*::'*a*,*Y*,*W*) & *product*(*X*::'*a*,*Y*,*Z*) −−> *equal*(*W*::'*a*,*Z*))
&
(∀ *X Y W Z*. *product*(*X*::'*a*,*W*,*Y*) & *product*(*X*::'*a*,*Z*,*Y*) −−> *equal*(*W*::'*a*,*Z*))
&
(∀ *Y X W Z*. *product*(*W*::'*a*,*Y*,*X*) & *product*(*Z*::'*a*,*Y*,*X*) −−> *equal*(*W*::'*a*,*Z*))
&

(∀ *Z1 Z2 Y X*. *product*(*X*::'*a*,*Y*,*Z1*) & *product*(*X*::'*a*,*Z1*,*Z2*) −−> *product*(*Z2*::'*a*,*Y*,*X*))
−−> *False*
⟨*proof*⟩

abbreviation *GRP004-0-ax INVERSE identity multiply equal* ≡
(∀ *X*. *equal*(*multiply*(*identity*::'*a*,*X*),*X*)) &
(∀ *X*. *equal*(*multiply*(*INVERSE*(*X*),*X*),*identity*)) &
(∀ *X Y Z*. *equal*(*multiply*(*multiply*(*X*::'*a*,*Y*),*Z*),*multiply*(*X*::'*a*,*multiply*(*Y*::'*a*,*Z*))))
&
(∀ *A B*. *equal*(*A*::'*a*,*B*) −−> *equal*(*INVERSE*(*A*),*INVERSE*(*B*))) &
(∀ *C D E*. *equal*(*C*::'*a*,*D*) −−> *equal*(*multiply*(*C*::'*a*,*E*),*multiply*(*D*::'*a*,*E*))) &
(∀ *F' H G*. *equal*(*F'*::'*a*,*G*) −−> *equal*(*multiply*(*H*::'*a*,*F'*),*multiply*(*H*::'*a*,*G*)))

abbreviation *GRP004-2-ax multiply least-upper-bound greatest-lower-bound equal*
≡
(∀ *Y X*. *equal*(*greatest-lower-bound*(*X*::'*a*,*Y*),*greatest-lower-bound*(*Y*::'*a*,*X*))) &
(∀ *Y X*. *equal*(*least-upper-bound*(*X*::'*a*,*Y*),*least-upper-bound*(*Y*::'*a*,*X*))) &
(∀ *X Y Z*. *equal*(*greatest-lower-bound*(*X*::'*a*,*greatest-lower-bound*(*Y*::'*a*,*Z*)),*greatest-lower-bound*(*greatest-lowe
&
(∀ *X Y Z*. *equal*(*least-upper-bound*(*X*::'*a*,*least-upper-bound*(*Y*::'*a*,*Z*)),*least-upper-bound*(*least-upper-bound*(*X:
&
(∀ *X*. *equal*(*least-upper-bound*(*X*::'*a*,*X*),*X*)) &
(∀ *X*. *equal*(*greatest-lower-bound*(*X*::'*a*,*X*),*X*)) &
(∀ *Y X*. *equal*(*least-upper-bound*(*X*::'*a*,*greatest-lower-bound*(*X*::'*a*,*Y*)),*X*)) &
(∀ *Y X*. *equal*(*greatest-lower-bound*(*X*::'*a*,*least-upper-bound*(*X*::'*a*,*Y*)),*X*)) &
(∀ *Y X Z*. *equal*(*multiply*(*X*::'*a*,*least-upper-bound*(*Y*::'*a*,*Z*)),*least-upper-bound*(*multiply*(*X*::'*a*,*Y*),*multiply*(*X:
&
(∀ *Y X Z*. *equal*(*multiply*(*X*::'*a*,*greatest-lower-bound*(*Y*::'*a*,*Z*)),*greatest-lower-bound*(*multiply*(*X*::'*a*,*Y*),*multip
&
(∀ *Y Z X*. *equal*(*multiply*(*least-upper-bound*(*Y*::'*a*,*Z*),*X*),*least-upper-bound*(*multiply*(*Y*::'*a*,*X*),*multiply*(*Z*::'*a*,
&
(∀ *Y Z X*. *equal*(*multiply*(*greatest-lower-bound*(*Y*::'*a*,*Z*),*X*),*greatest-lower-bound*(*multiply*(*Y*::'*a*,*X*),*multiply*(
&
(∀ *A B C*. *equal*(*A*::'*a*,*B*) −−> *equal*(*greatest-lower-bound*(*A*::'*a*,*C*),*greatest-lower-bound*(*B*::'*a*,*C*)))
&
(∀ *A C B*. *equal*(*A*::'*a*,*B*) −−> *equal*(*greatest-lower-bound*(*C*::'*a*,*A*),*greatest-lower-bound*(*C*::'*a*,*B*)))
&
(∀ *A B C*. *equal*(*A*::'*a*,*B*) −−> *equal*(*least-upper-bound*(*A*::'*a*,*C*),*least-upper-bound*(*B*::'*a*,*C*)))
&
(∀ *A C B*. *equal*(*A*::'*a*,*B*) −−> *equal*(*least-upper-bound*(*C*::'*a*,*A*),*least-upper-bound*(*C*::'*a*,*B*)))
&
(∀ *A B C*. *equal*(*A*::'*a*,*B*) −−> *equal*(*multiply*(*A*::'*a*,*C*),*multiply*(*B*::'*a*,*C*))) &
(∀ *A C B*. *equal*(*A*::'*a*,*B*) −−> *equal*(*multiply*(*C*::'*a*,*A*),*multiply*(*C*::'*a*,*B*)))

lemma *GRP156-1*:
EQU001-0-ax equal &
GRP004-0-ax INVERSE identity multiply equal &
GRP004-2-ax multiply least-upper-bound greatest-lower-bound equal &

(*equal*(*least-upper-bound*(*a*::'*a*,*b*),*b*)) &
(~*equal*(*greatest-lower-bound*(*multiply*(*a*::'*a*,*c*),*multiply*(*b*::'*a*,*c*)),*multiply*(*a*::'*a*,*c*)))
−−> *False*
⟨*proof*⟩

lemma *GRP168-1*:
EQU001-0-ax equal &
GRP004-0-ax INVERSE identity multiply equal &
GRP004-2-ax multiply least-upper-bound greatest-lower-bound equal &
(*equal*(*least-upper-bound*(*a*::'*a*,*b*),*b*)) &
(~*equal*(*least-upper-bound*(*multiply*(*INVERSE*(*c*),*multiply*(*a*::'*a*,*c*)),*multiply*(*INVERSE*(*c*),*multiply*(*b*::'*a*,*c*))
−−> *False*
⟨*proof*⟩

abbreviation *HEN002-0-ax identity Zero Divide equal mless-equal* ≡
(∀ *X Y*. *mless-equal*(*X*::'*a*,*Y*) −−> *equal*(*Divide*(*X*::'*a*,*Y*),*Zero*)) &
(∀ *X Y*. *equal*(*Divide*(*X*::'*a*,*Y*),*Zero*) −−> *mless-equal*(*X*::'*a*,*Y*)) &
(∀ *Y X*. *mless-equal*(*Divide*(*X*::'*a*,*Y*),*X*)) &
(∀ *X Y Z*. *mless-equal*(*Divide*(*Divide*(*X*::'*a*,*Z*),*Divide*(*Y*::'*a*,*Z*)),*Divide*(*Divide*(*X*::'*a*,*Y*),*Z*)))
&
(∀ *X*. *mless-equal*(*Zero*::'*a*,*X*)) &
(∀ *X Y*. *mless-equal*(*X*::'*a*,*Y*) & *mless-equal*(*Y*::'*a*,*X*) −−> *equal*(*X*::'*a*,*Y*)) &
(∀ *X*. *mless-equal*(*X*::'*a*,*identity*))

abbreviation *HEN002-0-eq mless-equal Divide equal* ≡
(∀ *A B C*. *equal*(*A*::'*a*,*B*) −−> *equal*(*Divide*(*A*::'*a*,*C*),*Divide*(*B*::'*a*,*C*))) &
(∀ *D F' E*. *equal*(*D*::'*a*,*E*) −−> *equal*(*Divide*(*F'*::'*a*,*D*),*Divide*(*F'*::'*a*,*E*))) &
(∀ *G H I'*. *equal*(*G*::'*a*,*H*) & *mless-equal*(*G*::'*a*,*I'*) −−> *mless-equal*(*H*::'*a*,*I'*)) &
(∀ *J L K'*. *equal*(*J*::'*a*,*K'*) & *mless-equal*(*L*::'*a*,*J*) −−> *mless-equal*(*L*::'*a*,*K'*))

lemma *HEN003-3*:
EQU001-0-ax equal &
HEN002-0-ax identity Zero Divide equal mless-equal &
HEN002-0-eq mless-equal Divide equal &
(~*equal*(*Divide*(*a*::'*a*,*a*),*Zero*)) −−> *False*
⟨*proof*⟩

lemma *HEN007-2*:
EQU001-0-ax equal &
(∀ *X Y*. *mless-equal*(*X*::'*a*,*Y*) −−> *quotient*(*X*::'*a*,*Y*,*Zero*)) &
(∀ *X Y*. *quotient*(*X*::'*a*,*Y*,*Zero*) −−> *mless-equal*(*X*::'*a*,*Y*)) &
(∀ *Y Z X*. *quotient*(*X*::'*a*,*Y*,*Z*) −−> *mless-equal*(*Z*::'*a*,*X*)) &
(∀ *Y X V3 V2 V1 Z V4 V5*. *quotient*(*X*::'*a*,*Y*,*V1*) & *quotient*(*Y*::'*a*,*Z*,*V2*) &
quotient(*X*::'*a*,*Z*,*V3*) & *quotient*(*V3*::'*a*,*V2*,*V4*) & *quotient*(*V1*::'*a*,*Z*,*V5*) −−>
mless-equal(*V4*::'*a*,*V5*)) &
(∀ *X*. *mless-equal*(*Zero*::'*a*,*X*)) &

$(\forall X\ Y.\ mless\text{-}equal(X::'a,Y)\ \&\ mless\text{-}equal(Y::'a,X)\ --> equal(X::'a,Y))\ \&$
$(\forall X.\ mless\text{-}equal(X::'a,identity))\ \&$
$(\forall X\ Y.\ quotient(X::'a,Y,Divide(X::'a,Y)))\ \&$
$(\forall X\ Y\ Z\ W.\ quotient(X::'a,Y,Z)\ \&\ quotient(X::'a,Y,W)\ --> equal(Z::'a,W))$
$\&$
$(\forall X\ Y\ W\ Z.\ equal(X::'a,Y)\ \&\ quotient(X::'a,W,Z)\ --> quotient(Y::'a,W,Z))$
$\&$
$(\forall X\ W\ Y\ Z.\ equal(X::'a,Y)\ \&\ quotient(W::'a,X,Z)\ --> quotient(W::'a,Y,Z))$
$\&$
$(\forall X\ W\ Z\ Y.\ equal(X::'a,Y)\ \&\ quotient(W::'a,Z,X)\ --> quotient(W::'a,Z,Y))$
$\&$
$(\forall X\ Z\ Y.\ equal(X::'a,Y)\ \&\ mless\text{-}equal(Z::'a,X)\ --> mless\text{-}equal(Z::'a,Y))\ \&$
$(\forall X\ Y\ Z.\ equal(X::'a,Y)\ \&\ mless\text{-}equal(X::'a,Z)\ --> mless\text{-}equal(Y::'a,Z))\ \&$
$(\forall X\ Y\ W.\ equal(X::'a,Y)\ --> equal(Divide(X::'a,W),Divide(Y::'a,W)))\ \&$
$(\forall X\ W\ Y.\ equal(X::'a,Y)\ --> equal(Divide(W::'a,X),Divide(W::'a,Y)))\ \&$
$(\forall X.\ quotient(X::'a,identity,Zero))\ \&$
$(\forall X.\ quotient(Zero::'a,X,Zero))\ \&$
$(\forall X.\ quotient(X::'a,X,Zero))\ \&$
$(\forall X.\ quotient(X::'a,Zero,X))\ \&$
$(\forall Y\ X\ Z.\ mless\text{-}equal(X::'a,Y)\ \&\ mless\text{-}equal(Y::'a,Z)\ --> mless\text{-}equal(X::'a,Z))$
$\&$
$(\forall W1\ X\ Z\ W2\ Y.\ quotient(X::'a,Y,W1)\ \&\ mless\text{-}equal(W1::'a,Z)\ \&\ quotient(X::'a,Z,W2)$
$--> mless\text{-}equal(W2::'a,Y))\ \&$
$(mless\text{-}equal(x::'a,y))\ \&$
$(quotient(z::'a,y,zQy))\ \&$
$(quotient(z::'a,x,zQx))\ \&$
$(\sim mless\text{-}equal(zQy::'a,zQx))\ --> False$
$\langle proof \rangle$

lemma *HEN008-4*:
EQU001-0-ax equal &
HEN002-0-ax identity Zero Divide equal mless-equal &
HEN002-0-eq mless-equal Divide equal &
$(\forall X.\ equal(Divide(X::'a,identity),Zero))\ \&$
$(\forall X.\ equal(Divide(Zero::'a,X),Zero))\ \&$
$(\forall X.\ equal(Divide(X::'a,X),Zero))\ \&$
$(equal(Divide(a::'a,Zero),a))\ \&$
$(\forall Y\ X\ Z.\ mless\text{-}equal(X::'a,Y)\ \&\ mless\text{-}equal(Y::'a,Z)\ --> mless\text{-}equal(X::'a,Z))$
$\&$
$(\forall X\ Z\ Y.\ mless\text{-}equal(Divide(X::'a,Y),Z)\ --> mless\text{-}equal(Divide(X::'a,Z),Y))$
$\&$
$(\forall Y\ Z\ X.\ mless\text{-}equal(X::'a,Y)\ --> mless\text{-}equal(Divide(Z::'a,Y),Divide(Z::'a,X)))$
$\&$
$(mless\text{-}equal(a::'a,b))\ \&$
$(\sim mless\text{-}equal(Divide(a::'a,c),Divide(b::'a,c)))\ --> False$
$\langle proof \rangle$

lemma *HEN009-5*:
EQU001-0-ax equal &
(∀ Y X. equal(Divide(Divide(X::'a,Y),X),Zero)) &
(∀ X Y Z. equal(Divide(Divide(Divide(X::'a,Z),Divide(Y::'a,Z)),Divide(Divide(X::'a,Y),Z)),Zero))
&
(∀ X. equal(Divide(Zero::'a,X),Zero)) &
(∀ X Y. equal(Divide(X::'a,Y),Zero) & equal(Divide(Y::'a,X),Zero) −−> equal(X::'a,Y))
&
(∀ X. equal(Divide(X::'a,identity),Zero)) &
(∀ A B C. equal(A::'a,B) −−> equal(Divide(A::'a,C),Divide(B::'a,C))) &
(∀ D F' E. equal(D::'a,E) −−> equal(Divide(F'::'a,D),Divide(F'::'a,E))) &
(∀ Y X Z. equal(Divide(X::'a,Y),Zero) & equal(Divide(Y::'a,Z),Zero) −−>
equal(Divide(X::'a,Z),Zero)) &
(∀ X Z Y. equal(Divide(Divide(X::'a,Y),Z),Zero) −−> equal(Divide(Divide(X::'a,Z),Y),Zero))
&
(∀ Y Z X. equal(Divide(X::'a,Y),Zero) −−> equal(Divide(Divide(Z::'a,Y),Divide(Z::'a,X)),Zero))
&
(~equal(Divide(identity::'a,a),Divide(identity::'a,Divide(identity::'a,Divide(identity::'a,a)))))
&
(equal(Divide(identity::'a,a),b)) &
(equal(Divide(identity::'a,b),c)) &
(equal(Divide(identity::'a,c),d)) &
(~equal(b::'a,d)) −−> False
⟨*proof*⟩

lemma *HEN012-3*:
EQU001-0-ax equal &
HEN002-0-ax identity Zero Divide equal mless-equal &
HEN002-0-eq mless-equal Divide equal &
(~mless-equal(a::'a,a)) −−> False
⟨*proof*⟩

lemma *LCL010-1*:
(∀ X Y. is-a-theorem(equivalent(X::'a,Y)) & is-a-theorem(X) −−> is-a-theorem(Y))
&
(∀ X Z Y. is-a-theorem(equivalent(equivalent(X::'a,Y),equivalent(equivalent(X::'a,Z),equivalent(Z::'a,Y)))))
&
(~is-a-theorem(equivalent(equivalent(a::'a,b),equivalent(equivalent(c::'a,b),equivalent(a::'a,c)))))
−−> False
⟨*proof*⟩

lemma *LCL077-2*:
(∀ X Y. is-a-theorem(implies(X,Y)) & is-a-theorem(X) −−> is-a-theorem(Y))
&
(∀ Y X. is-a-theorem(implies(X,implies(Y,X)))) &

(∀ Y X Z. is-a-theorem(implies(implies(X,implies(Y,Z)),implies(implies(X,Y),implies(X,Z)))))
&
(∀ Y X. is-a-theorem(implies(implies(not(X),not(Y)),implies(Y,X)))) &
(∀ X2 X1 X3. is-a-theorem(implies(X1,X2)) & is-a-theorem(implies(X2,X3))
−−> is-a-theorem(implies(X1,X3))) &
(~is-a-theorem(implies(not(not(a)),a))) −−> False
⟨proof⟩

lemma *LCL082-1*:
(∀ X Y. is-a-theorem(implies(X::'a,Y)) & is-a-theorem(X) −−> is-a-theorem(Y))
&
(∀ Y Z U X. is-a-theorem(implies(implies(implies(X::'a,Y),Z),implies(implies(Z::'a,X),implies(U::'a,X)))))
&
(~is-a-theorem(implies(a::'a,implies(b::'a,a)))) −−> False
⟨proof⟩

lemma *LCL111-1*:
(∀ X Y. is-a-theorem(implies(X,Y)) & is-a-theorem(X) −−> is-a-theorem(Y))
&
(∀ Y X. is-a-theorem(implies(X,implies(Y,X)))) &
(∀ Y X Z. is-a-theorem(implies(implies(X,Y),implies(implies(Y,Z),implies(X,Z)))))
&
(∀ Y X. is-a-theorem(implies(implies(implies(X,Y),Y),implies(implies(Y,X),X))))
&
(∀ Y X. is-a-theorem(implies(implies(not(X),not(Y)),implies(Y,X)))) &
(~is-a-theorem(implies(implies(a,b),implies(implies(c,a),implies(c,b))))) −−> False
⟨proof⟩

lemma *LCL143-1*:
(∀ X. equal(X,X)) &
(∀ Y X. equal(X,Y) −−> equal(Y,X)) &
(∀ Y X Z. equal(X,Y) & equal(Y,Z) −−> equal(X,Z)) &
(∀ X. equal(implies(true,X),X)) &
(∀ Y X Z. equal(implies(implies(X,Y),implies(implies(Y,Z),implies(X,Z))),true))
&
(∀ Y X. equal(implies(implies(X,Y),Y),implies(implies(Y,X),X))) &
(∀ Y X. equal(implies(implies(not(X),not(Y)),implies(Y,X)),true)) &
(∀ A B C. equal(A,B) −−> equal(implies(A,C),implies(B,C))) &
(∀ D F' E. equal(D,E) −−> equal(implies(F',D),implies(F',E))) &
(∀ G H. equal(G,H) −−> equal(not(G),not(H))) &
(∀ X Y. equal(big-V(X,Y),implies(implies(X,Y),Y))) &
(∀ X Y. equal(big-hat(X,Y),not(big-V(not(X),not(Y))))) &
(∀ X Y. ordered(X,Y) −−> equal(implies(X,Y),true)) &
(∀ X Y. equal(implies(X,Y),true) −−> ordered(X,Y)) &
(∀ A B C. equal(A,B) −−> equal(big-V(A,C),big-V(B,C))) &
(∀ D F' E. equal(D,E) −−> equal(big-V(F',D),big-V(F',E))) &

$(\forall\ G\ H\ I'.\ equal(G,H)\ \text{--}\!\!>\ equal(big\text{-}hat(G,I'),big\text{-}hat(H,I')))\ \&$
$(\forall\ J\ L\ K'.\ equal(J,K')\ \text{--}\!\!>\ equal(big\text{-}hat(L,J),big\text{-}hat(L,K')))\ \&$
$(\forall\ M\ N\ O'.\ equal(M,N)\ \&\ ordered(M,O')\ \text{--}\!\!>\ ordered(N,O'))\ \&$
$(\forall\ P\ R\ Q.\ equal(P,Q)\ \&\ ordered(R,P)\ \text{--}\!\!>\ ordered(R,Q))\ \&$
$(ordered(x,y))\ \&$
$(\sim ordered(implies(z,x),implies(z,y)))\ \text{--}\!\!>\ False$
$\langle proof \rangle$

lemma *LCL182-1*:
$(\forall\ A.\ axiom(or(not(or(A,A)),A)))\ \&$
$(\forall\ B\ A.\ axiom(or(not(A),or(B,A))))\ \&$
$(\forall\ B\ A.\ axiom(or(not(or(A,B)),or(B,A))))\ \&$
$(\forall\ B\ A\ C.\ axiom(or(not(or(A,or(B,C))),or(B,or(A,C)))))\ \&$
$(\forall\ A\ C\ B.\ axiom(or(not(or(not(A),B)),or(not(or(C,A)),or(C,B)))))\ \&$
$(\forall\ X.\ axiom(X)\ \text{--}\!\!>\ theorem(X))\ \&$
$(\forall\ X\ Y.\ axiom(or(not(Y),X))\ \&\ theorem(Y)\ \text{--}\!\!>\ theorem(X))\ \&$
$(\forall\ X\ Y\ Z.\ axiom(or(not(X),Y))\ \&\ theorem(or(not(Y),Z))\ \text{--}\!\!>\ theorem(or(not(X),Z)))$
$\&$
$(\sim theorem(or(not(or(not(p),q)),or(not(not(q)),not(p)))))\ \text{--}\!\!>\ False$
$\langle proof \rangle$

lemma *LCL200-1*:
$(\forall\ A.\ axiom(or(not(or(A,A)),A)))\ \&$
$(\forall\ B\ A.\ axiom(or(not(A),or(B,A))))\ \&$
$(\forall\ B\ A.\ axiom(or(not(or(A,B)),or(B,A))))\ \&$
$(\forall\ B\ A\ C.\ axiom(or(not(or(A,or(B,C))),or(B,or(A,C)))))\ \&$
$(\forall\ A\ C\ B.\ axiom(or(not(or(not(A),B)),or(not(or(C,A)),or(C,B)))))\ \&$
$(\forall\ X.\ axiom(X)\ \text{--}\!\!>\ theorem(X))\ \&$
$(\forall\ X\ Y.\ axiom(or(not(Y),X))\ \&\ theorem(Y)\ \text{--}\!\!>\ theorem(X))\ \&$
$(\forall\ X\ Y\ Z.\ axiom(or(not(X),Y))\ \&\ theorem(or(not(Y),Z))\ \text{--}\!\!>\ theorem(or(not(X),Z)))$
$\&$
$(\sim theorem(or(not(not(or(p,q))),not(q))))\ \text{--}\!\!>\ False$
$\langle proof \rangle$

lemma *LCL215-1*:
$(\forall\ A.\ axiom(or(not(or(A,A)),A)))\ \&$
$(\forall\ B\ A.\ axiom(or(not(A),or(B,A))))\ \&$
$(\forall\ B\ A.\ axiom(or(not(or(A,B)),or(B,A))))\ \&$
$(\forall\ B\ A\ C.\ axiom(or(not(or(A,or(B,C))),or(B,or(A,C)))))\ \&$
$(\forall\ A\ C\ B.\ axiom(or(not(or(not(A),B)),or(not(or(C,A)),or(C,B)))))\ \&$
$(\forall\ X.\ axiom(X)\ \text{--}\!\!>\ theorem(X))\ \&$
$(\forall\ X\ Y.\ axiom(or(not(Y),X))\ \&\ theorem(Y)\ \text{--}\!\!>\ theorem(X))\ \&$
$(\forall\ X\ Y\ Z.\ axiom(or(not(X),Y))\ \&\ theorem(or(not(Y),Z))\ \text{--}\!\!>\ theorem(or(not(X),Z)))$
$\&$
$(\sim theorem(or(not(or(not(p),q)),or(not(or(p,q)),q))))\ \text{--}\!\!>\ False$
$\langle proof \rangle$

lemma *LCL230-2*:
(*q* −−> *p* | *r*) &
(~*p*) &
(*q*) &
(~*r*) −−> *False*
⟨*proof*⟩

lemma *LDA003-1*:
EQU001-0-ax equal &
(∀ *Y X Z*. *equal*(*f*(*X*::'*a*,*f*(*Y*::'*a*,*Z*)),*f*(*f*(*X*::'*a*,*Y*),*f*(*X*::'*a*,*Z*)))) &
(∀ *X Y*. *left*(*X*::'*a*,*f*(*X*::'*a*,*Y*))) &
(∀ *Y X Z*. *left*(*X*::'*a*,*Y*) & *left*(*Y*::'*a*,*Z*) −−> *left*(*X*::'*a*,*Z*)) &
(*equal*(*num2*::'*a*,*f*(*num1*::'*a*,*num1*))) &
(*equal*(*num3*::'*a*,*f*(*num2*::'*a*,*num1*))) &
(*equal*(*u*::'*a*,*f*(*num2*::'*a*,*num2*))) &
(∀ *A B C*. *equal*(*A*::'*a*,*B*) −−> *equal*(*f*(*A*::'*a*,*C*),*f*(*B*::'*a*,*C*))) &
(∀ *D F' E*. *equal*(*D*::'*a*,*E*) −−> *equal*(*f*(*F'*::'*a*,*D*),*f*(*F'*::'*a*,*E*))) &
(∀ *G H I'*. *equal*(*G*::'*a*,*H*) & *left*(*G*::'*a*,*I'*) −−> *left*(*H*::'*a*,*I'*)) &
(∀ *J L K'*. *equal*(*J*::'*a*,*K'*) & *left*(*L*::'*a*,*J*) −−> *left*(*L*::'*a*,*K'*)) &
(~*left*(*num3*::'*a*,*u*)) −−> *False*
⟨*proof*⟩

lemma *MSC002-1*:
(*at*(*something*::'*a*,*here*,*now*)) &
(∀ *Place Situation*. *hand-at*(*Place*::'*a*,*Situation*) −−> *hand-at*(*Place*::'*a*,*let-go*(*Situation*)))
&
(∀ *Place Another-place Situation*. *hand-at*(*Place*::'*a*,*Situation*) −−> *hand-at*(*Another-place*::'*a*,*go*(*Another-pl*
&
(∀ *Thing Situation*. ~*held*(*Thing*::'*a*,*let-go*(*Situation*))) &
(∀ *Situation Thing*. *at*(*Thing*::'*a*,*here*,*Situation*) −−> *red*(*Thing*)) &
(∀ *Thing Place Situation*. *at*(*Thing*::'*a*,*Place*,*Situation*) −−> *at*(*Thing*::'*a*,*Place*,*let-go*(*Situation*)))
&
(∀ *Thing Place Situation*. *at*(*Thing*::'*a*,*Place*,*Situation*) −−> *at*(*Thing*::'*a*,*Place*,*pick-up*(*Situation*)))
&
(∀ *Thing Place Situation*. *at*(*Thing*::'*a*,*Place*,*Situation*) −−> *grabbed*(*Thing*::'*a*,*pick-up*(*go*(*Place*::'*a*,*let-go*(*Si*
&
(∀ *Thing Situation*. *red*(*Thing*) & *put*(*Thing*::'*a*,*there*,*Situation*) −−> *answer*(*Situation*))
&
(∀ *Place Thing Another-place Situation*. *at*(*Thing*::'*a*,*Place*,*Situation*) & *grabbed*(*Thing*::'*a*,*Situation*)
−−> *put*(*Thing*::'*a*,*Another-place*,*go*(*Another-place*::'*a*,*Situation*))) &
(∀ *Thing Place Another-place Situation*. *at*(*Thing*::'*a*,*Place*,*Situation*) −−> *held*(*Thing*::'*a*,*Situation*)
| *at*(*Thing*::'*a*,*Place*,*go*(*Another-place*::'*a*,*Situation*))) &
(∀ *One-place Thing Place Situation*. *hand-at*(*One-place*::'*a*,*Situation*) & *held*(*Thing*::'*a*,*Situation*)
−−> *at*(*Thing*::'*a*,*Place*,*go*(*Place*::'*a*,*Situation*))) &

(∀ *Place Thing Situation*. *hand-at*(*Place*::'*a*,*Situation*) & *at*(*Thing*::'*a*,*Place*,*Situation*)
−−> *held*(*Thing*::'*a*,*pick-up*(*Situation*))) &
(∀ *Situation*. ~*answer*(*Situation*)) −−> *False*
⟨*proof*⟩

lemma *MSC003-1*:
(∀ *Number-of-small-parts Small-part Big-part Number-of-mid-parts Mid-part*. *has-parts*(*Big-part*::'*a*,*Number-o
−−> *in'*(*object-in*(*Big-part*::'*a*,*Mid-part*,*Small-part*,*Number-of-mid-parts*,*Number-of-small-parts*),*Mid-part*)
| *has-parts*(*Big-part*::'*a*,*mtimes*(*Number-of-mid-parts*::'*a*,*Number-of-small-parts*),*Small-part*))
&
(∀ *Big-part Mid-part Number-of-mid-parts Number-of-small-parts Small-part*. *has-parts*(*Big-part*::'*a*,*Number-o
& *has-parts*(*object-in*(*Big-part*::'*a*,*Mid-part*,*Small-part*,*Number-of-mid-parts*,*Number-of-small-parts*),*Number-
−−> *has-parts*(*Big-part*::'*a*,*mtimes*(*Number-of-mid-parts*::'*a*,*Number-of-small-parts*),*Small-part*))
&
(*in'*(*john*::'*a*,*boy*)) &
(∀ *X*. *in'*(*X*::'*a*,*boy*) −−> *in'*(*X*::'*a*,*human*)) &
(∀ *X*. *in'*(*X*::'*a*,*hand*) −−> *has-parts*(*X*::'*a*,*num5*,*fingers*)) &
(∀ *X*. *in'*(*X*::'*a*,*human*) −−> *has-parts*(*X*::'*a*,*num2*,*arm*)) &
(∀ *X*. *in'*(*X*::'*a*,*arm*) −−> *has-parts*(*X*::'*a*,*num1*,*hand*)) &
(~*has-parts*(*john*::'*a*,*mtimes*(*num2*::'*a*,*num1*),*hand*)) −−> *False*
⟨*proof*⟩

lemma *MSC004-1*:
(∀ *Number-of-small-parts Small-part Big-part Number-of-mid-parts Mid-part*. *has-parts*(*Big-part*::'*a*,*Number-of
−−> *in'*(*object-in*(*Big-part*::'*a*,*Mid-part*,*Small-part*,*Number-of-mid-parts*,*Number-of-small-parts*),*Mid-part*)
| *has-parts*(*Big-part*::'*a*,*mtimes*(*Number-of-mid-parts*::'*a*,*Number-of-small-parts*),*Small-part*))
&
(∀ *Big-part Mid-part Number-of-mid-parts Number-of-small-parts Small-part*. *has-parts*(*Big-part*::'*a*,*Number-o
& *has-parts*(*object-in*(*Big-part*::'*a*,*Mid-part*,*Small-part*,*Number-of-mid-parts*,*Number-of-small-parts*),*Number-
−−> *has-parts*(*Big-part*::'*a*,*mtimes*(*Number-of-mid-parts*::'*a*,*Number-of-small-parts*),*Small-part*))
&
(*in'*(*john*::'*a*,*boy*)) &
(∀ *X*. *in'*(*X*::'*a*,*boy*) −−> *in'*(*X*::'*a*,*human*)) &
(∀ *X*. *in'*(*X*::'*a*,*hand*) −−> *has-parts*(*X*::'*a*,*num5*,*fingers*)) &
(∀ *X*. *in'*(*X*::'*a*,*human*) −−> *has-parts*(*X*::'*a*,*num2*,*arm*)) &
(∀ *X*. *in'*(*X*::'*a*,*arm*) −−> *has-parts*(*X*::'*a*,*num1*,*hand*)) &
(~*has-parts*(*john*::'*a*,*mtimes*(*mtimes*(*num2*::'*a*,*num1*),*num5*),*fingers*)) −−> *False*
⟨*proof*⟩

lemma *MSC005-1*:
(*value*(*truth*::'*a*,*truth*)) &
(*value*(*falsity*::'*a*,*falsity*)) &
(∀ *X Y*. *value*(*X*::'*a*,*truth*) & *value*(*Y*::'*a*,*truth*) −−> *value*(*xor*(*X*::'*a*, *Y*),*falsity*))
&
(∀ *X Y*. *value*(*X*::'*a*,*truth*) & *value*(*Y*::'*a*,*falsity*) −−> *value*(*xor*(*X*::'*a*, *Y*),*truth*))
&

$(\forall\, X\ Y.\ value(X::'a,falsity)\ \&\ value(Y::'a,truth) \longrightarrow value(xor(X::'a,Y),truth))$
$\&$
$(\forall\, X\ Y.\ value(X::'a,falsity)\ \&\ value(Y::'a,falsity) \longrightarrow value(xor(X::'a,Y),falsity))$
$\&$
$(\forall\, Value.\ \sim value(xor(xor(xor(xor(truth::'a,falsity),falsity),truth),falsity),Value))$
$\longrightarrow False$
$\langle proof \rangle$

lemma *MSC006-1*:
$(\forall\, Y\ X\ Z.\ p(X::'a,Y)\ \&\ p(Y::'a,Z) \longrightarrow p(X::'a,Z))\ \&$
$(\forall\, Y\ X\ Z.\ q(X::'a,Y)\ \&\ q(Y::'a,Z) \longrightarrow q(X::'a,Z))\ \&$
$(\forall\, Y\ X.\ q(X::'a,Y) \longrightarrow q(Y::'a,X))\ \&$
$(\forall\, X\ Y.\ p(X::'a,Y) \mid q(X::'a,Y))\ \&$
$(\sim p(a::'a,b))\ \&$
$(\sim q(c::'a,d)) \longrightarrow False$
$\langle proof \rangle$

lemma *NUM001-1*:
$(\forall\, A.\ equal(A::'a,A))\ \&$
$(\forall\, B\ A\ C.\ equal(A::'a,B)\ \&\ equal(B::'a,C) \longrightarrow equal(A::'a,C))\ \&$
$(\forall\, B\ A.\ equal(add(A::'a,B),add(B::'a,A)))\ \&$
$(\forall\, A\ B\ C.\ equal(add(A::'a,add(B::'a,C)),add(add(A::'a,B),C)))\ \&$
$(\forall\, B\ A.\ equal(subtract(add(A::'a,B),B),A))\ \&$
$(\forall\, A\ B.\ equal(A::'a,subtract(add(A::'a,B),B)))\ \&$
$(\forall\, A\ C\ B.\ equal(add(subtract(A::'a,B),C),subtract(add(A::'a,C),B)))\ \&$
$(\forall\, A\ C\ B.\ equal(subtract(add(A::'a,B),C),add(subtract(A::'a,C),B)))\ \&$
$(\forall\, A\ C\ B\ D.\ equal(A::'a,B)\ \&\ equal(C::'a,add(A::'a,D)) \longrightarrow equal(C::'a,add(B::'a,D)))$
$\&$
$(\forall\, A\ C\ D\ B.\ equal(A::'a,B)\ \&\ equal(C::'a,add(D::'a,A)) \longrightarrow equal(C::'a,add(D::'a,B)))$
$\&$
$(\forall\, A\ C\ B\ D.\ equal(A::'a,B)\ \&\ equal(C::'a,subtract(A::'a,D)) \longrightarrow equal(C::'a,subtract(B::'a,D)))$
$\&$
$(\forall\, A\ C\ D\ B.\ equal(A::'a,B)\ \&\ equal(C::'a,subtract(D::'a,A)) \longrightarrow equal(C::'a,subtract(D::'a,B)))$
$\&$
$(\sim equal(add(add(a::'a,b),c),add(a::'a,add(b::'a,c)))) \longrightarrow False$
$\langle proof \rangle$

abbreviation *NUM001-0-ax multiply successor num0 add equal* $\equiv$
$(\forall\, A.\ equal(add(A::'a,num0),A))\ \&$
$(\forall\, A\ B.\ equal(add(A::'a,successor(B)),successor(add(A::'a,B))))\ \&$
$(\forall\, A.\ equal(multiply(A::'a,num0),num0))\ \&$
$(\forall\, B\ A.\ equal(multiply(A::'a,successor(B)),add(multiply(A::'a,B),A)))\ \&$
$(\forall\, A\ B.\ equal(successor(A),successor(B)) \longrightarrow equal(A::'a,B))\ \&$
$(\forall\, A\ B.\ equal(A::'a,B) \longrightarrow equal(successor(A),successor(B)))$

abbreviation *NUM001-1-ax predecessor-of-1st-minus-2nd successor add equal mless*
$\equiv$

$(\forall\ A\ C\ B.\ mless(A::'a,B)\ \&\ mless(C::'a,A)\ --{>}\ mless(C::'a,B))\ \&$
$(\forall\ A\ B\ C.\ equal(add(successor(A),B),C)\ --{>}\ mless(B::'a,C))\ \&$
$(\forall\ A\ B.\ mless(A::'a,B)\ --{>}\ equal(add(successor(predecessor\text{-}of\text{-}1st\text{-}minus\text{-}2nd(B::'a,A)),A),B))$

abbreviation *NUM001-2-ax equal mless divides* $\equiv$
$(\forall\ A\ B.\ divides(A::'a,B)\ --{>}\ mless(A::'a,B)\ |\ equal(A::'a,B))\ \&$
$(\forall\ A\ B.\ mless(A::'a,B)\ --{>}\ divides(A::'a,B))\ \&$
$(\forall\ A\ B.\ equal(A::'a,B)\ --{>}\ divides(A::'a,B))$

lemma *NUM021-1*:
EQU001-0-ax equal &
NUM001-0-ax multiply successor num0 add equal &
NUM001-1-ax predecessor-of-1st-minus-2nd successor add equal mless &
NUM001-2-ax equal mless divides &
$(mless(b::'a,c))\ \&$
$(\sim mless(b::'a,a))\ \&$
$(divides(c::'a,a))\ \&$
$(\forall\ A.\ \sim equal(successor(A),num0))\ --{>}\ False$
$\langle proof \rangle$

lemma *NUM024-1*:
EQU001-0-ax equal &
NUM001-0-ax multiply successor num0 add equal &
NUM001-1-ax predecessor-of-1st-minus-2nd successor add equal mless &
$(\forall\ B\ A.\ equal(add(A::'a,B),add(B::'a,A)))\ \&$
$(\forall\ B\ A\ C.\ equal(add(A::'a,B),add(C::'a,B))\ --{>}\ equal(A::'a,C))\ \&$
$(mless(a::'a,a))\ \&$
$(\forall\ A.\ \sim equal(successor(A),num0))\ --{>}\ False$
$\langle proof \rangle$

abbreviation *SET004-0-ax not-homomorphism2 not-homomorphism1 homomorphism compatible operation cantor diagonalise subset-relation one-to-one choice apply regular function identity-relation single-valued-class compos powerClass sum-class omega inductive successor-relation successor image' rng domain range-of INVERSE flip rot domain-of null-class restrct difference union complement intersection element-relation second first cross-product ordered-pair singleton unordered-pair equal universal-class not-subclass-element member subclass* $\equiv$
$(\forall\ X\ U\ Y.\ subclass(X::'a,Y)\ \&\ member(U::'a,X)\ --{>}\ member(U::'a,Y))\ \&$
$(\forall\ X\ Y.\ member(not\text{-}subclass\text{-}element(X::'a,Y),X)\ |\ subclass(X::'a,Y))\ \&$
$(\forall\ X\ Y.\ member(not\text{-}subclass\text{-}element(X::'a,Y),Y)\ --{>}\ subclass(X::'a,Y))\ \&$
$(\forall\ X.\ subclass(X::'a,universal\text{-}class))\ \&$
$(\forall\ X\ Y.\ equal(X::'a,Y)\ --{>}\ subclass(X::'a,Y))\ \&$
$(\forall\ Y\ X.\ equal(X::'a,Y)\ --{>}\ subclass(Y::'a,X))\ \&$
$(\forall\ X\ Y.\ subclass(X::'a,Y)\ \&\ subclass(Y::'a,X)\ --{>}\ equal(X::'a,Y))\ \&$
$(\forall\ X\ U\ Y.\ member(U::'a,unordered\text{-}pair(X::'a,Y))\ --{>}\ equal(U::'a,X)\ |\ equal(U::'a,Y))$

$$
\begin{array}{l}
\& \\
\quad (\forall\, X\ Y.\ member(X::'a, universal\text{-}class) \longrightarrow member(X::'a, unordered\text{-}pair(X::'a, Y))) \\
\& \\
\quad (\forall\, X\ Y.\ member(Y::'a, universal\text{-}class) \longrightarrow member(Y::'a, unordered\text{-}pair(X::'a, Y))) \\
\& \\
\quad (\forall\, X\ Y.\ member(unordered\text{-}pair(X::'a, Y), universal\text{-}class)) \ \& \\
\quad (\forall\, X.\ equal(unordered\text{-}pair(X::'a, X), singleton(X))) \ \& \\
\quad (\forall\, X\ Y.\ equal(unordered\text{-}pair(singleton(X), unordered\text{-}pair(X::'a, singleton(Y))), ordered\text{-}pair(X::'a, Y))) \\
\& \\
\quad (\forall\, V\ Y\ U\ X.\ member(ordered\text{-}pair(U::'a, V), cross\text{-}product(X::'a, Y)) \longrightarrow mem- \\
ber(U::'a, X)) \ \& \\
\quad (\forall\, U\ X\ V\ Y.\ member(ordered\text{-}pair(U::'a, V), cross\text{-}product(X::'a, Y)) \longrightarrow mem- \\
ber(V::'a, Y)) \ \& \\
\quad (\forall\, U\ V\ X\ Y.\ member(U::'a, X) \ \& \ member(V::'a, Y) \longrightarrow member(ordered\text{-}pair(U::'a, V), cross\text{-}product(X::'a \\
\& \\
\quad (\forall\, X\ Y\ Z.\ member(Z::'a, cross\text{-}product(X::'a, Y)) \longrightarrow equal(ordered\text{-}pair(first(Z), second(Z)), Z)) \\
\& \\
\quad (subclass(element\text{-}relation::'a, cross\text{-}product(universal\text{-}class::'a, universal\text{-}class))) \\
\& \\
\quad (\forall\, X\ Y.\ member(ordered\text{-}pair(X::'a, Y), element\text{-}relation) \longrightarrow member(X::'a, Y)) \\
\& \\
\quad (\forall\, X\ Y.\ member(ordered\text{-}pair(X::'a, Y), cross\text{-}product(universal\text{-}class::'a, universal\text{-}class)) \\
\& \ member(X::'a, Y) \longrightarrow member(ordered\text{-}pair(X::'a, Y), element\text{-}relation)) \ \& \\
\quad (\forall\, Y\ Z\ X.\ member(Z::'a, intersection(X::'a, Y)) \longrightarrow member(Z::'a, X)) \ \& \\
\quad (\forall\, X\ Z\ Y.\ member(Z::'a, intersection(X::'a, Y)) \longrightarrow member(Z::'a, Y)) \ \& \\
\quad (\forall\, Z\ X\ Y.\ member(Z::'a, X) \ \& \ member(Z::'a, Y) \longrightarrow member(Z::'a, intersection(X::'a, Y))) \\
\& \\
\quad (\forall\, Z\ X.\ \sim(member(Z::'a, complement(X)) \ \& \ member(Z::'a, X))) \ \& \\
\quad (\forall\, Z\ X.\ member(Z::'a, universal\text{-}class) \longrightarrow member(Z::'a, complement(X)) \ | \\
member(Z::'a, X)) \ \& \\
\quad (\forall\, X\ Y.\ equal(complement(intersection(complement(X), complement(Y))), union(X::'a, Y))) \\
\& \\
\quad (\forall\, X\ Y.\ equal(intersection(complement(intersection(X::'a, Y)), complement(intersection(complement(X), com \\
\& \\
\quad (\forall\, Xr\ X\ Y.\ equal(intersection(Xr::'a, cross\text{-}product(X::'a, Y)), restrct(Xr::'a, X, Y))) \\
\& \\
\quad (\forall\, Xr\ X\ Y.\ equal(intersection(cross\text{-}product(X::'a, Y), Xr), restrct(Xr::'a, X, Y))) \\
\& \\
\quad (\forall\, Z\ X.\ \sim(equal(restrct(X::'a, singleton(Z), universal\text{-}class), null\text{-}class) \ \& \ mem- \\
ber(Z::'a, domain\text{-}of(X)))) \ \& \\
\quad (\forall\, Z\ X.\ member(Z::'a, universal\text{-}class) \longrightarrow equal(restrct(X::'a, singleton(Z), universal\text{-}class), null\text{-}class) \\
| \ member(Z::'a, domain\text{-}of(X))) \ \& \\
\quad (\forall\, X.\ subclass(rot(X), cross\text{-}product(cross\text{-}product(universal\text{-}class::'a, universal\text{-}class), universal\text{-}class))) \\
\& \\
\quad (\forall\, V\ W\ U\ X.\ member(ordered\text{-}pair(ordered\text{-}pair(U::'a, V), W), rot(X)) \longrightarrow mem- \\
ber(ordered\text{-}pair(ordered\text{-}pair(V::'a, W), U), X)) \ \& \\
\quad (\forall\, U\ V\ W\ X.\ member(ordered\text{-}pair(ordered\text{-}pair(V::'a, W), U), X) \ \& \ member(ordered\text{-}pair(ordered\text{-}pair(U::'a \\
\longrightarrow member(ordered\text{-}pair(ordered\text{-}pair(U::'a, V), W), rot(X))) \ \& \\
\quad (\forall\, X.\ subclass(flip(X), cross\text{-}product(cross\text{-}product(universal\text{-}class::'a, universal\text{-}class), universal\text{-}class)))
\end{array}
$$

$$
\begin{array}{l}
\& \\
\quad (\forall\, V\ U\ W\ X.\ member(ordered\text{-}pair(ordered\text{-}pair(U::'a,V),W),flip(X)) \longrightarrow member(ordered\text{-}pair(ordered\text{-}pair(V::'a,U),W),X)) \ \& \\
\quad (\forall\, U\ V\ W\ X.\ member(ordered\text{-}pair(ordered\text{-}pair(V::'a,U),W),X) \ \&\ member(ordered\text{-}pair(ordered\text{-}pair(U::'a \\
\longrightarrow member(ordered\text{-}pair(ordered\text{-}pair(U::'a,V),W),flip(X))) \ \& \\
\quad (\forall\, Y.\ equal(domain\text{-}of(flip(cross\text{-}product(Y::'a,universal\text{-}class))),INVERSE(Y))) \\
\& \\
\quad (\forall\, Z.\ equal(domain\text{-}of(INVERSE(Z)),range\text{-}of(Z))) \ \& \\
\quad (\forall\, Z\ X\ Y.\ equal(first(not\text{-}subclass\text{-}element(restrct(Z::'a,X,singleton(Y)),null\text{-}class)),domain(Z::'a,X,Y))) \\
\& \\
\quad (\forall\, Z\ X\ Y.\ equal(second(not\text{-}subclass\text{-}element(restrct(Z::'a,singleton(X),Y),null\text{-}class)),rng(Z::'a,X,Y))) \\
\& \\
\quad (\forall\, Xr\ X.\ equal(range\text{-}of(restrct(Xr::'a,X,universal\text{-}class)),image'(Xr::'a,X))) \ \& \\
\quad (\forall\, X.\ equal(union(X::'a,singleton(X)),successor(X))) \ \& \\
\quad (subclass(successor\text{-}relation::'a,cross\text{-}product(universal\text{-}class::'a,universal\text{-}class))) \\
\& \\
\quad (\forall\, X\ Y.\ member(ordered\text{-}pair(X::'a,Y),successor\text{-}relation) \longrightarrow equal(successor(X),Y)) \\
\& \\
\quad (\forall\, X\ Y.\ equal(successor(X),Y) \ \&\ member(ordered\text{-}pair(X::'a,Y),cross\text{-}product(universal\text{-}class::'a,universal\text{-} \\
\longrightarrow member(ordered\text{-}pair(X::'a,Y),successor\text{-}relation)) \ \& \\
\quad (\forall\, X.\ inductive(X) \longrightarrow member(null\text{-}class::'a,X)) \ \& \\
\quad (\forall\, X.\ inductive(X) \longrightarrow subclass(image'(successor\text{-}relation::'a,X),X)) \ \& \\
\quad (\forall\, X.\ member(null\text{-}class::'a,X) \ \&\ subclass(image'(successor\text{-}relation::'a,X),X) \\
\longrightarrow inductive(X)) \ \& \\
\quad (inductive(omega)) \ \& \\
\quad (\forall\, Y.\ inductive(Y) \longrightarrow subclass(omega::'a,Y)) \ \& \\
\quad (member(omega::'a,universal\text{-}class)) \ \& \\
\quad (\forall\, X.\ equal(domain\text{-}of(restrct(element\text{-}relation::'a,universal\text{-}class,X)),sum\text{-}class(X))) \\
\& \\
\quad (\forall\, X.\ member(X::'a,universal\text{-}class) \longrightarrow member(sum\text{-}class(X),universal\text{-}class)) \\
\& \\
\quad (\forall\, X.\ equal(complement(image'(element\text{-}relation::'a,complement(X))),powerClass(X))) \\
\& \\
\quad (\forall\, U.\ member(U::'a,universal\text{-}class) \longrightarrow member(powerClass(U),universal\text{-}class)) \\
\& \\
\quad (\forall\, Yr\ Xr.\ subclass(compos(Yr::'a,Xr),cross\text{-}product(universal\text{-}class::'a,universal\text{-}class))) \\
\& \\
\quad (\forall\, Z\ Yr\ Xr\ Y.\ member(ordered\text{-}pair(Y::'a,Z),compos(Yr::'a,Xr)) \longrightarrow member(Z::'a,image'(Yr::'a,image'(Xr::'a,singleton(Y))))) \ \& \\
\quad (\forall\, Y\ Z\ Yr\ Xr.\ member(Z::'a,image'(Yr::'a,image'(Xr::'a,singleton(Y)))) \ \&\ member(ordered\text{-}pair(Y::'a,Z),cross\text{-}product(universal\text{-}class::'a,universal\text{-}class)) \longrightarrow member(ordered\text{-}pair(Y::'a,Z),compos(Yr::'a,Xr))) \ \& \\
\quad (\forall\, X.\ single\text{-}valued\text{-}class(X) \longrightarrow subclass(compos(X::'a,INVERSE(X)),identity\text{-}relation)) \\
\& \\
\quad (\forall\, X.\ subclass(compos(X::'a,INVERSE(X)),identity\text{-}relation) \longrightarrow single\text{-}valued\text{-}class(X)) \\
\& \\
\quad (\forall\, Xf.\ function(Xf) \longrightarrow subclass(Xf::'a,cross\text{-}product(universal\text{-}class::'a,universal\text{-}class))) \\
\& \\
\quad (\forall\, Xf.\ function(Xf) \longrightarrow subclass(compos(Xf::'a,INVERSE(Xf)),identity\text{-}relation))
\end{array}
$$

$$
\begin{array}{l}
\&\\
\quad(\forall\, Xf.\ subclass(Xf::'a,cross\text{-}product(universal\text{-}class::'a,universal\text{-}class))\ \&\ sub\text{-}\\
class(compos(Xf::'a,INVERSE(Xf)),identity\text{-}relation)\ \text{--}\!>\ function(Xf))\ \&\\
\ (\forall\, Xf\ X.\ function(Xf)\ \&\ member(X::'a,universal\text{-}class)\ \text{--}\!>\ member(image'(Xf::'a,X),universal\text{-}class))\\
\&\\
\ (\forall\, X.\ equal(X::'a,null\text{-}class)\ |\ member(regular(X),X))\ \&\\
\ (\forall\, X.\ equal(X::'a,null\text{-}class)\ |\ equal(intersection(X::'a,regular(X)),null\text{-}class))\ \&\\
\ (\forall\, Xf\ Y.\ equal(sum\text{-}class(image'(Xf::'a,singleton(Y))),apply(Xf::'a,Y)))\ \&\\
\ (function(choice))\ \&\\
\ (\forall\, Y.\ member(Y::'a,universal\text{-}class)\ \text{--}\!>\ equal(Y::'a,null\text{-}class)\ |\ member(apply(choice::'a,Y),Y))\\
\&\\
\ (\forall\, Xf.\ one\text{-}to\text{-}one(Xf)\ \text{--}\!>\ function(Xf))\ \&\\
\ (\forall\, Xf.\ one\text{-}to\text{-}one(Xf)\ \text{--}\!>\ function(INVERSE(Xf)))\ \&\\
\ (\forall\, Xf.\ function(INVERSE(Xf))\ \&\ function(Xf)\ \text{--}\!>\ one\text{-}to\text{-}one(Xf))\ \&\\
\ (equal(intersection(cross\text{-}product(universal\text{-}class::'a,universal\text{-}class),intersection(cross\text{-}product(universal\text{-}clas\\
\&\\
\ (equal(intersection(INVERSE(subset\text{-}relation),subset\text{-}relation),identity\text{-}relation))\\
\&\\
\ (\forall\, Xr.\ equal(complement(domain\text{-}of(intersection(Xr::'a,identity\text{-}relation)))),diagonalise(Xr)))\\
\&\\
\ (\forall\, X.\ equal(intersection(domain\text{-}of(X),diagonalise(compos(INVERSE(element\text{-}relation),X))),cantor(X)))\\
\&\\
\ (\forall\, Xf.\ operation(Xf)\ \text{--}\!>\ function(Xf))\ \&\\
\ (\forall\, Xf.\ operation(Xf)\ \text{--}\!>\ equal(cross\text{-}product(domain\text{-}of(domain\text{-}of(Xf)),domain\text{-}of(domain\text{-}of(Xf))),dom\\
\&\\
\ (\forall\, Xf.\ operation(Xf)\ \text{--}\!>\ subclass(range\text{-}of(Xf),domain\text{-}of(domain\text{-}of(Xf))))\\
\&\\
\ (\forall\, Xf.\ function(Xf)\ \&\ equal(cross\text{-}product(domain\text{-}of(domain\text{-}of(Xf)),domain\text{-}of(domain\text{-}of(Xf))),domain\text{-}\\
\&\ subclass(range\text{-}of(Xf),domain\text{-}of(domain\text{-}of(Xf)))\ \text{--}\!>\ operation(Xf))\ \&\\
\ (\forall\, Xf1\ Xf2\ Xh.\ compatible(Xh::'a,Xf1,Xf2)\ \text{--}\!>\ function(Xh))\ \&\\
\ (\forall\, Xf2\ Xf1\ Xh.\ compatible(Xh::'a,Xf1,Xf2)\ \text{--}\!>\ equal(domain\text{-}of(domain\text{-}of(Xf1)),domain\text{-}of(Xh)))\\
\&\\
\ (\forall\, Xf1\ Xh\ Xf2.\ compatible(Xh::'a,Xf1,Xf2)\ \text{--}\!>\ subclass(range\text{-}of(Xh),domain\text{-}of(domain\text{-}of(Xf2))))\\
\&\\
\ (\forall\, Xh\ Xh1\ Xf1\ Xf2.\ function(Xh)\ \&\ equal(domain\text{-}of(domain\text{-}of(Xf1)),domain\text{-}of(Xh))\\
\&\ subclass(range\text{-}of(Xh),domain\text{-}of(domain\text{-}of(Xf2)))\ \text{--}\!>\ compatible(Xh1::'a,Xf1,Xf2))\\
\&\\
\ (\forall\, Xh\ Xf2\ Xf1.\ homomorphism(Xh::'a,Xf1,Xf2)\ \text{--}\!>\ operation(Xf1))\ \&\\
\ (\forall\, Xh\ Xf1\ Xf2.\ homomorphism(Xh::'a,Xf1,Xf2)\ \text{--}\!>\ operation(Xf2))\ \&\\
\ (\forall\, Xh\ Xf1\ Xf2.\ homomorphism(Xh::'a,Xf1,Xf2)\ \text{--}\!>\ compatible(Xh::'a,Xf1,Xf2))\\
\&\\
\ (\forall\, Xf2\ Xh\ Xf1\ X\ Y.\ homomorphism(Xh::'a,Xf1,Xf2)\ \&\ member(ordered\text{-}pair(X::'a,Y),domain\text{-}of(Xf1))\\
\text{--}\!>\ equal(apply(Xf2::'a,ordered\text{-}pair(apply(Xh::'a,X),apply(Xh::'a,Y))),apply(Xh::'a,apply(Xf1::'a,ordered\text{-}\\
\&\\
\ (\forall\, Xh\ Xf1\ Xf2.\ operation(Xf1)\ \&\ operation(Xf2)\ \&\ compatible(Xh::'a,Xf1,Xf2)\\
\text{--}\!>\ member(ordered\text{-}pair(not\text{-}homomorphism1(Xh::'a,Xf1,Xf2),not\text{-}homomorphism2(Xh::'a,Xf1,Xf2)),dom\\
|\ homomorphism(Xh::'a,Xf1,Xf2))\ \&\\
\ (\forall\, Xh\ Xf1\ Xf2.\ operation(Xf1)\ \&\ operation(Xf2)\ \&\ compatible(Xh::'a,Xf1,Xf2)\\
\&\ equal(apply(Xf2::'a,ordered\text{-}pair(apply(Xh::'a,not\text{-}homomorphism1(Xh::'a,Xf1,Xf2)),apply(Xh::'a,not\text{-}hom
\end{array}
$$

$--> homomorphism(Xh::'a,Xf1,Xf2))$

abbreviation *SET004-0-eq subclass single-valued-class operation one-to-one member inductive homomorphism function compatible unordered-pair union sum-class successor singleton second rot restrct regular range-of rng powerClass ordered-pair not-subclass-element not-homomorphism2 not-homomorphism1 INVERSE intersection image' flip first domain-of domain difference diagonalise cross-product compos complement cantor apply equal* $\equiv$

$(\forall D\ E\ F'.\ equal(D::'a,E) --> equal(apply(D::'a,F'),apply(E::'a,F'))) \&$
$(\forall G\ I'\ H.\ equal(G::'a,H) --> equal(apply(I'::'a,G),apply(I'::'a,H))) \&$
$(\forall J\ K'.\ equal(J::'a,K') --> equal(cantor(J),cantor(K'))) \&$
$(\forall L\ M.\ equal(L::'a,M) --> equal(complement(L),complement(M))) \&$
$(\forall N\ O'\ P.\ equal(N::'a,O') --> equal(compos(N::'a,P),compos(O'::'a,P))) \&$
$(\forall Q\ S'\ R.\ equal(Q::'a,R) --> equal(compos(S'::'a,Q),compos(S'::'a,R))) \&$
$(\forall T'\ U\ V.\ equal(T'::'a,U) --> equal(cross\text{-}product(T'::'a,V),cross\text{-}product(U::'a,V)))$
$\&$
$(\forall W\ Y\ X.\ equal(W::'a,X) --> equal(cross\text{-}product(Y::'a,W),cross\text{-}product(Y::'a,X)))$
$\&$
$(\forall Z\ A1.\ equal(Z::'a,A1) --> equal(diagonalise(Z),diagonalise(A1))) \&$
$(\forall B1\ C1\ D1.\ equal(B1::'a,C1) --> equal(difference(B1::'a,D1),difference(C1::'a,D1)))$
$\&$
$(\forall E1\ G1\ F1.\ equal(E1::'a,F1) --> equal(difference(G1::'a,E1),difference(G1::'a,F1)))$
$\&$
$(\forall H1\ I1\ J1\ K1.\ equal(H1::'a,I1) --> equal(domain(H1::'a,J1,K1),domain(I1::'a,J1,K1)))$
$\&$
$(\forall L1\ N1\ M1\ O1.\ equal(L1::'a,M1) --> equal(domain(N1::'a,L1,O1),domain(N1::'a,M1,O1)))$
$\&$
$(\forall P1\ R1\ S1\ Q1.\ equal(P1::'a,Q1) --> equal(domain(R1::'a,S1,P1),domain(R1::'a,S1,Q1)))$
$\&$
$(\forall T1\ U1.\ equal(T1::'a,U1) --> equal(domain\text{-}of(T1),domain\text{-}of(U1))) \&$
$(\forall V1\ W1.\ equal(V1::'a,W1) --> equal(first(V1),first(W1))) \&$
$(\forall X1\ Y1.\ equal(X1::'a,Y1) --> equal(flip(X1),flip(Y1))) \&$
$(\forall Z1\ A2\ B2.\ equal(Z1::'a,A2) --> equal(image'(Z1::'a,B2),image'(A2::'a,B2)))$
$\&$
$(\forall C2\ E2\ D2.\ equal(C2::'a,D2) --> equal(image'(E2::'a,C2),image'(E2::'a,D2)))$
$\&$
$(\forall F2\ G2\ H2.\ equal(F2::'a,G2) --> equal(intersection(F2::'a,H2),intersection(G2::'a,H2)))$
$\&$
$(\forall I2\ K2\ J2.\ equal(I2::'a,J2) --> equal(intersection(K2::'a,I2),intersection(K2::'a,J2)))$
$\&$
$(\forall L2\ M2.\ equal(L2::'a,M2) --> equal(INVERSE(L2),INVERSE(M2))) \&$
$(\forall N2\ O2\ P2\ Q2.\ equal(N2::'a,O2) --> equal(not\text{-}homomorphism1(N2::'a,P2,Q2),not\text{-}homomorphism1(O2$
$\&$
$(\forall R2\ T2\ S2\ U2.\ equal(R2::'a,S2) --> equal(not\text{-}homomorphism1(T2::'a,R2,U2),not\text{-}homomorphism1(T2:$
$\&$
$(\forall V2\ X2\ Y2\ W2.\ equal(V2::'a,W2) --> equal(not\text{-}homomorphism1(X2::'a,Y2,V2),not\text{-}homomorphism1(X$
$\&$
$(\forall Z2\ A3\ B3\ C3.\ equal(Z2::'a,A3) --> equal(not\text{-}homomorphism2(Z2::'a,B3,C3),not\text{-}homomorphism2(A3:$

```
&
 (∀ D3 F3 E3 G3. equal(D3::'a,E3) --> equal(not-homomorphism2(F3::'a,D3,G3),not-homomorphism2(F3
&
 (∀ H3 J3 K3 I3. equal(H3::'a,I3) --> equal(not-homomorphism2(J3::'a,K3,H3),not-homomorphism2(J3::'
&
 (∀ L3 M3 N3. equal(L3::'a,M3) --> equal(not-subclass-element(L3::'a,N3),not-subclass-element(M3::'a,N3
&
 (∀ O3 Q3 P3. equal(O3::'a,P3) --> equal(not-subclass-element(Q3::'a,O3),not-subclass-element(Q3::'a,P3
&
 (∀ R3 S3 T3. equal(R3::'a,S3) --> equal(ordered-pair(R3::'a,T3),ordered-pair(S3::'a,T3)))
&
 (∀ U3 W3 V3. equal(U3::'a,V3) --> equal(ordered-pair(W3::'a,U3),ordered-pair(W3::'a,V3)))
&
  (∀ X3 Y3. equal(X3::'a,Y3) --> equal(powerClass(X3),powerClass(Y3))) &
 (∀ Z3 A4 B4 C4. equal(Z3::'a,A4) --> equal(rng(Z3::'a,B4,C4),rng(A4::'a,B4,C4)))
&
 (∀ D4 F4 E4 G4. equal(D4::'a,E4) --> equal(rng(F4::'a,D4,G4),rng(F4::'a,E4,G4)))
&
 (∀ H4 J4 K4 I4. equal(H4::'a,I4) --> equal(rng(J4::'a,K4,H4),rng(J4::'a,K4,I4)))
&
  (∀ L4 M4. equal(L4::'a,M4) --> equal(range-of(L4),range-of(M4))) &
  (∀ N4 O4. equal(N4::'a,O4) --> equal(regular(N4),regular(O4))) &
 (∀ P4 Q4 R4 S4. equal(P4::'a,Q4) --> equal(restrct(P4::'a,R4,S4),restrct(Q4::'a,R4,S4)))
&
 (∀ T4 V4 U4 W4. equal(T4::'a,U4) --> equal(restrct(V4::'a,T4,W4),restrct(V4::'a,U4,W4)))
&
 (∀ X4 Z4 A5 Y4. equal(X4::'a,Y4) --> equal(restrct(Z4::'a,A5,X4),restrct(Z4::'a,A5,Y4)))
&
  (∀ B5 C5. equal(B5::'a,C5) --> equal(rot(B5),rot(C5))) &
  (∀ D5 E5. equal(D5::'a,E5) --> equal(second(D5),second(E5))) &
  (∀ F5 G5. equal(F5::'a,G5) --> equal(singleton(F5),singleton(G5))) &
  (∀ H5 I5. equal(H5::'a,I5) --> equal(successor(H5),successor(I5))) &
  (∀ J5 K5. equal(J5::'a,K5) --> equal(sum-class(J5),sum-class(K5))) &
 (∀ L5 M5 N5. equal(L5::'a,M5) --> equal(union(L5::'a,N5),union(M5::'a,N5)))
&
 (∀ O5 Q5 P5. equal(O5::'a,P5) --> equal(union(Q5::'a,O5),union(Q5::'a,P5)))
&
 (∀ R5 S5 T5. equal(R5::'a,S5) --> equal(unordered-pair(R5::'a,T5),unordered-pair(S5::'a,T5)))
&
 (∀ U5 W5 V5. equal(U5::'a,V5) --> equal(unordered-pair(W5::'a,U5),unordered-pair(W5::'a,V5)))
&
  (∀ X5 Y5 Z5 A6. equal(X5::'a,Y5) & compatible(X5::'a,Z5,A6) --> compati-
ble(Y5::'a,Z5,A6)) &
  (∀ B6 D6 C6 E6. equal(B6::'a,C6) & compatible(D6::'a,B6,E6) --> compati-
ble(D6::'a,C6,E6)) &
  (∀ F6 H6 I6 G6. equal(F6::'a,G6) & compatible(H6::'a,I6,F6) --> compati-
ble(H6::'a,I6,G6)) &
  (∀ J6 K6. equal(J6::'a,K6) & function(J6) --> function(K6)) &
  (∀ L6 M6 N6 O6. equal(L6::'a,M6) & homomorphism(L6::'a,N6,O6) --> ho-
```

momorphism(*M6*::'*a*,*N6*,*O6*)) &
(∀ *P6 R6 Q6 S6*. *equal*(*P6*::'*a*,*Q6*) & *homomorphism*(*R6*::'*a*,*P6*,*S6*) −−> *homomorphism*(*R6*::'*a*,*Q6*,*S6*)) &
(∀ *T6 V6 W6 U6*. *equal*(*T6*::'*a*,*U6*) & *homomorphism*(*V6*::'*a*,*W6*,*T6*) −−> *homomorphism*(*V6*::'*a*,*W6*,*U6*)) &
(∀ *X6 Y6*. *equal*(*X6*::'*a*,*Y6*) & *inductive*(*X6*) −−> *inductive*(*Y6*)) &
(∀ *Z6 A7 B7*. *equal*(*Z6*::'*a*,*A7*) & *member*(*Z6*::'*a*,*B7*) −−> *member*(*A7*::'*a*,*B7*))
&
(∀ *C7 E7 D7*. *equal*(*C7*::'*a*,*D7*) & *member*(*E7*::'*a*,*C7*) −−> *member*(*E7*::'*a*,*D7*))
&
(∀ *F7 G7*. *equal*(*F7*::'*a*,*G7*) & *one-to-one*(*F7*) −−> *one-to-one*(*G7*)) &
(∀ *H7 I7*. *equal*(*H7*::'*a*,*I7*) & *operation*(*H7*) −−> *operation*(*I7*)) &
(∀ *J7 K7*. *equal*(*J7*::'*a*,*K7*) & *single-valued-class*(*J7*) −−> *single-valued-class*(*K7*))
&
(∀ *L7 M7 N7*. *equal*(*L7*::'*a*,*M7*) & *subclass*(*L7*::'*a*,*N7*) −−> *subclass*(*M7*::'*a*,*N7*))
&
(∀ *O7 Q7 P7*. *equal*(*O7*::'*a*,*P7*) & *subclass*(*Q7*::'*a*,*O7*) −−> *subclass*(*Q7*::'*a*,*P7*))

abbreviation *SET004-1-ax range-of function maps apply*
application-function singleton-relation element-relation complement
intersection single-valued3 singleton image' domain single-valued2
second single-valued1 identity-relation INVERSE not-subclass-element
first domain-of domain-relation composition-function compos equal
ordered-pair member universal-class cross-product compose-class
subclass ≡
(∀ *X*. *subclass*(*compose-class*(*X*),*cross-product*(*universal-class*::'*a*,*universal-class*)))
&
(∀ *X Y Z*. *member*(*ordered-pair*(*Y*::'*a*,*Z*),*compose-class*(*X*)) −−> *equal*(*compos*(*X*::'*a*,*Y*),*Z*))
&
(∀ *Y Z X*. *member*(*ordered-pair*(*Y*::'*a*,*Z*),*cross-product*(*universal-class*::'*a*,*universal-class*))
& *equal*(*compos*(*X*::'*a*,*Y*),*Z*) −−> *member*(*ordered-pair*(*Y*::'*a*,*Z*),*compose-class*(*X*)))
&
(*subclass*(*composition-function*::'*a*,*cross-product*(*universal-class*::'*a*,*cross-product*(*universal-class*::'*a*,*universa
&
(∀ *X Y Z*. *member*(*ordered-pair*(*X*::'*a*,*ordered-pair*(*Y*::'*a*,*Z*)),*composition-function*)
−−> *equal*(*compos*(*X*::'*a*,*Y*),*Z*)) &
(∀ *X Y*. *member*(*ordered-pair*(*X*::'*a*,*Y*),*cross-product*(*universal-class*::'*a*,*universal-class*))
−−> *member*(*ordered-pair*(*X*::'*a*,*ordered-pair*(*Y*::'*a*,*compos*(*X*::'*a*,*Y*))),*composition-function*))
&
(*subclass*(*domain-relation*::'*a*,*cross-product*(*universal-class*::'*a*,*universal-class*))) &
(∀ *X Y*. *member*(*ordered-pair*(*X*::'*a*,*Y*),*domain-relation*) −−> *equal*(*domain-of*(*X*),*Y*))
&
(∀ *X*. *member*(*X*::'*a*,*universal-class*) −−> *member*(*ordered-pair*(*X*::'*a*,*domain-of*(*X*)),*domain-relation*))
&
(∀ *X*. *equal*(*first*(*not-subclass-element*(*compos*(*X*::'*a*,*INVERSE*(*X*)),*identity-relation*)),*single-valued1*(*X*)))
&
(∀ *X*. *equal*(*second*(*not-subclass-element*(*compos*(*X*::'*a*,*INVERSE*(*X*)),*identity-relation*)),*single-valued2*(*X*))
&
(∀ *X*. *equal*(*domain*(*X*::'*a*,*image'*(*INVERSE*(*X*),*singleton*(*single-valued1*(*X*))),*single-valued2*(*X*)),*single-valu

$\&$
$(equal(intersection(complement(compos(element\text{-}relation::'a,complement(identity\text{-}relation))),element\text{-}relation$
$\&$
$(subclass(application\text{-}function::'a,cross\text{-}product(universal\text{-}class::'a,cross\text{-}product(universal\text{-}class::'a,universal\text{-}$
$\&$
$(\forall\, Z\ Y\ X.\ member(ordered\text{-}pair(X::'a,ordered\text{-}pair(Y::'a,Z)),application\text{-}function)$
$--> member(Y::'a,domain\text{-}of(X))) \&$
$(\forall\, X\ Y\ Z.\ member(ordered\text{-}pair(X::'a,ordered\text{-}pair(Y::'a,Z)),application\text{-}function)$
$--> equal(apply(X::'a,Y),Z)) \&$
$(\forall\, Z\ X\ Y.\ member(ordered\text{-}pair(X::'a,ordered\text{-}pair(Y::'a,Z)),cross\text{-}product(universal\text{-}class::'a,cross\text{-}product($
$\& member(Y::'a,domain\text{-}of(X)) --> member(ordered\text{-}pair(X::'a,ordered\text{-}pair(Y::'a,apply(X::'a,Y))),applic$
$\&$
$(\forall\, X\ Y\ Xf.\ maps(Xf::'a,X,Y) --> function(Xf)) \&$
$(\forall\, Y\ Xf\ X.\ maps(Xf::'a,X,Y) --> equal(domain\text{-}of(Xf),X)) \&$
$(\forall\, X\ Xf\ Y.\ maps(Xf::'a,X,Y) --> subclass(range\text{-}of(Xf),Y)) \&$
$(\forall\, Xf\ Y.\ function(Xf) \&\ subclass(range\text{-}of(Xf),Y) --> maps(Xf::'a,domain\text{-}of(Xf),Y))$

abbreviation *SET004-1-eq maps single-valued3 single-valued2 single-valued1 compose-class*
$equal \equiv$
$(\forall\, L\ M.\ equal(L::'a,M) --> equal(compose\text{-}class(L),compose\text{-}class(M))) \&$
$(\forall\, N2\ O2.\ equal(N2::'a,O2) --> equal(single\text{-}valued1(N2),single\text{-}valued1(O2)))$
$\&$
$(\forall\, P2\ Q2.\ equal(P2::'a,Q2) --> equal(single\text{-}valued2(P2),single\text{-}valued2(Q2)))$
$\&$
$(\forall\, R2\ S2.\ equal(R2::'a,S2) --> equal(single\text{-}valued3(R2),single\text{-}valued3(S2)))$
$\&$
$(\forall\, X2\ Y2\ Z2\ A3.\ equal(X2::'a,Y2) \&\ maps(X2::'a,Z2,A3) --> maps(Y2::'a,Z2,A3))$
$\&$
$(\forall\, B3\ D3\ C3\ E3.\ equal(B3::'a,C3) \&\ maps(D3::'a,B3,E3) --> maps(D3::'a,C3,E3))$
$\&$
$(\forall\, F3\ H3\ I3\ G3.\ equal(F3::'a,G3) \&\ maps(H3::'a,I3,F3) --> maps(H3::'a,I3,G3))$

abbreviation *NUM004-0-ax integer-of omega ordinal-multiply*
add-relation ordinal-add recursion apply range-of union-of-range-map
function recursion-equation-functions rest-relation rest-of
limit-ordinals kind-1-ordinals successor-relation image'
universal-class sum-class element-relation ordinal-numbers section
not-well-ordering ordered-pair least member well-ordering singleton
domain-of segment null-class intersection asymmetric compos transitive
cross-product connected identity-relation complement restrct subclass
irreflexive symmetrization-of INVERSE union equal $\equiv$
$(\forall\, X.\ equal(union(X::'a,INVERSE(X)),symmetrization\text{-}of(X))) \&$
$(\forall\, X\ Y.\ irreflexive(X::'a,Y) --> subclass(restrct(X::'a,Y,Y),complement(identity\text{-}relation)))$
$\&$
$(\forall\, X\ Y.\ subclass(restrct(X::'a,Y,Y),complement(identity\text{-}relation)) --> irreflex\text{-}$
$ive(X::'a,Y)) \&$
$(\forall\, Y\ X.\ connected(X::'a,Y) --> subclass(cross\text{-}product(Y::'a,Y),union(identity\text{-}relation::'a,symmetrizatio$
$\&$
$(\forall\, X\ Y.\ subclass(cross\text{-}product(Y::'a,Y),union(identity\text{-}relation::'a,symmetrization\text{-}of(X)))$

$--> connected(X::'a,Y))$ &
$(\forall Xr\ Y.\ transitive(Xr::'a,Y) --> subclass(compos(restrct(Xr::'a,Y,Y),restrct(Xr::'a,Y,Y)),restrct(Xr::'a$
&
$(\forall Xr\ Y.\ subclass(compos(restrct(Xr::'a,Y,Y),restrct(Xr::'a,Y,Y)),restrct(Xr::'a,Y,Y))$
$--> transitive(Xr::'a,Y))$ &
$(\forall Xr\ Y.\ asymmetric(Xr::'a,Y) --> equal(restrct(intersection(Xr::'a,INVERSE(Xr)),Y,Y),null\text{-}class))$
&
$(\forall Xr\ Y.\ equal(restrct(intersection(Xr::'a,INVERSE(Xr)),Y,Y),null\text{-}class) -->$
$asymmetric(Xr::'a,Y))$ &
$(\forall Xr\ Y\ Z.\ equal(segment(Xr::'a,Y,Z),domain\text{-}of(restrct(Xr::'a,Y,singleton(Z)))))$
&
$(\forall X\ Y.\ well\text{-}ordering(X::'a,Y) --> connected(X::'a,Y))$ &
$(\forall Y\ Xr\ U.\ well\text{-}ordering(Xr::'a,Y)$ & $subclass(U::'a,Y) --> equal(U::'a,null\text{-}class)$
$\mid member(least(Xr::'a,U),U))$ &
$(\forall Y\ V\ Xr\ U.\ well\text{-}ordering(Xr::'a,Y)$ & $subclass(U::'a,Y)$ & $member(V::'a,U)$
$--> member(least(Xr::'a,U),U))$ &
$(\forall Y\ Xr\ U.\ well\text{-}ordering(Xr::'a,Y)$ & $subclass(U::'a,Y) --> equal(segment(Xr::'a,U,least(Xr::'a,U)),null\text{-}$
&
$(\forall Y\ V\ U\ Xr.\ \sim(well\text{-}ordering(Xr::'a,Y)$ & $subclass(U::'a,Y)$ & $member(V::'a,U)$
& $member(ordered\text{-}pair(V::'a,least(Xr::'a,U)),Xr)))$ &
$(\forall Xr\ Y.\ connected(Xr::'a,Y)$ & $equal(not\text{-}well\text{-}ordering(Xr::'a,Y),null\text{-}class)$
$--> well\text{-}ordering(Xr::'a,Y))$ &
$(\forall Xr\ Y.\ connected(Xr::'a,Y) --> subclass(not\text{-}well\text{-}ordering(Xr::'a,Y),Y) \mid$
$well\text{-}ordering(Xr::'a,Y))$ &
$(\forall V\ Xr\ Y.\ member(V::'a,not\text{-}well\text{-}ordering(Xr::'a,Y))$ & $equal(segment(Xr::'a,not\text{-}well\text{-}ordering(Xr::'a,Y),$
& $connected(Xr::'a,Y) --> well\text{-}ordering(Xr::'a,Y))$ &
$(\forall Xr\ Y\ Z.\ section(Xr::'a,Y,Z) --> subclass(Y::'a,Z))$ &
$(\forall Xr\ Z\ Y.\ section(Xr::'a,Y,Z) --> subclass(domain\text{-}of(restrct(Xr::'a,Z,Y)),Y))$
&
$(\forall Xr\ Y\ Z.\ subclass(Y::'a,Z)$ & $subclass(domain\text{-}of(restrct(Xr::'a,Z,Y)),Y) -->$
$section(Xr::'a,Y,Z))$ &
$(\forall X.\ member(X::'a,ordinal\text{-}numbers) --> well\text{-}ordering(element\text{-}relation::'a,X))$
&
$(\forall X.\ member(X::'a,ordinal\text{-}numbers) --> subclass(sum\text{-}class(X),X))$ &
$(\forall X.\ well\text{-}ordering(element\text{-}relation::'a,X)$ & $subclass(sum\text{-}class(X),X)$ & $mem\text{-}$
$ber(X::'a,universal\text{-}class) --> member(X::'a,ordinal\text{-}numbers))$ &
$(\forall X.\ well\text{-}ordering(element\text{-}relation::'a,X)$ & $subclass(sum\text{-}class(X),X) -->$
$member(X::'a,ordinal\text{-}numbers) \mid equal(X::'a,ordinal\text{-}numbers))$ &
$(equal(union(singleton(null\text{-}class),image'(successor\text{-}relation::'a,ordinal\text{-}numbers)),kind\text{-}1\text{-}ordinals))$
&
$(equal(intersection(complement(kind\text{-}1\text{-}ordinals),ordinal\text{-}numbers),limit\text{-}ordinals))$
&
$(\forall X.\ subclass(rest\text{-}of(X),cross\text{-}product(universal\text{-}class::'a,universal\text{-}class)))$ &
$(\forall V\ U\ X.\ member(ordered\text{-}pair(U::'a,V),rest\text{-}of(X)) --> member(U::'a,domain\text{-}of(X)))$
&
$(\forall X\ U\ V.\ member(ordered\text{-}pair(U::'a,V),rest\text{-}of(X)) --> equal(restrct(X::'a,U,universal\text{-}class),V))$
&
$(\forall U\ V\ X.\ member(U::'a,domain\text{-}of(X))$ & $equal(restrct(X::'a,U,universal\text{-}class),V)$
$--> member(ordered\text{-}pair(U::'a,V),rest\text{-}of(X)))$ &

(*subclass*(*rest-relation*::'*a*,*cross-product*(*universal-class*::'*a*,*universal-class*))) &
(∀ *X Y*. *member*(*ordered-pair*(*X*::'*a*,*Y*),*rest-relation*) −−> *equal*(*rest-of*(*X*),*Y*))
&
(∀ *X*. *member*(*X*::'*a*,*universal-class*) −−> *member*(*ordered-pair*(*X*::'*a*,*rest-of*(*X*)),*rest-relation*))
&
(∀ *X Z*. *member*(*X*::'*a*,*recursion-equation-functions*(*Z*)) −−> *function*(*Z*)) &
(∀ *Z X*. *member*(*X*::'*a*,*recursion-equation-functions*(*Z*)) −−> *function*(*X*)) &
(∀ *Z X*. *member*(*X*::'*a*,*recursion-equation-functions*(*Z*)) −−> *member*(*domain-of*(*X*),*ordinal-numbers*))
&
(∀ *Z X*. *member*(*X*::'*a*,*recursion-equation-functions*(*Z*)) −−> *equal*(*compos*(*Z*::'*a*,*rest-of*(*X*)),*X*))
&
(∀ *X Z*. *function*(*Z*) & *function*(*X*) & *member*(*domain-of*(*X*),*ordinal-numbers*) &
equal(*compos*(*Z*::'*a*,*rest-of*(*X*)),*X*) −−> *member*(*X*::'*a*,*recursion-equation-functions*(*Z*)))
&
(*subclass*(*union-of-range-map*::'*a*,*cross-product*(*universal-class*::'*a*,*universal-class*)))
&
(∀ *X Y*. *member*(*ordered-pair*(*X*::'*a*,*Y*),*union-of-range-map*) −−> *equal*(*sum-class*(*range-of*(*X*)),*Y*))
&
(∀ *X Y*. *member*(*ordered-pair*(*X*::'*a*,*Y*),*cross-product*(*universal-class*::'*a*,*universal-class*))
& *equal*(*sum-class*(*range-of*(*X*)),*Y*) −−> *member*(*ordered-pair*(*X*::'*a*,*Y*),*union-of-range-map*))
&
(∀ *X Y*. *equal*(*apply*(*recursion*(*X*::'*a*,*successor-relation*,*union-of-range-map*),*Y*),*ordinal-add*(*X*::'*a*,*Y*)))
&
(∀ *X Y*. *equal*(*recursion*(*null-class*::'*a*,*apply*(*add-relation*::'*a*,*X*),*union-of-range-map*),*ordinal-multiply*(*X*::'*a*,
&
(∀ *X*. *member*(*X*::'*a*,*omega*) −−> *equal*(*integer-of*(*X*),*X*)) &
(∀ *X*. *member*(*X*::'*a*,*omega*) | *equal*(*integer-of*(*X*),*null-class*))

abbreviation *NUM004-0-eq well-ordering transitive section irreflexive*
connected asymmetric symmetrization-of segment rest-of
recursion-equation-functions recursion ordinal-multiply ordinal-add
not-well-ordering least integer-of equal ≡
(∀ *D E*. *equal*(*D*::'*a*,*E*) −−> *equal*(*integer-of*(*D*),*integer-of*(*E*))) &
(∀ *F′ G H*. *equal*(*F′*::'*a*,*G*) −−> *equal*(*least*(*F′*::'*a*,*H*),*least*(*G*::'*a*,*H*))) &
(∀ *I′ K′ J*. *equal*(*I′*::'*a*,*J*) −−> *equal*(*least*(*K′*::'*a*,*I′*),*least*(*K′*::'*a*,*J*))) &
(∀ *L M N*. *equal*(*L*::'*a*,*M*) −−> *equal*(*not-well-ordering*(*L*::'*a*,*N*),*not-well-ordering*(*M*::'*a*,*N*)))
&
(∀ *O′ Q P*. *equal*(*O′*::'*a*,*P*) −−> *equal*(*not-well-ordering*(*Q*::'*a*,*O′*),*not-well-ordering*(*Q*::'*a*,*P*)))
&
(∀ *R S′ T′*. *equal*(*R*::'*a*,*S′*) −−> *equal*(*ordinal-add*(*R*::'*a*,*T′*),*ordinal-add*(*S′*::'*a*,*T′*)))
&
(∀ *U W V*. *equal*(*U*::'*a*,*V*) −−> *equal*(*ordinal-add*(*W*::'*a*,*U*),*ordinal-add*(*W*::'*a*,*V*)))
&
(∀ *X Y Z*. *equal*(*X*::'*a*,*Y*) −−> *equal*(*ordinal-multiply*(*X*::'*a*,*Z*),*ordinal-multiply*(*Y*::'*a*,*Z*)))
&
(∀ *A1 C1 B1*. *equal*(*A1*::'*a*,*B1*) −−> *equal*(*ordinal-multiply*(*C1*::'*a*,*A1*),*ordinal-multiply*(*C1*::'*a*,*B1*)))
&
(∀ *F1 G1 H1 I1*. *equal*(*F1*::'*a*,*G1*) −−> *equal*(*recursion*(*F1*::'*a*,*H1*,*I1*),*recursion*(*G1*::'*a*,*H1*,*I1*)))
&

$(\forall\, J1\ L1\ K1\ M1.\ equal(J1::'a,K1) \longrightarrow equal(recursion(L1::'a,J1,M1),recursion(L1::'a,K1,M1)))$
&
$(\forall\, N1\ P1\ Q1\ O1.\ equal(N1::'a,O1) \longrightarrow equal(recursion(P1::'a,Q1,N1),recursion(P1::'a,Q1,O1)))$
&
$(\forall\, R1\ S1.\ equal(R1::'a,S1) \longrightarrow equal(recursion\text{-}equation\text{-}functions(R1),recursion\text{-}equation\text{-}functions(S1)))$
&
$(\forall\, T1\ U1.\ equal(T1::'a,U1) \longrightarrow equal(rest\text{-}of(T1),rest\text{-}of(U1))) \ \&$
$(\forall\, V1\ W1\ X1\ Y1.\ equal(V1::'a,W1) \longrightarrow equal(segment(V1::'a,X1,Y1),segment(W1::'a,X1,Y1)))$
&
$(\forall\, Z1\ B2\ A2\ C2.\ equal(Z1::'a,A2) \longrightarrow equal(segment(B2::'a,Z1,C2),segment(B2::'a,A2,C2)))$
&
$(\forall\, D2\ F2\ G2\ E2.\ equal(D2::'a,E2) \longrightarrow equal(segment(F2::'a,G2,D2),segment(F2::'a,G2,E2)))$
&
$(\forall\, H2\ I2.\ equal(H2::'a,I2) \longrightarrow equal(symmetrization\text{-}of(H2),symmetrization\text{-}of(I2)))$
&
$(\forall\, J2\ K2\ L2.\ equal(J2::'a,K2)\ \&\ asymmetric(J2::'a,L2) \longrightarrow asymmetric(K2::'a,L2))$
&
$(\forall\, M2\ O2\ N2.\ equal(M2::'a,N2)\ \&\ asymmetric(O2::'a,M2) \longrightarrow asymmetric(O2::'a,N2))$
&
$(\forall\, P2\ Q2\ R2.\ equal(P2::'a,Q2)\ \&\ connected(P2::'a,R2) \longrightarrow connected(Q2::'a,R2))$
&
$(\forall\, S2\ U2\ T2.\ equal(S2::'a,T2)\ \&\ connected(U2::'a,S2) \longrightarrow connected(U2::'a,T2))$
&
$(\forall\, V2\ W2\ X2.\ equal(V2::'a,W2)\ \&\ irreflexive(V2::'a,X2) \longrightarrow irreflexive(W2::'a,X2))$
&
$(\forall\, Y2\ A3\ Z2.\ equal(Y2::'a,Z2)\ \&\ irreflexive(A3::'a,Y2) \longrightarrow irreflexive(A3::'a,Z2))$
&
$(\forall\, B3\ C3\ D3\ E3.\ equal(B3::'a,C3)\ \&\ section(B3::'a,D3,E3) \longrightarrow section(C3::'a,D3,E3))$
&
$(\forall\, F3\ H3\ G3\ I3.\ equal(F3::'a,G3)\ \&\ section(H3::'a,F3,I3) \longrightarrow section(H3::'a,G3,I3))$
&
$(\forall\, J3\ L3\ M3\ K3.\ equal(J3::'a,K3)\ \&\ section(L3::'a,M3,J3) \longrightarrow section(L3::'a,M3,K3))$
&
$(\forall\, N3\ O3\ P3.\ equal(N3::'a,O3)\ \&\ transitive(N3::'a,P3) \longrightarrow transitive(O3::'a,P3))$
&
$(\forall\, Q3\ S3\ R3.\ equal(Q3::'a,R3)\ \&\ transitive(S3::'a,Q3) \longrightarrow transitive(S3::'a,R3))$
&
$(\forall\, T3\ U3\ V3.\ equal(T3::'a,U3)\ \&\ well\text{-}ordering(T3::'a,V3) \longrightarrow well\text{-}ordering(U3::'a,V3))$
&
$(\forall\, W3\ Y3\ X3.\ equal(W3::'a,X3)\ \&\ well\text{-}ordering(Y3::'a,W3) \longrightarrow well\text{-}ordering(Y3::'a,X3))$

lemma *NUM180-1*:
EQU001-0-ax equal &
SET004-0-ax not-homomorphism2 not-homomorphism1
homomorphism compatible operation cantor diagonalise subset-relation
one-to-one choice apply regular function identity-relation
single-valued-class compos powerClass sum-class omega inductive
successor-relation successor image' rng domain range-of INVERSE flip

rot domain-of null-class restrct difference union complement
intersection element-relation second first cross-product ordered-pair
singleton unordered-pair equal universal-class not-subclass-element
member subclass &
SET004-0-eq subclass single-valued-class operation
one-to-one member inductive homomorphism function compatible
unordered-pair union sum-class successor singleton second rot restrct
regular range-of rng powerClass ordered-pair not-subclass-element
not-homomorphism2 not-homomorphism1 INVERSE intersection image′ flip
first domain-of domain difference diagonalise cross-product compos
complement cantor apply equal &
SET004-1-ax range-of function maps apply
application-function singleton-relation element-relation complement
intersection single-valued3 singleton image′ domain single-valued2
second single-valued1 identity-relation INVERSE not-subclass-element
first domain-of domain-relation composition-function compos equal
ordered-pair member universal-class cross-product compose-class
subclass &
SET004-1-eq maps single-valued3 single-valued2 single-valued1 compose-class equal
&
NUM004-0-ax integer-of omega ordinal-multiply
add-relation ordinal-add recursion apply range-of union-of-range-map
function recursion-equation-functions rest-relation rest-of
limit-ordinals kind-1-ordinals successor-relation image′
universal-class sum-class element-relation ordinal-numbers section
not-well-ordering ordered-pair least member well-ordering singleton
domain-of segment null-class intersection asymmetric compos transitive
cross-product connected identity-relation complement restrct subclass
irreflexive symmetrization-of INVERSE union equal &
NUM004-0-eq well-ordering transitive section irreflexive
connected asymmetric symmetrization-of segment rest-of
recursion-equation-functions recursion ordinal-multiply ordinal-add
not-well-ordering least integer-of equal &
(~*subclass*(*limit-ordinals*::′*a*,*ordinal-numbers*)) −−> *False*
⟨*proof*⟩

lemma *NUM228-1*:
EQU001-0-ax equal &
SET004-0-ax not-homomorphism2 not-homomorphism1
homomorphism compatible operation cantor diagonalise subset-relation
one-to-one choice apply regular function identity-relation
single-valued-class compos powerClass sum-class omega inductive
successor-relation successor image′ rng domain range-of INVERSE flip
rot domain-of null-class restrct difference union complement
intersection element-relation second first cross-product ordered-pair
singleton unordered-pair equal universal-class not-subclass-element
member subclass &

SET004-0-eq subclass single-valued-class operation
one-to-one member inductive homomorphism function compatible
unordered-pair union sum-class successor singleton second rot restrct
regular range-of rng powerClass ordered-pair not-subclass-element
not-homomorphism2 not-homomorphism1 INVERSE intersection image′ flip
first domain-of domain difference diagonalise cross-product compos
complement cantor apply equal &
SET004-1-ax range-of function maps apply
application-function singleton-relation element-relation complement
intersection single-valued3 singleton image′ domain single-valued2
second single-valued1 identity-relation INVERSE not-subclass-element
first domain-of domain-relation composition-function compos equal
ordered-pair member universal-class cross-product compose-class
subclass &
SET004-1-eq maps single-valued3 single-valued2 single-valued1 compose-class equal
&
NUM004-0-ax integer-of omega ordinal-multiply
add-relation ordinal-add recursion apply range-of union-of-range-map
function recursion-equation-functions rest-relation rest-of
limit-ordinals kind-1-ordinals successor-relation image′
universal-class sum-class element-relation ordinal-numbers section
not-well-ordering ordered-pair least member well-ordering singleton
domain-of segment null-class intersection asymmetric compos transitive
cross-product connected identity-relation complement restrct subclass
irreflexive symmetrization-of INVERSE union equal &
NUM004-0-eq well-ordering transitive section irreflexive
connected asymmetric symmetrization-of segment rest-of
recursion-equation-functions recursion ordinal-multiply ordinal-add
not-well-ordering least integer-of equal &
(~*function*(*z*)) &
(~*equal*(*recursion-equation-functions*(*z*),*null-class*)) −−> *False*
⟨*proof*⟩

lemma *PLA002-1*:
(∀ *Situation1 Situation2*. *warm*(*Situation1*) | *cold*(*Situation2*)) &
(∀ *Situation*. *at*(*a*::′*a*,*Situation*) −−> *at*(*b*::′*a*,*walk*(*b*::′*a*,*Situation*))) &
(∀ *Situation*. *at*(*a*::′*a*,*Situation*) −−> *at*(*b*::′*a*,*drive*(*b*::′*a*,*Situation*))) &
(∀ *Situation*. *at*(*b*::′*a*,*Situation*) −−> *at*(*a*::′*a*,*walk*(*a*::′*a*,*Situation*))) &
(∀ *Situation*. *at*(*b*::′*a*,*Situation*) −−> *at*(*a*::′*a*,*drive*(*a*::′*a*,*Situation*))) &
(∀ *Situation*. *cold*(*Situation*) & *at*(*b*::′*a*,*Situation*) −−> *at*(*c*::′*a*,*skate*(*c*::′*a*,*Situation*)))
&
(∀ *Situation*. *cold*(*Situation*) & *at*(*c*::′*a*,*Situation*) −−> *at*(*b*::′*a*,*skate*(*b*::′*a*,*Situation*)))
&
(∀ *Situation*. *warm*(*Situation*) & *at*(*b*::′*a*,*Situation*) −−> *at*(*d*::′*a*,*climb*(*d*::′*a*,*Situation*)))
&
(∀ *Situation*. *warm*(*Situation*) & *at*(*d*::′*a*,*Situation*) −−> *at*(*b*::′*a*,*climb*(*b*::′*a*,*Situation*)))
&

$(\forall\, Situation.\ at(c::'a,Situation) \longrightarrow at(d::'a,go(d::'a,Situation)))\ \&$
$(\forall\, Situation.\ at(d::'a,Situation) \longrightarrow at(c::'a,go(c::'a,Situation)))\ \&$
$(\forall\, Situation.\ at(c::'a,Situation) \longrightarrow at(e::'a,go(e::'a,Situation)))\ \&$
$(\forall\, Situation.\ at(e::'a,Situation) \longrightarrow at(c::'a,go(c::'a,Situation)))\ \&$
$(\forall\, Situation.\ at(d::'a,Situation) \longrightarrow at(f::'a,go(f::'a,Situation)))\ \&$
$(\forall\, Situation.\ at(f::'a,Situation) \longrightarrow at(d::'a,go(d::'a,Situation)))\ \&$
$(at(f::'a,s0))\ \&$
$(\forall\, S'.\ \sim at(a::'a,S')) \longrightarrow False$
$\langle proof \rangle$

abbreviation *PLA001-0-ax putdown on pickup do holding table differ clear EMPTY and′ holds* $\equiv$
$(\forall\, X\ Y\ State.\ holds(X::'a,State)\ \&\ holds(Y::'a,State) \longrightarrow holds(and'(X::'a,Y),State))$
$\&$
$(\forall\, State\ X.\ holds(EMPTY::'a,State)\ \&\ holds(clear(X),State)\ \&\ differ(X::'a,table)$
$\longrightarrow holds(holding(X),do(pickup(X),State)))\ \&$
$(\forall\, Y\ X\ State.\ holds(on(X::'a,Y),State)\ \&\ holds(clear(X),State)\ \&\ holds(EMPTY::'a,State)$
$\longrightarrow holds(clear(Y),do(pickup(X),State)))\ \&$
$(\forall\, Y\ State\ X\ Z.\ holds(on(X::'a,Y),State)\ \&\ differ(X::'a,Z) \longrightarrow holds(on(X::'a,Y),do(pickup(Z),State)))$
$\&$
$(\forall\, State\ X\ Z.\ holds(clear(X),State)\ \&\ differ(X::'a,Z) \longrightarrow holds(clear(X),do(pickup(Z),State)))$
$\&$
$(\forall\, X\ Y\ State.\ holds(holding(X),State)\ \&\ holds(clear(Y),State) \longrightarrow holds(EMPTY::'a,do(putdown(X::'a,Y$
$\&$
$(\forall\, X\ Y\ State.\ holds(holding(X),State)\ \&\ holds(clear(Y),State) \longrightarrow holds(on(X::'a,Y),do(putdown(X::'a,Y$
$\&$
$(\forall\, X\ Y\ State.\ holds(holding(X),State)\ \&\ holds(clear(Y),State) \longrightarrow holds(clear(X),do(putdown(X::'a,Y),St$
$\&$
$(\forall\, Z\ W\ X\ Y\ State.\ holds(on(X::'a,Y),State) \longrightarrow holds(on(X::'a,Y),do(putdown(Z::'a,W),State)))$
$\&$
$(\forall\, X\ State\ Z\ Y.\ holds(clear(Z),State)\ \&\ differ(Z::'a,Y) \longrightarrow holds(clear(Z),do(putdown(X::'a,Y),State)))$

abbreviation *PLA001-1-ax EMPTY clear s0 on holds table d c b a differ* $\equiv$
$(\forall\, Y\ X.\ differ(Y::'a,X) \longrightarrow differ(X::'a,Y))\ \&$
$(differ(a::'a,b))\ \&$
$(differ(a::'a,c))\ \&$
$(differ(a::'a,d))\ \&$
$(differ(a::'a,table))\ \&$
$(differ(b::'a,c))\ \&$
$(differ(b::'a,d))\ \&$
$(differ(b::'a,table))\ \&$
$(differ(c::'a,d))\ \&$
$(differ(c::'a,table))\ \&$
$(differ(d::'a,table))\ \&$
$(holds(on(a::'a,table),s0))\ \&$
$(holds(on(b::'a,table),s0))\ \&$
$(holds(on(c::'a,d),s0))\ \&$
$(holds(on(d::'a,table),s0))\ \&$
$(holds(clear(a),s0))\ \&$

(*holds*(*clear*(*b*),*s0*)) &
(*holds*(*clear*(*c*),*s0*)) &
(*holds*(*EMPTY*::'*a*,*s0*)) &
(∀ *State*. *holds*(*clear*(*table*),*State*))

lemma *PLA006-1*:
PLA001-0-ax putdown on pickup do holding table differ clear EMPTY and' holds
&
PLA001-1-ax EMPTY clear s0 on holds table d c b a differ &
(∀ *State*. ~*holds*(*on*(*c*::'*a*,*table*),*State*)) −−> *False*
⟨*proof*⟩

lemma *PLA017-1*:
PLA001-0-ax putdown on pickup do holding table differ clear EMPTY and' holds
&
PLA001-1-ax EMPTY clear s0 on holds table d c b a differ &
(∀ *State*. ~*holds*(*on*(*a*::'*a*,*c*),*State*)) −−> *False*
⟨*proof*⟩

lemma *PLA022-1*:
PLA001-0-ax putdown on pickup do holding table differ clear EMPTY and' holds
&
PLA001-1-ax EMPTY clear s0 on holds table d c b a differ &
(∀ *State*. ~*holds*(*and'*(*on*(*c*::'*a*,*d*),*on*(*a*::'*a*,*c*)),*State*)) −−> *False*
⟨*proof*⟩

lemma *PLA022-2*:
PLA001-0-ax putdown on pickup do holding table differ clear EMPTY and' holds
&
PLA001-1-ax EMPTY clear s0 on holds table d c b a differ &
(∀ *State*. ~*holds*(*and'*(*on*(*a*::'*a*,*c*),*on*(*c*::'*a*,*d*)),*State*)) −−> *False*
⟨*proof*⟩

lemma *PRV001-1*:
(∀ *X Y Z*. *q1*(*X*::'*a*,*Y*,*Z*) & *mless-or-equal*(*X*::'*a*,*Y*) −−> *q2*(*X*::'*a*,*Y*,*Z*)) &
(∀ *X Y Z*. *q1*(*X*::'*a*,*Y*,*Z*) −−> *mless-or-equal*(*X*::'*a*,*Y*) | *q3*(*X*::'*a*,*Y*,*Z*)) &
(∀ *Z X Y*. *q2*(*X*::'*a*,*Y*,*Z*) −−> *q4*(*X*::'*a*,*Y*,*Y*)) &
(∀ *Z Y X*. *q3*(*X*::'*a*,*Y*,*Z*) −−> *q4*(*X*::'*a*,*Y*,*X*)) &
(∀ *X*. *mless-or-equal*(*X*::'*a*,*X*)) &
(∀ *X Y*. *mless-or-equal*(*X*::'*a*,*Y*) & *mless-or-equal*(*Y*::'*a*,*X*) −−> *equal*(*X*::'*a*,*Y*))
&
(∀ *Y X Z*. *mless-or-equal*(*X*::'*a*,*Y*) & *mless-or-equal*(*Y*::'*a*,*Z*) −−> *mless-or-equal*(*X*::'*a*,*Z*))
&
(∀ *Y X*. *mless-or-equal*(*X*::'*a*,*Y*) | *mless-or-equal*(*Y*::'*a*,*X*)) &

$(\forall\, X\ Y.\ equal(X::'a,Y) \longrightarrow mless\text{-}or\text{-}equal(X::'a,Y))$ &
$(\forall\, X\ Y\ Z.\ equal(X::'a,Y)$ & $mless\text{-}or\text{-}equal(X::'a,Z) \longrightarrow mless\text{-}or\text{-}equal(Y::'a,Z))$
&
$(\forall\, X\ Z\ Y.\ equal(X::'a,Y)$ & $mless\text{-}or\text{-}equal(Z::'a,X) \longrightarrow mless\text{-}or\text{-}equal(Z::'a,Y))$
&
$(q1(a::'a,b,c))$ &
$(\forall\, W.\ \sim(q4(a::'a,b,W)$ & $mless\text{-}or\text{-}equal(a::'a,W)$ & $mless\text{-}or\text{-}equal(b::'a,W)$ &
$mless\text{-}or\text{-}equal(W::'a,a)))$ &
$(\forall\, W.\ \sim(q4(a::'a,b,W)$ & $mless\text{-}or\text{-}equal(a::'a,W)$ & $mless\text{-}or\text{-}equal(b::'a,W)$ &
$mless\text{-}or\text{-}equal(W::'a,b))) \longrightarrow False$
$\langle proof \rangle$

abbreviation *SWV001-1-ax mless-THAN successor predecessor equal* $\equiv$
$(\forall\, X.\ equal(predecessor(successor(X)),X))$ &
$(\forall\, X.\ equal(successor(predecessor(X)),X))$ &
$(\forall\, X\ Y.\ equal(predecessor(X),predecessor(Y)) \longrightarrow equal(X::'a,Y))$ &
$(\forall\, X\ Y.\ equal(successor(X),successor(Y)) \longrightarrow equal(X::'a,Y))$ &
$(\forall\, X.\ mless\text{-}THAN(predecessor(X),X))$ &
$(\forall\, X.\ mless\text{-}THAN(X::'a,successor(X)))$ &
$(\forall\, X\ Y\ Z.\ mless\text{-}THAN(X::'a,Y)$ & $mless\text{-}THAN(Y::'a,Z) \longrightarrow mless\text{-}THAN(X::'a,Z))$
&
$(\forall\, X\ Y.\ mless\text{-}THAN(X::'a,Y) \mid mless\text{-}THAN(Y::'a,X) \mid equal(X::'a,Y))$ &
$(\forall\, X.\ \sim mless\text{-}THAN(X::'a,X))$ &
$(\forall\, Y\ X.\ \sim(mless\text{-}THAN(X::'a,Y)$ & $mless\text{-}THAN(Y::'a,X)))$ &
$(\forall\, Y\ X\ Z.\ equal(X::'a,Y)$ & $mless\text{-}THAN(X::'a,Z) \longrightarrow mless\text{-}THAN(Y::'a,Z))$
&
$(\forall\, Y\ Z\ X.\ equal(X::'a,Y)$ & $mless\text{-}THAN(Z::'a,X) \longrightarrow mless\text{-}THAN(Z::'a,Y))$

abbreviation *SWV001-0-eq a successor predecessor equal* $\equiv$
$(\forall\, X\ Y.\ equal(X::'a,Y) \longrightarrow equal(predecessor(X),predecessor(Y)))$ &
$(\forall\, X\ Y.\ equal(X::'a,Y) \longrightarrow equal(successor(X),successor(Y)))$ &
$(\forall\, X\ Y.\ equal(X::'a,Y) \longrightarrow equal(a(X),a(Y)))$

lemma *PRV003-1*:
EQU001-0-ax equal &
SWV001-1-ax mless-THAN successor predecessor equal &
SWV001-0-eq a successor predecessor equal &
$(\sim mless\text{-}THAN(n::'a,j))$ &
$(mless\text{-}THAN(k::'a,j))$ &
$(\sim mless\text{-}THAN(k::'a,i))$ &
$(mless\text{-}THAN(i::'a,n))$ &
$(mless\text{-}THAN(a(j),a(k)))$ &
$(\forall\, X.\ mless\text{-}THAN(X::'a,j)$ & $mless\text{-}THAN(a(X),a(k)) \longrightarrow mless\text{-}THAN(X::'a,i))$
&
$(\forall\, X.\ mless\text{-}THAN(One::'a,i)$ & $mless\text{-}THAN(a(X),a(predecessor(i))) \longrightarrow mless\text{-}THAN(X::'a,i)$
$\mid mless\text{-}THAN(n::'a,X))$ &
$(\forall\, X.\ \sim(mless\text{-}THAN(One::'a,X)$ & $mless\text{-}THAN(X::'a,i)$ & $mless\text{-}THAN(a(X),a(predecessor(X)))))$

&
(*mless-THAN*(*j*::'*a*,*i*)) −−> *False*
⟨*proof*⟩

lemma *PRV005-1*:
EQU001-0-ax equal &
SWV001-1-ax mless-THAN successor predecessor equal &
SWV001-0-eq a successor predecessor equal &
(~*mless-THAN*(*n*::'*a*,*k*)) &
(~*mless-THAN*(*k*::'*a*,*l*)) &
(~*mless-THAN*(*k*::'*a*,*i*)) &
(*mless-THAN*(*l*::'*a*,*n*)) &
(*mless-THAN*(*One*::'*a*,*l*)) &
(*mless-THAN*(*a*(*k*),*a*(*predecessor*(*l*)))) &
(∀ *X*. *mless-THAN*(*X*::'*a*,*successor*(*n*)) & *mless-THAN*(*a*(*X*),*a*(*k*)) −−> *mless-THAN*(*X*::'*a*,*l*))
&
(∀ *X*. *mless-THAN*(*One*::'*a*,*l*) & *mless-THAN*(*a*(*X*),*a*(*predecessor*(*l*))) −−> *mless-THAN*(*X*::'*a*,*l*)
| *mless-THAN*(*n*::'*a*,*X*)) &
(∀ *X*. ~(*mless-THAN*(*One*::'*a*,*X*) & *mless-THAN*(*X*::'*a*,*l*) & *mless-THAN*(*a*(*X*),*a*(*predecessor*(*X*)))))
−−> *False*
⟨*proof*⟩

lemma *PRV006-1*:
EQU001-0-ax equal &
SWV001-1-ax mless-THAN successor predecessor equal &
SWV001-0-eq a successor predecessor equal &
(~*mless-THAN*(*n*::'*a*,*m*)) &
(*mless-THAN*(*i*::'*a*,*m*)) &
(*mless-THAN*(*i*::'*a*,*n*)) &
(~*mless-THAN*(*i*::'*a*,*One*)) &
(*mless-THAN*(*a*(*i*),*a*(*m*))) &
(∀ *X*. *mless-THAN*(*X*::'*a*,*successor*(*n*)) & *mless-THAN*(*a*(*X*),*a*(*m*)) −−> *mless-THAN*(*X*::'*a*,*i*))
&
(∀ *X*. *mless-THAN*(*One*::'*a*,*i*) & *mless-THAN*(*a*(*X*),*a*(*predecessor*(*i*))) −−> *mless-THAN*(*X*::'*a*,*i*)
| *mless-THAN*(*n*::'*a*,*X*)) &
(∀ *X*. ~(*mless-THAN*(*One*::'*a*,*X*) & *mless-THAN*(*X*::'*a*,*i*) & *mless-THAN*(*a*(*X*),*a*(*predecessor*(*X*)))))
−−> *False*
⟨*proof*⟩

lemma *PRV009-1*:
(∀ *Y X*. *mless-or-equal*(*X*::'*a*, *Y*) | *mless*(*Y*::'*a*,*X*)) &
(*mless*(*j*::'*a*,*i*)) &
(*mless-or-equal*(*m*::'*a*,*p*)) &
(*mless-or-equal*(*p*::'*a*,*q*)) &
(*mless-or-equal*(*q*::'*a*,*n*)) &
(∀ *X Y*. *mless-or-equal*(*m*::'*a*,*X*) & *mless*(*X*::'*a*,*i*) & *mless*(*j*::'*a*, *Y*) & *mless-or-equal*(*Y*::'*a*,*n*)

$--> mless\text{-}or\text{-}equal(a(X),a(Y))) \;\&$
$(\forall X\ Y.\ mless\text{-}or\text{-}equal(m::'a,X)\ \&\ mless\text{-}or\text{-}equal(X::'a,Y)\ \&\ mless\text{-}or\text{-}equal(Y::'a,j)$
$--> mless\text{-}or\text{-}equal(a(X),a(Y))) \;\&$
$(\forall X\ Y.\ mless\text{-}or\text{-}equal(i::'a,X)\ \&\ mless\text{-}or\text{-}equal(X::'a,Y)\ \&\ mless\text{-}or\text{-}equal(Y::'a,n)$
$--> mless\text{-}or\text{-}equal(a(X),a(Y))) \;\&$
$(\sim mless\text{-}or\text{-}equal(a(p),a(q))) --> False$
$\langle proof \rangle$

lemma *PUZ012-1*:
$(\forall X.\ equal\text{-}fruits(X::'a,X)) \;\&$
$(\forall X.\ equal\text{-}boxes(X::'a,X)) \;\&$
$(\forall X\ Y.\ \sim(label(X::'a,Y)\ \&\ contains(X::'a,Y))) \;\&$
$(\forall X.\ contains(boxa::'a,X) \mid contains(boxb::'a,X) \mid contains(boxc::'a,X)) \;\&$
$(\forall X.\ contains(X::'a,apples) \mid contains(X::'a,bananas) \mid contains(X::'a,oranges))$
$\&$
$(\forall X\ Y\ Z.\ contains(X::'a,Y)\ \&\ contains(X::'a,Z) --> equal\text{-}fruits(Y::'a,Z)) \;\&$
$(\forall Y\ X\ Z.\ contains(X::'a,Y)\ \&\ contains(Z::'a,Y) --> equal\text{-}boxes(X::'a,Z)) \;\&$
$(\sim equal\text{-}boxes(boxa::'a,boxb)) \;\&$
$(\sim equal\text{-}boxes(boxb::'a,boxc)) \;\&$
$(\sim equal\text{-}boxes(boxa::'a,boxc)) \;\&$
$(\sim equal\text{-}fruits(apples::'a,bananas)) \;\&$
$(\sim equal\text{-}fruits(bananas::'a,oranges)) \;\&$
$(\sim equal\text{-}fruits(apples::'a,oranges)) \;\&$
$(label(boxa::'a,apples)) \;\&$
$(label(boxb::'a,oranges)) \;\&$
$(label(boxc::'a,bananas)) \;\&$
$(contains(boxb::'a,apples)) \;\&$
$(\sim(contains(boxa::'a,bananas)\ \&\ contains(boxc::'a,oranges))) --> False$
$\langle proof \rangle$

lemma *PUZ020-1*:
$EQU001\text{-}0\text{-}ax\ equal \;\&$
$(\forall A\ B.\ equal(A::'a,B) --> equal(statement\text{-}by(A),statement\text{-}by(B))) \;\&$
$(\forall X.\ person(X) --> knight(X) \mid knave(X)) \;\&$
$(\forall X.\ \sim(person(X)\ \&\ knight(X)\ \&\ knave(X))) \;\&$
$(\forall X\ Y.\ says(X::'a,Y)\ \&\ a\text{-}truth(Y) --> a\text{-}truth(Y)) \;\&$
$(\forall X\ Y.\ \sim(says(X::'a,Y)\ \&\ equal(X::'a,Y))) \;\&$
$(\forall Y\ X.\ says(X::'a,Y) --> equal(Y::'a,statement\text{-}by(X))) \;\&$
$(\forall X\ Y.\ \sim(person(X)\ \&\ equal(X::'a,statement\text{-}by(Y)))) \;\&$
$(\forall X.\ person(X)\ \&\ a\text{-}truth(statement\text{-}by(X)) --> knight(X)) \;\&$
$(\forall X.\ person(X) --> a\text{-}truth(statement\text{-}by(X)) \mid knave(X)) \;\&$
$(\forall X\ Y.\ equal(X::'a,Y)\ \&\ knight(X) --> knight(Y)) \;\&$
$(\forall X\ Y.\ equal(X::'a,Y)\ \&\ knave(X) --> knave(Y)) \;\&$
$(\forall X\ Y.\ equal(X::'a,Y)\ \&\ person(X) --> person(Y)) \;\&$
$(\forall X\ Y\ Z.\ equal(X::'a,Y)\ \&\ says(X::'a,Z) --> says(Y::'a,Z)) \;\&$
$(\forall X\ Z\ Y.\ equal(X::'a,Y)\ \&\ says(Z::'a,X) --> says(Z::'a,Y)) \;\&$
$(\forall X\ Y.\ equal(X::'a,Y)\ \&\ a\text{-}truth(X) --> a\text{-}truth(Y)) \;\&$

$(\forall X\ Y.\ knight(X)\ \&\ says(X::'a,Y)\ -->\ a\text{-}truth(Y))\ \&$
$(\forall X\ Y.\ \sim(knave(X)\ \&\ says(X::'a,Y)\ \&\ a\text{-}truth(Y)))\ \&$
$(person(husband))\ \&$
$(person(wife))\ \&$
$(\sim equal(husband::'a,wife))\ \&$
$(says(husband::'a,statement\text{-}by(husband)))\ \&$
$(a\text{-}truth(statement\text{-}by(husband))\ \&\ knight(husband)\ -->\ knight(wife))\ \&$
$(knight(husband)\ -->\ a\text{-}truth(statement\text{-}by(husband)))\ \&$
$(a\text{-}truth(statement\text{-}by(husband))\ |\ knight(wife))\ \&$
$(knight(wife)\ -->\ a\text{-}truth(statement\text{-}by(husband)))\ \&$
$(\sim knight(husband))\ -->\ False$
$\langle proof\rangle$

lemma *PUZ025-1*:
$(\forall X.\ a\text{-}truth(truthteller(X))\ |\ a\text{-}truth(liar(X)))\ \&$
$(\forall X.\ \sim(a\text{-}truth(truthteller(X))\ \&\ a\text{-}truth(liar(X))))\ \&$
$(\forall Truthteller\ Statement.\ a\text{-}truth(truthteller(Truthteller))\ \&\ a\text{-}truth(says(Truthteller::'a,Statement))$
$-->\ a\text{-}truth(Statement))\ \&$
$(\forall Liar\ Statement.\ \sim(a\text{-}truth(liar(Liar))\ \&\ a\text{-}truth(says(Liar::'a,Statement))\ \&$
$a\text{-}truth(Statement)))\ \&$
$(\forall Statement\ Truthteller.\ a\text{-}truth(Statement)\ \&\ a\text{-}truth(says(Truthteller::'a,Statement))$
$-->\ a\text{-}truth(truthteller(Truthteller)))\ \&$
$(\forall Statement\ Liar.\ a\text{-}truth(says(Liar::'a,Statement))\ -->\ a\text{-}truth(Statement)\ |$
$a\text{-}truth(liar(Liar)))\ \&$
$(\forall Z\ X\ Y.\ people(X::'a,Y,Z)\ \&\ a\text{-}truth(liar(X))\ \&\ a\text{-}truth(liar(Y))\ -->\ a\text{-}truth(equal\text{-}type(X::'a,Y)))$
$\&$
$(\forall Z\ X\ Y.\ people(X::'a,Y,Z)\ \&\ a\text{-}truth(truthteller(X))\ \&\ a\text{-}truth(truthteller(Y))$
$-->\ a\text{-}truth(equal\text{-}type(X::'a,Y)))\ \&$
$(\forall X\ Y.\ a\text{-}truth(equal\text{-}type(X::'a,Y))\ \&\ a\text{-}truth(truthteller(X))\ -->\ a\text{-}truth(truthteller(Y)))$
$\&$
$(\forall X\ Y.\ a\text{-}truth(equal\text{-}type(X::'a,Y))\ \&\ a\text{-}truth(liar(X))\ -->\ a\text{-}truth(liar(Y)))$
$\&$
$(\forall X\ Y.\ a\text{-}truth(truthteller(X))\ -->\ a\text{-}truth(equal\text{-}type(X::'a,Y))\ |\ a\text{-}truth(liar(Y)))$
$\&$
$(\forall X\ Y.\ a\text{-}truth(liar(X))\ -->\ a\text{-}truth(equal\text{-}type(X::'a,Y))\ |\ a\text{-}truth(truthteller(Y)))$
$\&$
$(\forall Y\ X.\ a\text{-}truth(equal\text{-}type(X::'a,Y))\ -->\ a\text{-}truth(equal\text{-}type(Y::'a,X)))\ \&$
$(\forall X\ Y.\ ask\text{-}1\text{-}if\text{-}2(X::'a,Y)\ \&\ a\text{-}truth(truthteller(X))\ \&\ a\text{-}truth(Y)\ -->\ an$
$swer(yes))\ \&$
$(\forall X\ Y.\ ask\text{-}1\text{-}if\text{-}2(X::'a,Y)\ \&\ a\text{-}truth(truthteller(X))\ -->\ a\text{-}truth(Y)\ |\ an$
$swer(no))\ \&$
$(\forall X\ Y.\ ask\text{-}1\text{-}if\text{-}2(X::'a,Y)\ \&\ a\text{-}truth(liar(X))\ \&\ a\text{-}truth(Y)\ -->\ answer(no))$
$\&$
$(\forall X\ Y.\ ask\text{-}1\text{-}if\text{-}2(X::'a,Y)\ \&\ a\text{-}truth(liar(X))\ -->\ a\text{-}truth(Y)\ |\ answer(yes))$
$\&$
$(people(b::'a,c,a))\ \&$
$(people(a::'a,b,a))\ \&$
$(people(a::'a,c,b))\ \&$

(*people*(*c*::'*a*,*b*,*a*)) &
(*a-truth*(*says*(*a*::'*a*,*equal-type*(*b*::'*a*,*c*)))) &
(*ask-1-if-2*(*c*::'*a*,*equal-type*(*a*::'*a*,*b*))) &
(∀ *Answer*. ~*answer*(*Answer*)) −−> *False*
⟨*proof*⟩

lemma *PUZ029-1*:
(∀ *X*. *dances-on-tightropes*(*X*) | *eats-pennybuns*(*X*) | *old*(*X*)) &
(∀ *X*. *pig*(*X*) & *liable-to-giddiness*(*X*) −−> *treated-with-respect*(*X*)) &
(∀ *X*. *wise*(*X*) & *balloonist*(*X*) −−> *has-umbrella*(*X*)) &
(∀ *X*. ~(*looks-ridiculous*(*X*) & *eats-pennybuns*(*X*) & *eats-lunch-in-public*(*X*))) &
(∀ *X*. *balloonist*(*X*) & *young*(*X*) −−> *liable-to-giddiness*(*X*)) &
(∀ *X*. *fat*(*X*) & *looks-ridiculous*(*X*) −−> *dances-on-tightropes*(*X*) | *eats-lunch-in-public*(*X*))
&
(∀ *X*. ~(*liable-to-giddiness*(*X*) & *wise*(*X*) & *dances-on-tightropes*(*X*))) &
(∀ *X*. *pig*(*X*) & *has-umbrella*(*X*) −−> *looks-ridiculous*(*X*)) &
(∀ *X*. *treated-with-respect*(*X*) −−> *dances-on-tightropes*(*X*) | *fat*(*X*)) &
(∀ *X*. *young*(*X*) | *old*(*X*)) &
(∀ *X*. ~(*young*(*X*) & *old*(*X*))) &
(*wise*(*piggy*)) &
(*young*(*piggy*)) &
(*pig*(*piggy*)) &
(*balloonist*(*piggy*)) −−> *False*
⟨*proof*⟩

abbreviation *RNG001-0-ax equal additive-inverse add multiply product additive-identity sum* ≡
(∀ *X*. *sum*(*additive-identity*::'*a*,*X*,*X*)) &
(∀ *X*. *sum*(*X*::'*a*,*additive-identity*,*X*)) &
(∀ *X Y*. *product*(*X*::'*a*,*Y*,*multiply*(*X*::'*a*,*Y*))) &
(∀ *X Y*. *sum*(*X*::'*a*,*Y*,*add*(*X*::'*a*,*Y*))) &
(∀ *X*. *sum*(*additive-inverse*(*X*),*X*,*additive-identity*)) &
(∀ *X*. *sum*(*X*::'*a*,*additive-inverse*(*X*),*additive-identity*)) &
(∀ *Y U Z X V W*. *sum*(*X*::'*a*,*Y*,*U*) & *sum*(*Y*::'*a*,*Z*,*V*) & *sum*(*U*::'*a*,*Z*,*W*) −−>
sum(*X*::'*a*,*V*,*W*)) &
(∀ *Y X V U Z W*. *sum*(*X*::'*a*,*Y*,*U*) & *sum*(*Y*::'*a*,*Z*,*V*) & *sum*(*X*::'*a*,*V*,*W*) −−>
sum(*U*::'*a*,*Z*,*W*)) &
(∀ *Y X Z*. *sum*(*X*::'*a*,*Y*,*Z*) −−> *sum*(*Y*::'*a*,*X*,*Z*)) &
(∀ *Y U Z X V W*. *product*(*X*::'*a*,*Y*,*U*) & *product*(*Y*::'*a*,*Z*,*V*) & *product*(*U*::'*a*,*Z*,*W*)
−−> *product*(*X*::'*a*,*V*,*W*)) &
(∀ *Y X V U Z W*. *product*(*X*::'*a*,*Y*,*U*) & *product*(*Y*::'*a*,*Z*,*V*) & *product*(*X*::'*a*,*V*,*W*)
−−> *product*(*U*::'*a*,*Z*,*W*)) &
(∀ *Y Z X V3 V1 V2 V4*. *product*(*X*::'*a*,*Y*,*V1*) & *product*(*X*::'*a*,*Z*,*V2*) & *sum*(*Y*::'*a*,*Z*,*V3*)
& *product*(*X*::'*a*,*V3*,*V4*) −−> *sum*(*V1*::'*a*,*V2*,*V4*)) &
(∀ *Y Z V1 V2 X V3 V4*. *product*(*X*::'*a*,*Y*,*V1*) & *product*(*X*::'*a*,*Z*,*V2*) & *sum*(*Y*::'*a*,*Z*,*V3*)
& *sum*(*V1*::'*a*,*V2*,*V4*) −−> *product*(*X*::'*a*,*V3*,*V4*)) &
(∀ *Y Z V3 X V1 V2 V4*. *product*(*Y*::'*a*,*X*,*V1*) & *product*(*Z*::'*a*,*X*,*V2*) & *sum*(*Y*::'*a*,*Z*,*V3*)

& $product(V3::'a,X,V4)$ −−> $sum(V1::'a,V2,V4)$) &
($\forall$ Y Z $V1$ $V2$ $V3$ X $V4$. $product(Y::'a,X,V1)$ & $product(Z::'a,X,V2)$ & $sum(Y::'a,Z,V3)$
& $sum(V1::'a,V2,V4)$ −−> $product(V3::'a,X,V4)$) &
($\forall$ X Y U V. $sum(X::'a,Y,U)$ & $sum(X::'a,Y,V)$ −−> $equal(U::'a,V)$) &
($\forall$ X Y U V. $product(X::'a,Y,U)$ & $product(X::'a,Y,V)$ −−> $equal(U::'a,V)$)

abbreviation *RNG001-0-eq product multiply sum add additive-inverse equal* ≡
($\forall$ X Y. $equal(X::'a,Y)$ −−> $equal(additive\text{-}inverse(X),additive\text{-}inverse(Y))$) &
($\forall$ X Y W. $equal(X::'a,Y)$ −−> $equal(add(X::'a,W),add(Y::'a,W))$) &
($\forall$ X W Y. $equal(X::'a,Y)$ −−> $equal(add(W::'a,X),add(W::'a,Y))$) &
($\forall$ X Y W Z. $equal(X::'a,Y)$ & $sum(X::'a,W,Z)$ −−> $sum(Y::'a,W,Z)$) &
($\forall$ X W Y Z. $equal(X::'a,Y)$ & $sum(W::'a,X,Z)$ −−> $sum(W::'a,Y,Z)$) &
($\forall$ X W Z Y. $equal(X::'a,Y)$ & $sum(W::'a,Z,X)$ −−> $sum(W::'a,Z,Y)$) &
($\forall$ X Y W. $equal(X::'a,Y)$ −−> $equal(multiply(X::'a,W),multiply(Y::'a,W))$)
&
($\forall$ X W Y. $equal(X::'a,Y)$ −−> $equal(multiply(W::'a,X),multiply(W::'a,Y))$)
&
($\forall$ X Y W Z. $equal(X::'a,Y)$ & $product(X::'a,W,Z)$ −−> $product(Y::'a,W,Z)$)
&
($\forall$ X W Y Z. $equal(X::'a,Y)$ & $product(W::'a,X,Z)$ −−> $product(W::'a,Y,Z)$)
&
($\forall$ X W Z Y. $equal(X::'a,Y)$ & $product(W::'a,Z,X)$ −−> $product(W::'a,Z,Y)$)

lemma *RNG001-3*:
($\forall$ X. $sum(additive\text{-}identity::'a,X,X)$) &
($\forall$ X. $sum(additive\text{-}inverse(X),X,additive\text{-}identity)$) &
($\forall$ Y U Z X V W. $sum(X::'a,Y,U)$ & $sum(Y::'a,Z,V)$ & $sum(U::'a,Z,W)$ −−>
$sum(X::'a,V,W)$) &
($\forall$ Y X V U Z W. $sum(X::'a,Y,U)$ & $sum(Y::'a,Z,V)$ & $sum(X::'a,V,W)$ −−>
$sum(U::'a,Z,W)$) &
($\forall$ X Y. $product(X::'a,Y,multiply(X::'a,Y))$) &
($\forall$ Y Z X $V3$ $V1$ $V2$ $V4$. $product(X::'a,Y,V1)$ & $product(X::'a,Z,V2)$ & $sum(Y::'a,Z,V3)$
& $product(X::'a,V3,V4)$ −−> $sum(V1::'a,V2,V4)$) &
($\forall$ Y Z $V1$ $V2$ X $V3$ $V4$. $product(X::'a,Y,V1)$ & $product(X::'a,Z,V2)$ & $sum(Y::'a,Z,V3)$
& $sum(V1::'a,V2,V4)$ −−> $product(X::'a,V3,V4)$) &
(~$product(a::'a,additive\text{-}identity,additive\text{-}identity)$) −−> $False$
⟨*proof*⟩

abbreviation *RNG-other-ax multiply add equal product additive-identity additive-inverse
sum* ≡
($\forall$ X. $sum(X::'a,additive\text{-}inverse(X),additive\text{-}identity)$) &
($\forall$ Y U Z X V W. $sum(X::'a,Y,U)$ & $sum(Y::'a,Z,V)$ & $sum(U::'a,Z,W)$ −−>
$sum(X::'a,V,W)$) &
($\forall$ Y X V U Z W. $sum(X::'a,Y,U)$ & $sum(Y::'a,Z,V)$ & $sum(X::'a,V,W)$ −−>
$sum(U::'a,Z,W)$) &
($\forall$ Y X Z. $sum(X::'a,Y,Z)$ −−> $sum(Y::'a,X,Z)$) &
($\forall$ Y U Z X V W. $product(X::'a,Y,U)$ & $product(Y::'a,Z,V)$ & $product(U::'a,Z,W)$
−−> $product(X::'a,V,W)$) &

(∀ Y X V U Z W. product(X::'a,Y,U) & product(Y::'a,Z,V) & product(X::'a,V,W)
−−> product(U::'a,Z,W)) &
(∀ Y Z X V3 V1 V2 V4. product(X::'a,Y,V1) & product(X::'a,Z,V2) & sum(Y::'a,Z,V3)
& product(X::'a,V3,V4) −−> sum(V1::'a,V2,V4)) &
(∀ Y Z V1 V2 X V3 V4. product(X::'a,Y,V1) & product(X::'a,Z,V2) & sum(Y::'a,Z,V3)
& sum(V1::'a,V2,V4) −−> product(X::'a,V3,V4)) &
(∀ Y Z V3 X V1 V2 V4. product(Y::'a,X,V1) & product(Z::'a,X,V2) & sum(Y::'a,Z,V3)
& product(V3::'a,X,V4) −−> sum(V1::'a,V2,V4)) &
(∀ Y Z V1 V2 V3 X V4. product(Y::'a,X,V1) & product(Z::'a,X,V2) & sum(Y::'a,Z,V3)
& sum(V1::'a,V2,V4) −−> product(V3::'a,X,V4)) &
(∀ X Y U V. sum(X::'a,Y,U) & sum(X::'a,Y,V) −−> equal(U::'a,V)) &
(∀ X Y U V. product(X::'a,Y,U) & product(X::'a,Y,V) −−> equal(U::'a,V))
&
(∀ X Y. equal(X::'a,Y) −−> equal(additive-inverse(X),additive-inverse(Y))) &
(∀ X Y W. equal(X::'a,Y) −−> equal(add(X::'a,W),add(Y::'a,W))) &
(∀ X Y W Z. equal(X::'a,Y) & sum(X::'a,W,Z) −−> sum(Y::'a,W,Z)) &
(∀ X W Y Z. equal(X::'a,Y) & sum(W::'a,X,Z) −−> sum(W::'a,Y,Z)) &
(∀ X W Z Y. equal(X::'a,Y) & sum(W::'a,Z,X) −−> sum(W::'a,Z,Y)) &
(∀ X Y W. equal(X::'a,Y) −−> equal(multiply(X::'a,W),multiply(Y::'a,W)))
&
(∀ X Y W Z. equal(X::'a,Y) & product(X::'a,W,Z) −−> product(Y::'a,W,Z))
&
(∀ X W Y Z. equal(X::'a,Y) & product(W::'a,X,Z) −−> product(W::'a,Y,Z))
&
(∀ X W Z Y. equal(X::'a,Y) & product(W::'a,Z,X) −−> product(W::'a,Z,Y))

lemma *RNG001-5*:
EQU001-0-ax equal &
(∀ X. sum(additive-identity::'a,X,X)) &
(∀ X. sum(X::'a,additive-identity,X)) &
(∀ X Y. product(X::'a,Y,multiply(X::'a,Y))) &
(∀ X Y. sum(X::'a,Y,add(X::'a,Y))) &
(∀ X. sum(additive-inverse(X),X,additive-identity)) &
RNG-other-ax multiply add equal product additive-identity additive-inverse sum
&
(~product(a::'a,additive-identity,additive-identity)) −−> *False*
⟨*proof*⟩

lemma *RNG011-5*:
EQU001-0-ax equal &
(∀ A B C. equal(A::'a,B) −−> equal(add(A::'a,C),add(B::'a,C))) &
(∀ D F' E. equal(D::'a,E) −−> equal(add(F'::'a,D),add(F'::'a,E))) &
(∀ G H. equal(G::'a,H) −−> equal(additive-inverse(G),additive-inverse(H))) &
(∀ I' J K'. equal(I'::'a,J) −−> equal(multiply(I'::'a,K'),multiply(J::'a,K'))) &
(∀ L N M. equal(L::'a,M) −−> equal(multiply(N::'a,L),multiply(N::'a,M))) &
(∀ A B C D. equal(A::'a,B) −−> equal(associator(A::'a,C,D),associator(B::'a,C,D)))

$$
\begin{array}{l}
\& \\
\quad (\forall\, E\ G\ F'\ H.\ equal(E::'a,F') \dashrightarrow equal(associator(G::'a,E,H),associator(G::'a,F',H))) \\
\& \\
\quad (\forall\, I'\ K'\ L\ J.\ equal(I'::'a,J) \dashrightarrow equal(associator(K'::'a,L,I'),associator(K'::'a,L,J))) \\
\& \\
\quad (\forall\, M\ N\ O'.\ equal(M::'a,N) \dashrightarrow equal(commutator(M::'a,O'),commutator(N::'a,O'))) \\
\& \\
\quad (\forall\, P\ R\ Q.\ equal(P::'a,Q) \dashrightarrow equal(commutator(R::'a,P),commutator(R::'a,Q))) \\
\& \\
\quad (\forall\, Y\ X.\ equal(add(X::'a,Y),add(Y::'a,X))) \ \& \\
\quad (\forall\, X\ Y\ Z.\ equal(add(add(X::'a,Y),Z),add(X::'a,add(Y::'a,Z)))) \ \& \\
\quad (\forall\, X.\ equal(add(X::'a,additive\text{-}identity),X)) \ \& \\
\quad (\forall\, X.\ equal(add(additive\text{-}identity::'a,X),X)) \ \& \\
\quad (\forall\, X.\ equal(add(X::'a,additive\text{-}inverse(X)),additive\text{-}identity)) \ \& \\
\quad (\forall\, X.\ equal(add(additive\text{-}inverse(X),X),additive\text{-}identity)) \ \& \\
\quad (equal(additive\text{-}inverse(additive\text{-}identity),additive\text{-}identity)) \ \& \\
\quad (\forall\, X\ Y.\ equal(add(X::'a,add(additive\text{-}inverse(X),Y)),Y)) \ \& \\
\quad (\forall\, X\ Y.\ equal(additive\text{-}inverse(add(X::'a,Y)),add(additive\text{-}inverse(X),additive\text{-}inverse(Y)))) \\
\& \\
\quad (\forall\, X.\ equal(additive\text{-}inverse(additive\text{-}inverse(X)),X)) \ \& \\
\quad (\forall\, X.\ equal(multiply(X::'a,additive\text{-}identity),additive\text{-}identity)) \ \& \\
\quad (\forall\, X.\ equal(multiply(additive\text{-}identity::'a,X),additive\text{-}identity)) \ \& \\
\quad (\forall\, X\ Y.\ equal(multiply(additive\text{-}inverse(X),additive\text{-}inverse(Y)),multiply(X::'a,Y))) \\
\& \\
\quad (\forall\, X\ Y.\ equal(multiply(X::'a,additive\text{-}inverse(Y)),additive\text{-}inverse(multiply(X::'a,Y)))) \\
\& \\
\quad (\forall\, X\ Y.\ equal(multiply(additive\text{-}inverse(X),Y),additive\text{-}inverse(multiply(X::'a,Y)))) \\
\& \\
\quad (\forall\, Y\ X\ Z.\ equal(multiply(X::'a,add(Y::'a,Z)),add(multiply(X::'a,Y),multiply(X::'a,Z)))) \\
\& \\
\quad (\forall\, X\ Y\ Z.\ equal(multiply(add(X::'a,Y),Z),add(multiply(X::'a,Z),multiply(Y::'a,Z)))) \\
\& \\
\quad (\forall\, X\ Y.\ equal(multiply(multiply(X::'a,Y),Y),multiply(X::'a,multiply(Y::'a,Y)))) \\
\& \\
\quad (\forall\, X\ Y\ Z.\ equal(associator(X::'a,Y,Z),add(multiply(multiply(X::'a,Y),Z),additive\text{-}inverse(multiply(X::'a,m \\
\& \\
\quad (\forall\, X\ Y.\ equal(commutator(X::'a,Y),add(multiply(Y::'a,X),additive\text{-}inverse(multiply(X::'a,Y))))) \\
\& \\
\quad (\forall\, X\ Y.\ equal(multiply(multiply(associator(X::'a,X,Y),X),associator(X::'a,X,Y)),additive\text{-}identity)) \\
\& \\
\quad (\sim equal(multiply(multiply(associator(a::'a,a,b),a),associator(a::'a,a,b)),additive\text{-}identity)) \\
\dashrightarrow False \\
\quad \langle proof \rangle
\end{array}
$$

lemma *RNG023-6*:

$$
\begin{array}{l}
\quad EQU001\text{-}0\text{-}ax\ equal\ \& \\
\quad (\forall\, Y\ X.\ equal(add(X::'a,Y),add(Y::'a,X))) \ \& \\
\quad (\forall\, X\ Y\ Z.\ equal(add(X::'a,add(Y::'a,Z)),add(add(X::'a,Y),Z))) \ \&
\end{array}
$$

$(\forall X.\ equal(add(additive\text{-}identity::'a,X),X))\ \&$
$(\forall X.\ equal(add(X::'a,additive\text{-}identity),X))\ \&$
$(\forall X.\ equal(multiply(additive\text{-}identity::'a,X),additive\text{-}identity))\ \&$
$(\forall X.\ equal(multiply(X::'a,additive\text{-}identity),additive\text{-}identity))\ \&$
$(\forall X.\ equal(add(additive\text{-}inverse(X),X),additive\text{-}identity))\ \&$
$(\forall X.\ equal(add(X::'a,additive\text{-}inverse(X)),additive\text{-}identity))\ \&$
$(\forall Y\ X\ Z.\ equal(multiply(X::'a,add(Y::'a,Z)),add(multiply(X::'a,Y),multiply(X::'a,Z))))$
$\&$
$(\forall X\ Y\ Z.\ equal(multiply(add(X::'a,Y),Z),add(multiply(X::'a,Z),multiply(Y::'a,Z))))$
$\&$
$(\forall X.\ equal(additive\text{-}inverse(additive\text{-}inverse(X)),X))\ \&$
$(\forall X\ Y.\ equal(multiply(multiply(X::'a,Y),Y),multiply(X::'a,multiply(Y::'a,Y))))$
$\&$
$(\forall X\ Y.\ equal(multiply(multiply(X::'a,X),Y),multiply(X::'a,multiply(X::'a,Y))))$
$\&$
$(\forall X\ Y\ Z.\ equal(associator(X::'a,Y,Z),add(multiply(multiply(X::'a,Y),Z),additive\text{-}inverse(multiply(X::'a,m$
$\&$
$(\forall X\ Y.\ equal(commutator(X::'a,Y),add(multiply(Y::'a,X),additive\text{-}inverse(multiply(X::'a,Y)))))$
$\&$
$(\forall D\ E\ F'.\ equal(D::'a,E)\ \text{--}\!\!>\ equal(add(D::'a,F'),add(E::'a,F')))\ \&$
$(\forall G\ I'\ H.\ equal(G::'a,H)\ \text{--}\!\!>\ equal(add(I'::'a,G),add(I'::'a,H)))\ \&$
$(\forall J\ K'.\ equal(J::'a,K')\ \text{--}\!\!>\ equal(additive\text{-}inverse(J),additive\text{-}inverse(K')))\ \&$
$(\forall L\ M\ N\ O'.\ equal(L::'a,M)\ \text{--}\!\!>\ equal(associator(L::'a,N,O'),associator(M::'a,N,O')))$
$\&$
$(\forall P\ R\ Q\ S'.\ equal(P::'a,Q)\ \text{--}\!\!>\ equal(associator(R::'a,P,S'),associator(R::'a,Q,S')))$
$\&$
$(\forall T'\ V\ W\ U.\ equal(T'::'a,U)\ \text{--}\!\!>\ equal(associator(V::'a,W,T'),associator(V::'a,W,U)))$
$\&$
$(\forall X\ Y\ Z.\ equal(X::'a,Y)\ \text{--}\!\!>\ equal(commutator(X::'a,Z),commutator(Y::'a,Z)))$
$\&$
$(\forall A1\ C1\ B1.\ equal(A1::'a,B1)\ \text{--}\!\!>\ equal(commutator(C1::'a,A1),commutator(C1::'a,B1)))$
$\&$
$(\forall D1\ E1\ F1.\ equal(D1::'a,E1)\ \text{--}\!\!>\ equal(multiply(D1::'a,F1),multiply(E1::'a,F1)))$
$\&$
$(\forall G1\ I1\ H1.\ equal(G1::'a,H1)\ \text{--}\!\!>\ equal(multiply(I1::'a,G1),multiply(I1::'a,H1)))$
$\&$
$(\sim equal(associator(x::'a,x,y),additive\text{-}identity))\ \text{--}\!\!>\ False$
$\langle proof \rangle$

lemma *RNG028-2*:
EQU001-0-ax equal $\&$
$(\forall X.\ equal(add(additive\text{-}identity::'a,X),X))\ \&$
$(\forall X.\ equal(multiply(additive\text{-}identity::'a,X),additive\text{-}identity))\ \&$
$(\forall X.\ equal(multiply(X::'a,additive\text{-}identity),additive\text{-}identity))\ \&$
$(\forall X.\ equal(add(additive\text{-}inverse(X),X),additive\text{-}identity))\ \&$
$(\forall X\ Y.\ equal(additive\text{-}inverse(add(X::'a,Y)),add(additive\text{-}inverse(X),additive\text{-}inverse(Y))))$
$\&$
$(\forall X.\ equal(additive\text{-}inverse(additive\text{-}inverse(X)),X))\ \&$

$(\forall\ Y\ X\ Z.\ equal(multiply(X::'a,add(Y::'a,Z)),add(multiply(X::'a,Y),multiply(X::'a,Z))))$
&
$(\forall\ X\ Y\ Z.\ equal(multiply(add(X::'a,Y),Z),add(multiply(X::'a,Z),multiply(Y::'a,Z))))$
&
$(\forall\ X\ Y.\ equal(multiply(multiply(X::'a,Y),Y),multiply(X::'a,multiply(Y::'a,Y))))$
&
$(\forall\ X\ Y.\ equal(multiply(multiply(X::'a,X),Y),multiply(X::'a,multiply(X::'a,Y))))$
&
$(\forall\ X\ Y.\ equal(multiply(additive\text{-}inverse(X),Y),additive\text{-}inverse(multiply(X::'a,Y))))$
&
$(\forall\ X\ Y.\ equal(multiply(X::'a,additive\text{-}inverse(Y)),additive\text{-}inverse(multiply(X::'a,Y))))$
&
$(equal(additive\text{-}inverse(additive\text{-}identity),additive\text{-}identity))$ &
$(\forall\ Y\ X.\ equal(add(X::'a,Y),add(Y::'a,X)))$ &
$(\forall\ X\ Y\ Z.\ equal(add(X::'a,add(Y::'a,Z)),add(add(X::'a,Y),Z)))$ &
$(\forall\ Z\ X\ Y.\ equal(add(X::'a,Z),add(Y::'a,Z)) \longrightarrow equal(X::'a,Y))$ &
$(\forall\ Z\ X\ Y.\ equal(add(Z::'a,X),add(Z::'a,Y)) \longrightarrow equal(X::'a,Y))$ &
$(\forall\ D\ E\ F'.\ equal(D::'a,E) \longrightarrow equal(add(D::'a,F'),add(E::'a,F')))$ &
$(\forall\ G\ I'\ H.\ equal(G::'a,H) \longrightarrow equal(add(I'::'a,G),add(I'::'a,H)))$ &
$(\forall\ J\ K'.\ equal(J::'a,K') \longrightarrow equal(additive\text{-}inverse(J),additive\text{-}inverse(K')))$ &
$(\forall\ D1\ E1\ F1.\ equal(D1::'a,E1) \longrightarrow equal(multiply(D1::'a,F1),multiply(E1::'a,F1)))$
&
$(\forall\ G1\ I1\ H1.\ equal(G1::'a,H1) \longrightarrow equal(multiply(I1::'a,G1),multiply(I1::'a,H1)))$
&
$(\forall\ X\ Y\ Z.\ equal(associator(X::'a,Y,Z),add(multiply(multiply(X::'a,Y),Z),additive\text{-}inverse(multiply(X::'a,m$
&
$(\forall\ L\ M\ N\ O'.\ equal(L::'a,M) \longrightarrow equal(associator(L::'a,N,O'),associator(M::'a,N,O')))$
&
$(\forall\ P\ R\ Q\ S'.\ equal(P::'a,Q) \longrightarrow equal(associator(R::'a,P,S'),associator(R::'a,Q,S')))$
&
$(\forall\ T'\ V\ W\ U.\ equal(T'::'a,U) \longrightarrow equal(associator(V::'a,W,T'),associator(V::'a,W,U)))$
&
$(\forall\ X\ Y.\ \sim equal(multiply(multiply(Y::'a,X),Y),multiply(Y::'a,multiply(X::'a,Y))))$
&
$(\forall\ X\ Y\ Z.\ \sim equal(associator(Y::'a,X,Z),additive\text{-}inverse(associator(X::'a,Y,Z))))$
&
$(\forall\ X\ Y\ Z.\ \sim equal(associator(Z::'a,Y,X),additive\text{-}inverse(associator(X::'a,Y,Z))))$
&
$(\sim equal(multiply(multiply(cx::'a,multiply(cy::'a,cx)),cz),multiply(cx::'a,multiply(cy::'a,multiply(cx::'a,cz))))$
$\longrightarrow False$
$\langle proof \rangle$

lemma *RNG038-2*:
$(\forall\ X.\ sum(X::'a,additive\text{-}identity,X))$ &
$(\forall\ X\ Y.\ product(X::'a,Y,multiply(X::'a,Y)))$ &
$(\forall\ X\ Y.\ sum(X::'a,Y,add(X::'a,Y)))$ &
RNG-other-ax multiply add equal product additive-identity additive-inverse sum
&

$(\forall X.\ product(additive\text{-}identity::'a,X,additive\text{-}identity))$ &
$(\forall X.\ product(X::'a,additive\text{-}identity,additive\text{-}identity))$ &
$(\forall X\ Y.\ equal(X::'a,additive\text{-}identity) \longrightarrow product(X::'a,h(X::'a),Y))$ &
$(product(a::'a,b,additive\text{-}identity))$ &
$(\sim equal(a::'a,additive\text{-}identity))$ &
$(\sim equal(b::'a,additive\text{-}identity)) \longrightarrow False$
$\langle proof \rangle$

lemma *RNG040-2*:
EQU001-0-ax equal &
RNG001-0-eq product multiply sum add additive-inverse equal &
$(\forall X.\ sum(additive\text{-}identity::'a,X,X))$ &
$(\forall X.\ sum(X::'a,additive\text{-}identity,X))$ &
$(\forall X\ Y.\ product(X::'a,Y,multiply(X::'a,Y)))$ &
$(\forall X\ Y.\ sum(X::'a,Y,add(X::'a,Y)))$ &
$(\forall X.\ sum(additive\text{-}inverse(X),X,additive\text{-}identity))$ &
$(\forall X.\ sum(X::'a,additive\text{-}inverse(X),additive\text{-}identity))$ &
$(\forall Y\ U\ Z\ X\ V\ W.\ sum(X::'a,Y,U)\ \&\ sum(Y::'a,Z,V)\ \&\ sum(U::'a,Z,W) \longrightarrow sum(X::'a,V,W))$ &
$(\forall Y\ X\ V\ U\ Z\ W.\ sum(X::'a,Y,U)\ \&\ sum(Y::'a,Z,V)\ \&\ sum(X::'a,V,W) \longrightarrow sum(U::'a,Z,W))$ &
$(\forall Y\ X\ Z.\ sum(X::'a,Y,Z) \longrightarrow sum(Y::'a,X,Z))$ &
$(\forall Y\ U\ Z\ X\ V\ W.\ product(X::'a,Y,U)\ \&\ product(Y::'a,Z,V)\ \&\ product(U::'a,Z,W) \longrightarrow product(X::'a,V,W))$ &
$(\forall Y\ X\ V\ U\ Z\ W.\ product(X::'a,Y,U)\ \&\ product(Y::'a,Z,V)\ \&\ product(X::'a,V,W) \longrightarrow product(U::'a,Z,W))$ &
$(\forall Y\ Z\ X\ V3\ V1\ V2\ V4.\ product(X::'a,Y,V1)\ \&\ product(X::'a,Z,V2)\ \&\ sum(Y::'a,Z,V3)\ \&\ product(X::'a,V3,V4) \longrightarrow sum(V1::'a,V2,V4))$ &
$(\forall Y\ Z\ V1\ V2\ X\ V3\ V4.\ product(X::'a,Y,V1)\ \&\ product(X::'a,Z,V2)\ \&\ sum(Y::'a,Z,V3)\ \&\ sum(V1::'a,V2,V4) \longrightarrow product(X::'a,V3,V4))$ &
$(\forall X\ Y\ U\ V.\ sum(X::'a,Y,U)\ \&\ sum(X::'a,Y,V) \longrightarrow equal(U::'a,V))$ &
$(\forall X\ Y\ U\ V.\ product(X::'a,Y,U)\ \&\ product(X::'a,Y,V) \longrightarrow equal(U::'a,V))$ &
$(\forall A.\ product(A::'a,multiplicative\text{-}identity,A))$ &
$(\forall A.\ product(multiplicative\text{-}identity::'a,A,A))$ &
$(\forall A.\ product(A::'a,h(A),multiplicative\text{-}identity) \mid equal(A::'a,additive\text{-}identity))$ &
$(\forall A.\ product(h(A),A,multiplicative\text{-}identity) \mid equal(A::'a,additive\text{-}identity))$ &
$(\forall B\ A\ C.\ product(A::'a,B,C) \longrightarrow product(B::'a,A,C))$ &
$(\forall A\ B.\ equal(A::'a,B) \longrightarrow equal(h(A),h(B)))$ &
$(sum(b::'a,c,d))$ &
$(product(d::'a,a,additive\text{-}identity))$ &
$(product(b::'a,a,l))$ &
$(product(c::'a,a,n))$ &
$(\sim sum(l::'a,n,additive\text{-}identity)) \longrightarrow False$
$\langle proof \rangle$

lemma *RNG041-1*:
EQU001-0-ax equal &
RNG001-0-ax equal additive-inverse add multiply product additive-identity sum &
RNG001-0-eq product multiply sum add additive-inverse equal &
$(\forall\ A\ B.\ equal(A::'a,B)\ -->\ equal(h(A),h(B)))$ &
$(\forall\ A.\ product(additive\text{-}identity::'a,A,additive\text{-}identity))$ &
$(\forall\ A.\ product(A::'a,additive\text{-}identity,additive\text{-}identity))$ &
$(\forall\ A.\ product(A::'a,multiplicative\text{-}identity,A))$ &
$(\forall\ A.\ product(multiplicative\text{-}identity::'a,A,A))$ &
$(\forall\ A.\ product(A::'a,h(A),multiplicative\text{-}identity)\ |\ equal(A::'a,additive\text{-}identity))$
&
$(\forall\ A.\ product(h(A),A,multiplicative\text{-}identity)\ |\ equal(A::'a,additive\text{-}identity))$ &
$(product(a::'a,b,additive\text{-}identity))$ &
$(\sim equal(a::'a,additive\text{-}identity))$ &
$(\sim equal(b::'a,additive\text{-}identity))\ -->\ False$
$\langle proof \rangle$

lemma *ROB010-1*:
EQU001-0-ax equal &
$(\forall\ Y\ X.\ equal(add(X::'a,Y),add(Y::'a,X)))$ &
$(\forall\ X\ Y\ Z.\ equal(add(add(X::'a,Y),Z),add(X::'a,add(Y::'a,Z))))$ &
$(\forall\ Y\ X.\ equal(negate(add(negate(add(X::'a,Y)),negate(add(X::'a,negate(Y))))),X))$
&
$(\forall\ A\ B\ C.\ equal(A::'a,B)\ -->\ equal(add(A::'a,C),add(B::'a,C)))$ &
$(\forall\ D\ F'\ E.\ equal(D::'a,E)\ -->\ equal(add(F'::'a,D),add(F'::'a,E)))$ &
$(\forall\ G\ H.\ equal(G::'a,H)\ -->\ equal(negate(G),negate(H)))$ &
$(equal(negate(add(a::'a,negate(b))),c))$ &
$(\sim equal(negate(add(c::'a,negate(add(b::'a,a))))),a))\ -->\ False$
$\langle proof \rangle$

lemma *ROB013-1*:
EQU001-0-ax equal &
$(\forall\ Y\ X.\ equal(add(X::'a,Y),add(Y::'a,X)))$ &
$(\forall\ X\ Y\ Z.\ equal(add(add(X::'a,Y),Z),add(X::'a,add(Y::'a,Z))))$ &
$(\forall\ Y\ X.\ equal(negate(add(negate(add(X::'a,Y)),negate(add(X::'a,negate(Y))))),X))$
&
$(\forall\ A\ B\ C.\ equal(A::'a,B)\ -->\ equal(add(A::'a,C),add(B::'a,C)))$ &
$(\forall\ D\ F'\ E.\ equal(D::'a,E)\ -->\ equal(add(F'::'a,D),add(F'::'a,E)))$ &
$(\forall\ G\ H.\ equal(G::'a,H)\ -->\ equal(negate(G),negate(H)))$ &
$(equal(negate(add(a::'a,b)),c))$ &
$(\sim equal(negate(add(c::'a,negate(add(negate(b),a))))),a))\ -->\ False$
$\langle proof \rangle$

lemma *ROB016-1*:
EQU001-0-ax equal &

$(\forall\ Y\ X.\ equal(add(X::'a,Y),add(Y::'a,X)))\ \&$
$(\forall\ X\ Y\ Z.\ equal(add(add(X::'a,Y),Z),add(X::'a,add(Y::'a,Z))))\ \&$
$(\forall\ Y\ X.\ equal(negate(add(negate(add(X::'a,Y)),negate(add(X::'a,negate(Y))))),X))$
$\&$
$(\forall\ A\ B\ C.\ equal(A::'a,B) \longrightarrow equal(add(A::'a,C),add(B::'a,C)))\ \&$
$(\forall\ D\ F'\ E.\ equal(D::'a,E) \longrightarrow equal(add(F'::'a,D),add(F'::'a,E)))\ \&$
$(\forall\ G\ H.\ equal(G::'a,H) \longrightarrow equal(negate(G),negate(H)))\ \&$
$(\forall\ J\ K'\ L.\ equal(J::'a,K') \longrightarrow equal(multiply(J::'a,L),multiply(K'::'a,L)))\ \&$
$(\forall\ M\ O'\ N.\ equal(M::'a,N) \longrightarrow equal(multiply(O'::'a,M),multiply(O'::'a,N)))$
$\&$
$(\forall\ P\ Q.\ equal(P::'a,Q) \longrightarrow equal(successor(P),successor(Q)))\ \&$
$(\forall\ R\ S'.\ equal(R::'a,S')\ \&\ positive\text{-}integer(R) \longrightarrow positive\text{-}integer(S'))\ \&$
$(\forall\ X.\ equal(multiply(One::'a,X),X))\ \&$
$(\forall\ V\ X.\ positive\text{-}integer(X) \longrightarrow equal(multiply(successor(V),X),add(X::'a,multiply(V::'a,X))))$
$\&$
$(positive\text{-}integer(One))\ \&$
$(\forall\ X.\ positive\text{-}integer(X) \longrightarrow positive\text{-}integer(successor(X)))\ \&$
$(equal(negate(add(d::'a,e)),negate(e)))\ \&$
$(positive\text{-}integer(k))\ \&$
$(\forall\ Vk\ X\ Y.\ equal(negate(add(negate(Y),negate(add(X::'a,negate(Y))))),X)\ \&$
$positive\text{-}integer(Vk) \longrightarrow equal(negate(add(Y::'a,multiply(Vk::'a,add(X::'a,negate(add(X::'a,negate(Y))))))$
$\&$
$(\sim equal(negate(add(e::'a,multiply(k::'a,add(d::'a,negate(add(d::'a,negate(e))))))),negate(e)))$
$\longrightarrow False$
$\langle proof \rangle$

lemma *ROB021-1*:
$EQU001\text{-}0\text{-}ax\ equal\ \&$
$(\forall\ Y\ X.\ equal(add(X::'a,Y),add(Y::'a,X)))\ \&$
$(\forall\ X\ Y\ Z.\ equal(add(add(X::'a,Y),Z),add(X::'a,add(Y::'a,Z))))\ \&$
$(\forall\ Y\ X.\ equal(negate(add(negate(add(X::'a,Y)),negate(add(X::'a,negate(Y))))),X))$
$\&$
$(\forall\ A\ B\ C.\ equal(A::'a,B) \longrightarrow equal(add(A::'a,C),add(B::'a,C)))\ \&$
$(\forall\ D\ F'\ E.\ equal(D::'a,E) \longrightarrow equal(add(F'::'a,D),add(F'::'a,E)))\ \&$
$(\forall\ G\ H.\ equal(G::'a,H) \longrightarrow equal(negate(G),negate(H)))\ \&$
$(\forall\ X\ Y.\ equal(negate(X),negate(Y)) \longrightarrow equal(X::'a,Y))\ \&$
$(\sim equal(add(negate(add(a::'a,negate(b))),negate(add(negate(a),negate(b)))),b))$
$\longrightarrow False$
$\langle proof \rangle$

lemma *SET005-1*:
$(\forall\ Subset\ Element\ Superset.\ member(Element::'a,Subset)\ \&\ subset(Subset::'a,Superset)$
$\longrightarrow member(Element::'a,Superset))\ \&$
$(\forall\ Superset\ Subset.\ subset(Subset::'a,Superset) \mid member(member\text{-}of\text{-}1\text{-}not\text{-}of\text{-}2(Subset::'a,Superset),Subset))$
$\&$
$(\forall\ Subset\ Superset.\ member(member\text{-}of\text{-}1\text{-}not\text{-}of\text{-}2(Subset::'a,Superset),Superset)$
$\longrightarrow subset(Subset::'a,Superset))\ \&$

(∀ *Subset Superset*. *equal-sets*(*Subset*::'*a*,*Superset*) −−> *subset*(*Subset*::'*a*,*Superset*))
&
(∀ *Subset Superset*. *equal-sets*(*Superset*::'*a*,*Subset*) −−> *subset*(*Subset*::'*a*,*Superset*))
&
(∀ *Set2 Set1*. *subset*(*Set1*::'*a*,*Set2*) & *subset*(*Set2*::'*a*,*Set1*) −−> *equal-sets*(*Set2*::'*a*,*Set1*))
&
(∀ *Set2 Intersection Element Set1*. *intersection*(*Set1*::'*a*,*Set2*,*Intersection*) & *member*(*Element*::'*a*,*Intersection*) −−> *member*(*Element*::'*a*,*Set1*)) &
(∀ *Set1 Intersection Element Set2*. *intersection*(*Set1*::'*a*,*Set2*,*Intersection*) & *member*(*Element*::'*a*,*Intersection*) −−> *member*(*Element*::'*a*,*Set2*)) &
(∀ *Set2 Set1 Element Intersection*. *intersection*(*Set1*::'*a*,*Set2*,*Intersection*) & *member*(*Element*::'*a*,*Set2*) & *member*(*Element*::'*a*,*Set1*) −−> *member*(*Element*::'*a*,*Intersection*))
&
(∀ *Set2 Intersection Set1*. *member*(*h*(*Set1*::'*a*,*Set2*,*Intersection*),*Intersection*) | *intersection*(*Set1*::'*a*,*Set2*,*Intersection*) | *member*(*h*(*Set1*::'*a*,*Set2*,*Intersection*),*Set1*))
&
(∀ *Set1 Intersection Set2*. *member*(*h*(*Set1*::'*a*,*Set2*,*Intersection*),*Intersection*) | *intersection*(*Set1*::'*a*,*Set2*,*Intersection*) | *member*(*h*(*Set1*::'*a*,*Set2*,*Intersection*),*Set2*))
&
(∀ *Set1 Set2 Intersection*. *member*(*h*(*Set1*::'*a*,*Set2*,*Intersection*),*Intersection*) & *member*(*h*(*Set1*::'*a*,*Set2*,*Intersection*),*Set2*) & *member*(*h*(*Set1*::'*a*,*Set2*,*Intersection*),*Set1*) −−> *intersection*(*Set1*::'*a*,*Set2*,*Intersection*)) &
(*intersection*(*a*::'*a*,*b*,*aIb*)) &
(*intersection*(*b*::'*a*,*c*,*bIc*)) &
(*intersection*(*a*::'*a*,*bIc*,*aIbIc*)) &
(~*intersection*(*aIb*::'*a*,*c*,*aIbIc*)) −−> *False*
⟨*proof*⟩

lemma *SET009-1*:
(∀ *Subset Element Superset*. *member*(*Element*::'*a*,*Subset*) & *ssubset*(*Subset*::'*a*,*Superset*) −−> *member*(*Element*::'*a*,*Superset*)) &
(∀ *Superset Subset*. *ssubset*(*Subset*::'*a*,*Superset*) | *member*(*member-of-1-not-of-2*(*Subset*::'*a*,*Superset*),*Subset*))
&
(∀ *Subset Superset*. *member*(*member-of-1-not-of-2*(*Subset*::'*a*,*Superset*),*Superset*) −−> *ssubset*(*Subset*::'*a*,*Superset*)) &
(∀ *Subset Superset*. *equal-sets*(*Subset*::'*a*,*Superset*) −−> *ssubset*(*Subset*::'*a*,*Superset*))
&
(∀ *Subset Superset*. *equal-sets*(*Superset*::'*a*,*Subset*) −−> *ssubset*(*Subset*::'*a*,*Superset*))
&
(∀ *Set2 Set1*. *ssubset*(*Set1*::'*a*,*Set2*) & *ssubset*(*Set2*::'*a*,*Set1*) −−> *equal-sets*(*Set2*::'*a*,*Set1*))
&
(∀ *Set2 Difference Element Set1*. *difference*(*Set1*::'*a*,*Set2*,*Difference*) & *member*(*Element*::'*a*,*Difference*) −−> *member*(*Element*::'*a*,*Set1*)) &
(∀ *Element A-set Set1 Set2*. ~(*member*(*Element*::'*a*,*Set1*) & *member*(*Element*::'*a*,*Set2*) & *difference*(*A-set*::'*a*,*Set1*,*Set2*))) &
(∀ *Set1 Difference Element Set2*. *member*(*Element*::'*a*,*Set1*) & *difference*(*Set1*::'*a*,*Set2*,*Difference*) −−> *member*(*Element*::'*a*,*Difference*) | *member*(*Element*::'*a*,*Set2*)) &

$(\forall\ Set1\ Set2\ Difference.\ difference(Set1::'a,Set2,Difference) \mid member(k(Set1::'a,Set2,Difference),Set1)$
$\mid member(k(Set1::'a,Set2,Difference),Difference))\ \&$
$(\forall\ Set1\ Set2\ Difference.\ member(k(Set1::'a,Set2,Difference),Set2)\ --{>}\ member(k(Set1::'a,Set2,Difference),Difference) \mid difference(Set1::'a,Set2,Difference))\ \&$
$(\forall\ Set1\ Set2\ Difference.\ member(k(Set1::'a,Set2,Difference),Difference)\ \&\ member(k(Set1::'a,Set2,Difference),Set1)\ --{>}\ member(k(Set1::'a,Set2,Difference),Set2)$
$\mid difference(Set1::'a,Set2,Difference))\ \&$
$(ssubset(d::'a,a))\ \&$
$(difference(b::'a,a,bDa))\ \&$
$(difference(b::'a,d,bDd))\ \&$
$(\sim ssubset(bDa::'a,bDd))\ --{>}\ False$
$\langle proof \rangle$

lemma *SET025-4*:
$EQU001\text{-}0\text{-}ax\ equal\ \&$
$(\forall\ Y\ X.\ member(X::'a,Y)\ --{>}\ little\text{-}set(X))\ \&$
$(\forall\ X\ Y.\ little\text{-}set(f1(X::'a,Y)) \mid equal(X::'a,Y))\ \&$
$(\forall\ X\ Y.\ member(f1(X::'a,Y),X) \mid member(f1(X::'a,Y),Y) \mid equal(X::'a,Y))\ \&$
$(\forall\ X\ Y.\ member(f1(X::'a,Y),X)\ \&\ member(f1(X::'a,Y),Y)\ --{>}\ equal(X::'a,Y))$
$\&$
$(\forall\ X\ U\ Y.\ member(U::'a,non\text{-}ordered\text{-}pair(X::'a,Y))\ --{>}\ equal(U::'a,X) \mid equal(U::'a,Y))$
$\&$
$(\forall\ Y\ U\ X.\ little\text{-}set(U)\ \&\ equal(U::'a,X)\ --{>}\ member(U::'a,non\text{-}ordered\text{-}pair(X::'a,Y)))$
$\&$
$(\forall\ X\ U\ Y.\ little\text{-}set(U)\ \&\ equal(U::'a,Y)\ --{>}\ member(U::'a,non\text{-}ordered\text{-}pair(X::'a,Y)))$
$\&$
$(\forall\ X\ Y.\ little\text{-}set(non\text{-}ordered\text{-}pair(X::'a,Y)))\ \&$
$(\forall\ X.\ equal(singleton\text{-}set(X),non\text{-}ordered\text{-}pair(X::'a,X)))\ \&$
$(\forall\ X\ Y.\ equal(ordered\text{-}pair(X::'a,Y),non\text{-}ordered\text{-}pair(singleton\text{-}set(X),non\text{-}ordered\text{-}pair(X::'a,Y))))$
$\&$
$(\forall\ X.\ ordered\text{-}pair\text{-}predicate(X)\ --{>}\ little\text{-}set(f2(X)))\ \&$
$(\forall\ X.\ ordered\text{-}pair\text{-}predicate(X)\ --{>}\ little\text{-}set(f3(X)))\ \&$
$(\forall\ X.\ ordered\text{-}pair\text{-}predicate(X)\ --{>}\ equal(X::'a,ordered\text{-}pair(f2(X),f3(X))))\ \&$
$(\forall\ X\ Y\ Z.\ little\text{-}set(Y)\ \&\ little\text{-}set(Z)\ \&\ equal(X::'a,ordered\text{-}pair(Y::'a,Z))\ --{>}$
$ordered\text{-}pair\text{-}predicate(X))\ \&$
$(\forall\ Z\ X.\ member(Z::'a,first(X))\ --{>}\ little\text{-}set(f4(Z::'a,X)))\ \&$
$(\forall\ Z\ X.\ member(Z::'a,first(X))\ --{>}\ little\text{-}set(f5(Z::'a,X)))\ \&$
$(\forall\ Z\ X.\ member(Z::'a,first(X))\ --{>}\ equal(X::'a,ordered\text{-}pair(f4(Z::'a,X),f5(Z::'a,X))))$
$\&$
$(\forall\ Z\ X.\ member(Z::'a,first(X))\ --{>}\ member(Z::'a,f4(Z::'a,X)))\ \&$
$(\forall\ X\ V\ Z\ U.\ little\text{-}set(U)\ \&\ little\text{-}set(V)\ \&\ equal(X::'a,ordered\text{-}pair(U::'a,V))$
$\&\ member(Z::'a,U)\ --{>}\ member(Z::'a,first(X)))\ \&$
$(\forall\ Z\ X.\ member(Z::'a,second(X))\ --{>}\ little\text{-}set(f6(Z::'a,X)))\ \&$
$(\forall\ Z\ X.\ member(Z::'a,second(X))\ --{>}\ little\text{-}set(f7(Z::'a,X)))\ \&$
$(\forall\ Z\ X.\ member(Z::'a,second(X))\ --{>}\ equal(X::'a,ordered\text{-}pair(f6(Z::'a,X),f7(Z::'a,X))))$
$\&$
$(\forall\ Z\ X.\ member(Z::'a,second(X))\ --{>}\ member(Z::'a,f7(Z::'a,X)))\ \&$
$(\forall\ X\ U\ Z\ V.\ little\text{-}set(U)\ \&\ little\text{-}set(V)\ \&\ equal(X::'a,ordered\text{-}pair(U::'a,V))$

$$
\begin{array}{l}
\& \ member(Z::'a,V) \ --> \ member(Z::'a,second(X))) \ \& \\
\ (\forall Z.\ member(Z::'a,estin) \ --> \ ordered\text{-}pair\text{-}predicate(Z)) \ \& \\
\ (\forall Z.\ member(Z::'a,estin) \ --> \ member(first(Z),second(Z))) \ \& \\
\ \ (\forall Z.\ little\text{-}set(Z) \ \& \ ordered\text{-}pair\text{-}predicate(Z) \ \& \ member(first(Z),second(Z)) \\
--> \ member(Z::'a,estin)) \ \& \\
\ (\forall Y\ Z\ X.\ member(Z::'a,intersection(X::'a,Y)) \ --> \ member(Z::'a,X)) \ \& \\
\ (\forall X\ Z\ Y.\ member(Z::'a,intersection(X::'a,Y)) \ --> \ member(Z::'a,Y)) \ \& \\
(\forall X\ Z\ Y.\ member(Z::'a,X) \ \& \ member(Z::'a,Y) \ --> \ member(Z::'a,intersection(X::'a,Y))) \\
\& \\
\ (\forall Z\ X.\ \sim(member(Z::'a,complement(X)) \ \& \ member(Z::'a,X))) \ \& \\
\ (\forall Z\ X.\ little\text{-}set(Z) \ --> \ member(Z::'a,complement(X)) \mid member(Z::'a,X)) \ \& \\
(\forall X\ Y.\ equal(union(X::'a,Y),complement(intersection(complement(X),complement(Y))))) \\
\& \\
\ (\forall Z\ X.\ member(Z::'a,domain\text{-}of(X)) \ --> \ ordered\text{-}pair\text{-}predicate(f8(Z::'a,X))) \\
\& \\
\ (\forall Z\ X.\ member(Z::'a,domain\text{-}of(X)) \ --> \ member(f8(Z::'a,X),X)) \ \& \\
\ (\forall Z\ X.\ member(Z::'a,domain\text{-}of(X)) \ --> \ equal(Z::'a,first(f8(Z::'a,X)))) \ \& \\
\ \ (\forall X\ Z\ Xp.\ little\text{-}set(Z) \ \& \ ordered\text{-}pair\text{-}predicate(Xp) \ \& \ member(Xp::'a,X) \ \& \\
equal(Z::'a,first(Xp)) \ --> \ member(Z::'a,domain\text{-}of(X))) \ \& \\
\ (\forall X\ Y\ Z.\ member(Z::'a,cross\text{-}product(X::'a,Y)) \ --> \ ordered\text{-}pair\text{-}predicate(Z)) \\
\& \\
\ (\forall Y\ Z\ X.\ member(Z::'a,cross\text{-}product(X::'a,Y)) \ --> \ member(first(Z),X)) \ \& \\
\ \ (\forall X\ Z\ Y.\ member(Z::'a,cross\text{-}product(X::'a,Y)) \ --> \ member(second(Z),Y)) \\
\& \\
\ \ (\forall X\ Z\ Y.\ little\text{-}set(Z) \ \& \ ordered\text{-}pair\text{-}predicate(Z) \ \& \ member(first(Z),X) \ \& \\
member(second(Z),Y) \ --> \ member(Z::'a,cross\text{-}product(X::'a,Y))) \ \& \\
\ (\forall X\ Z.\ member(Z::'a,inv1\ X) \ --> \ ordered\text{-}pair\text{-}predicate(Z)) \ \& \\
\ (\forall Z\ X.\ member(Z::'a,inv1\ X) \ --> \ member(ordered\text{-}pair(second(Z),first(Z)),X)) \\
\& \\
\ (\forall Z\ X.\ little\text{-}set(Z) \ \& \ ordered\text{-}pair\text{-}predicate(Z) \ \& \ member(ordered\text{-}pair(second(Z),first(Z)),X) \\
--> \ member(Z::'a,inv1\ X)) \ \& \\
\ (\forall Z\ X.\ member(Z::'a,rot\text{-}right(X)) \ --> \ little\text{-}set(f9(Z::'a,X))) \ \& \\
\ (\forall Z\ X.\ member(Z::'a,rot\text{-}right(X)) \ --> \ little\text{-}set(f10(Z::'a,X))) \ \& \\
\ (\forall Z\ X.\ member(Z::'a,rot\text{-}right(X)) \ --> \ little\text{-}set(f11(Z::'a,X))) \ \& \\
\ (\forall Z\ X.\ member(Z::'a,rot\text{-}right(X)) \ --> \ equal(Z::'a,ordered\text{-}pair(f9(Z::'a,X),ordered\text{-}pair(f10(Z::'a,X),f11 \\
\& \\
\ (\forall Z\ X.\ member(Z::'a,rot\text{-}right(X)) \ --> \ member(ordered\text{-}pair(f10(Z::'a,X),ordered\text{-}pair(f11(Z::'a,X),f9(Z \\
\& \\
\ \ (\forall Z\ V\ W\ U\ X.\ little\text{-}set(Z) \ \& \ little\text{-}set(U) \ \& \ little\text{-}set(V) \ \& \ little\text{-}set(W) \ \& \\
equal(Z::'a,ordered\text{-}pair(U::'a,ordered\text{-}pair(V::'a,W))) \ \& \ member(ordered\text{-}pair(V::'a,ordered\text{-}pair(W::'a,U)) \\
--> \ member(Z::'a,rot\text{-}right(X))) \ \& \\
\ (\forall Z\ X.\ member(Z::'a,flip\text{-}range\text{-}of(X)) \ --> \ little\text{-}set(f12(Z::'a,X))) \ \& \\
\ (\forall Z\ X.\ member(Z::'a,flip\text{-}range\text{-}of(X)) \ --> \ little\text{-}set(f13(Z::'a,X))) \ \& \\
\ (\forall Z\ X.\ member(Z::'a,flip\text{-}range\text{-}of(X)) \ --> \ little\text{-}set(f14(Z::'a,X))) \ \& \\
\ (\forall Z\ X.\ member(Z::'a,flip\text{-}range\text{-}of(X)) \ --> \ equal(Z::'a,ordered\text{-}pair(f12(Z::'a,X),ordered\text{-}pair(f13(Z::'a,X \\
\& \\
\ (\forall Z\ X.\ member(Z::'a,flip\text{-}range\text{-}of(X)) \ --> \ member(ordered\text{-}pair(f12(Z::'a,X),ordered\text{-}pair(f14(Z::'a,X), \\
\& \\
\ \ (\forall Z\ U\ W\ V\ X.\ little\text{-}set(Z) \ \& \ little\text{-}set(U) \ \& \ little\text{-}set(V) \ \& \ little\text{-}set(W) \ \&
\end{array}
$$

$equal(Z::'a,ordered\text{-}pair(U::'a,ordered\text{-}pair(V::'a,W))) \;\&\; member(ordered\text{-}pair(U::'a,ordered\text{-}pair(W::'a,V))$
$--> member(Z::'a,flip\text{-}range\text{-}of(X))) \;\&$
$(\forall X.\ equal(successor(X),union(X::'a,singleton\text{-}set(X)))) \;\&$
$(\forall Z.\ {\sim}member(Z::'a,empty\text{-}set)) \;\&$
$(\forall Z.\ little\text{-}set(Z) --> member(Z::'a,universal\text{-}set)) \;\&$
$(little\text{-}set(infinity)) \;\&$
$(member(empty\text{-}set::'a,infinity)) \;\&$
$(\forall X.\ member(X::'a,infinity) --> member(successor(X),infinity)) \;\&$
$(\forall Z\ X.\ member(Z::'a,sigma(X)) --> member(f16(Z::'a,X),X)) \;\&$
$(\forall Z\ X.\ member(Z::'a,sigma(X)) --> member(Z::'a,f16(Z::'a,X))) \;\&$
$(\forall X\ Z\ Y.\ member(Y::'a,X) \;\&\; member(Z::'a,Y) --> member(Z::'a,sigma(X)))$
$\&$
$(\forall U.\ little\text{-}set(U) --> little\text{-}set(sigma(U))) \;\&$
$(\forall X\ U\ Y.\ ssubset(X::'a,Y) \;\&\; member(U::'a,X) --> member(U::'a,Y)) \;\&$
$(\forall Y\ X.\ ssubset(X::'a,Y) \mid member(f17(X::'a,Y),X)) \;\&$
$(\forall X\ Y.\ member(f17(X::'a,Y),Y) --> ssubset(X::'a,Y)) \;\&$
$(\forall X\ Y.\ proper\text{-}subset(X::'a,Y) --> ssubset(X::'a,Y)) \;\&$
$(\forall X\ Y.\ {\sim}(proper\text{-}subset(X::'a,Y) \;\&\; equal(X::'a,Y))) \;\&$
$(\forall X\ Y.\ ssubset(X::'a,Y) --> proper\text{-}subset(X::'a,Y) \mid equal(X::'a,Y)) \;\&$
$(\forall Z\ X.\ member(Z::'a,powerset(X)) --> ssubset(Z::'a,X)) \;\&$
$(\forall Z\ X.\ little\text{-}set(Z) \;\&\; ssubset(Z::'a,X) --> member(Z::'a,powerset(X))) \;\&$
$(\forall U.\ little\text{-}set(U) --> little\text{-}set(powerset(U))) \;\&$
$(\forall Z\ X.\ relation(Z) \;\&\; member(X::'a,Z) --> ordered\text{-}pair\text{-}predicate(X)) \;\&$
$(\forall Z.\ relation(Z) \mid member(f18(Z),Z)) \;\&$
$(\forall Z.\ ordered\text{-}pair\text{-}predicate(f18(Z)) --> relation(Z)) \;\&$
$(\forall U\ X\ V\ W.\ single\text{-}valued\text{-}set(X) \;\&\; little\text{-}set(U) \;\&\; little\text{-}set(V) \;\&\; little\text{-}set(W)$
$\&\; member(ordered\text{-}pair(U::'a,V),X) \;\&\; member(ordered\text{-}pair(U::'a,W),X) -->$
$equal(V::'a,W)) \;\&$
$(\forall X.\ single\text{-}valued\text{-}set(X) \mid little\text{-}set(f19(X))) \;\&$
$(\forall X.\ single\text{-}valued\text{-}set(X) \mid little\text{-}set(f20(X))) \;\&$
$(\forall X.\ single\text{-}valued\text{-}set(X) \mid little\text{-}set(f21(X))) \;\&$
$(\forall X.\ single\text{-}valued\text{-}set(X) \mid member(ordered\text{-}pair(f19(X),f20(X)),X)) \;\&$
$(\forall X.\ single\text{-}valued\text{-}set(X) \mid member(ordered\text{-}pair(f19(X),f21(X)),X)) \;\&$
$(\forall X.\ equal(f20(X),f21(X)) --> single\text{-}valued\text{-}set(X)) \;\&$
$(\forall Xf.\ function(Xf) --> relation(Xf)) \;\&$
$(\forall Xf.\ function(Xf) --> single\text{-}valued\text{-}set(Xf)) \;\&$
$(\forall Xf.\ relation(Xf) \;\&\; single\text{-}valued\text{-}set(Xf) --> function(Xf)) \;\&$
$(\forall Z\ X\ Xf.\ member(Z::'a,image'(X::'a,Xf)) --> ordered\text{-}pair\text{-}predicate(f22(Z::'a,X,Xf)))$
$\&$
$(\forall Z\ X\ Xf.\ member(Z::'a,image'(X::'a,Xf)) --> member(f22(Z::'a,X,Xf),Xf))$
$\&$
$(\forall Z\ Xf\ X.\ member(Z::'a,image'(X::'a,Xf)) --> member(first(f22(Z::'a,X,Xf)),X))$
$\&$
$(\forall X\ Xf\ Z.\ member(Z::'a,image'(X::'a,Xf)) --> equal(second(f22(Z::'a,X,Xf)),Z))$
$\&$
$(\forall Xf\ X\ Y\ Z.\ little\text{-}set(Z) \;\&\; ordered\text{-}pair\text{-}predicate(Y) \;\&\; member(Y::'a,Xf) \;\&$
$member(first(Y),X) \;\&\; equal(second(Y),Z) --> member(Z::'a,image'(X::'a,Xf)))$
$\&$
$(\forall X\ Xf.\ little\text{-}set(X) \;\&\; function(Xf) --> little\text{-}set(image'(X::'a,Xf))) \;\&$

$(\forall\, X\ U\ Y.\ \sim(disjoint(X::'a,Y)\ \&\ member(U::'a,X)\ \&\ member(U::'a,Y)))\ \&$
$(\forall\, Y\ X.\ disjoint(X::'a,Y)\mid member(f23(X::'a,Y),X))\ \&$
$(\forall\, X\ Y.\ disjoint(X::'a,Y)\mid member(f23(X::'a,Y),Y))\ \&$
$(\forall\, X.\ equal(X::'a,empty\text{-}set)\mid member(f24(X),X))\ \&$
$(\forall\, X.\ equal(X::'a,empty\text{-}set)\mid disjoint(f24(X),X))\ \&$
$(function(f25))\ \&$
$(\forall\, X.\ little\text{-}set(X)\ -->\ equal(X::'a,empty\text{-}set)\mid member(f26(X),X))\ \&$
$(\forall\, X.\ little\text{-}set(X)\ -->\ equal(X::'a,empty\text{-}set)\mid member(ordered\text{-}pair(X::'a,f26(X)),f25))$
$\&$
$(\forall\, Z\ X.\ member(Z::'a,range\text{-}of(X))\ -->\ ordered\text{-}pair\text{-}predicate(f27(Z::'a,X)))$
$\&$
$(\forall\, Z\ X.\ member(Z::'a,range\text{-}of(X))\ -->\ member(f27(Z::'a,X),X))\ \&$
$(\forall\, Z\ X.\ member(Z::'a,range\text{-}of(X))\ -->\ equal(Z::'a,second(f27(Z::'a,X))))\ \&$
$(\forall\, X\ Z\ Xp.\ little\text{-}set(Z)\ \&\ ordered\text{-}pair\text{-}predicate(Xp)\ \&\ member(Xp::'a,X)\ \&$
$equal(Z::'a,second(Xp))\ -->\ member(Z::'a,range\text{-}of(X)))\ \&$
$(\forall\, Z.\ member(Z::'a,identity\text{-}relation)\ -->\ ordered\text{-}pair\text{-}predicate(Z))\ \&$
$(\forall\, Z.\ member(Z::'a,identity\text{-}relation)\ -->\ equal(first(Z),second(Z)))\ \&$
$(\forall\, Z.\ little\text{-}set(Z)\ \&\ ordered\text{-}pair\text{-}predicate(Z)\ \&\ equal(first(Z),second(Z))\ -->$
$member(Z::'a,identity\text{-}relation))\ \&$
$(\forall\, X\ Y.\ equal(restrct(X::'a,Y),intersection(X::'a,cross\text{-}product(Y::'a,universal\text{-}set))))$
$\&$
$(\forall\, Xf.\ one\text{-}to\text{-}one\text{-}function(Xf)\ -->\ function(Xf))\ \&$
$(\forall\, Xf.\ one\text{-}to\text{-}one\text{-}function(Xf)\ -->\ function(inv1\ Xf))\ \&$
$(\forall\, Xf.\ function(Xf)\ \&\ function(inv1\ Xf)\ -->\ one\text{-}to\text{-}one\text{-}function(Xf))\ \&$
$(\forall\, Z\ Xf\ Y.\ member(Z::'a,apply(Xf::'a,Y))\ -->\ ordered\text{-}pair\text{-}predicate(f28(Z::'a,Xf,Y)))$
$\&$
$(\forall\, Z\ Y\ Xf.\ member(Z::'a,apply(Xf::'a,Y))\ -->\ member(f28(Z::'a,Xf,Y),Xf))$
$\&$
$(\forall\, Z\ Xf\ Y.\ member(Z::'a,apply(Xf::'a,Y))\ -->\ equal(first(f28(Z::'a,Xf,Y)),Y))$
$\&$
$(\forall\, Z\ Xf\ Y.\ member(Z::'a,apply(Xf::'a,Y))\ -->\ member(Z::'a,second(f28(Z::'a,Xf,Y))))$
$\&$
$(\forall\, Xf\ Y\ Z\ W.\ ordered\text{-}pair\text{-}predicate(W)\ \&\ member(W::'a,Xf)\ \&\ equal(first(W),Y)$
$\&\ member(Z::'a,second(W))\ -->\ member(Z::'a,apply(Xf::'a,Y)))\ \&$
$(\forall\, Xf\ X\ Y.\ equal(apply\text{-}to\text{-}two\text{-}arguments(Xf::'a,X,Y),apply(Xf::'a,ordered\text{-}pair(X::'a,Y))))$
$\&$
$(\forall\, X\ Y\ Xf.\ maps(Xf::'a,X,Y)\ -->\ function(Xf))\ \&$
$(\forall\, Y\ Xf\ X.\ maps(Xf::'a,X,Y)\ -->\ equal(domain\text{-}of(Xf),X))\ \&$
$(\forall\, X\ Xf\ Y.\ maps(Xf::'a,X,Y)\ -->\ ssubset(range\text{-}of(Xf),Y))\ \&$
$(\forall\, X\ Xf\ Y.\ function(Xf)\ \&\ equal(domain\text{-}of(Xf),X)\ \&\ ssubset(range\text{-}of(Xf),Y)$
$-->\ maps(Xf::'a,X,Y))\ \&$
$(\forall\, Xf\ Xs.\ closed(Xs::'a,Xf)\ -->\ little\text{-}set(Xs))\ \&$
$(\forall\, Xs\ Xf.\ closed(Xs::'a,Xf)\ -->\ little\text{-}set(Xf))\ \&$
$(\forall\, Xf\ Xs.\ closed(Xs::'a,Xf)\ -->\ maps(Xf::'a,cross\text{-}product(Xs::'a,Xs),Xs))\ \&$
$(\forall\, Xf\ Xs.\ little\text{-}set(Xs)\ \&\ little\text{-}set(Xf)\ \&\ maps(Xf::'a,cross\text{-}product(Xs::'a,Xs),Xs)$
$-->\ closed(Xs::'a,Xf))\ \&$
$(\forall\, Z\ Xf\ Xg.\ member(Z::'a,composition(Xf::'a,Xg))\ -->\ little\text{-}set(f29(Z::'a,Xf,Xg)))$
$\&$
$(\forall\, Z\ Xf\ Xg.\ member(Z::'a,composition(Xf::'a,Xg))\ -->\ little\text{-}set(f30(Z::'a,Xf,Xg)))$

$$
\begin{array}{l}
\& \\
\quad (\forall\, Z\ Xf\ Xg.\ member(Z::'a,composition(Xf::'a,Xg)) \longrightarrow little\text{-}set(f31(Z::'a,Xf,Xg))) \\
\& \\
\quad (\forall\, Z\ Xf\ Xg.\ member(Z::'a,composition(Xf::'a,Xg)) \longrightarrow equal(Z::'a,ordered\text{-}pair(f29(Z::'a,Xf,Xg),f30(Z::'a \\
\& \\
\quad (\forall\, Z\ Xg\ Xf.\ member(Z::'a,composition(Xf::'a,Xg)) \longrightarrow member(ordered\text{-}pair(f29(Z::'a,Xf,Xg),f31(Z::'a,X \\
\& \\
\quad (\forall\, Z\ Xf\ Xg.\ member(Z::'a,composition(Xf::'a,Xg)) \longrightarrow member(ordered\text{-}pair(f31(Z::'a,Xf,Xg),f30(Z::'a,X \\
\& \\
\quad (\forall\, Z\ X\ Xf\ W\ Y\ Xg.\ little\text{-}set(Z)\ \&\ little\text{-}set(X)\ \&\ little\text{-}set(Y)\ \&\ little\text{-}set(W)\ \& \\
equal(Z::'a,ordered\text{-}pair(X::'a,Y))\ \&\ member(ordered\text{-}pair(X::'a,W),Xf)\ \&\ mem\text{-} \\
ber(ordered\text{-}pair(W::'a,Y),Xg) \longrightarrow member(Z::'a,composition(Xf::'a,Xg)))\ \& \\
\quad (\forall\, Xh\ Xs2\ Xf2\ Xs1\ Xf1.\ homomorphism(Xh::'a,Xs1,Xf1,Xs2,Xf2) \longrightarrow closed(Xs1::'a,Xf1)) \\
\& \\
\quad (\forall\, Xh\ Xs1\ Xf1\ Xs2\ Xf2.\ homomorphism(Xh::'a,Xs1,Xf1,Xs2,Xf2) \longrightarrow closed(Xs2::'a,Xf2)) \\
\& \\
\quad (\forall\, Xf1\ Xf2\ Xh\ Xs1\ Xs2.\ homomorphism(Xh::'a,Xs1,Xf1,Xs2,Xf2) \longrightarrow maps(Xh::'a,Xs1,Xs2)) \\
\& \\
\quad (\forall\, Xs2\ Xs1\ Xf1\ Xf2\ X\ Xh\ Y.\ homomorphism(Xh::'a,Xs1,Xf1,Xs2,Xf2)\ \&\ mem\text{-} \\
ber(X::'a,Xs1)\ \&\ member(Y::'a,Xs1) \longrightarrow equal(apply(Xh::'a,apply\text{-}to\text{-}two\text{-}arguments(Xf1::'a,X,Y)),apply\text{-}t \\
\& \\
\quad (\forall\, Xh\ Xf1\ Xs2\ Xf2\ Xs1.\ closed(Xs1::'a,Xf1)\ \&\ closed(Xs2::'a,Xf2)\ \&\ maps(Xh::'a,Xs1,Xs2) \\
\longrightarrow homomorphism(Xh::'a,Xs1,Xf1,Xs2,Xf2) \mid member(f32(Xh::'a,Xs1,Xf1,Xs2,Xf2),Xs1)) \\
\& \\
\quad (\forall\, Xh\ Xf1\ Xs2\ Xf2\ Xs1.\ closed(Xs1::'a,Xf1)\ \&\ closed(Xs2::'a,Xf2)\ \&\ maps(Xh::'a,Xs1,Xs2) \\
\longrightarrow homomorphism(Xh::'a,Xs1,Xf1,Xs2,Xf2) \mid member(f33(Xh::'a,Xs1,Xf1,Xs2,Xf2),Xs1)) \\
\& \\
\quad (\forall\, Xh\ Xs1\ Xf1\ Xs2\ Xf2.\ closed(Xs1::'a,Xf1)\ \&\ closed(Xs2::'a,Xf2)\ \&\ maps(Xh::'a,Xs1,Xs2) \\
\&\ equal(apply(Xh::'a,apply\text{-}to\text{-}two\text{-}arguments(Xf1::'a,f32(Xh::'a,Xs1,Xf1,Xs2,Xf2),f33(Xh::'a,Xs1,Xf1,Xs2, \\
\longrightarrow homomorphism(Xh::'a,Xs1,Xf1,Xs2,Xf2))\ \& \\
\quad (\forall\, A\ B\ C.\ equal(A::'a,B) \longrightarrow equal(f1(A::'a,C),f1(B::'a,C)))\ \& \\
\quad (\forall\, D\ F'\ E.\ equal(D::'a,E) \longrightarrow equal(f1(F'::'a,D),f1(F'::'a,E)))\ \& \\
\quad (\forall\, A2\ B2.\ equal(A2::'a,B2) \longrightarrow equal(f2(A2),f2(B2)))\ \& \\
\quad (\forall\, G4\ H4.\ equal(G4::'a,H4) \longrightarrow equal(f3(G4),f3(H4)))\ \& \\
\quad (\forall\, O7\ P7\ Q7.\ equal(O7::'a,P7) \longrightarrow equal(f4(O7::'a,Q7),f4(P7::'a,Q7)))\ \& \\
\quad (\forall\, R7\ T7\ S7.\ equal(R7::'a,S7) \longrightarrow equal(f4(T7::'a,R7),f4(T7::'a,S7)))\ \& \\
\quad (\forall\, U7\ V7\ W7.\ equal(U7::'a,V7) \longrightarrow equal(f5(U7::'a,W7),f5(V7::'a,W7)))\ \& \\
\quad (\forall\, X7\ Z7\ Y7.\ equal(X7::'a,Y7) \longrightarrow equal(f5(Z7::'a,X7),f5(Z7::'a,Y7)))\ \& \\
\quad (\forall\, A8\ B8\ C8.\ equal(A8::'a,B8) \longrightarrow equal(f6(A8::'a,C8),f6(B8::'a,C8)))\ \& \\
\quad (\forall\, D8\ F8\ E8.\ equal(D8::'a,E8) \longrightarrow equal(f6(F8::'a,D8),f6(F8::'a,E8)))\ \& \\
\quad (\forall\, G8\ H8\ I8.\ equal(G8::'a,H8) \longrightarrow equal(f7(G8::'a,I8),f7(H8::'a,I8)))\ \& \\
\quad (\forall\, J8\ L8\ K8.\ equal(J8::'a,K8) \longrightarrow equal(f7(L8::'a,J8),f7(L8::'a,K8)))\ \& \\
\quad (\forall\, M8\ N8\ O8.\ equal(M8::'a,N8) \longrightarrow equal(f8(M8::'a,O8),f8(N8::'a,O8)))\ \& \\
\quad (\forall\, P8\ R8\ Q8.\ equal(P8::'a,Q8) \longrightarrow equal(f8(R8::'a,P8),f8(R8::'a,Q8)))\ \& \\
\quad (\forall\, S8\ T8\ U8.\ equal(S8::'a,T8) \longrightarrow equal(f9(S8::'a,U8),f9(T8::'a,U8)))\ \& \\
\quad (\forall\, V8\ X8\ W8.\ equal(V8::'a,W8) \longrightarrow equal(f9(X8::'a,V8),f9(X8::'a,W8)))\ \& \\
\quad (\forall\, G\ H\ I'.\ equal(G::'a,H) \longrightarrow equal(f10(G::'a,I'),f10(H::'a,I')))\ \& \\
\quad (\forall\, J\ L\ K'.\ equal(J::'a,K') \longrightarrow equal(f10(L::'a,J),f10(L::'a,K')))\ \& \\
\quad (\forall\, M\ N\ O'.\ equal(M::'a,N) \longrightarrow equal(f11(M::'a,O'),f11(N::'a,O')))\ \&
\end{array}
$$

$$
\begin{array}{l}
(\forall\, P\ R\ Q.\ equal(P::'a,Q) \longrightarrow equal(f11(R::'a,P),f11(R::'a,Q))) \ \&\\
(\forall\, S'\ T'\ U.\ equal(S'::'a,T') \longrightarrow equal(f12(S'::'a,U),f12(T'::'a,U))) \ \&\\
(\forall\, V\ X\ W.\ equal(V::'a,W) \longrightarrow equal(f12(X::'a,V),f12(X::'a,W))) \ \&\\
(\forall\, Y\ Z\ A1.\ equal(Y::'a,Z) \longrightarrow equal(f13(Y::'a,A1),f13(Z::'a,A1))) \ \&\\
(\forall\, B1\ D1\ C1.\ equal(B1::'a,C1) \longrightarrow equal(f13(D1::'a,B1),f13(D1::'a,C1))) \ \&\\
(\forall\, E1\ F1\ G1.\ equal(E1::'a,F1) \longrightarrow equal(f14(E1::'a,G1),f14(F1::'a,G1))) \ \&\\
(\forall\, H1\ J1\ I1.\ equal(H1::'a,I1) \longrightarrow equal(f14(J1::'a,H1),f14(J1::'a,I1))) \ \&\\
(\forall\, K1\ L1\ M1.\ equal(K1::'a,L1) \longrightarrow equal(f16(K1::'a,M1),f16(L1::'a,M1))) \ \&\\
(\forall\, N1\ P1\ O1.\ equal(N1::'a,O1) \longrightarrow equal(f16(P1::'a,N1),f16(P1::'a,O1))) \ \&\\
(\forall\, Q1\ R1\ S1.\ equal(Q1::'a,R1) \longrightarrow equal(f17(Q1::'a,S1),f17(R1::'a,S1))) \ \&\\
(\forall\, T1\ V1\ U1.\ equal(T1::'a,U1) \longrightarrow equal(f17(V1::'a,T1),f17(V1::'a,U1))) \ \&\\
(\forall\, W1\ X1.\ equal(W1::'a,X1) \longrightarrow equal(f18(W1),f18(X1))) \ \&\\
(\forall\, Y1\ Z1.\ equal(Y1::'a,Z1) \longrightarrow equal(f19(Y1),f19(Z1))) \ \&\\
(\forall\, C2\ D2.\ equal(C2::'a,D2) \longrightarrow equal(f20(C2),f20(D2))) \ \&\\
(\forall\, E2\ F2.\ equal(E2::'a,F2) \longrightarrow equal(f21(E2),f21(F2))) \ \&\\
(\forall\, G2\ H2\ I2\ J2.\ equal(G2::'a,H2) \longrightarrow equal(f22(G2::'a,I2,J2),f22(H2::'a,I2,J2))) \\
\&\\
(\forall\, K2\ M2\ L2\ N2.\ equal(K2::'a,L2) \longrightarrow equal(f22(M2::'a,K2,N2),f22(M2::'a,L2,N2))) \\
\&\\
(\forall\, O2\ Q2\ R2\ P2.\ equal(O2::'a,P2) \longrightarrow equal(f22(Q2::'a,R2,O2),f22(Q2::'a,R2,P2))) \\
\&\\
(\forall\, S2\ T2\ U2.\ equal(S2::'a,T2) \longrightarrow equal(f23(S2::'a,U2),f23(T2::'a,U2))) \ \&\\
(\forall\, V2\ X2\ W2.\ equal(V2::'a,W2) \longrightarrow equal(f23(X2::'a,V2),f23(X2::'a,W2))) \\
\&\\
(\forall\, Y2\ Z2.\ equal(Y2::'a,Z2) \longrightarrow equal(f24(Y2),f24(Z2))) \ \&\\
(\forall\, A3\ B3.\ equal(A3::'a,B3) \longrightarrow equal(f26(A3),f26(B3))) \ \&\\
(\forall\, C3\ D3\ E3.\ equal(C3::'a,D3) \longrightarrow equal(f27(C3::'a,E3),f27(D3::'a,E3))) \ \&\\
(\forall\, F3\ H3\ G3.\ equal(F3::'a,G3) \longrightarrow equal(f27(H3::'a,F3),f27(H3::'a,G3))) \ \&\\
(\forall\, I3\ J3\ K3\ L3.\ equal(I3::'a,J3) \longrightarrow equal(f28(I3::'a,K3,L3),f28(J3::'a,K3,L3))) \\
\&\\
(\forall\, M3\ O3\ N3\ P3.\ equal(M3::'a,N3) \longrightarrow equal(f28(O3::'a,M3,P3),f28(O3::'a,N3,P3))) \\
\&\\
(\forall\, Q3\ S3\ T3\ R3.\ equal(Q3::'a,R3) \longrightarrow equal(f28(S3::'a,T3,Q3),f28(S3::'a,T3,R3))) \\
\&\\
(\forall\, U3\ V3\ W3\ X3.\ equal(U3::'a,V3) \longrightarrow equal(f29(U3::'a,W3,X3),f29(V3::'a,W3,X3))) \\
\&\\
(\forall\, Y3\ A4\ Z3\ B4.\ equal(Y3::'a,Z3) \longrightarrow equal(f29(A4::'a,Y3,B4),f29(A4::'a,Z3,B4))) \\
\&\\
(\forall\, C4\ E4\ F4\ D4.\ equal(C4::'a,D4) \longrightarrow equal(f29(E4::'a,F4,C4),f29(E4::'a,F4,D4))) \\
\&\\
(\forall\, I4\ J4\ K4\ L4.\ equal(I4::'a,J4) \longrightarrow equal(f30(I4::'a,K4,L4),f30(J4::'a,K4,L4))) \\
\&\\
(\forall\, M4\ O4\ N4\ P4.\ equal(M4::'a,N4) \longrightarrow equal(f30(O4::'a,M4,P4),f30(O4::'a,N4,P4))) \\
\&\\
(\forall\, Q4\ S4\ T4\ R4.\ equal(Q4::'a,R4) \longrightarrow equal(f30(S4::'a,T4,Q4),f30(S4::'a,T4,R4))) \\
\&\\
(\forall\, U4\ V4\ W4\ X4.\ equal(U4::'a,V4) \longrightarrow equal(f31(U4::'a,W4,X4),f31(V4::'a,W4,X4))) \\
\&\\
(\forall\, Y4\ A5\ Z4\ B5.\ equal(Y4::'a,Z4) \longrightarrow equal(f31(A5::'a,Y4,B5),f31(A5::'a,Z4,B5)))
\end{array}
$$

$$
\begin{array}{l}
\& \\
\quad (\forall\, C5\ E5\ F5\ D5.\ equal(C5::'a,D5) \dashrightarrow equal(f31(E5::'a,F5,C5),f31(E5::'a,F5,D5))) \\
\& \\
\quad (\forall\, G5\ H5\ I5\ J5\ K5\ L5.\ equal(G5::'a,H5) \dashrightarrow equal(f32(G5::'a,I5,J5,K5,L5),f32(H5::'a,I5,J5,K5,L5))) \\
\& \\
\quad (\forall\, M5\ O5\ N5\ P5\ Q5\ R5.\ equal(M5::'a,N5) \dashrightarrow equal(f32(O5::'a,M5,P5,Q5,R5),f32(O5::'a,N5,P5,Q5,R5 \\
\& \\
\quad (\forall\, S5\ U5\ V5\ T5\ W5\ X5.\ equal(S5::'a,T5) \dashrightarrow equal(f32(U5::'a,V5,S5,W5,X5),f32(U5::'a,V5,T5,W5,X5) \\
\& \\
\quad (\forall\, Y5\ A6\ B6\ C6\ Z5\ D6.\ equal(Y5::'a,Z5) \dashrightarrow equal(f32(A6::'a,B6,C6,Y5,D6),f32(A6::'a,B6,C6,Z5,D6))) \\
\& \\
\quad (\forall\, E6\ G6\ H6\ I6\ J6\ F6.\ equal(E6::'a,F6) \dashrightarrow equal(f32(G6::'a,H6,I6,J6,E6),f32(G6::'a,H6,I6,J6,F6))) \\
\& \\
\quad (\forall\, K6\ L6\ M6\ N6\ O6\ P6.\ equal(K6::'a,L6) \dashrightarrow equal(f33(K6::'a,M6,N6,O6,P6),f33(L6::'a,M6,N6,O6,P6) \\
\& \\
\quad (\forall\, Q6\ S6\ R6\ T6\ U6\ V6.\ equal(Q6::'a,R6) \dashrightarrow equal(f33(S6::'a,Q6,T6,U6,V6),f33(S6::'a,R6,T6,U6,V6))) \\
\& \\
\quad (\forall\, W6\ Y6\ Z6\ X6\ A7\ B7.\ equal(W6::'a,X6) \dashrightarrow equal(f33(Y6::'a,Z6,W6,A7,B7),f33(Y6::'a,Z6,X6,A7,B7) \\
\& \\
\quad (\forall\, C7\ E7\ F7\ G7\ D7\ H7.\ equal(C7::'a,D7) \dashrightarrow equal(f33(E7::'a,F7,G7,C7,H7),f33(E7::'a,F7,G7,D7,H7)) \\
\& \\
\quad (\forall\, I7\ K7\ L7\ M7\ N7\ J7.\ equal(I7::'a,J7) \dashrightarrow equal(f33(K7::'a,L7,M7,N7,I7),f33(K7::'a,L7,M7,N7,J7))) \\
\& \\
\quad (\forall\, A\ B\ C.\ equal(A::'a,B) \dashrightarrow equal(apply(A::'a,C),apply(B::'a,C))) \ \& \\
\quad (\forall\, D\ F'\ E.\ equal(D::'a,E) \dashrightarrow equal(apply(F'::'a,D),apply(F'::'a,E))) \ \& \\
\quad (\forall\, G\ H\ I'\ J.\ equal(G::'a,H) \dashrightarrow equal(apply\text{-}to\text{-}two\text{-}arguments(G::'a,I',J),apply\text{-}to\text{-}two\text{-}arguments(H::'a,I', \\
\& \\
\quad (\forall\, K'\ M\ L\ N.\ equal(K'::'a,L) \dashrightarrow equal(apply\text{-}to\text{-}two\text{-}arguments(M::'a,K',N),apply\text{-}to\text{-}two\text{-}arguments(M::'a \\
\& \\
\quad (\forall\, O'\ Q\ R\ P.\ equal(O'::'a,P) \dashrightarrow equal(apply\text{-}to\text{-}two\text{-}arguments(Q::'a,R,O'),apply\text{-}to\text{-}two\text{-}arguments(Q::'a, \\
\& \\
\quad (\forall\, S'\ T'.\ equal(S'::'a,T') \dashrightarrow equal(complement(S'),complement(T'))) \ \& \\
\quad (\forall\, U\ V\ W.\ equal(U::'a,V) \dashrightarrow equal(composition(U::'a,W),composition(V::'a,W))) \\
\& \\
\quad (\forall\, X\ Z\ Y.\ equal(X::'a,Y) \dashrightarrow equal(composition(Z::'a,X),composition(Z::'a,Y))) \\
\& \\
\quad (\forall\, A1\ B1.\ equal(A1::'a,B1) \dashrightarrow equal(inv1\ A1,inv1\ B1)) \ \& \\
\quad (\forall\, C1\ D1\ E1.\ equal(C1::'a,D1) \dashrightarrow equal(cross\text{-}product(C1::'a,E1),cross\text{-}product(D1::'a,E1))) \\
\& \\
\quad (\forall\, F1\ H1\ G1.\ equal(F1::'a,G1) \dashrightarrow equal(cross\text{-}product(H1::'a,F1),cross\text{-}product(H1::'a,G1))) \\
\& \\
\quad (\forall\, I1\ J1.\ equal(I1::'a,J1) \dashrightarrow equal(domain\text{-}of(I1),domain\text{-}of(J1))) \ \& \\
\quad (\forall\, I10\ J10.\ equal(I10::'a,J10) \dashrightarrow equal(first(I10),first(J10))) \ \& \\
\quad (\forall\, Q10\ R10.\ equal(Q10::'a,R10) \dashrightarrow equal(flip\text{-}range\text{-}of(Q10),flip\text{-}range\text{-}of(R10))) \\
\& \\
\quad (\forall\, S10\ T10\ U10.\ equal(S10::'a,T10) \dashrightarrow equal(image'(S10::'a,U10),image'(T10::'a,U10))) \\
\& \\
\quad (\forall\, V10\ X10\ W10.\ equal(V10::'a,W10) \dashrightarrow equal(image'(X10::'a,V10),image'(X10::'a,W10))) \\
\&
\end{array}
$$

$$(\forall\ Y10\ Z10\ A11.\ equal(Y10::'a,Z10) \longrightarrow equal(intersection(Y10::'a,A11),intersection(Z10::'a,A11)))$$
$$\&$$
$$(\forall\ B11\ D11\ C11.\ equal(B11::'a,C11) \longrightarrow equal(intersection(D11::'a,B11),intersection(D11::'a,C11)))$$
$$\&$$
$$(\forall\ E11\ F11\ G11.\ equal(E11::'a,F11) \longrightarrow equal(non\text{-}ordered\text{-}pair(E11::'a,G11),non\text{-}ordered\text{-}pair(F11::'a,G$$
$$\&$$
$$(\forall\ H11\ J11\ I11.\ equal(H11::'a,I11) \longrightarrow equal(non\text{-}ordered\text{-}pair(J11::'a,H11),non\text{-}ordered\text{-}pair(J11::'a,I11)$$
$$\&$$
$$(\forall\ K11\ L11\ M11.\ equal(K11::'a,L11) \longrightarrow equal(ordered\text{-}pair(K11::'a,M11),ordered\text{-}pair(L11::'a,M11)))$$
$$\&$$
$$(\forall\ N11\ P11\ O11.\ equal(N11::'a,O11) \longrightarrow equal(ordered\text{-}pair(P11::'a,N11),ordered\text{-}pair(P11::'a,O11)))$$
$$\&$$
$$(\forall\ Q11\ R11.\ equal(Q11::'a,R11) \longrightarrow equal(powerset(Q11),powerset(R11))) \ \&$$
$$(\forall\ S11\ T11.\ equal(S11::'a,T11) \longrightarrow equal(range\text{-}of(S11),range\text{-}of(T11))) \ \&$$
$$(\forall\ U11\ V11\ W11.\ equal(U11::'a,V11) \longrightarrow equal(restrct(U11::'a,W11),restrct(V11::'a,W11)))$$
$$\&$$
$$(\forall\ X11\ Z11\ Y11.\ equal(X11::'a,Y11) \longrightarrow equal(restrct(Z11::'a,X11),restrct(Z11::'a,Y11)))$$
$$\&$$
$$(\forall\ A12\ B12.\ equal(A12::'a,B12) \longrightarrow equal(rot\text{-}right(A12),rot\text{-}right(B12))) \ \&$$
$$(\forall\ C12\ D12.\ equal(C12::'a,D12) \longrightarrow equal(second(C12),second(D12))) \ \&$$
$$(\forall\ K12\ L12.\ equal(K12::'a,L12) \longrightarrow equal(sigma(K12),sigma(L12))) \ \&$$
$$(\forall\ M12\ N12.\ equal(M12::'a,N12) \longrightarrow equal(singleton\text{-}set(M12),singleton\text{-}set(N12)))$$
$$\&$$
$$(\forall\ O12\ P12.\ equal(O12::'a,P12) \longrightarrow equal(successor(O12),successor(P12))) \ \&$$
$$(\forall\ Q12\ R12\ S12.\ equal(Q12::'a,R12) \longrightarrow equal(union(Q12::'a,S12),union(R12::'a,S12)))$$
$$\&$$
$$(\forall\ T12\ V12\ U12.\ equal(T12::'a,U12) \longrightarrow equal(union(V12::'a,T12),union(V12::'a,U12)))$$
$$\&$$
$$(\forall\ W12\ X12\ Y12.\ equal(W12::'a,X12)\ \&\ closed(W12::'a,Y12) \longrightarrow closed(X12::'a,Y12))$$
$$\&$$
$$(\forall\ Z12\ B13\ A13.\ equal(Z12::'a,A13)\ \&\ closed(B13::'a,Z12) \longrightarrow closed(B13::'a,A13))$$
$$\&$$
$$(\forall\ C13\ D13\ E13.\ equal(C13::'a,D13)\ \&\ disjoint(C13::'a,E13) \longrightarrow disjoint(D13::'a,E13))$$
$$\&$$
$$(\forall\ F13\ H13\ G13.\ equal(F13::'a,G13)\ \&\ disjoint(H13::'a,F13) \longrightarrow disjoint(H13::'a,G13))$$
$$\&$$
$$(\forall\ I13\ J13.\ equal(I13::'a,J13)\ \&\ function(I13) \longrightarrow function(J13)) \ \&$$
$$(\forall\ K13\ L13\ M13\ N13\ O13\ P13.\ equal(K13::'a,L13)\ \&\ homomorphism(K13::'a,M13,N13,O13,P13)$$
$$\longrightarrow homomorphism(L13::'a,M13,N13,O13,P13)) \ \&$$
$$(\forall\ Q13\ S13\ R13\ T13\ U13\ V13.\ equal(Q13::'a,R13)\ \&\ homomorphism(S13::'a,Q13,T13,U13,V13)$$
$$\longrightarrow homomorphism(S13::'a,R13,T13,U13,V13)) \ \&$$
$$(\forall\ W13\ Y13\ Z13\ X13\ A14\ B14.\ equal(W13::'a,X13)\ \&\ homomorphism(Y13::'a,Z13,W13,A14,B14)$$
$$\longrightarrow homomorphism(Y13::'a,Z13,X13,A14,B14)) \ \&$$
$$(\forall\ C14\ E14\ F14\ G14\ D14\ H14.\ equal(C14::'a,D14)\ \&\ homomorphism(E14::'a,F14,G14,C14,H14)$$
$$\longrightarrow homomorphism(E14::'a,F14,G14,D14,H14)) \ \&$$
$$(\forall\ I14\ K14\ L14\ M14\ N14\ J14.\ equal(I14::'a,J14)\ \&\ homomorphism(K14::'a,L14,M14,N14,I14)$$
$$\longrightarrow homomorphism(K14::'a,L14,M14,N14,J14)) \ \&$$
$$(\forall\ O14\ P14.\ equal(O14::'a,P14)\ \&\ little\text{-}set(O14) \longrightarrow little\text{-}set(P14)) \ \&$$
$$(\forall\ Q14\ R14\ S14\ T14.\ equal(Q14::'a,R14)\ \&\ maps(Q14::'a,S14,T14) \longrightarrow maps(R14::'a,S14,T14))$$

$$
\begin{array}{l}
\& \\
\quad (\forall\, U14\ W14\ V14\ X14.\ equal(U14::'a,V14)\ \&\ maps(W14::'a,U14,X14)\ \text{--->} \\
maps(W14::'a,V14,X14))\ \& \\
\ (\forall\, Y14\ A15\ B15\ Z14.\ equal(Y14::'a,Z14)\ \&\ maps(A15::'a,B15,Y14)\ \text{--->}\ maps(A15::'a,B15,Z14)) \\
\& \\
\ (\forall\, C15\ D15\ E15.\ equal(C15::'a,D15)\ \&\ member(C15::'a,E15)\ \text{--->}\ member(D15::'a,E15)) \\
\& \\
\ (\forall\, F15\ H15\ G15.\ equal(F15::'a,G15)\ \&\ member(H15::'a,F15)\ \text{--->}\ member(H15::'a,G15)) \\
\& \\
\ (\forall\, I15\ J15.\ equal(I15::'a,J15)\ \&\ one\text{-}to\text{-}one\text{-}function(I15)\ \text{--->}\ one\text{-}to\text{-}one\text{-}function(J15)) \\
\& \\
\ (\forall\, K15\ L15.\ equal(K15::'a,L15)\ \&\ ordered\text{-}pair\text{-}predicate(K15)\ \text{--->}\ ordered\text{-}pair\text{-}predicate(L15)) \\
\& \\
\ (\forall\, M15\ N15\ O15.\ equal(M15::'a,N15)\ \&\ proper\text{-}subset(M15::'a,O15)\ \text{--->}\ proper\text{-}subset(N15::'a,O15)) \\
\& \\
\ (\forall\, P15\ R15\ Q15.\ equal(P15::'a,Q15)\ \&\ proper\text{-}subset(R15::'a,P15)\ \text{--->}\ proper\text{-}subset(R15::'a,Q15)) \\
\& \\
\quad (\forall\, S15\ T15.\ equal(S15::'a,T15)\ \&\ relation(S15)\ \text{--->}\ relation(T15))\ \& \\
\ (\forall\, U15\ V15.\ equal(U15::'a,V15)\ \&\ single\text{-}valued\text{-}set(U15)\ \text{--->}\ single\text{-}valued\text{-}set(V15)) \\
\& \\
\ (\forall\, W15\ X15\ Y15.\ equal(W15::'a,X15)\ \&\ ssubset(W15::'a,Y15)\ \text{--->}\ ssubset(X15::'a,Y15)) \\
\& \\
\ (\forall\, Z15\ B16\ A16.\ equal(Z15::'a,A16)\ \&\ ssubset(B16::'a,Z15)\ \text{--->}\ ssubset(B16::'a,A16)) \\
\& \\
\quad (\sim little\text{-}set(ordered\text{-}pair(a::'a,b)))\ \text{--->}\ False \\
\quad \langle proof \rangle
\end{array}
$$

lemma *SET046-5*:

$$
\begin{array}{l}
(\forall\, Y\ X.\ \sim(element(X::'a,a)\ \&\ element(X::'a,Y)\ \&\ element(Y::'a,X)))\ \& \\
\quad (\forall\, X.\ element(X::'a,f(X))\ |\ element(X::'a,a))\ \& \\
\quad (\forall\, X.\ element(f(X),X)\ |\ element(X::'a,a))\ \text{--->}\ False \\
\quad \langle proof \rangle
\end{array}
$$

lemma *SET047-5*:

$$
\begin{array}{l}
(\forall\, X\ Z\ Y.\ set\text{-}equal(X::'a,Y)\ \&\ element(Z::'a,X)\ \text{--->}\ element(Z::'a,Y))\ \& \\
\quad (\forall\, Y\ Z\ X.\ set\text{-}equal(X::'a,Y)\ \&\ element(Z::'a,Y)\ \text{--->}\ element(Z::'a,X))\ \& \\
\quad (\forall\, X\ Y.\ element(f(X::'a,Y),X)\ |\ element(f(X::'a,Y),Y)\ |\ set\text{-}equal(X::'a,Y)) \\
\& \\
\ (\forall\, X\ Y.\ element(f(X::'a,Y),Y)\ \&\ element(f(X::'a,Y),X)\ \text{--->}\ set\text{-}equal(X::'a,Y)) \\
\& \\
\quad (set\text{-}equal(a::'a,b)\ |\ set\text{-}equal(b::'a,a))\ \& \\
\quad (\sim(set\text{-}equal(b::'a,a)\ \&\ set\text{-}equal(a::'a,b)))\ \text{--->}\ False \\
\quad \langle proof \rangle
\end{array}
$$

lemma *SYN034-1*:

$(\forall A.\ p(A::'a,a) \mid p(A::'a,f(A)))\ \&$
$(\forall A.\ p(A::'a,a) \mid p(f(A),A))\ \&$
$(\forall A\ B.\ \sim(p(A::'a,B)\ \&\ p(B::'a,A)\ \&\ p(B::'a,a)))\ --{>}\ \mathit{False}$
$\langle \mathit{proof} \rangle$

lemma *SYN071-1*:
$\mathit{EQU001\text{-}0\text{-}ax\ equal}\ \&$
$(\mathit{equal}(a::'a,b) \mid \mathit{equal}(c::'a,d))\ \&$
$(\mathit{equal}(a::'a,c) \mid \mathit{equal}(b::'a,d))\ \&$
$(\sim \mathit{equal}(a::'a,d))\ \&$
$(\sim \mathit{equal}(b::'a,c))\ --{>}\ \mathit{False}$
$\langle \mathit{proof} \rangle$

lemma *SYN349-1*:
$(\forall X\ Y.\ f(w(X),g(X::'a,Y))\ --{>}\ f(X::'a,g(X::'a,Y)))\ \&$
$(\forall X\ Y.\ f(X::'a,g(X::'a,Y))\ --{>}\ f(w(X),g(X::'a,Y)))\ \&$
$(\forall Y\ X.\ f(X::'a,g(X::'a,Y))\ \&\ f(Y::'a,g(X::'a,Y))\ --{>}\ f(g(X::'a,Y),Y)\ \mid$
$f(g(X::'a,Y),w(X)))\ \&$
$(\forall Y\ X.\ f(g(X::'a,Y),Y)\ \&\ f(Y::'a,g(X::'a,Y))\ --{>}\ f(X::'a,g(X::'a,Y))\ \mid$
$f(g(X::'a,Y),w(X)))\ \&$
$(\forall Y\,X.\ f(X::'a,g(X::'a,Y)) \mid f(g(X::'a,Y),Y) \mid f(Y::'a,g(X::'a,Y)) \mid f(g(X::'a,Y),w(X)))$
$\&$
$(\forall Y\ X.\ f(X::'a,g(X::'a,Y))\ \&\ f(g(X::'a,Y),Y)\ --{>}\ f(Y::'a,g(X::'a,Y))\ \mid$
$f(g(X::'a,Y),w(X)))\ \&$
$(\forall Y\ X.\ f(X::'a,g(X::'a,Y))\ \&\ f(g(X::'a,Y),w(X))\ --{>}\ f(g(X::'a,Y),Y)\ \mid$
$f(Y::'a,g(X::'a,Y)))\ \&$
$(\forall Y\ X.\ f(g(X::'a,Y),Y)\ \&\ f(g(X::'a,Y),w(X))\ --{>}\ f(X::'a,g(X::'a,Y))\ \mid$
$f(Y::'a,g(X::'a,Y)))\ \&$
$(\forall Y\,X.\ f(Y::'a,g(X::'a,Y))\ \&\ f(g(X::'a,Y),w(X))\ --{>}\ f(X::'a,g(X::'a,Y))\ \mid$
$f(g(X::'a,Y),Y))\ \&$
$(\forall Y\ X.\ \sim(f(X::'a,g(X::'a,Y))\ \&\ f(g(X::'a,Y),Y)\ \&\ f(Y::'a,g(X::'a,Y))\ \&$
$f(g(X::'a,Y),w(X))))\ --{>}\ \mathit{False}$
$\langle \mathit{proof} \rangle$

lemma *SYN352-1*:
$(f(a::'a,b))\ \&$
$(\forall X\ Y.\ f(X::'a,Y)\ --{>}\ f(b::'a,z(X::'a,Y)) \mid f(Y::'a,z(X::'a,Y)))\ \&$
$(\forall X\ Y.\ f(X::'a,Y) \mid f(z(X::'a,Y),z(X::'a,Y)))\ \&$
$(\forall X\ Y.\ f(b::'a,z(X::'a,Y)) \mid f(X::'a,z(X::'a,Y)) \mid f(z(X::'a,Y),z(X::'a,Y)))\ \&$
$(\forall X\ Y.\ f(b::'a,z(X::'a,Y))\ \&\ f(X::'a,z(X::'a,Y))\ --{>}\ f(z(X::'a,Y),z(X::'a,Y)))$
$\&$
$(\forall X\ Y.\ \sim(f(X::'a,Y)\ \&\ f(X::'a,z(X::'a,Y))\ \&\ f(Y::'a,z(X::'a,Y))))\ \&$
$(\forall X\ Y.\ f(X::'a,Y)\ --{>}\ f(X::'a,z(X::'a,Y)) \mid f(Y::'a,z(X::'a,Y)))\ --{>}\ \mathit{False}$
$\langle \mathit{proof} \rangle$

lemma *TOP001-2*:
$(\forall\ Vf\ U.\ element\text{-}of\text{-}set(U::'a,union\text{-}of\text{-}members(Vf)) \longrightarrow element\text{-}of\text{-}set(U::'a,f1(Vf::'a,U)))$
&
$(\forall\ U\ Vf.\ element\text{-}of\text{-}set(U::'a,union\text{-}of\text{-}members(Vf)) \longrightarrow element\text{-}of\text{-}collection(f1(Vf::'a,U),Vf))$
&
$(\forall\ U\ \ Uu1\ \ Vf.\ \ element\text{-}of\text{-}set(U::'a,Uu1)\ \ \&\ \ element\text{-}of\text{-}collection(Uu1::'a,Vf)$
$\longrightarrow element\text{-}of\text{-}set(U::'a,union\text{-}of\text{-}members(Vf)))\ \&$
$(\forall\ Vf\ X.\ basis(X::'a,Vf) \longrightarrow equal\text{-}sets(union\text{-}of\text{-}members(Vf),X))\ \&$
$(\forall\ Vf\ U\ X.\ element\text{-}of\text{-}collection(U::'a,top\text{-}of\text{-}basis(Vf))\ \&\ element\text{-}of\text{-}set(X::'a,U)$
$\longrightarrow element\text{-}of\text{-}set(X::'a,f10(Vf::'a,U,X)))\ \&$
$(\forall\ U\ X\ Vf.\ element\text{-}of\text{-}collection(U::'a,top\text{-}of\text{-}basis(Vf))\ \&\ element\text{-}of\text{-}set(X::'a,U)$
$\longrightarrow element\text{-}of\text{-}collection(f10(Vf::'a,U,X),Vf))\ \&$
$(\forall\ X.\ subset\text{-}sets(X::'a,X))\ \&$
$(\forall\ X\ U\ Y.\ subset\text{-}sets(X::'a,Y)\ \&\ element\text{-}of\text{-}set(U::'a,X) \longrightarrow element\text{-}of\text{-}set(U::'a,Y))$
&
$(\forall\ X\ \ Y.\ equal\text{-}sets(X::'a,Y) \longrightarrow subset\text{-}sets(X::'a,Y))\ \&$
$(\forall\ Y\ X.\ subset\text{-}sets(X::'a,Y) \mid element\text{-}of\text{-}set(in\text{-}1st\text{-}set(X::'a,Y),X))\ \&$
$(\forall\ X\ \ Y.\ element\text{-}of\text{-}set(in\text{-}1st\text{-}set(X::'a,Y),Y) \longrightarrow subset\text{-}sets(X::'a,Y))\ \&$
$(basis(cx::'a,f))\ \&$
$(\sim subset\text{-}sets(union\text{-}of\text{-}members(top\text{-}of\text{-}basis(f)),cx)) \longrightarrow False$
$\langle proof \rangle$

lemma *TOP002-2*:
$(\forall\ Vf\ U.\ element\text{-}of\text{-}collection(U::'a,top\text{-}of\text{-}basis(Vf)) \mid element\text{-}of\text{-}set(f11(Vf::'a,U),U))$
&
$(\forall\ X.\ \sim element\text{-}of\text{-}set(X::'a,empty\text{-}set))\ \&$
$(\sim element\text{-}of\text{-}collection(empty\text{-}set::'a,top\text{-}of\text{-}basis(f))) \longrightarrow False$
$\langle proof \rangle$

lemma *TOP004-1*:
$(\forall\ Vf\ U.\ element\text{-}of\text{-}set(U::'a,union\text{-}of\text{-}members(Vf)) \longrightarrow element\text{-}of\text{-}set(U::'a,f1(Vf::'a,U)))$
&
$(\forall\ U\ Vf.\ element\text{-}of\text{-}set(U::'a,union\text{-}of\text{-}members(Vf)) \longrightarrow element\text{-}of\text{-}collection(f1(Vf::'a,U),Vf))$
&
$(\forall\ U\ \ Uu1\ \ Vf.\ \ element\text{-}of\text{-}set(U::'a,Uu1)\ \ \&\ \ element\text{-}of\text{-}collection(Uu1::'a,Vf)$
$\longrightarrow element\text{-}of\text{-}set(U::'a,union\text{-}of\text{-}members(Vf)))\ \&$
$(\forall\ Vf\ U\ Va.\ element\text{-}of\text{-}set(U::'a,intersection\text{-}of\text{-}members(Vf))\ \&\ element\text{-}of\text{-}collection(Va::'a,Vf)$
$\longrightarrow element\text{-}of\text{-}set(U::'a,Va))\ \&$
$(\forall\ U\ Vf.\ element\text{-}of\text{-}set(U::'a,intersection\text{-}of\text{-}members(Vf)) \mid element\text{-}of\text{-}collection(f2(Vf::'a,U),Vf))$
&
$(\forall\ Vf\ U.\ element\text{-}of\text{-}set(U::'a,f2(Vf::'a,U)) \longrightarrow element\text{-}of\text{-}set(U::'a,intersection\text{-}of\text{-}members(Vf)))$
&
$(\forall\ Vt\ X.\ topological\text{-}space(X::'a,Vt) \longrightarrow equal\text{-}sets(union\text{-}of\text{-}members(Vt),X))$
&
$(\forall\ X\ Vt.\ topological\text{-}space(X::'a,Vt) \longrightarrow element\text{-}of\text{-}collection(empty\text{-}set::'a,Vt))$
&
$(\forall\ X\ \ Vt.\ topological\text{-}space(X::'a,Vt) \longrightarrow element\text{-}of\text{-}collection(X::'a,Vt))\ \&$

$$(\forall X\ Y\ Z\ Vt.\ topological\text{-}space(X::'a,Vt)\ \&\ element\text{-}of\text{-}collection(Y::'a,Vt)\ \&\ element\text{-}of\text{-}collection(Z::'a,Vt) \longrightarrow element\text{-}of\text{-}collection(intersection\text{-}of\text{-}sets(Y::'a,Z),Vt))$$
$$\&$$
$$(\forall X\ Vf\ Vt.\ topological\text{-}space(X::'a,Vt)\ \&\ subset\text{-}collections(Vf::'a,Vt) \longrightarrow element\text{-}of\text{-}collection(union\text{-}of\text{-}members(Vf),Vt))\ \&$$
$$(\forall X\ Vt.\ equal\text{-}sets(union\text{-}of\text{-}members(Vt),X)\ \&\ element\text{-}of\text{-}collection(empty\text{-}set::'a,Vt)\ \&\ element\text{-}of\text{-}collection(X::'a,Vt) \longrightarrow topological\text{-}space(X::'a,Vt) \mid element\text{-}of\text{-}collection(f3(X::'a,Vt),Vt) \mid subset\text{-}collections(f5(X::'a,Vt),Vt))\ \&$$
$$(\forall X\ Vt.\ equal\text{-}sets(union\text{-}of\text{-}members(Vt),X)\ \&\ element\text{-}of\text{-}collection(empty\text{-}set::'a,Vt)\ \&\ element\text{-}of\text{-}collection(X::'a,Vt)\ \&\ element\text{-}of\text{-}collection(union\text{-}of\text{-}members(f5(X::'a,Vt)),Vt) \longrightarrow topological\text{-}space(X::'a,Vt) \mid element\text{-}of\text{-}collection(f3(X::'a,Vt),Vt))\ \&$$
$$(\forall X\ Vt.\ equal\text{-}sets(union\text{-}of\text{-}members(Vt),X)\ \&\ element\text{-}of\text{-}collection(empty\text{-}set::'a,Vt)\ \&\ element\text{-}of\text{-}collection(X::'a,Vt) \longrightarrow topological\text{-}space(X::'a,Vt) \mid element\text{-}of\text{-}collection(f4(X::'a,Vt),Vt) \mid subset\text{-}collections(f5(X::'a,Vt),Vt))\ \&$$
$$(\forall X\ Vt.\ equal\text{-}sets(union\text{-}of\text{-}members(Vt),X)\ \&\ element\text{-}of\text{-}collection(empty\text{-}set::'a,Vt)\ \&\ element\text{-}of\text{-}collection(X::'a,Vt)\ \&\ element\text{-}of\text{-}collection(union\text{-}of\text{-}members(f5(X::'a,Vt)),Vt) \longrightarrow topological\text{-}space(X::'a,Vt) \mid element\text{-}of\text{-}collection(f4(X::'a,Vt),Vt))\ \&$$
$$(\forall X\ Vt.\ equal\text{-}sets(union\text{-}of\text{-}members(Vt),X)\ \&\ element\text{-}of\text{-}collection(empty\text{-}set::'a,Vt)\ \&\ element\text{-}of\text{-}collection(X::'a,Vt)\ \&\ element\text{-}of\text{-}collection(intersection\text{-}of\text{-}sets(f3(X::'a,Vt),f4(X::'a,Vt)),Vt) \longrightarrow topological\text{-}space(X::'a,Vt) \mid subset\text{-}collections(f5(X::'a,Vt),Vt))\ \&$$
$$(\forall X\ Vt.\ equal\text{-}sets(union\text{-}of\text{-}members(Vt),X)\ \&\ element\text{-}of\text{-}collection(empty\text{-}set::'a,Vt)\ \&\ element\text{-}of\text{-}collection(X::'a,Vt)\ \&\ element\text{-}of\text{-}collection(intersection\text{-}of\text{-}sets(f3(X::'a,Vt),f4(X::'a,Vt)),Vt)\ \&\ element\text{-}of\text{-}collection(union\text{-}of\text{-}members(f5(X::'a,Vt)),Vt) \longrightarrow topological\text{-}space(X::'a,Vt))$$
$$\&$$
$$(\forall U\ X\ Vt.\ open(U::'a,X,Vt) \longrightarrow topological\text{-}space(X::'a,Vt))\ \&$$
$$(\forall X\ U\ Vt.\ open(U::'a,X,Vt) \longrightarrow element\text{-}of\text{-}collection(U::'a,Vt))\ \&$$
$$(\forall X\ U\ Vt.\ topological\text{-}space(X::'a,Vt)\ \&\ element\text{-}of\text{-}collection(U::'a,Vt) \longrightarrow open(U::'a,X,Vt))\ \&$$
$$(\forall U\ X\ Vt.\ closed(U::'a,X,Vt) \longrightarrow topological\text{-}space(X::'a,Vt))\ \&$$
$$(\forall U\ X\ Vt.\ closed(U::'a,X,Vt) \longrightarrow open(relative\text{-}complement\text{-}sets(U::'a,X),X,Vt))$$
$$\&$$
$$(\forall U\ X\ Vt.\ topological\text{-}space(X::'a,Vt)\ \&\ open(relative\text{-}complement\text{-}sets(U::'a,X),X,Vt) \longrightarrow closed(U::'a,X,Vt))\ \&$$
$$(\forall Vs\ X\ Vt.\ finer(Vt::'a,Vs,X) \longrightarrow topological\text{-}space(X::'a,Vt))\ \&$$
$$(\forall Vt\ X\ Vs.\ finer(Vt::'a,Vs,X) \longrightarrow topological\text{-}space(X::'a,Vs))\ \&$$
$$(\forall X\ Vs\ Vt.\ finer(Vt::'a,Vs,X) \longrightarrow subset\text{-}collections(Vs::'a,Vt))\ \&$$
$$(\forall X\ Vs\ Vt.\ topological\text{-}space(X::'a,Vt)\ \&\ topological\text{-}space(X::'a,Vs)\ \&\ subset\text{-}collections(Vs::'a,Vt) \longrightarrow finer(Vt::'a,Vs,X))\ \&$$
$$(\forall Vf\ X.\ basis(X::'a,Vf) \longrightarrow equal\text{-}sets(union\text{-}of\text{-}members(Vf),X))\ \&$$
$$(\forall X\ Vf\ Y\ Vb1\ Vb2.\ basis(X::'a,Vf)\ \&\ element\text{-}of\text{-}set(Y::'a,X)\ \&\ element\text{-}of\text{-}collection(Vb1::'a,Vf)\ \&\ element\text{-}of\text{-}collection(Vb2::'a,Vf)\ \&\ element\text{-}of\text{-}set(Y::'a,intersection\text{-}of\text{-}sets(Vb1::'a,Vb2)) \longrightarrow element\text{-}of\text{-}set(Y::'a,f6(X::'a,Vf,Y,Vb1,Vb2)))\ \&$$
$$(\forall X\ Y\ Vb1\ Vb2\ Vf.\ basis(X::'a,Vf)\ \&\ element\text{-}of\text{-}set(Y::'a,X)\ \&\ element\text{-}of\text{-}collection(Vb1::'a,Vf)\ \&\ element\text{-}of\text{-}collection(Vb2::'a,Vf)\ \&\ element\text{-}of\text{-}set(Y::'a,intersection\text{-}of\text{-}sets(Vb1::'a,Vb2)) \longrightarrow element\text{-}of\text{-}collection(f6(X::'a,Vf,Y,Vb1,Vb2),Vf))\ \&$$
$$(\forall X\ Vf\ Y\ Vb1\ Vb2.\ basis(X::'a,Vf)\ \&\ element\text{-}of\text{-}set(Y::'a,X)\ \&\ element\text{-}of\text{-}collection(Vb1::'a,Vf)\ \&\ element\text{-}of\text{-}collection(Vb2::'a,Vf)\ \&\ element\text{-}of\text{-}set(Y::'a,intersection\text{-}of\text{-}sets(Vb1::'a,Vb2)) \longrightarrow subset\text{-}sets(f6(X::'a,Vf,Y,Vb1,Vb2),intersection\text{-}of\text{-}sets(Vb1::'a,Vb2)))\ \&$$
$$(\forall Vf\ X.\ equal\text{-}sets(union\text{-}of\text{-}members(Vf),X) \longrightarrow basis(X::'a,Vf) \mid element\text{-}of\text{-}set(f7(X::'a,Vf),X))$$

$$
\begin{array}{l}
\& \\
\quad (\forall\, X\ Vf.\ \mathit{equal\text{-}sets}(\mathit{union\text{-}of\text{-}members}(Vf),X) \longrightarrow \mathit{basis}(X::'a,Vf) \mid \mathit{element\text{-}of\text{-}collection}(f8(X::'a,Vf),Vf) \\
\& \\
\quad (\forall\, X\ Vf.\ \mathit{equal\text{-}sets}(\mathit{union\text{-}of\text{-}members}(Vf),X) \longrightarrow \mathit{basis}(X::'a,Vf) \mid \mathit{element\text{-}of\text{-}collection}(f9(X::'a,Vf),Vf) \\
\& \\
\quad (\forall\, X\ Vf.\ \mathit{equal\text{-}sets}(\mathit{union\text{-}of\text{-}members}(Vf),X) \longrightarrow \mathit{basis}(X::'a,Vf) \mid \mathit{element\text{-}of\text{-}set}(f7(X::'a,Vf),\mathit{intersectio} \\
\& \\
\quad (\forall\, Uu9\ X\ Vf.\ \mathit{equal\text{-}sets}(\mathit{union\text{-}of\text{-}members}(Vf),X) \ \&\ \mathit{element\text{-}of\text{-}set}(f7(X::'a,Vf),Uu9) \\
\&\ \mathit{element\text{-}of\text{-}collection}(Uu9::'a,Vf) \ \&\ \mathit{subset\text{-}sets}(Uu9::'a,\mathit{intersection\text{-}of\text{-}sets}(f8(X::'a,Vf),f9(X::'a,Vf))) \\
\longrightarrow \mathit{basis}(X::'a,Vf)) \ \& \\
\quad (\forall\, Vf\ U\ X.\ \mathit{element\text{-}of\text{-}collection}(U::'a,\mathit{top\text{-}of\text{-}basis}(Vf)) \ \&\ \mathit{element\text{-}of\text{-}set}(X::'a,U) \\
\longrightarrow \mathit{element\text{-}of\text{-}set}(X::'a,f10(Vf::'a,U,X))) \ \& \\
\quad (\forall\, U\ X\ Vf.\ \mathit{element\text{-}of\text{-}collection}(U::'a,\mathit{top\text{-}of\text{-}basis}(Vf)) \ \&\ \mathit{element\text{-}of\text{-}set}(X::'a,U) \\
\longrightarrow \mathit{element\text{-}of\text{-}collection}(f10(Vf::'a,U,X),Vf)) \ \& \\
\quad (\forall\, Vf\ X\ U.\ \mathit{element\text{-}of\text{-}collection}(U::'a,\mathit{top\text{-}of\text{-}basis}(Vf)) \ \&\ \mathit{element\text{-}of\text{-}set}(X::'a,U) \\
\longrightarrow \mathit{subset\text{-}sets}(f10(Vf::'a,U,X),U)) \ \& \\
\quad (\forall\, Vf\ U.\ \mathit{element\text{-}of\text{-}collection}(U::'a,\mathit{top\text{-}of\text{-}basis}(Vf)) \mid \mathit{element\text{-}of\text{-}set}(f11(Vf::'a,U),U)) \\
\& \\
\quad (\forall\, Vf\ Uu11\ U.\ \mathit{element\text{-}of\text{-}set}(f11(Vf::'a,U),Uu11) \ \&\ \mathit{element\text{-}of\text{-}collection}(Uu11::'a,Vf) \\
\&\ \mathit{subset\text{-}sets}(Uu11::'a,U) \longrightarrow \mathit{element\text{-}of\text{-}collection}(U::'a,\mathit{top\text{-}of\text{-}basis}(Vf))) \ \& \\
\quad\ (\forall\, U\ Y\ X\ Vt.\ \mathit{element\text{-}of\text{-}collection}(U::'a,\mathit{subspace\text{-}topology}(X::'a,Vt,Y)) \longrightarrow \\
\mathit{topological\text{-}space}(X::'a,Vt)) \ \& \\
\quad\ (\forall\, U\ Vt\ Y\ X.\ \mathit{element\text{-}of\text{-}collection}(U::'a,\mathit{subspace\text{-}topology}(X::'a,Vt,Y)) \longrightarrow \\
\mathit{subset\text{-}sets}(Y::'a,X)) \ \& \\
\quad\ (\forall\, X\ Y\ U\ Vt.\ \mathit{element\text{-}of\text{-}collection}(U::'a,\mathit{subspace\text{-}topology}(X::'a,Vt,Y)) \longrightarrow \\
\mathit{element\text{-}of\text{-}collection}(f12(X::'a,Vt,Y,U),Vt)) \ \& \\
\quad\ (\forall\, X\ Vt\ Y\ U.\ \mathit{element\text{-}of\text{-}collection}(U::'a,\mathit{subspace\text{-}topology}(X::'a,Vt,Y)) \longrightarrow \\
\mathit{equal\text{-}sets}(U::'a,\mathit{intersection\text{-}of\text{-}sets}(Y::'a,f12(X::'a,Vt,Y,U)))) \ \& \\
\quad (\forall\, X\ Vt\ U\ Y\ Uu12.\ \mathit{topological\text{-}space}(X::'a,Vt) \ \&\ \mathit{subset\text{-}sets}(Y::'a,X) \ \&\ \mathit{element\text{-}of\text{-}collection}(Uu12::'a,Vt) \\
\&\ \mathit{equal\text{-}sets}(U::'a,\mathit{intersection\text{-}of\text{-}sets}(Y::'a,Uu12)) \longrightarrow \mathit{element\text{-}of\text{-}collection}(U::'a,\mathit{subspace\text{-}topology}(X::'a \\
\& \\
\quad (\forall\, U\ Y\ X\ Vt.\ \mathit{element\text{-}of\text{-}set}(U::'a,\mathit{interior}(Y::'a,X,Vt)) \longrightarrow \mathit{topological\text{-}space}(X::'a,Vt)) \\
\& \\
\quad (\forall\, U\ Vt\ Y\ X.\ \mathit{element\text{-}of\text{-}set}(U::'a,\mathit{interior}(Y::'a,X,Vt)) \longrightarrow \mathit{subset\text{-}sets}(Y::'a,X)) \\
\& \\
\quad (\forall\, Y\ X\ Vt\ U.\ \mathit{element\text{-}of\text{-}set}(U::'a,\mathit{interior}(Y::'a,X,Vt)) \longrightarrow \mathit{element\text{-}of\text{-}set}(U::'a,f13(Y::'a,X,Vt,U))) \\
\& \\
\quad (\forall\, X\ Vt\ U\ Y.\ \mathit{element\text{-}of\text{-}set}(U::'a,\mathit{interior}(Y::'a,X,Vt)) \longrightarrow \mathit{subset\text{-}sets}(f13(Y::'a,X,Vt,U),Y)) \\
\& \\
\quad (\forall\, Y\ U\ X\ Vt.\ \mathit{element\text{-}of\text{-}set}(U::'a,\mathit{interior}(Y::'a,X,Vt)) \longrightarrow \mathit{open}(f13(Y::'a,X,Vt,U),X,Vt)) \\
\& \\
\quad (\forall\, U\ Y\ Uu13\ X\ Vt.\ \mathit{topological\text{-}space}(X::'a,Vt) \ \&\ \mathit{subset\text{-}sets}(Y::'a,X) \ \&\ \mathit{element\text{-}of\text{-}set}(U::'a,Uu13) \\
\&\ \mathit{subset\text{-}sets}(Uu13::'a,Y) \ \&\ \mathit{open}(Uu13::'a,X,Vt) \longrightarrow \mathit{element\text{-}of\text{-}set}(U::'a,\mathit{interior}(Y::'a,X,Vt))) \\
\& \\
\quad (\forall\, U\ Y\ X\ Vt.\ \mathit{element\text{-}of\text{-}set}(U::'a,\mathit{closure}(Y::'a,X,Vt)) \longrightarrow \mathit{topological\text{-}space}(X::'a,Vt)) \\
\& \\
\quad (\forall\, U\ Vt\ Y\ X.\ \mathit{element\text{-}of\text{-}set}(U::'a,\mathit{closure}(Y::'a,X,Vt)) \longrightarrow \mathit{subset\text{-}sets}(Y::'a,X)) \\
\& \\
\quad (\forall\, Y\ X\ Vt\ U\ V.\ \mathit{element\text{-}of\text{-}set}(U::'a,\mathit{closure}(Y::'a,X,Vt)) \ \&\ \mathit{subset\text{-}sets}(Y::'a,V)
\end{array}
$$

& closed(V::'a,X,Vt) --> element-of-set(U::'a,V)) &
 (∀ Y X Vt U. topological-space(X::'a,Vt) & subset-sets(Y::'a,X) --> element-of-set(U::'a,closure(Y::'a,X,
| subset-sets(Y::'a,f14(Y::'a,X,Vt,U))) &
 (∀ Y U X Vt. topological-space(X::'a,Vt) & subset-sets(Y::'a,X) --> element-of-set(U::'a,closure(Y::'a,X,
| closed(f14(Y::'a,X,Vt,U),X,Vt)) &
 (∀ Y X Vt U. topological-space(X::'a,Vt) & subset-sets(Y::'a,X) & element-of-set(U::'a,f14(Y::'a,X,Vt,U))
--> element-of-set(U::'a,closure(Y::'a,X,Vt))) &
 (∀ U Y X Vt. neighborhood(U::'a,Y,X,Vt) --> topological-space(X::'a,Vt)) &
 (∀ Y U X Vt. neighborhood(U::'a,Y,X,Vt) --> open(U::'a,X,Vt)) &
 (∀ X Vt Y U. neighborhood(U::'a,Y,X,Vt) --> element-of-set(Y::'a,U)) &
 (∀ X Vt Y U. topological-space(X::'a,Vt) & open(U::'a,X,Vt) & element-of-set(Y::'a,U)
--> neighborhood(U::'a,Y,X,Vt)) &
 (∀ Z Y X Vt. limit-point(Z::'a,Y,X,Vt) --> topological-space(X::'a,Vt)) &
 (∀ Z Vt Y X. limit-point(Z::'a,Y,X,Vt) --> subset-sets(Y::'a,X)) &
 (∀ Z X Vt U Y. limit-point(Z::'a,Y,X,Vt) & neighborhood(U::'a,Z,X,Vt) -->
element-of-set(f15(Z::'a,Y,X,Vt,U),intersection-of-sets(U::'a,Y))) &
 (∀ Y X Vt U Z. ~(limit-point(Z::'a,Y,X,Vt) & neighborhood(U::'a,Z,X,Vt) &
eq-p(f15(Z::'a,Y,X,Vt,U),Z))) &
 (∀ Y Z X Vt. topological-space(X::'a,Vt) & subset-sets(Y::'a,X) --> limit-point(Z::'a,Y,X,Vt)
| neighborhood(f16(Z::'a,Y,X,Vt),Z,X,Vt)) &
 (∀ X Vt Y Uu16 Z. topological-space(X::'a,Vt) & subset-sets(Y::'a,X) & element-of-set(Uu16::'a,intersection
--> limit-point(Z::'a,Y,X,Vt) | eq-p(Uu16::'a,Z)) &
 (∀ U Y X Vt. element-of-set(U::'a,boundary(Y::'a,X,Vt)) --> topological-space(X::'a,Vt))
&
 (∀ U Y X Vt. element-of-set(U::'a,boundary(Y::'a,X,Vt)) --> element-of-set(U::'a,closure(Y::'a,X,Vt)))
&
 (∀ U Y X Vt. element-of-set(U::'a,boundary(Y::'a,X,Vt)) --> element-of-set(U::'a,closure(relative-comple
&
 (∀ U Y X Vt. topological-space(X::'a,Vt) & element-of-set(U::'a,closure(Y::'a,X,Vt))
& element-of-set(U::'a,closure(relative-complement-sets(Y::'a,X),X,Vt)) --> element-of-set(U::'a,boundary
&
 (∀ X Vt. hausdorff(X::'a,Vt) --> topological-space(X::'a,Vt)) &
 (∀ X-2 X-1 X Vt. hausdorff(X::'a,Vt) & element-of-set(X-1::'a,X) & element-of-set(X-2::'a,X)
--> eq-p(X-1::'a,X-2) | neighborhood(f17(X::'a,Vt,X-1,X-2),X-1,X,Vt)) &
 (∀ X-1 X-2 X Vt. hausdorff(X::'a,Vt) & element-of-set(X-1::'a,X) & element-of-set(X-2::'a,X)
--> eq-p(X-1::'a,X-2) | neighborhood(f18(X::'a,Vt,X-1,X-2),X-2,X,Vt)) &
 (∀ X Vt X-1 X-2. hausdorff(X::'a,Vt) & element-of-set(X-1::'a,X) & element-of-set(X-2::'a,X)
--> eq-p(X-1::'a,X-2) | disjoint-s(f17(X::'a,Vt,X-1,X-2),f18(X::'a,Vt,X-1,X-2)))
&
 (∀ Vt X. topological-space(X::'a,Vt) --> hausdorff(X::'a,Vt) | element-of-set(f19(X::'a,Vt),X))
&
 (∀ Vt X. topological-space(X::'a,Vt) --> hausdorff(X::'a,Vt) | element-of-set(f20(X::'a,Vt),X))
&
  (∀ X Vt. topological-space(X::'a,Vt) & eq-p(f19(X::'a,Vt),f20(X::'a,Vt)) -->
hausdorff(X::'a,Vt)) &
 (∀ X Vt Uu19 Uu20. topological-space(X::'a,Vt) & neighborhood(Uu19::'a,f19(X::'a,Vt),X,Vt)
& neighborhood(Uu20::'a,f20(X::'a,Vt),X,Vt) & disjoint-s(Uu19::'a,Uu20) -->
hausdorff(X::'a,Vt)) &
 (∀ Va1 Va2 X Vt. separation(Va1::'a,Va2,X,Vt) --> topological-space(X::'a,Vt))

$$
\begin{array}{l}
\& \\
\quad (\forall\ Va2\ X\ Vt\ Va1.\ \sim(separation(Va1::'a, Va2, X, Vt)\ \&\ equal\text{-}sets(Va1::'a, empty\text{-}set))) \\
\& \\
\quad (\forall\ Va1\ X\ Vt\ Va2.\ \sim(separation(Va1::'a, Va2, X, Vt)\ \&\ equal\text{-}sets(Va2::'a, empty\text{-}set))) \\
\& \\
\quad (\forall\ Va2\ X\ Va1\ Vt.\ separation(Va1::'a, Va2, X, Vt) \longrightarrow element\text{-}of\text{-}collection(Va1::'a, Vt)) \\
\& \\
\quad (\forall\ Va1\ X\ Va2\ Vt.\ separation(Va1::'a, Va2, X, Vt) \longrightarrow element\text{-}of\text{-}collection(Va2::'a, Vt)) \\
\& \\
\quad (\forall\ Vt\ Va1\ Va2\ X.\ separation(Va1::'a, Va2, X, Vt) \longrightarrow equal\text{-}sets(union\text{-}of\text{-}sets(Va1::'a, Va2), X)) \\
\& \\
\quad (\forall\ X\ Vt\ Va1\ Va2.\ separation(Va1::'a, Va2, X, Vt) \longrightarrow disjoint\text{-}s(Va1::'a, Va2)) \\
\& \\
\quad (\forall\ Vt\ X\ Va1\ Va2.\ topological\text{-}space(X::'a, Vt)\ \&\ element\text{-}of\text{-}collection(Va1::'a, Vt) \\
\& \ element\text{-}of\text{-}collection(Va2::'a, Vt)\ \&\ equal\text{-}sets(union\text{-}of\text{-}sets(Va1::'a, Va2), X)\ \& \\
disjoint\text{-}s(Va1::'a, Va2) \longrightarrow separation(Va1::'a, Va2, X, Vt) \mid equal\text{-}sets(Va1::'a, empty\text{-}set) \\
\mid equal\text{-}sets(Va2::'a, empty\text{-}set))\ \& \\
\quad (\forall\ X\ Vt.\ connected\text{-}space(X::'a, Vt) \longrightarrow topological\text{-}space(X::'a, Vt))\ \& \\
\quad (\forall\ Va1\ Va2\ X\ Vt.\ \sim(connected\text{-}space(X::'a, Vt)\ \&\ separation(Va1::'a, Va2, X, Vt))) \\
\& \\
\quad (\forall\ X\ Vt.\ topological\text{-}space(X::'a, Vt) \longrightarrow connected\text{-}space(X::'a, Vt) \mid separa\text{-} \\
tion(f21(X::'a, Vt), f22(X::'a, Vt), X, Vt))\ \& \\
\quad (\forall\ Va\ X\ Vt.\ connected\text{-}set(Va::'a, X, Vt) \longrightarrow topological\text{-}space(X::'a, Vt))\ \& \\
\quad (\forall\ Vt\ Va\ X.\ connected\text{-}set(Va::'a, X, Vt) \longrightarrow subset\text{-}sets(Va::'a, X))\ \& \\
\quad (\forall\ X\ Vt\ Va.\ connected\text{-}set(Va::'a, X, Vt) \longrightarrow connected\text{-}space(Va::'a, subspace\text{-}topology(X::'a, Vt, Va))) \\
\& \\
\quad (\forall\ X\ Vt\ Va.\ topological\text{-}space(X::'a, Vt)\ \&\ subset\text{-}sets(Va::'a, X)\ \&\ connected\text{-}space(Va::'a, subspace\text{-}topology(X \\
\longrightarrow connected\text{-}set(Va::'a, X, Vt))\ \& \\
\quad (\forall\ Vf\ X\ Vt.\ open\text{-}covering(Vf::'a, X, Vt) \longrightarrow topological\text{-}space(X::'a, Vt))\ \& \\
\quad (\forall\ X\ Vf\ Vt.\ open\text{-}covering(Vf::'a, X, Vt) \longrightarrow subset\text{-}collections(Vf::'a, Vt))\ \& \\
\quad (\forall\ Vt\ Vf\ X.\ open\text{-}covering(Vf::'a, X, Vt) \longrightarrow equal\text{-}sets(union\text{-}of\text{-}members(Vf), X)) \\
\& \\
\quad (\forall\ Vt\ Vf\ X.\ topological\text{-}space(X::'a, Vt)\ \&\ subset\text{-}collections(Vf::'a, Vt)\ \&\ equal\text{-}sets(union\text{-}of\text{-}members(Vf), X \\
\longrightarrow open\text{-}covering(Vf::'a, X, Vt))\ \& \\
\quad (\forall\ X\ Vt.\ compact\text{-}space(X::'a, Vt) \longrightarrow topological\text{-}space(X::'a, Vt))\ \& \\
\quad (\forall\ X\ Vt\ Vf1.\ compact\text{-}space(X::'a, Vt)\ \&\ open\text{-}covering(Vf1::'a, X, Vt) \longrightarrow fi\text{-} \\
nite'(f23(X::'a, Vt, Vf1)))\ \& \\
\quad (\forall\ X\ Vt\ Vf1.\ compact\text{-}space(X::'a, Vt)\ \&\ open\text{-}covering(Vf1::'a, X, Vt) \longrightarrow subset\text{-}collections(f23(X::'a, Vt, V \\
\& \\
\quad (\forall\ Vf1\ X\ Vt.\ compact\text{-}space(X::'a, Vt)\ \&\ open\text{-}covering(Vf1::'a, X, Vt) \longrightarrow open\text{-}covering(f23(X::'a, Vt, Vf1) \\
\& \\
\quad (\forall\ X\ Vt.\ topological\text{-}space(X::'a, Vt) \longrightarrow compact\text{-}space(X::'a, Vt) \mid open\text{-}covering(f24(X::'a, Vt), X, Vt)) \\
\& \\
\quad (\forall\ Uu24\ X\ Vt.\ topological\text{-}space(X::'a, Vt)\ \&\ finite'(Uu24)\ \&\ subset\text{-}collections(Uu24::'a, f24(X::'a, Vt)) \\
\& \ open\text{-}covering(Uu24::'a, X, Vt) \longrightarrow compact\text{-}space(X::'a, Vt))\ \& \\
\quad (\forall\ Va\ X\ Vt.\ compact\text{-}set(Va::'a, X, Vt) \longrightarrow topological\text{-}space(X::'a, Vt))\ \& \\
\quad (\forall\ Vt\ Va\ X.\ compact\text{-}set(Va::'a, X, Vt) \longrightarrow subset\text{-}sets(Va::'a, X))\ \& \\
\quad (\forall\ X\ Vt\ Va.\ compact\text{-}set(Va::'a, X, Vt) \longrightarrow compact\text{-}space(Va::'a, subspace\text{-}topology(X::'a, Vt, Va))) \\
\&
\end{array}
$$

(∀ X Vt Va. topological-space(X::'a,Vt) & subset-sets(Va::'a,X) & compact-space(Va::'a,subspace-topology(X:
−−> compact-set(Va::'a,X,Vt)) &
(basis(cx::'a,f)) &
(∀ U. element-of-collection(U::'a,top-of-basis(f))) &
(∀ V. element-of-collection(V::'a,top-of-basis(f))) &
(∀ U V. ~element-of-collection(intersection-of-sets(U::'a,V),top-of-basis(f))) −−>
False
⟨proof⟩

lemma *TOP004-2*:
(∀ U Uu1 Vf. element-of-set(U::'a,Uu1) & element-of-collection(Uu1::'a,Vf) −−>
element-of-set(U::'a,union-of-members(Vf))) &
(∀ Vf X. basis(X::'a,Vf) −−> equal-sets(union-of-members(Vf),X)) &
(∀ X Vf Y Vb1 Vb2. basis(X::'a,Vf) & element-of-set(Y::'a,X) & element-of-collection(Vb1::'a,Vf)
& element-of-collection(Vb2::'a,Vf) & element-of-set(Y::'a,intersection-of-sets(Vb1::'a,Vb2))
−−> element-of-set(Y::'a,f6(X::'a,Vf,Y,Vb1,Vb2))) &
(∀ X Y Vb1 Vb2 Vf. basis(X::'a,Vf) & element-of-set(Y::'a,X) & element-of-collection(Vb1::'a,Vf)
& element-of-collection(Vb2::'a,Vf) & element-of-set(Y::'a,intersection-of-sets(Vb1::'a,Vb2))
−−> element-of-collection(f6(X::'a,Vf,Y,Vb1,Vb2),Vf)) &
(∀ X Vf Y Vb1 Vb2. basis(X::'a,Vf) & element-of-set(Y::'a,X) & element-of-collection(Vb1::'a,Vf)
& element-of-collection(Vb2::'a,Vf) & element-of-set(Y::'a,intersection-of-sets(Vb1::'a,Vb2))
−−> subset-sets(f6(X::'a,Vf,Y,Vb1,Vb2),intersection-of-sets(Vb1::'a,Vb2))) &
(∀ Vf U X. element-of-collection(U::'a,top-of-basis(Vf)) & element-of-set(X::'a,U)
−−> element-of-set(X::'a,f10(Vf::'a,U,X))) &
(∀ U X Vf. element-of-collection(U::'a,top-of-basis(Vf)) & element-of-set(X::'a,U)
−−> element-of-collection(f10(Vf::'a,U,X),Vf)) &
(∀ Vf X U. element-of-collection(U::'a,top-of-basis(Vf)) & element-of-set(X::'a,U)
−−> subset-sets(f10(Vf::'a,U,X),U)) &
(∀ Vf U. element-of-collection(U::'a,top-of-basis(Vf)) | element-of-set(f11(Vf::'a,U),U))
&
(∀ Vf Uu11 U. element-of-set(f11(Vf::'a,U),Uu11) & element-of-collection(Uu11::'a,Vf)
& subset-sets(Uu11::'a,U) −−> element-of-collection(U::'a,top-of-basis(Vf))) &
(∀ Y X Z. subset-sets(X::'a,Y) & subset-sets(Y::'a,Z) −−> subset-sets(X::'a,Z))
&
(∀ Y Z X. element-of-set(Z::'a,intersection-of-sets(X::'a,Y)) −−> element-of-set(Z::'a,X))
&
(∀ X Z Y. element-of-set(Z::'a,intersection-of-sets(X::'a,Y)) −−> element-of-set(Z::'a,Y))
&
(∀ X Z Y. element-of-set(Z::'a,X) & element-of-set(Z::'a,Y) −−> element-of-set(Z::'a,intersection-of-sets(X
&
(∀ X U Y V. subset-sets(X::'a,Y) & subset-sets(U::'a,V) −−> subset-sets(intersection-of-sets(X::'a,U),inte
&
(∀ X Z Y. equal-sets(X::'a,Y) & element-of-set(Z::'a,X) −−> element-of-set(Z::'a,Y))
&
(∀ Y X. equal-sets(intersection-of-sets(X::'a,Y),intersection-of-sets(Y::'a,X))) &
(basis(cx::'a,f)) &
(∀ U. element-of-collection(U::'a,top-of-basis(f))) &

(∀ V. element-of-collection(V::'a,top-of-basis(f))) &
(∀ U V. ~element-of-collection(intersection-of-sets(U::'a,V),top-of-basis(f))) −−>
False
⟨proof⟩

lemma *TOP005-2*:
(∀ Vf U. element-of-set(U::'a,union-of-members(Vf)) −−> element-of-set(U::'a,f1(Vf::'a,U)))
&
(∀ U Vf. element-of-set(U::'a,union-of-members(Vf)) −−> element-of-collection(f1(Vf::'a,U),Vf))
&
(∀ Vf U X. element-of-collection(U::'a,top-of-basis(Vf)) & element-of-set(X::'a,U)
−−> element-of-set(X::'a,f10(Vf::'a,U,X))) &
(∀ U X Vf. element-of-collection(U::'a,top-of-basis(Vf)) & element-of-set(X::'a,U)
−−> element-of-collection(f10(Vf::'a,U,X),Vf)) &
(∀ Vf X U. element-of-collection(U::'a,top-of-basis(Vf)) & element-of-set(X::'a,U)
−−> subset-sets(f10(Vf::'a,U,X),U)) &
(∀ Vf U. element-of-collection(U::'a,top-of-basis(Vf)) | element-of-set(f11(Vf::'a,U),U))
&
(∀ Vf Uu11 U. element-of-set(f11(Vf::'a,U),Uu11) & element-of-collection(Uu11::'a,Vf)
& subset-sets(Uu11::'a,U) −−> element-of-collection(U::'a,top-of-basis(Vf))) &
(∀ X U Y. element-of-set(U::'a,X) −−> subset-sets(X::'a,Y) | element-of-set(U::'a,Y))
&
(∀ Y X Z. subset-sets(X::'a,Y) & element-of-collection(Y::'a,Z) −−> subset-sets(X::'a,union-of-members(Z
&
(∀ X U Y. subset-collections(X::'a,Y) & element-of-collection(U::'a,X) −−>
element-of-collection(U::'a,Y)) &
(subset-collections(g::'a,top-of-basis(f))) &
(~element-of-collection(union-of-members(g),top-of-basis(f))) −−> False
⟨proof⟩

end

38 Examples and regression tests for automated termination proofs

theory *Termination*
imports *Main Multiset*
begin

The *fun* command uses the method *lexicographic-order* by default.

38.1 Trivial examples

fun *identity* :: *nat* ⇒ *nat*
where
identity n = *n*

```
fun yaSuc :: nat ⇒ nat
where
  yaSuc 0 = 0
| yaSuc (Suc n) = Suc (yaSuc n)
```

38.2 Examples on natural numbers

```
fun bin :: (nat * nat) ⇒ nat
where
  bin (0, 0) = 1
| bin (Suc n, 0) = 0
| bin (0, Suc m) = 0
| bin (Suc n, Suc m) = bin (n, m) + bin (Suc n, m)


fun t :: (nat * nat) ⇒ nat
where
  t (0,n) = 0
| t (n,0) = 0
| t (Suc n, Suc m) = (if (n mod 2 = 0) then (t (Suc n, m)) else (t (n, Suc m)))


fun k :: (nat * nat) * (nat * nat) ⇒ nat
where
  k ((0,0),(0,0)) = 0
| k ((Suc z, y), (u,v)) = k((z, y), (u, v))
| k ((0, Suc y), (u,v)) = k((1, y), (u, v))
| k ((0,0), (Suc u, v)) = k((1, 1), (u, v))
| k ((0,0), (0, Suc v)) = k((1,1), (1,v))


fun gcd2 :: nat ⇒ nat ⇒ nat
where
  gcd2 x 0 = x
| gcd2 0 y = y
| gcd2 (Suc x) (Suc y) = (if x < y then gcd2 (Suc x) (y − x)
                                   else gcd2 (x − y) (Suc y))

fun ack :: (nat * nat) ⇒ nat
where
  ack (0, m) = Suc m
| ack (Suc n, 0) = ack(n, 1)
| ack (Suc n, Suc m) = ack (n, ack (Suc n, m))


fun greedy :: nat * nat * nat * nat * nat => nat
where
  greedy (Suc a, Suc b, Suc c, Suc d, Suc e) =
```

```
  (if (a < 10) then greedy (Suc a, Suc b, c, d + 2, Suc e) else
  (if (a < 20) then greedy (Suc a, b, Suc c, d, Suc e) else
  (if (a < 30) then greedy (Suc a, b, Suc c, d, Suc e) else
  (if (a < 40) then greedy (Suc a, b, Suc c, d, Suc e) else
  (if (a < 50) then greedy (Suc a, b, Suc c, d, Suc e) else
  (if (a < 60) then greedy (a, Suc b, Suc c, d, Suc e) else
  (if (a < 70) then greedy (a, Suc b, Suc c, d, Suc e) else
  (if (a < 80) then greedy (a, Suc b, Suc c, d, Suc e) else
  (if (a < 90) then greedy (Suc a, Suc b, Suc c, d, e) else
  greedy (Suc a, Suc b, Suc c, d, e)))))))))
| greedy (a, b, c, d, e) = 0


fun blowup :: nat => nat => nat => nat => nat => nat => nat => nat =>
nat => nat
where
  blowup 0 0 0 0 0 0 0 0 0 = 0
| blowup 0 0 0 0 0 0 0 0 (Suc i) = Suc (blowup i i i i i i i i i)
| blowup 0 0 0 0 0 0 0 (Suc h) i = Suc (blowup h h h h h h h h i)
| blowup 0 0 0 0 0 0 (Suc g) h i = Suc (blowup g g g g g g g h i)
| blowup 0 0 0 0 0 (Suc f) g h i = Suc (blowup f f f f f f g h i)
| blowup 0 0 0 0 (Suc e) f g h i = Suc (blowup e e e e e f g h i)
| blowup 0 0 0 (Suc d) e f g h i = Suc (blowup d d d d e f g h i)
| blowup 0 0 (Suc c) d e f g h i = Suc (blowup c c c d e f g h i)
| blowup 0 (Suc b) c d e f g h i = Suc (blowup b b c d e f g h i)
| blowup (Suc a) b c d e f g h i = Suc (blowup a b c d e f g h i)
```

38.3 Simple examples with other datatypes than nat, e.g. trees and lists

```
datatype tree = Node | Branch tree tree

fun g-tree :: tree * tree ⇒ tree
where
  g-tree (Node, Node) = Node
| g-tree (Node, Branch a b) = Branch Node (g-tree (a,b))
| g-tree (Branch a b, Node) = Branch (g-tree (a,Node)) b
| g-tree (Branch a b, Branch c d) = Branch (g-tree (a,c)) (g-tree (b,d))


fun acklist :: 'a list * 'a list ⇒ 'a list
where
  acklist ([], m) = ((hd m)#m)
|  acklist (n#ns, []) = acklist (ns, [n])
|  acklist ((n#ns), (m#ms)) = acklist (ns, acklist ((n#ns), ms))
```

38.4 Examples with mutual recursion

```
fun evn od :: nat ⇒ bool
```

```
where
  evn 0 = True
| od 0 = False
| evn (Suc n) = od (Suc n)
| od (Suc n) = evn n


fun sizechange-f :: 'a list => 'a list => 'a list and
sizechange-g :: 'a list => 'a list => 'a list => 'a list
where
  sizechange-f i x = (if i=[] then x else sizechange-g (tl i) x i)
| sizechange-g a b c = sizechange-f a (b @ c)

fun
  pedal :: nat => nat => nat => nat
and
  coast :: nat => nat => nat => nat
where
  pedal 0 m c = c
| pedal n 0 c = c
| pedal n m c =
    (if n < m then coast (n - 1) (m - 1) (c + m)
              else pedal (n - 1) m (c + m))

| coast n m c =
    (if n < m then coast n (m - 1) (c + n)
              else pedal n m (c + n))
```

38.5 Refined analysis: The *sizechange* method

Unsolvable for *lexicographic-order*

```
function fun1 :: nat * nat ⇒ nat
where
  fun1 (0,0) = 1
| fun1 (0, Suc b) = 0
| fun1 (Suc a, 0) = 0
| fun1 (Suc a, Suc b) = fun1 (b, a)
⟨proof⟩
termination ⟨proof⟩
```

lexicographic-order can do the following, but it is much slower.

```
function
  prod :: nat => nat => nat => nat and
  eprod :: nat => nat => nat => nat and
  oprod :: nat => nat => nat => nat
where
  prod x y z = (if y mod 2 = 0 then eprod x y z else oprod x y z)
| oprod x y z = eprod x (y - 1) (z+x)
| eprod x y z = (if y=0 then z else prod (2*x) (y div 2) z)
```

⟨*proof*⟩
termination ⟨*proof*⟩

Permutations of arguments:

function *perm* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat*
where
perm m n r = (*if r* > *0 then perm m* (*r* − *1*) *n*
else if n > *0 then perm r* (*n* − *1*) *m*
else m)
⟨*proof*⟩
termination ⟨*proof*⟩

Artificial examples and regression tests:

function
fun2 :: *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat*
where
fun2 x y z =
(*if x* > *1000* ∧ *z* > *0 then*
fun2 (*min x y*) *y* (*z* − *1*)
else if y > *0* ∧ *x* > *100 then*
fun2 x (*y* − *1*) (*2* ∗ *z*)
else if z > *0 then*
fun2 (*min y* (*z* − *1*)) *x x*
else
0
)
⟨*proof*⟩
termination ⟨*proof*⟩

end

39 Coherent Logic Problems

theory *Coherent* **imports** *Main* **begin**

39.1 Equivalence of two versions of Pappus' Axiom

no-notation
comp (**infixl** *o 55*) **and**
rel-comp (**infixr** *O 75*)

lemma *p1p2*:
assumes
col a b c l ∧ *col d e f m*
col b f g n ∧ *col c e g o*
col b d h p ∧ *col a e h q*
col c d i r ∧ *col a f i s*

$el\ n\ o \Longrightarrow goal$
$el\ p\ q \Longrightarrow goal$
$el\ s\ r \Longrightarrow goal$
$\bigwedge A.\ el\ A\ A \Longrightarrow pl\ g\ A \Longrightarrow pl\ h\ A \Longrightarrow pl\ i\ A \Longrightarrow goal$
$\bigwedge A\ B\ C\ D.\ col\ A\ B\ C\ D \Longrightarrow pl\ A\ D$
$\bigwedge A\ B\ C\ D.\ col\ A\ B\ C\ D \Longrightarrow pl\ B\ D$
$\bigwedge A\ B\ C\ D.\ col\ A\ B\ C\ D \Longrightarrow pl\ C\ D$
$\bigwedge A\ B.\ pl\ A\ B \Longrightarrow ep\ A\ A$
$\bigwedge A\ B.\ ep\ A\ B \Longrightarrow ep\ B\ A$
$\bigwedge A\ B\ C.\ ep\ A\ B \Longrightarrow ep\ B\ C \Longrightarrow ep\ A\ C$
$\bigwedge A\ B.\ pl\ A\ B \Longrightarrow el\ B\ B$
$\bigwedge A\ B.\ el\ A\ B \Longrightarrow el\ B\ A$
$\bigwedge A\ B\ C.\ el\ A\ B \Longrightarrow el\ B\ C \Longrightarrow el\ A\ C$
$\bigwedge A\ B\ C.\ ep\ A\ B \Longrightarrow pl\ B\ C \Longrightarrow pl\ A\ C$
$\bigwedge A\ B\ C.\ pl\ A\ B \Longrightarrow el\ B\ C \Longrightarrow pl\ A\ C$
$\bigwedge A\ B\ C\ D\ E\ F\ G\ H\ I\ J\ K\ L\ M\ N\ O\ P\ Q.$
$col\ A\ B\ C\ D \Longrightarrow col\ E\ F\ G\ H \Longrightarrow col\ B\ G\ I\ J \Longrightarrow col\ C\ F\ I\ K \Longrightarrow$
$col\ B\ E\ L\ M \Longrightarrow col\ A\ F\ L\ N \Longrightarrow col\ C\ E\ O\ P \Longrightarrow col\ A\ G\ O\ Q \Longrightarrow$
$(\exists\ R.\ col\ I\ L\ O\ R) \vee pl\ A\ H \vee pl\ B\ H \vee pl\ C\ H \vee pl\ E\ D \vee pl\ F\ D \vee pl\ G\ D$
$\bigwedge A\ B\ C\ D.\ pl\ A\ B \Longrightarrow pl\ A\ C \Longrightarrow pl\ D\ B \Longrightarrow pl\ D\ C \Longrightarrow ep\ A\ D \vee el\ B\ C$
$\bigwedge A\ B.\ ep\ A\ A \Longrightarrow ep\ B\ B \Longrightarrow \exists C.\ pl\ A\ C \wedge pl\ B\ C$
shows *goal* ⟨*proof*⟩

lemma *p2p1*:
assumes
$col\ a\ b\ c\ l \wedge col\ d\ e\ f\ m$
$col\ b\ f\ g\ n \wedge col\ c\ e\ g\ o$
$col\ b\ d\ h\ p \wedge col\ a\ e\ h\ q$
$col\ c\ d\ i\ r \wedge col\ a\ f\ i\ s$
$pl\ a\ m \Longrightarrow goal$
$pl\ b\ m \Longrightarrow goal$
$pl\ c\ m \Longrightarrow goal$
$pl\ d\ l \Longrightarrow goal$
$pl\ e\ l \Longrightarrow goal$
$pl\ f\ l \Longrightarrow goal$
$\bigwedge A.\ pl\ g\ A \Longrightarrow pl\ h\ A \Longrightarrow pl\ i\ A \Longrightarrow goal$
$\bigwedge A\ B\ C\ D.\ col\ A\ B\ C\ D \Longrightarrow pl\ A\ D$
$\bigwedge A\ B\ C\ D.\ col\ A\ B\ C\ D \Longrightarrow pl\ B\ D$
$\bigwedge A\ B\ C\ D.\ col\ A\ B\ C\ D \Longrightarrow pl\ C\ D$
$\bigwedge A\ B.\ pl\ A\ B \Longrightarrow ep\ A\ A$
$\bigwedge A\ B.\ ep\ A\ B \Longrightarrow ep\ B\ A$
$\bigwedge A\ B\ C.\ ep\ A\ B \Longrightarrow ep\ B\ C \Longrightarrow ep\ A\ C$
$\bigwedge A\ B.\ pl\ A\ B \Longrightarrow el\ B\ B$
$\bigwedge A\ B.\ el\ A\ B \Longrightarrow el\ B\ A$
$\bigwedge A\ B\ C.\ el\ A\ B \Longrightarrow el\ B\ C \Longrightarrow el\ A\ C$
$\bigwedge A\ B\ C.\ ep\ A\ B \Longrightarrow pl\ B\ C \Longrightarrow pl\ A\ C$
$\bigwedge A\ B\ C.\ pl\ A\ B \Longrightarrow el\ B\ C \Longrightarrow pl\ A\ C$
$\bigwedge A\ B\ C\ D\ E\ F\ G\ H\ I\ J\ K\ L\ M\ N\ O\ P\ Q.$
$col\ A\ B\ C\ J \Longrightarrow col\ D\ E\ F\ K \Longrightarrow col\ B\ F\ G\ L \Longrightarrow col\ C\ E\ G\ M \Longrightarrow$

col B D H N $\Longrightarrow$ *col A E H O* $\Longrightarrow$ *col C D I P* $\Longrightarrow$ *col A F I Q* $\Longrightarrow$
(∃ *R*. *col G H I R*) ∨ *el L M* ∨ *el N O* ∨ *el P Q*
$\bigwedge$*A B C D*. *pl C A* $\Longrightarrow$ *pl C B* $\Longrightarrow$ *pl D A* $\Longrightarrow$ *pl D B* $\Longrightarrow$ *ep C D* ∨ *el A B*
$\bigwedge$*A B C*. *ep A A* $\Longrightarrow$ *ep B B* $\Longrightarrow$ ∃ *C*. *pl A C* ∧ *pl B C*
shows *goal* ⟨*proof*⟩

39.2 Preservation of the Diamond Property under reflexive closure

lemma *diamond*:
assumes
reflexive-rewrite a b reflexive-rewrite a c
$\bigwedge$*A*. *reflexive-rewrite b A* $\Longrightarrow$ *reflexive-rewrite c A* $\Longrightarrow$ *goal*
$\bigwedge$*A*. *equalish A A*
$\bigwedge$*A B*. *equalish A B* $\Longrightarrow$ *equalish B A*
$\bigwedge$*A B C*. *equalish A B* $\Longrightarrow$ *reflexive-rewrite B C* $\Longrightarrow$ *reflexive-rewrite A C*
$\bigwedge$*A B*. *equalish A B* $\Longrightarrow$ *reflexive-rewrite A B*
$\bigwedge$*A B*. *rewrite A B* $\Longrightarrow$ *reflexive-rewrite A B*
$\bigwedge$*A B*. *reflexive-rewrite A B* $\Longrightarrow$ *equalish A B* ∨ *rewrite A B*
$\bigwedge$*A B C*. *rewrite A B* $\Longrightarrow$ *rewrite A C* $\Longrightarrow$ ∃ *D*. *rewrite B D* ∧ *rewrite C D*
shows *goal* ⟨*proof*⟩

end

40 Some examples for Presburger Arithmetic

theory *PresburgerEx*
imports *Presburger*
begin

lemma $\bigwedge$*m n ja ia*. ⟦¬ *m* ≤ *j*; ¬ (*n*::*nat*) ≤ *i*; (*e*::*nat*) ≠ *0*; *Suc j* ≤ *ja*⟧ $\Longrightarrow$ ∃ *m*. ∀ *ja ia*. *m* ≤ *ja* ⟶ (*if j* = *ja* ∧ *i* = *ia then e else 0*) = *0* ⟨*proof*⟩
lemma (*0*::*nat*) < *emBits mod 8* $\Longrightarrow$ *8* + *emBits div 8* ∗ *8* − *emBits* = *8* − *emBits mod 8*
⟨*proof*⟩
lemma (*0*::*nat*) < *emBits mod 8* $\Longrightarrow$ *8* + *emBits div 8* ∗ *8* − *emBits* = *8* − *emBits mod 8*
⟨*proof*⟩

theorem (∀(*y*::*int*). *3 dvd y*) ==> ∀(*x*::*int*). *b* < *x* −−> *a* ≤ *x*
⟨*proof*⟩

theorem !! (*y*::*int*) (*z*::*int*) (*n*::*int*). *3 dvd z* ==> *2 dvd* (*y*::*int*) ==>
(∃(*x*::*int*). *2*∗*x* = *y*) & (∃(*k*::*int*). *3*∗*k* = *z*)
⟨*proof*⟩

theorem !! (*y*::*int*) (*z*::*int*) *n*. *Suc*(*n*::*nat*) < *6* ==> *3 dvd z* ==>
2 dvd (*y*::*int*) ==> (∃(*x*::*int*). *2*∗*x* = *y*) & (∃(*k*::*int*). *3*∗*k* = *z*)

⟨*proof*⟩

theorem $\forall (x{::}nat).\ \exists (y{::}nat).\ (0{::}nat) \leq 5 \longrightarrow y = 5 + x$
⟨*proof*⟩

Slow: about 7 seconds on a 1.6GHz machine.

theorem $\forall (x{::}nat).\ \exists (y{::}nat).\ y = 5 + x \mid x\ div\ 6 + 1 = 2$
⟨*proof*⟩

theorem $\exists (x{::}int).\ 0 < x$
⟨*proof*⟩

theorem $\forall (x{::}int)\ y.\ x < y \longrightarrow 2 * x + 1 < 2 * y$
⟨*proof*⟩

theorem $\forall (x{::}int)\ y.\ 2 * x + 1 \neq 2 * y$
⟨*proof*⟩

theorem $\exists (x{::}int)\ y.\ 0 < x\ \&\ 0 \leq y\ \&\ 3 * x - 5 * y = 1$
⟨*proof*⟩

theorem $\sim (\exists (x{::}int)\ (y{::}int)\ (z{::}int).\ 4{*}x + (-6{::}int){*}y = 1)$
⟨*proof*⟩

theorem $\forall (x{::}int).\ b < x \longrightarrow a \leq x$
⟨*proof*⟩

theorem $\sim (\exists (x{::}int).\ False)$
⟨*proof*⟩

theorem $\forall (x{::}int).\ (a{::}int) < 3 * x \longrightarrow b < 3 * x$
⟨*proof*⟩

theorem $\forall (x{::}int).\ (2\ dvd\ x) \longrightarrow (\exists (y{::}int).\ x = 2{*}y)$
⟨*proof*⟩

theorem $\forall (x{::}int).\ (2\ dvd\ x) \longrightarrow (\exists (y{::}int).\ x = 2{*}y)$
⟨*proof*⟩

theorem $\forall (x{::}int).\ (2\ dvd\ x) = (\exists (y{::}int).\ x = 2{*}y)$
⟨*proof*⟩

theorem $\forall (x{::}int).\ ((2\ dvd\ x) = (\forall (y{::}int).\ x \neq 2{*}y + 1))$
⟨*proof*⟩

theorem $\sim (\forall (x{::}int).$
$((2\ dvd\ x) = (\forall (y{::}int).\ x \neq 2{*}y{+}1) \mid$
$(\exists (q{::}int)\ (u{::}int)\ i.\ 3{*}i + 2{*}q - u < 17)$
$\longrightarrow 0 < x \mid ((\sim\ 3\ dvd\ x)\ \&(x + 8 = 0))))$

⟨*proof*⟩

theorem ~ (∀(*i*::*int*). *4* ≤ *i* −−> (∃ *x y*. *0* ≤ *x* & *0* ≤ *y* & *3* ∗ *x* + *5* ∗ *y* = *i*))
⟨*proof*⟩

theorem ∀(*i*::*int*). *8* ≤ *i* −−> (∃ *x y*. *0* ≤ *x* & *0* ≤ *y* & *3* ∗ *x* + *5* ∗ *y* = *i*)
⟨*proof*⟩

theorem ∃(*j*::*int*). ∀ *i*. *j* ≤ *i* −−> (∃ *x y*. *0* ≤ *x* & *0* ≤ *y* & *3* ∗ *x* + *5* ∗ *y* = *i*)
⟨*proof*⟩

theorem ~ (∀ *j* (*i*::*int*). *j* ≤ *i* −−> (∃ *x y*. *0* ≤ *x* & *0* ≤ *y* & *3* ∗ *x* + *5* ∗ *y* = *i*))
⟨*proof*⟩

Slow: about 5 seconds on a 1.6GHz machine.

theorem (∃ *m*::*nat*. *n* = *2* ∗ *m*) −−> (*n* + *1*) *div* *2* = *n div* *2*
⟨*proof*⟩

This following theorem proves that all solutions to the recurrence relation $x_{i+2} = |x_{i+1}| - x_i$ are periodic with period 9. The example was brought to our attention by John Harrison. It does does not require Presburger arithmetic but merely quantifier-free linear arithmetic and holds for the rationals as well.

Warning: it takes (in 2006) over 4.2 minutes!

lemma ⟦ *x3* = *abs x2* − *x1*; *x4* = *abs x3* − *x2*; *x5* = *abs x4* − *x3*;
x6 = *abs x5* − *x4*; *x7* = *abs x6* − *x5*; *x8* = *abs x7* − *x6*;
x9 = *abs x8* − *x7*; *x10* = *abs x9* − *x8*; *x11* = *abs x10* − *x9* ⟧
⟹ *x1* = *x10* & *x2* = (*x11*::*int*)
⟨*proof*⟩

end

41 Generic reflection and reification

theory *Reflection*
imports *Main*
uses *reify-data.ML* (*reflection.ML*)
begin

⟨*ML*⟩

lemma *ext2*: (∀ *x*. *f x* = *g x*) ⟹ *f* = *g*
⟨*proof*⟩

⟨*ML*⟩

end

42 Examples for generic reflection and reification

theory *ReflectionEx*
imports *Reflection*
begin

This theory presents two methods: reify and reflection

Consider an HOL type 'a, the structure of which is not recongnisable on the theory level. This is the case of bool, arithmetical terms such as int, real etc ... In order to implement a simplification on terms of type 'a we often need its structure. Traditionnaly such simplifications are written in ML, proofs are synthesized. An other strategy is to declare an HOL-datatype tau and an HOL function (the interpretation) that maps elements of tau to elements of 'a. The functionality of *reify* is to compute a term s::tau, which is the representant of t. For this it needs equations for the interpretation.

NB: All the interpretations supported by *reify* must have the type $'b\ list \Rightarrow tau \Rightarrow 'a$. The method *reify* can also be told which subterm of the current subgoal should be reified. The general call for *reify* is: *reify eqs* (t), where *eqs* are the defining equations of the interpretation and (t) is an optional parameter which specifies the subterm to which reification should be applied to. If (t) is abscent, *reify* tries to reify the whole subgoal.

The method reflection uses *reify* and has a very similar signature: *reflection corr-thm eqs* (t). Here again *eqs* and (t) are as described above and *corr-thm* is a thorem proving $I\ vs\ (f\ t) = I\ vs\ t$. We assume that I is the interpretation and f is some useful and executable simplification of type $tau \Rightarrow tau$. The method *reflection* applies reification and hence the theorem $t = I\ xs\ s$ and hence using *corr-thm* derives $t = I\ xs\ (f\ s)$. It then uses normalization by evaluation to prove $f\ s = s'$ which almost finishes the proof of $t = t'$ where $I\ xs\ s' = t'$.

Example 1 : Propositional formulae and NNF.

The type *fm* represents simple propositional formulae:

datatype *form* = *TrueF* | *FalseF* | *Less nat nat* |
And form form | *Or form form* | *Neg form* | *ExQ form*

fun *interp* :: *form* ⇒ ('*a*::*ord*) *list* ⇒ *bool* **where**
interp TrueF e = *True* |
interp FalseF e = *False* |
interp (*Less i j*) *e* = (*e*!*i* < *e*!*j*) |
interp (*And f1 f2*) *e* = (*interp f1 e* & *interp f2 e*) |
interp (*Or f1 f2*) *e* = (*interp f1 e* | *interp f2 e*) |

interp (*Neg f*) *e* = (~ *interp f e*) |
interp (*ExQ f*) *e* = (*EX x*. *interp f* (*x*#*e*))

lemmas *interp-reify-eqs* = *interp.simps*
declare *interp-reify-eqs*[*reify*]

lemma *EX x*. *x* < *y* & *x* < *z*
⟨*proof*⟩

datatype *fm* = *And fm fm* | *Or fm fm* | *Imp fm fm* | *Iff fm fm* | *NOT fm* | *At nat*

consts *Ifm* :: *fm* ⇒ *bool list* ⇒ *bool*
primrec
Ifm (*At n*) *vs* = *vs*!*n*
Ifm (*And p q*) *vs* = (*Ifm p vs* ∧ *Ifm q vs*)
Ifm (*Or p q*) *vs* = (*Ifm p vs* ∨ *Ifm q vs*)
Ifm (*Imp p q*) *vs* = (*Ifm p vs* ⟶ *Ifm q vs*)
Ifm (*Iff p q*) *vs* = (*Ifm p vs* = *Ifm q vs*)
Ifm (*NOT p*) *vs* = (¬ (*Ifm p vs*))

lemma *Q* ⟶ (*D* & *F* & ((~ *D*) & (~ *F*)))
⟨*proof*⟩

Method *reify* maps a bool to an fm. For this it needs the semantics of fm, i.e. the rewrite rules in *Ifm.simps*.

lemma *Q* ⟶ (*D* & *F* & ((~ *D*) & (~ *F*)))
⟨*proof*⟩

Let's perform NNF. This is a version that tends to generate disjunctions

consts *fmsize* :: *fm* ⇒ *nat*
primrec
fmsize (*At n*) = *1*
fmsize (*NOT p*) = *1* + *fmsize p*
fmsize (*And p q*) = *1* + *fmsize p* + *fmsize q*
fmsize (*Or p q*) = *1* + *fmsize p* + *fmsize q*
fmsize (*Imp p q*) = *2* + *fmsize p* + *fmsize q*
fmsize (*Iff p q*) = *2* + *2*∗ *fmsize p* + *2*∗ *fmsize q*

consts *nnf* :: *fm* ⇒ *fm*
recdef *nnf measure fmsize*
nnf (*At n*) = *At n*
nnf (*And p q*) = *And* (*nnf p*) (*nnf q*)
nnf (*Or p q*) = *Or* (*nnf p*) (*nnf q*)
nnf (*Imp p q*) = *Or* (*nnf* (*NOT p*)) (*nnf q*)
nnf (*Iff p q*) = *Or* (*And* (*nnf p*) (*nnf q*)) (*And* (*nnf* (*NOT p*)) (*nnf* (*NOT q*)))
nnf (*NOT* (*And p q*)) = *Or* (*nnf* (*NOT p*)) (*nnf* (*NOT q*))
nnf (*NOT* (*Or p q*)) = *And* (*nnf* (*NOT p*)) (*nnf* (*NOT q*))
nnf (*NOT* (*Imp p q*)) = *And* (*nnf p*) (*nnf* (*NOT q*))

nnf (NOT (Iff p q)) = Or (And (nnf p) (nnf (NOT q))) (And (nnf (NOT p)) (nnf q))
nnf (NOT (NOT p)) = nnf p
nnf (NOT p) = NOT p

The correctness theorem of nnf: it preserves the semantics of fm

lemma *nnf*[*reflection*]: *Ifm (nnf p) vs = Ifm p vs*
⟨*proof*⟩

Now let's perform NNF using our *nnf* function defined above. First to the whole subgoal.

lemma $(\neg (A = B)) \wedge (B \longrightarrow (A \neq (B \mid C \wedge (B \longrightarrow A \mid D)))) \longrightarrow A \vee B \wedge D$
⟨*proof*⟩

Now we specify on which subterm it should be applied

lemma $(\neg (A = B)) \wedge (B \longrightarrow (A \neq (B \mid C \wedge (B \longrightarrow A \mid D)))) \longrightarrow A \vee B \wedge D$
⟨*proof*⟩

The type *num* reflects linear expressions over natural number

datatype *num = C nat | Add num num | Mul nat num | Var nat | CN nat nat num*

This is just technical to make recursive definitions easier.

consts *num-size* :: *num* ⇒ *nat*
primrec
num-size (C c) = 1
num-size (Var n) = 1
num-size (Add a b) = 1 + num-size a + num-size b
num-size (Mul c a) = 1 + num-size a
num-size (CN n c a) = 4 + num-size a

The semantics of num

consts *Inum*:: *num* ⇒ *nat list* ⇒ *nat*
primrec
Inum-C : *Inum (C i) vs = i*
Inum-Var: *Inum (Var n) vs = vs!n*
Inum-Add: *Inum (Add s t) vs = Inum s vs + Inum t vs*
Inum-Mul: *Inum (Mul c t) vs = c * Inum t vs*
Inum-CN : *Inum (CN n c t) vs = c*(vs!n) + Inum t vs*

Let's reify some nat expressions ...

lemma *4 * (2*x + (y::nat)) + f a ≠ 0*
⟨*proof*⟩

We're in a bad situation!! x, y and f a have been recongnized as a constants, which is correct but does not correspond to our intuition of the constructor C. It should encapsulate constants, i.e. numbers, i.e. numerals.

So let's leave the *Inum-C* equation at the end and see what happens ...

lemma *4 * (2*x + (y::nat)) ≠ 0*
⟨*proof*⟩

Hmmm let's specialize *Inum-C* with numerals.

lemma *Inum-number*: *Inum (C (number-of t)) vs = number-of t* ⟨*proof*⟩
lemmas *Inum-eqs = Inum-Var Inum-Add Inum-Mul Inum-CN Inum-number*

Second attempt

lemma *1 * (2*x + (y::nat)) ≠ 0*
⟨*proof*⟩

That was fine, so let's try another one ...

lemma *1 * (2* x + (y::nat) + 0 + 1) ≠ 0*
⟨*proof*⟩

Oh!! 0 is not a variable ... Oh! 0 is not a *number-of* ... thing. The same for 1. So let's add those equations too

lemma *Inum-01*: *Inum (C 0) vs = 0 Inum (C 1) vs = 1 Inum (C(Suc n)) vs = Suc n*
⟨*proof*⟩

lemmas *Inum-eqs'= Inum-eqs Inum-01*

Third attempt:

lemma *1 * (2*x + (y::nat) + 0 + 1) ≠ 0*
⟨*proof*⟩

Okay, let's try reflection. Some simplifications on num follow. You can skim until the main theorem *linum*

consts *lin-add* :: *num × num ⇒ num*
recdef *lin-add measure* (λ(*x*,*y*). ((*size x*) + (*size y*)))
lin-add (CN n1 c1 r1,CN n2 c2 r2) =
(if n1=n2 then
(let c = c1 + c2
in (if c=0 then lin-add(r1,r2) else CN n1 c (lin-add (r1,r2))))
else if n1 ≤ n2 then (CN n1 c1 (lin-add (r1,CN n2 c2 r2)))
else (CN n2 c2 (lin-add (CN n1 c1 r1,r2))))
lin-add (CN n1 c1 r1,t) = CN n1 c1 (lin-add (r1, t))
lin-add (t,CN n2 c2 r2) = CN n2 c2 (lin-add (t,r2))
lin-add (C b1, C b2) = C (b1+b2)
lin-add (a,b) = Add a b
lemma *lin-add*: *Inum (lin-add (t,s)) bs = Inum (Add t s) bs*
⟨*proof*⟩

consts *lin-mul* :: *num ⇒ nat ⇒ num*
recdef *lin-mul measure size*

lin-mul (*C j*) = (λ *i*. *C* (*i*∗*j*))
lin-mul (*CN n c a*) = (λ *i*. *if i=0 then* (*C 0*) *else CN n* (*i*∗*c*) (*lin-mul a i*))
lin-mul t = (λ *i*. *Mul i t*)

lemma *lin-mul*: *Inum* (*lin-mul t i*) *bs* = *Inum* (*Mul i t*) *bs*
⟨*proof*⟩

consts *linum*:: *num* ⇒ *num*
recdef *linum measure num-size*
linum (*C b*) = *C b*
linum (*Var n*) = *CN n 1* (*C 0*)
linum (*Add t s*) = *lin-add* (*linum t*, *linum s*)
linum (*Mul c t*) = *lin-mul* (*linum t*) *c*
linum (*CN n c t*) = *lin-add* (*linum* (*Mul c* (*Var n*)),*linum t*)

lemma *linum*[*reflection*] : *Inum* (*linum t*) *bs* = *Inum t bs*
⟨*proof*⟩

Now we can use linum to simplify nat terms using reflection

lemma (*Suc* (*Suc 1*)) ∗ (*x* + (*Suc 1*)∗*y*) = *3*∗*x* + *6*∗*y*
⟨*proof*⟩

Let's lift this to formulae and see what happens

datatype *aform* = *Lt num num* | *Eq num num* | *Ge num num* | *NEq num num*
|
Conj aform aform | *Disj aform aform* | *NEG aform* | *T* | *F*
consts *linaformsize*:: *aform* ⇒ *nat*
recdef *linaformsize measure size*
linaformsize T = *1*
linaformsize F = *1*
linaformsize (*Lt a b*) = *1*
linaformsize (*Ge a b*) = *1*
linaformsize (*Eq a b*) = *1*
linaformsize (*NEq a b*) = *1*
linaformsize (*NEG p*) = *2* + *linaformsize p*
linaformsize (*Conj p q*) = *1* + *linaformsize p* + *linaformsize q*
linaformsize (*Disj p q*) = *1* + *linaformsize p* + *linaformsize q*

consts *is-aform* :: *aform* => *nat list* => *bool*
primrec
is-aform T vs = *True*
is-aform F vs = *False*
is-aform (*Lt a b*) *vs* = (*Inum a vs* < *Inum b vs*)
is-aform (*Eq a b*) *vs* = (*Inum a vs* = *Inum b vs*)
is-aform (*Ge a b*) *vs* = (*Inum a vs* ≥ *Inum b vs*)
is-aform (*NEq a b*) *vs* = (*Inum a vs* ≠ *Inum b vs*)
is-aform (*NEG p*) *vs* = (¬ (*is-aform p vs*))
is-aform (*Conj p q*) *vs* = (*is-aform p vs* ∧ *is-aform q vs*)

is-aform (Disj p q) vs = (is-aform p vs ∨ is-aform q vs)

Let's reify and do reflection

lemma *(3::nat)*x + t < 0 ∧ (2 * x + y ≠ 17)*
⟨*proof*⟩

Note that reification handles several interpretations at the same time

lemma *(3::nat)*x + t < 0 & x*x + t*x + 3 + 1 = z*t*4*z | x + x + 1 < 0*
⟨*proof*⟩

For reflection we now define a simple transformation on aform: NNF + linum on atoms

consts *linaform*:: *aform ⇒ aform*
recdef *linaform measure linaformsize*
linaform (Lt s t) = Lt (linum s) (linum t)
linaform (Eq s t) = Eq (linum s) (linum t)
linaform (Ge s t) = Ge (linum s) (linum t)
linaform (NEq s t) = NEq (linum s) (linum t)
linaform (Conj p q) = Conj (linaform p) (linaform q)
linaform (Disj p q) = Disj (linaform p) (linaform q)
linaform (NEG T) = F
linaform (NEG F) = T
linaform (NEG (Lt a b)) = Ge a b
linaform (NEG (Ge a b)) = Lt a b
linaform (NEG (Eq a b)) = NEq a b
linaform (NEG (NEq a b)) = Eq a b
linaform (NEG (NEG p)) = linaform p
linaform (NEG (Conj p q)) = Disj (linaform (NEG p)) (linaform (NEG q))
linaform (NEG (Disj p q)) = Conj (linaform (NEG p)) (linaform (NEG q))
linaform p = p

lemma *linaform*: *is-aform (linaform p) vs = is-aform p vs*
⟨*proof*⟩

lemma *(((Suc(Suc (Suc 0)))*((x::nat) + (Suc (Suc 0)))) + (Suc (Suc (Suc 0)))*
** ((Suc(Suc (Suc 0)))*((x::nat) + (Suc (Suc 0))))< 0) ∧ (Suc 0 + Suc 0< 0)*
⟨*proof*⟩

declare *linaform*[*reflection*]
lemma *(((Suc(Suc (Suc 0)))*((x::nat) + (Suc (Suc 0)))) + (Suc (Suc (Suc 0)))*
** ((Suc(Suc (Suc 0)))*((x::nat) + (Suc (Suc 0))))< 0) ∧ (Suc 0 + Suc 0< 0)*
⟨*proof*⟩

We now give an example where Interpretaions have 0 or more than only one envornement of different types and show that automatic reification also deals with binding

datatype *rb = BC bool| BAnd rb rb | BOr rb rb*
consts *Irb :: rb ⇒ bool*

primrec
Irb (*BC p*) = *p*
Irb (*BAnd s t*) = (*Irb s* ∧ *Irb t*)
Irb (*BOr s t*) = (*Irb s* ∨ *Irb t*)

lemma *A* ∧ (*B* ∨ *D* ∧ *B*) ∧ *A* ∧ (*B* ∨ *D* ∧ *B*) ∨ *A* ∧ (*B* ∨ *D* ∧ *B*) ∨ *A* ∧ (*B* ∨ *D* ∧ *B*)
⟨*proof*⟩

datatype *rint* = *IC int*| *IVar nat* | *IAdd rint rint* | *IMult rint rint* | *INeg rint* | *ISub rint rint*
consts *Irint* :: *rint* ⇒ *int list* ⇒ *int*
primrec
Irint-Var: *Irint* (*IVar n*) *vs* = *vs*!*n*
Irint-Neg: *Irint* (*INeg t*) *vs* = − *Irint t vs*
Irint-Add: *Irint* (*IAdd s t*) *vs* = *Irint s vs* + *Irint t vs*
Irint-Sub: *Irint* (*ISub s t*) *vs* = *Irint s vs* − *Irint t vs*
Irint-Mult: *Irint* (*IMult s t*) *vs* = *Irint s vs* ∗ *Irint t vs*
Irint-C: *Irint* (*IC i*) *vs* = *i*
lemma *Irint-C0*: *Irint* (*IC 0*) *vs* = *0*
⟨*proof*⟩
lemma *Irint-C1*: *Irint* (*IC 1*) *vs* = *1*
⟨*proof*⟩
lemma *Irint-Cnumberof*: *Irint* (*IC* (*number-of x*)) *vs* = *number-of x*
⟨*proof*⟩
lemmas *Irint-simps* = *Irint-Var Irint-Neg Irint-Add Irint-Sub Irint-Mult Irint-C0 Irint-C1 Irint-Cnumberof*
lemma (*3*::*int*) ∗ *x* + *y*∗*y* − *9* + (− *z*) = *0*
⟨*proof*⟩
datatype *rlist* = *LVar nat*| *LEmpty*| *LCons rint rlist* | *LAppend rlist rlist*
consts *Irlist* :: *rlist* ⇒ *int list* ⇒ (*int list*) *list* ⇒ (*int list*)
primrec
Irlist (*LEmpty*) *is vs* = []
Irlist (*LVar n*) *is vs* = *vs*!*n*
Irlist (*LCons i t*) *is vs* = ((*Irint i is*)#(*Irlist t is vs*))
Irlist (*LAppend s t*) *is vs* = (*Irlist s is vs*) @ (*Irlist t is vs*)
lemma [(*1*::*int*)] = []
⟨*proof*⟩

lemma ([(*3*::*int*) ∗ *x* + *y*∗*y* − *9* + (− *z*)] @ []) @ *xs* = [*y*∗*y* − *z* − *9* + (*3*::*int*) ∗ *x*]
⟨*proof*⟩

datatype *rnat* = *NC nat*| *NVar nat*| *NSuc rnat* | *NAdd rnat rnat* | *NMult rnat rnat* | *NNeg rnat* | *NSub rnat rnat* | *Nlgth rlist*
consts *Irnat* :: *rnat* ⇒ *int list* ⇒ (*int list*) *list* ⇒ *nat list* ⇒ *nat*
primrec
Irnat-Suc: *Irnat* (*NSuc t*) *is ls vs* = *Suc* (*Irnat t is ls vs*)

Irnat-Var: *Irnat* (*NVar n*) *is ls vs* = *vs*!*n*
Irnat-Neg: *Irnat* (*NNeg t*) *is ls vs* = *0*
Irnat-Add: *Irnat* (*NAdd s t*) *is ls vs* = *Irnat s is ls vs* + *Irnat t is ls vs*
Irnat-Sub: *Irnat* (*NSub s t*) *is ls vs* = *Irnat s is ls vs* − *Irnat t is ls vs*
Irnat-Mult: *Irnat* (*NMult s t*) *is ls vs* = *Irnat s is ls vs* ∗ *Irnat t is ls vs*
Irnat-lgth: *Irnat* (*Nlgth rxs*) *is ls vs* = *length* (*Irlist rxs is ls*)
Irnat-C: *Irnat* (*NC i*) *is ls vs* = *i*
lemma *Irnat-C0*: *Irnat* (*NC 0*) *is ls vs* = *0*
⟨*proof*⟩
lemma *Irnat-C1*: *Irnat* (*NC 1*) *is ls vs* = *1*
⟨*proof*⟩
lemma *Irnat-Cnumberof*: *Irnat* (*NC* (*number-of x*)) *is ls vs* = *number-of x*
⟨*proof*⟩
lemmas *Irnat-simps* = *Irnat-Suc Irnat-Var Irnat-Neg Irnat-Add Irnat-Sub Irnat-Mult Irnat-lgth*
Irnat-C0 Irnat-C1 Irnat-Cnumberof
lemma (*Suc n*) ∗ *length* ((*[*(*3*::*int*) ∗ *x* + *y*∗*y* − *9* + (− *z*)*]* @ *[]*) @ *xs*) = *length xs*
⟨*proof*⟩
datatype *rifm* = *RT* | *RF* | *RVar nat*
| *RNLT rnat rnat* | *RNILT rnat rint* | *RNEQ rnat rnat*
|*RAnd rifm rifm* | *ROr rifm rifm* | *RImp rifm rifm*| *RIff rifm rifm*
| *RNEX rifm* | *RIEX rifm*| *RLEX rifm* | *RNALL rifm* | *RIALL rifm*| *RLALL rifm*
| *RBEX rifm* | *RBALL rifm*

consts *Irifm* :: *rifm* ⇒ *bool list* ⇒ *int list* ⇒ (*int list*) *list* ⇒ *nat list* ⇒ *bool*
primrec
Irifm RT ps is ls ns = *True*
Irifm RF ps is ls ns = *False*
Irifm (*RVar n*) *ps is ls ns* = *ps*!*n*
Irifm (*RNLT s t*) *ps is ls ns* = (*Irnat s is ls ns* < *Irnat t is ls ns*)
Irifm (*RNILT s t*) *ps is ls ns* = (*int* (*Irnat s is ls ns*) < *Irint t is*)
Irifm (*RNEQ s t*) *ps is ls ns* = (*Irnat s is ls ns* = *Irnat t is ls ns*)
Irifm (*RAnd p q*) *ps is ls ns* = (*Irifm p ps is ls ns* ∧ *Irifm q ps is ls ns*)
Irifm (*ROr p q*) *ps is ls ns* = (*Irifm p ps is ls ns* ∨ *Irifm q ps is ls ns*)
Irifm (*RImp p q*) *ps is ls ns* = (*Irifm p ps is ls ns* ⟶ *Irifm q ps is ls ns*)
Irifm (*RIff p q*) *ps is ls ns* = (*Irifm p ps is ls ns* = *Irifm q ps is ls ns*)
Irifm (*RNEX p*) *ps is ls ns* = (∃ *x*. *Irifm p ps is ls* (*x*#*ns*))
Irifm (*RIEX p*) *ps is ls ns* = (∃ *x*. *Irifm p ps* (*x*#*is*) *ls ns*)
Irifm (*RLEX p*) *ps is ls ns* = (∃ *x*. *Irifm p ps is* (*x*#*ls*) *ns*)
Irifm (*RBEX p*) *ps is ls ns* = (∃ *x*. *Irifm p* (*x*#*ps*) *is ls ns*)
Irifm (*RNALL p*) *ps is ls ns* = (∀ *x*. *Irifm p ps is ls* (*x*#*ns*))
Irifm (*RIALL p*) *ps is ls ns* = (∀ *x*. *Irifm p ps* (*x*#*is*) *ls ns*)
Irifm (*RLALL p*) *ps is ls ns* = (∀ *x*. *Irifm p ps is* (*x*#*ls*) *ns*)
Irifm (*RBALL p*) *ps is ls ns* = (∀ *x*. *Irifm p* (*x*#*ps*) *is ls ns*)

lemma ∀ *x*. ∃ *n*. ((*Suc n*) ∗ *length* ((*[*(*3*::*int*) ∗ *x* + (*f t*)∗*y* − *9* + (− *z*)*]* @ *[]*) @ *xs*) = *length xs*) ∧ *m* < *5*∗*n* − *length* (*xs* @ *[2*,*3*,*4*,*x*∗*z* + *8* − *y]*) ⟶ (∃ *p*. ∀ *q*.

$p \wedge q \longrightarrow r$)
⟨*proof*⟩

datatype *prod* = *Zero* | *One* | *Var nat* | *Mul prod prod*
| *Pw prod nat* | *PNM nat nat prod*
consts *Iprod* :: *prod* ⇒ ('*a*::{*ordered-idom*,*recpower*}) *list* ⇒'*a*
primrec
Iprod Zero vs = *0*
Iprod One vs = *1*
Iprod (*Var n*) *vs* = *vs*!*n*
Iprod (*Mul a b*) *vs* = (*Iprod a vs* ∗ *Iprod b vs*)
Iprod (*Pw a n*) *vs* = ((*Iprod a vs*) ^ *n*)
Iprod (*PNM n k t*) *vs* = (*vs* ! *n*)^*k* ∗ *Iprod t vs*
consts *prodmul*:: *prod* × *prod* ⇒ *prod*
datatype *sgn* = *Pos prod* | *Neg prod* | *ZeroEq prod* | *NZeroEq prod* | *Tr* | *F*
| *Or sgn sgn* | *And sgn sgn*

consts *Isgn* :: *sgn* ⇒ ('*a*::{*ordered-idom*, *recpower*}) *list* ⇒*bool*
primrec
Isgn Tr vs = *True*
Isgn F vs = *False*
Isgn (*ZeroEq t*) *vs* = (*Iprod t vs* = *0*)
Isgn (*NZeroEq t*) *vs* = (*Iprod t vs* ≠ *0*)
Isgn (*Pos t*) *vs* = (*Iprod t vs* > *0*)
Isgn (*Neg t*) *vs* = (*Iprod t vs* < *0*)
Isgn (*And p q*) *vs* = (*Isgn p vs* ∧ *Isgn q vs*)
Isgn (*Or p q*) *vs* = (*Isgn p vs* ∨ *Isgn q vs*)

lemmas *eqs* = *Isgn.simps Iprod.simps*

lemma (*x*::'*a*::{*ordered-idom*, *recpower*})^*4* ∗ *y* ∗ *z* ∗ *y*^*2* ∗ *z*^*23* > *0*
⟨*proof*⟩

end

43 Square roots of primes are irrational

theory *Sqrt*
imports *Complex-Main Primes*
begin

The square root of any prime number (including *2*) is irrational.

theorem *sqrt-prime-irrational*:
assumes *prime p*
shows *sqrt* (*real p*) ∉ ℚ
⟨*proof*⟩

corollary *sqrt* (*real* (*2*::*nat*)) $\notin \mathbb{Q}$
⟨*proof*⟩

43.1 Variations

Here is an alternative version of the main proof, using mostly linear forward-reasoning. While this results in less top-down structure, it is probably closer to proofs seen in mathematics.

theorem
assumes *prime p*
shows *sqrt* (*real p*) $\notin \mathbb{Q}$
⟨*proof*⟩

end

44 Square roots of primes are irrational (script version)

theory *Sqrt-Script*
imports *Complex-Main Primes*
begin

Contrast this linear Isabelle/Isar script with Markus Wenzel's more mathematical version.

44.1 Preliminaries

lemma *prime-nonzero*: *prime p* $\Longrightarrow$ *p* $\neq$ *0*
⟨*proof*⟩

lemma *prime-dvd-other-side*:
n $*$ *n* = *p* $*$ (*k* $*$ *k*) $\Longrightarrow$ *prime p* $\Longrightarrow$ *p dvd n*
⟨*proof*⟩

lemma *reduction*: *prime p* $\Longrightarrow$
0 < *k* $\Longrightarrow$ *k* $*$ *k* = *p* $*$ (*j* $*$ *j*) $\Longrightarrow$ *k* < *p* $*$ *j* $\wedge$ *0* < *j*
⟨*proof*⟩

lemma *rearrange*: (*j*::*nat*) $*$ (*p* $*$ *j*) = *k* $*$ *k* $\Longrightarrow$ *k* $*$ *k* = *p* $*$ (*j* $*$ *j*)
⟨*proof*⟩

lemma *prime-not-square*:
prime p $\Longrightarrow$ ($\bigwedge$*k*. *0* < *k* $\Longrightarrow$ *m* $*$ *m* $\neq$ *p* $*$ (*k* $*$ *k*))
⟨*proof*⟩

44.2 Main theorem

The square root of any prime number (including *2*) is irrational.

theorem *prime-sqrt-irrational*:
$prime\ p \Longrightarrow x * x = real\ p \Longrightarrow 0 \leq x \Longrightarrow x \notin \mathbb{Q}$
⟨*proof*⟩

lemmas *two-sqrt-irrational* =
prime-sqrt-irrational [*OF two-is-prime*]

end

45 Arithmetic Series for Reals

theory *Arithmetic-Series-Complex*
imports *Complex-Main*
begin

lemma *arith-series-real*:
$(2::real) * (\sum i \in \{..<n\}.\ a + of\text{-}nat\ i * d) =$
$of\text{-}nat\ n * (a + (a + of\text{-}nat(n - 1) * d))$
⟨*proof*⟩

end

46 Divergence of the Harmonic Series

theory *HarmonicSeries*
imports *Complex-Main*
begin

47 Abstract

The following document presents a proof of the Divergence of Harmonic Series theorem formalised in the Isabelle/Isar theorem proving system.

Theorem: The series $\sum_{n=1}^{\infty} \frac{1}{n}$ does not converge to any number.

Informal Proof: The informal proof is based on the following auxillary lemmas:

- aux: $\sum_{n=2^m-1}^{2^m} \frac{1}{n} \geq \frac{1}{2}$
- aux2: $\sum_{n=1}^{2^M} \frac{1}{n} = 1 + \sum_{m=1}^{M} \sum_{n=2^m-1}^{2^m} \frac{1}{n}$

From *aux* and *aux2* we can deduce that $\sum_{n=1}^{2^M} \frac{1}{n} \geq 1 + \frac{M}{2}$ for all M. Now for contradiction, assume that $\sum_{n=1}^{\infty} \frac{1}{n} = s$ for some s. Because $\forall n. \frac{1}{n} > 0$ all the partial sums in the series must be less than s. However with our deduction above we can choose $N > 2 * s - 2$ and thus $\sum_{n=1}^{2^N} \frac{1}{n} > s$. This leads to a contradiction and hence $\sum_{n=1}^{\infty} \frac{1}{n}$ is not summable. QED.

48 Formal Proof

lemma *two-pow-sub*:
0 < m ⟹ (2::nat) ^ m − 2 ^ (m − 1) = 2 ^ (m − 1)
⟨*proof*⟩

We first prove the following auxillary lemma. This lemma simply states that the finite sums: $\frac{1}{2}$, $\frac{1}{3} + \frac{1}{4}$, $\frac{1}{5} + \frac{1}{6} + \frac{1}{7} + \frac{1}{8}$ etc. are all greater than or equal to $\frac{1}{2}$. We do this by observing that each term in the sum is greater than or equal to the last term, e.g. $\frac{1}{3} > \frac{1}{4}$ and thus $\frac{1}{3} + \frac{1}{4} > \frac{1}{4} + \frac{1}{4} = \frac{1}{2}$.

lemma *harmonic-aux*:
∀ m>0. (∑ n∈{(2::nat) ^ (m − 1)+1..2 ^ m}. 1/real n) ≥ 1/2
(**is** *∀ m>0. (∑ n∈(?S m). 1/real n) ≥ 1/2*)
⟨*proof*⟩

We then show that the sum of a finite number of terms from the harmonic series can be regrouped in increasing powers of 2. For example: $1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \frac{1}{6} + \frac{1}{7} + \frac{1}{8} = 1 + (\frac{1}{2}) + (\frac{1}{3} + \frac{1}{4}) + (\frac{1}{5} + \frac{1}{6} + \frac{1}{7} + \frac{1}{8})$.

lemma *harmonic-aux2* [*rule-format*]:
0<M ⟹ (∑ n∈{1..(2::nat) ^ M}. 1/real n) =
(1 + (∑ m∈{1..M}. ∑ n∈{(2::nat) ^ (m − 1)+1..2 ^ m}. 1/real n))
(**is** *0<M ⟹ ?LHS M = ?RHS M*)
⟨*proof*⟩

Using *harmonic-aux* and *harmonic-aux2* we now show that each group sum is greater than or equal to $\frac{1}{2}$ and thus the finite sum is bounded below by a value proportional to the number of elements we choose.

lemma *harmonic-aux3* [*rule-format*]:
shows *∀ (M::nat). (∑ n∈{1..(2::nat) ^ M}. 1 / real n) ≥ 1 + (real M)/2*
(**is** *∀ M. ?P M ≥ -*)
⟨*proof*⟩

The final theorem shows that as we take more and more elements (see *harmonic-aux3*) we get an ever increasing sum. By assuming the sum converges, the lemma *series-pos-less* (⟦*summable ?f*; *∀ m≥?n. 0 < ?f m*⟧ ⟹ *setsum ?f {0..<?n} < suminf ?f*) states that each sum is bounded above by the series' limit. This contradicts our first statement and thus we prove that the harmonic series is divergent.

theorem *DivergenceOfHarmonicSeries*:

shows ¬*summable* (λ*n*. *1*/*real* (*Suc n*))
(**is** ¬*summable ?f*)
⟨*proof*⟩

end

49 Examples for the 'refute' command

theory *Refute-Examples* **imports** *Main*
begin

refute-params [*satsolver*=*dpll*]

lemma *P* ∧ *Q*
⟨*proof*⟩
refute *1* — refutes *P*
refute *2* — refutes *Q*
refute — equivalent to 'refute 1'
— here 'refute 3' would cause an exception, since we only have 2 subgoals
refute [*maxsize*=*5*] — we can override parameters ...
refute [*satsolver*=*dpll*] *2* — ... and specify a subgoal at the same time
⟨*proof*⟩

49.1 Examples and Test Cases

49.1.1 Propositional logic

lemma *True*
refute
⟨*proof*⟩

lemma *False*
refute
⟨*proof*⟩

lemma *P*
refute
⟨*proof*⟩

lemma ~ *P*
refute
⟨*proof*⟩

lemma *P* & *Q*
refute
⟨*proof*⟩

lemma *P* | *Q*
refute

⟨*proof*⟩

lemma $P \longrightarrow Q$
refute
⟨*proof*⟩

lemma $(P::bool) = Q$
refute
⟨*proof*⟩

lemma $(P \mid Q) \longrightarrow (P \;\&\; Q)$
refute
⟨*proof*⟩

49.1.2 Predicate logic

lemma $P\ x\ y\ z$
refute
⟨*proof*⟩

lemma $P\ x\ y \longrightarrow P\ y\ x$
refute
⟨*proof*⟩

lemma $P\ (f\ (f\ x)) \longrightarrow P\ x \longrightarrow P\ (f\ x)$
refute
⟨*proof*⟩

49.1.3 Equality

lemma $P = True$
refute
⟨*proof*⟩

lemma $P = False$
refute
⟨*proof*⟩

lemma $x = y$
refute
⟨*proof*⟩

lemma $f\ x = g\ x$
refute
⟨*proof*⟩

lemma $(f::'a \Rightarrow 'b) = g$
refute
⟨*proof*⟩

lemma $(f::('d \Rightarrow 'd) \Rightarrow ('c \Rightarrow 'd)) = g$
refute
⟨*proof*⟩

lemma *distinct* [*a*,*b*]
refute
⟨*proof*⟩
refute
⟨*proof*⟩

49.1.4 First-Order Logic

lemma $\exists\, x.\ P\ x$
refute
⟨*proof*⟩

lemma $\forall\, x.\ P\ x$
refute
⟨*proof*⟩

lemma $EX!\ x.\ P\ x$
refute
⟨*proof*⟩

lemma *Ex P*
refute
⟨*proof*⟩

lemma *All P*
refute
⟨*proof*⟩

lemma *Ex1 P*
refute
⟨*proof*⟩

lemma $(\exists\, x.\ P\ x) \longrightarrow (\forall\, x.\ P\ x)$
refute
⟨*proof*⟩

lemma $(\forall\, x.\ \exists\, y.\ P\ x\ y) \longrightarrow (\exists\, y.\ \forall\, x.\ P\ x\ y)$
refute
⟨*proof*⟩

lemma $(\exists\, x.\ P\ x) \longrightarrow (EX!\ x.\ P\ x)$
refute
⟨*proof*⟩

A true statement (also testing names of free and bound variables being identical)

lemma $(\forall x\ y.\ P\ x\ y \longrightarrow P\ y\ x) \longrightarrow (\forall x.\ P\ x\ y) \longrightarrow P\ y\ x$
refute [*maxsize=4*]
⟨*proof*⟩

"A type has at most 4 elements."

lemma $a{=}b \mid a{=}c \mid a{=}d \mid a{=}e \mid b{=}c \mid b{=}d \mid b{=}e \mid c{=}d \mid c{=}e \mid d{=}e$
refute
⟨*proof*⟩

lemma $\forall a\ b\ c\ d\ e.\ a{=}b \mid a{=}c \mid a{=}d \mid a{=}e \mid b{=}c \mid b{=}d \mid b{=}e \mid c{=}d \mid c{=}e \mid d{=}e$
refute
⟨*proof*⟩

"Every reflexive and symmetric relation is transitive."

lemma $[\![\ \forall x.\ P\ x\ x; \forall x\ y.\ P\ x\ y \longrightarrow P\ y\ x\ ]\!] \Longrightarrow P\ x\ y \longrightarrow P\ y\ z \longrightarrow P\ x\ z$
refute
⟨*proof*⟩

The "Drinker's theorem" ...

lemma $\exists x.\ f\ x = g\ x \longrightarrow f = g$
refute [*maxsize=4*]
⟨*proof*⟩

... and an incorrect version of it

lemma $(\exists x.\ f\ x = g\ x) \longrightarrow f = g$
refute
⟨*proof*⟩

"Every function has a fixed point."

lemma $\exists x.\ f\ x = x$
refute
⟨*proof*⟩

"Function composition is commutative."

lemma $f\ (g\ x) = g\ (f\ x)$
refute
⟨*proof*⟩

"Two functions that are equivalent wrt. the same predicate 'P' are equal."

lemma $((P{::}('a{\Rightarrow}'b){\Rightarrow}bool)\ f = P\ g) \longrightarrow (f\ x = g\ x)$
refute
⟨*proof*⟩

49.1.5 Higher-Order Logic

lemma $\exists P.\ P$
refute
⟨*proof*⟩

lemma $\forall P.\ P$
  refute
⟨*proof*⟩

lemma $EX!\ P.\ P$
  refute
  ⟨*proof*⟩

lemma $EX!\ P.\ P\ x$
  refute
⟨*proof*⟩

lemma $P\ Q \mid Q\ x$
  refute
⟨*proof*⟩

lemma $x \neq All$
  refute
⟨*proof*⟩

lemma $x \neq Ex$
  refute
⟨*proof*⟩

lemma $x \neq Ex1$
  refute
⟨*proof*⟩

"The transitive closure 'T' of an arbitrary relation 'P' is non-empty."

constdefs
  $trans :: ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$
  $trans\ P == (ALL\ x\ y\ z.\ P\ x\ y \longrightarrow P\ y\ z \longrightarrow P\ x\ z)$
  $subset :: ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$
  $subset\ P\ Q == (ALL\ x\ y.\ P\ x\ y \longrightarrow Q\ x\ y)$
  $trans\text{-}closure :: ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$
  $trans\text{-}closure\ P\ Q == (subset\ Q\ P)\ \&\ (trans\ P)\ \&\ (ALL\ R.\ subset\ Q\ R \longrightarrow trans\ R \longrightarrow subset\ P\ R)$

lemma $trans\text{-}closure\ T\ P \longrightarrow (\exists x\ y.\ T\ x\ y)$
  refute
⟨*proof*⟩

"The union of transitive closures is equal to the transitive closure of unions."

lemma $(\forall x\ y.\ (P\ x\ y \mid R\ x\ y) \longrightarrow T\ x\ y) \longrightarrow trans\ T \longrightarrow (\forall Q.\ (\forall x\ y.\ (P\ x\ y \mid R\ x\ y) \longrightarrow Q\ x\ y) \longrightarrow trans\ Q \longrightarrow subset\ T\ Q)$
  $\longrightarrow trans\text{-}closure\ TP\ P$
  $\longrightarrow trans\text{-}closure\ TR\ R$
  $\longrightarrow (T\ x\ y = (TP\ x\ y \mid TR\ x\ y))$

refute
⟨*proof*⟩

"Every surjective function is invertible."

lemma $(\forall y.\ \exists x.\ y = f\ x) \longrightarrow (\exists g.\ \forall x.\ g\ (f\ x) = x)$
refute
⟨*proof*⟩

"Every invertible function is surjective."

lemma $(\exists g.\ \forall x.\ g\ (f\ x) = x) \longrightarrow (\forall y.\ \exists x.\ y = f\ x)$
refute
⟨*proof*⟩

Every point is a fixed point of some function.

lemma $\exists f.\ f\ x = x$
refute [*maxsize=4*]
⟨*proof*⟩

Axiom of Choice: first an incorrect version ...

lemma $(\forall x.\ \exists y.\ P\ x\ y) \longrightarrow (EX!f.\ \forall x.\ P\ x\ (f\ x))$
refute
⟨*proof*⟩

... and now two correct ones

lemma $(\forall x.\ \exists y.\ P\ x\ y) \longrightarrow (\exists f.\ \forall x.\ P\ x\ (f\ x))$
refute [*maxsize=4*]
⟨*proof*⟩

lemma $(\forall x.\ EX!y.\ P\ x\ y) \longrightarrow (EX!f.\ \forall x.\ P\ x\ (f\ x))$
refute [*maxsize=2*]
⟨*proof*⟩

49.1.6 Meta-logic

lemma $!!x.\ P\ x$
refute
⟨*proof*⟩

lemma $f\ x == g\ x$
refute
⟨*proof*⟩

lemma $P \Longrightarrow Q$
refute
⟨*proof*⟩

lemma $[\![\ P;\ Q;\ R\ ]\!] \Longrightarrow S$
refute

⟨*proof*⟩

lemma $(x == all) \Longrightarrow False$
refute
⟨*proof*⟩

lemma $(x == (op ==)) \Longrightarrow False$
refute
⟨*proof*⟩

lemma $(x == (op \Longrightarrow)) \Longrightarrow False$
refute
⟨*proof*⟩

49.1.7 Schematic variables

lemma *?P*
refute
⟨*proof*⟩

lemma $x = ?y$
refute
⟨*proof*⟩

49.1.8 Abstractions

lemma $(\lambda x.\ x) = (\lambda x.\ y)$
refute
⟨*proof*⟩

lemma $(\lambda f.\ f\ x) = (\lambda f.\ True)$
refute
⟨*proof*⟩

lemma $(\lambda x.\ x) = (\lambda y.\ y)$
refute
⟨*proof*⟩

49.1.9 Sets

lemma $P\ (A::'a\ set)$
refute
⟨*proof*⟩

lemma $P\ (A::'a\ set\ set)$
refute
⟨*proof*⟩

lemma $\{x.\ P\ x\} = \{y.\ P\ y\}$
refute

⟨*proof*⟩

lemma $x : \{x.\ P\ x\}$
refute
⟨*proof*⟩

lemma $P\ op$:
refute
⟨*proof*⟩

lemma $P\ (op{:}\ x)$
refute
⟨*proof*⟩

lemma $P\ Collect$
refute
⟨*proof*⟩

lemma $A\ Un\ B = A\ Int\ B$
refute
⟨*proof*⟩

lemma $(A\ Int\ B)\ Un\ C = (A\ Un\ C)\ Int\ B$
refute
⟨*proof*⟩

lemma $Ball\ A\ P \longrightarrow Bex\ A\ P$
refute
⟨*proof*⟩

49.1.10 undefined

lemma *undefined*
refute
⟨*proof*⟩

lemma $P\ undefined$
refute
⟨*proof*⟩

lemma $undefined\ x$
refute
⟨*proof*⟩

lemma *undefined undefined*
refute
⟨*proof*⟩

49.1.11 The

lemma *The P*
refute
⟨*proof*⟩

lemma *P The*
refute
⟨*proof*⟩

lemma *P* (*The P*)
refute
⟨*proof*⟩

lemma (*THE x. x=y*) = *z*
refute
⟨*proof*⟩

lemma *Ex P* ⟶ *P* (*The P*)
refute
⟨*proof*⟩

49.1.12 Eps

lemma *Eps P*
refute
⟨*proof*⟩

lemma *P Eps*
refute
⟨*proof*⟩

lemma *P* (*Eps P*)
refute
⟨*proof*⟩

lemma (*SOME x. x=y*) = *z*
refute
⟨*proof*⟩

lemma *Ex P* ⟶ *P* (*Eps P*)
refute [*maxsize=3*]
⟨*proof*⟩

49.1.13 Subtypes (typedef), typedecl

A completely unspecified non-empty subset of *'a*:

typedef *'a myTdef* = *insert* (*undefined*::*'a*) (*undefined*::*'a set*)
⟨*proof*⟩

lemma $(x::'a\ myTdef) = y$
refute
⟨*proof*⟩

typedecl *myTdecl*

typedef $'a\ T\text{-}bij = \{(f::'a \Rightarrow 'a).\ \forall\, y.\ \exists!x.\ f\, x = y\}$
⟨*proof*⟩

lemma $P\ (f::(myTdecl\ myTdef)\ T\text{-}bij)$
refute
⟨*proof*⟩

49.1.14 Inductive datatypes

With *quick-and-dirty* set, the datatype package does not generate certain axioms for recursion operators. Without these axioms, refute may find spurious countermodels.

unit

lemma $P\ (x::unit)$
refute
⟨*proof*⟩

lemma $\forall\, x::unit.\ P\ x$
refute
⟨*proof*⟩

lemma $P\ ()$
refute
⟨*proof*⟩

lemma $unit\text{-}rec\ u\ x = u$
refute
⟨*proof*⟩

lemma $P\ (unit\text{-}rec\ u\ x)$
refute
⟨*proof*⟩

lemma $P\ (case\ x\ of\ () \Rightarrow u)$
refute
⟨*proof*⟩

option

lemma $P\ (x::'a\ option)$
refute
⟨*proof*⟩

lemma $\forall x::'a\ option.\ P\ x$
refute
⟨*proof*⟩

lemma $P\ None$
refute
⟨*proof*⟩

lemma $P\ (Some\ x)$
refute
⟨*proof*⟩

lemma $option\text{-}rec\ n\ s\ None = n$
refute
⟨*proof*⟩

lemma $option\text{-}rec\ n\ s\ (Some\ x) = s\ x$
refute [*maxsize=4*]
⟨*proof*⟩

lemma $P\ (option\text{-}rec\ n\ s\ x)$
refute
⟨*proof*⟩

lemma $P\ (case\ x\ of\ None \Rightarrow n \mid Some\ u \Rightarrow s\ u)$
refute
⟨*proof*⟩

*

lemma $P\ (x::'a*'b)$
refute
⟨*proof*⟩

lemma $\forall x::'a*'b.\ P\ x$
refute
⟨*proof*⟩

lemma $P\ (x,\ y)$
refute
⟨*proof*⟩

lemma $P\ (fst\ x)$
refute
⟨*proof*⟩

lemma $P\ (snd\ x)$
refute
⟨*proof*⟩

lemma *P Pair*
refute
⟨*proof*⟩

lemma *prod-rec p* (*a*, *b*) = *p a b*
refute [*maxsize=2*]
⟨*proof*⟩

lemma *P* (*prod-rec p x*)
refute
⟨*proof*⟩

lemma *P* (*case x of Pair a b* ⇒ *p a b*)
refute
⟨*proof*⟩

+

lemma *P* (*x*::'*a*+'*b*)
refute
⟨*proof*⟩

lemma ∀ *x*::'*a*+'*b*. *P x*
refute
⟨*proof*⟩

lemma *P* (*Inl x*)
refute
⟨*proof*⟩

lemma *P* (*Inr x*)
refute
⟨*proof*⟩

lemma *P Inl*
refute
⟨*proof*⟩

lemma *sum-rec l r* (*Inl x*) = *l x*
refute [*maxsize=3*]
⟨*proof*⟩

lemma *sum-rec l r* (*Inr x*) = *r x*
refute [*maxsize=3*]
⟨*proof*⟩

lemma *P* (*sum-rec l r x*)
refute
⟨*proof*⟩

lemma P (*case* x *of Inl* $a \Rightarrow l\ a \mid Inr\ b \Rightarrow r\ b$)
refute
⟨*proof*⟩

Non-recursive datatypes

datatype *T1* = $A \mid B$

lemma P (x::*T1*)
refute
⟨*proof*⟩

lemma $\forall x$::*T1*. $P\ x$
refute
⟨*proof*⟩

lemma $P\ A$
refute
⟨*proof*⟩

lemma $P\ B$
refute
⟨*proof*⟩

lemma *T1-rec* $a\ b\ A = a$
refute
⟨*proof*⟩

lemma *T1-rec* $a\ b\ B = b$
refute
⟨*proof*⟩

lemma P (*T1-rec* $a\ b\ x$)
refute
⟨*proof*⟩

lemma P (*case* x *of* $A \Rightarrow a \mid B \Rightarrow b$)
refute
⟨*proof*⟩

datatype $'a$ *T2* = *C T1* | *D* $'a$

lemma P (x::$'a$ *T2*)
refute
⟨*proof*⟩

lemma $\forall x$::$'a$ *T2*. $P\ x$
refute
⟨*proof*⟩

lemma *P D*
refute
⟨*proof*⟩

lemma *T2-rec c d (C x) = c x*
refute [*maxsize=4*]
⟨*proof*⟩

lemma *T2-rec c d (D x) = d x*
refute [*maxsize=4*]
⟨*proof*⟩

lemma *P (T2-rec c d x)*
refute
⟨*proof*⟩

lemma *P (case x of C u ⇒ c u | D v ⇒ d v)*
refute
⟨*proof*⟩

datatype *('a,'b) T3 = E 'a ⇒ 'b*

lemma *P (x::('a,'b) T3)*
refute
⟨*proof*⟩

lemma *∀x::('a,'b) T3. P x*
refute
⟨*proof*⟩

lemma *P E*
refute
⟨*proof*⟩

lemma *T3-rec e (E x) = e x*
refute [*maxsize=2*]
⟨*proof*⟩

lemma *P (T3-rec e x)*
refute
⟨*proof*⟩

lemma *P (case x of E f ⇒ e f)*
refute
⟨*proof*⟩

Recursive datatypes

nat

lemma $P\ (x{::}nat)$
refute
⟨*proof*⟩

lemma $\forall x{::}nat.\ P\ x$
refute
⟨*proof*⟩

lemma $P\ (Suc\ 0)$
refute
⟨*proof*⟩

lemma *P Suc*
refute — *Suc* is a partial function (regardless of the size of the model), hence *P Suc* is undefined, hence no model will be found
⟨*proof*⟩

lemma *nat-rec zero suc 0 = zero*
refute
⟨*proof*⟩

lemma *nat-rec zero suc (Suc x) = suc x (nat-rec zero suc x)*
refute [*maxsize=2*]
⟨*proof*⟩

lemma *P (nat-rec zero suc x)*
refute
⟨*proof*⟩

lemma *P (case x of 0 ⇒ zero | Suc n ⇒ suc n)*
refute
⟨*proof*⟩

'a list

lemma $P\ (xs{::}'a\ list)$
refute
⟨*proof*⟩

lemma $\forall xs{::}'a\ list.\ P\ xs$
refute
⟨*proof*⟩

lemma $P\ [x,\ y]$
refute
⟨*proof*⟩

lemma *list-rec nil cons [] = nil*
refute [*maxsize=3*]
⟨*proof*⟩

lemma *list-rec nil cons (x#xs) = cons x xs (list-rec nil cons xs)*
refute [*maxsize=2*]
⟨*proof*⟩

lemma *P (list-rec nil cons xs)*
refute
⟨*proof*⟩

lemma *P (case x of Nil ⇒ nil | Cons a b ⇒ cons a b)*
refute
⟨*proof*⟩

lemma *(xs::'a list) = ys*
refute
⟨*proof*⟩

lemma *a # xs = b # xs*
refute
⟨*proof*⟩

datatype *BitList = BitListNil | Bit0 BitList | Bit1 BitList*

lemma *P (x::BitList)*
refute
⟨*proof*⟩

lemma *∀x::BitList. P x*
refute
⟨*proof*⟩

lemma *P (Bit0 (Bit1 BitListNil))*
refute
⟨*proof*⟩

lemma *BitList-rec nil bit0 bit1 BitListNil = nil*
refute [*maxsize=4*]
⟨*proof*⟩

lemma *BitList-rec nil bit0 bit1 (Bit0 xs) = bit0 xs (BitList-rec nil bit0 bit1 xs)*
refute [*maxsize=2*]
⟨*proof*⟩

lemma *BitList-rec nil bit0 bit1 (Bit1 xs) = bit1 xs (BitList-rec nil bit0 bit1 xs)*
refute [*maxsize=2*]
⟨*proof*⟩

lemma *P (BitList-rec nil bit0 bit1 x)*
refute

⟨*proof*⟩

datatype *'a BinTree = Leaf 'a | Node 'a BinTree 'a BinTree*

lemma *P (x::'a BinTree)*
  refute
⟨*proof*⟩

lemma *∀x::'a BinTree. P x*
  refute
⟨*proof*⟩

lemma *P (Node (Leaf x) (Leaf y))*
  refute
⟨*proof*⟩

lemma *BinTree-rec l n (Leaf x) = l x*
  refute [*maxsize=1*]
  ⟨*proof*⟩

lemma *BinTree-rec l n (Node x y) = n x y (BinTree-rec l n x) (BinTree-rec l n y)*
  refute [*maxsize=1*]
  ⟨*proof*⟩

lemma *P (BinTree-rec l n x)*
  refute
⟨*proof*⟩

lemma *P (case x of Leaf a ⇒ l a | Node a b ⇒ n a b)*
  refute
⟨*proof*⟩

Mutually recursive datatypes

datatype *'a aexp = Number 'a | ITE 'a bexp 'a aexp 'a aexp*
    and *'a bexp = Equal 'a aexp 'a aexp*

lemma *P (x::'a aexp)*
  refute
⟨*proof*⟩

lemma *∀x::'a aexp. P x*
  refute
⟨*proof*⟩

lemma *P (ITE (Equal (Number x) (Number y)) (Number x) (Number y))*
  refute
⟨*proof*⟩

lemma *P (x::'a bexp)*

refute
⟨*proof*⟩

lemma ∀ *x*::*'a bexp*. *P x*
refute
⟨*proof*⟩

lemma *aexp-bexp-rec-1 number ite equal* (*Number x*) = *number x*
refute [*maxsize=1*]
⟨*proof*⟩

lemma *aexp-bexp-rec-1 number ite equal* (*ITE x y z*) = *ite x y z* (*aexp-bexp-rec-2 number ite equal x*) (*aexp-bexp-rec-1 number ite equal y*) (*aexp-bexp-rec-1 number ite equal z*)
refute [*maxsize=1*]
⟨*proof*⟩

lemma *P* (*aexp-bexp-rec-1 number ite equal x*)
refute
⟨*proof*⟩

lemma *P* (*case x of Number a* ⇒ *number a* | *ITE b a1 a2* ⇒ *ite b a1 a2*)
refute
⟨*proof*⟩

lemma *aexp-bexp-rec-2 number ite equal* (*Equal x y*) = *equal x y* (*aexp-bexp-rec-1 number ite equal x*) (*aexp-bexp-rec-1 number ite equal y*)
refute [*maxsize=1*]
⟨*proof*⟩

lemma *P* (*aexp-bexp-rec-2 number ite equal x*)
refute
⟨*proof*⟩

lemma *P* (*case x of Equal a1 a2* ⇒ *equal a1 a2*)
refute
⟨*proof*⟩

datatype *X* = *A* | *B X* | *C Y*
and *Y* = *D X* | *E Y* | *F*

lemma *P* (*x*::*X*)
refute
⟨*proof*⟩

lemma *P* (*y*::*Y*)
refute
⟨*proof*⟩

lemma $P\ (B\ (B\ A))$
refute
⟨*proof*⟩

lemma $P\ (B\ (C\ F))$
refute
⟨*proof*⟩

lemma $P\ (C\ (D\ A))$
refute
⟨*proof*⟩

lemma $P\ (C\ (E\ F))$
refute
⟨*proof*⟩

lemma $P\ (D\ (B\ A))$
refute
⟨*proof*⟩

lemma $P\ (D\ (C\ F))$
refute
⟨*proof*⟩

lemma $P\ (E\ (D\ A))$
refute
⟨*proof*⟩

lemma $P\ (E\ (E\ F))$
refute
⟨*proof*⟩

lemma $P\ (C\ (D\ (C\ F)))$
refute
⟨*proof*⟩

lemma $X\text{-}Y\text{-}rec\text{-}1\ a\ b\ c\ d\ e\ f\ A = a$
refute [*maxsize=3*]
⟨*proof*⟩

lemma $X\text{-}Y\text{-}rec\text{-}1\ a\ b\ c\ d\ e\ f\ (B\ x) = b\ x\ (X\text{-}Y\text{-}rec\text{-}1\ a\ b\ c\ d\ e\ f\ x)$
refute [*maxsize=1*]
⟨*proof*⟩

lemma $X\text{-}Y\text{-}rec\text{-}1\ a\ b\ c\ d\ e\ f\ (C\ y) = c\ y\ (X\text{-}Y\text{-}rec\text{-}2\ a\ b\ c\ d\ e\ f\ y)$
refute [*maxsize=1*]
⟨*proof*⟩

lemma $X\text{-}Y\text{-}rec\text{-}2\ a\ b\ c\ d\ e\ f\ (D\ x) = d\ x\ (X\text{-}Y\text{-}rec\text{-}1\ a\ b\ c\ d\ e\ f\ x)$

refute [*maxsize=1*]
⟨*proof*⟩

lemma *X-Y-rec-2 a b c d e f* (*E y*) = *e y* (*X-Y-rec-2 a b c d e f y*)
refute [*maxsize=1*]
⟨*proof*⟩

lemma *X-Y-rec-2 a b c d e f F* = *f*
refute [*maxsize=3*]
⟨*proof*⟩

lemma *P* (*X-Y-rec-1 a b c d e f x*)
refute
⟨*proof*⟩

lemma *P* (*X-Y-rec-2 a b c d e f y*)
refute
⟨*proof*⟩

Other datatype examples

Indirect recursion is implemented via mutual recursion.

datatype *XOpt* = *CX XOpt option* | *DX bool* ⇒ *XOpt option*

lemma *P* (*x*::*XOpt*)
refute
⟨*proof*⟩

lemma *P* (*CX None*)
refute
⟨*proof*⟩

lemma *P* (*CX* (*Some* (*CX None*)))
refute
⟨*proof*⟩

lemma *XOpt-rec-1 cx dx n1 s1 n2 s2* (*CX x*) = *cx x* (*XOpt-rec-2 cx dx n1 s1 n2 s2 x*)
refute [*maxsize=1*]
⟨*proof*⟩

lemma *XOpt-rec-1 cx dx n1 s1 n2 s2* (*DX x*) = *dx x* (λ*b*. *XOpt-rec-3 cx dx n1 s1 n2 s2* (*x b*))
refute [*maxsize=1*]
⟨*proof*⟩

lemma *XOpt-rec-2 cx dx n1 s1 n2 s2 None* = *n1*
refute [*maxsize=2*]
⟨*proof*⟩

lemma *XOpt-rec-2 cx dx n1 s1 n2 s2 (Some x) = s1 x (XOpt-rec-1 cx dx n1 s1 n2 s2 x)*
refute [*maxsize=1*]
⟨*proof*⟩

lemma *XOpt-rec-3 cx dx n1 s1 n2 s2 None = n2*
refute [*maxsize=2*]
⟨*proof*⟩

lemma *XOpt-rec-3 cx dx n1 s1 n2 s2 (Some x) = s2 x (XOpt-rec-1 cx dx n1 s1 n2 s2 x)*
refute [*maxsize=1*]
⟨*proof*⟩

lemma *P (XOpt-rec-1 cx dx n1 s1 n2 s2 x)*
refute
⟨*proof*⟩

lemma *P (XOpt-rec-2 cx dx n1 s1 n2 s2 x)*
refute
⟨*proof*⟩

lemma *P (XOpt-rec-3 cx dx n1 s1 n2 s2 x)*
refute
⟨*proof*⟩

datatype *'a YOpt = CY ('a ⇒ 'a YOpt) option*

lemma *P (x::'a YOpt)*
refute
⟨*proof*⟩

lemma *P (CY None)*
refute
⟨*proof*⟩

lemma *P (CY (Some (λa. CY None)))*
refute
⟨*proof*⟩

lemma *YOpt-rec-1 cy n s (CY x) = cy x (YOpt-rec-2 cy n s x)*
refute [*maxsize=1*]
⟨*proof*⟩

lemma *YOpt-rec-2 cy n s None = n*
refute [*maxsize=2*]
⟨*proof*⟩

lemma *YOpt-rec-2 cy n s (Some x) = s x (λa. YOpt-rec-1 cy n s (x a))*

refute [*maxsize=1*]
⟨*proof*⟩

lemma *P* (*YOpt-rec-1 cy n s x*)
refute
⟨*proof*⟩

lemma *P* (*YOpt-rec-2 cy n s x*)
refute
⟨*proof*⟩

datatype *Trie* = *TR Trie list*

lemma *P* (*x*::*Trie*)
refute
⟨*proof*⟩

lemma ∀ *x*::*Trie*. *P x*
refute
⟨*proof*⟩

lemma *P* (*TR* [*TR* []])
refute
⟨*proof*⟩

lemma *Trie-rec-1 tr nil cons* (*TR x*) = *tr x* (*Trie-rec-2 tr nil cons x*)
refute [*maxsize=1*]
⟨*proof*⟩

lemma *Trie-rec-2 tr nil cons* [] = *nil*
refute [*maxsize=3*]
⟨*proof*⟩

lemma *Trie-rec-2 tr nil cons* (*x*#*xs*) = *cons x xs* (*Trie-rec-1 tr nil cons x*) (*Trie-rec-2 tr nil cons xs*)
refute [*maxsize=1*]
⟨*proof*⟩

lemma *P* (*Trie-rec-1 tr nil cons x*)
refute
⟨*proof*⟩

lemma *P* (*Trie-rec-2 tr nil cons x*)
refute
⟨*proof*⟩

datatype *InfTree* = *Leaf* | *Node nat* ⇒ *InfTree*

lemma *P* (*x*::*InfTree*)

refute
⟨*proof*⟩

lemma ∀ *x*::*InfTree*. *P x*
refute
⟨*proof*⟩

lemma *P* (*Node* (λ*n*. *Leaf*))
refute
⟨*proof*⟩

lemma *InfTree-rec leaf node Leaf* = *leaf*
refute [*maxsize=2*]
⟨*proof*⟩

lemma *InfTree-rec leaf node* (*Node x*) = *node x* (λ*n*. *InfTree-rec leaf node* (*x n*))
refute [*maxsize=1*]
⟨*proof*⟩

lemma *P* (*InfTree-rec leaf node x*)
refute
⟨*proof*⟩

datatype *'a lambda* = *Var 'a* | *App 'a lambda 'a lambda* | *Lam 'a* ⇒ *'a lambda*

lemma *P* (*x*::*'a lambda*)
refute
⟨*proof*⟩

lemma ∀ *x*::*'a lambda*. *P x*
refute
⟨*proof*⟩

lemma *P* (*Lam* (λ*a*. *Var a*))
refute
⟨*proof*⟩

lemma *lambda-rec var app lam* (*Var x*) = *var x*
refute [*maxsize=1*]
⟨*proof*⟩

lemma *lambda-rec var app lam* (*App x y*) = *app x y* (*lambda-rec var app lam x*) (*lambda-rec var app lam y*)
refute [*maxsize=1*]
⟨*proof*⟩

lemma *lambda-rec var app lam* (*Lam x*) = *lam x* (λ*a*. *lambda-rec var app lam* (*x a*))
refute [*maxsize=1*]

⟨*proof*⟩

lemma *P* (*lambda-rec v a l x*)
refute
⟨*proof*⟩

Taken from "Inductive datatypes in HOL", p.8:

datatype ('*a*, '*b*) *T* = *C* '*a* ⇒ *bool* | *D* '*b list*
datatype '*c U* = *E* ('*c*, '*c U*) *T*

lemma *P* (*x*::'*c U*)
refute
⟨*proof*⟩

lemma ∀ *x*::'*c U*. *P x*
refute
⟨*proof*⟩

lemma *P* (*E* (*C* (λ*a*. *True*)))
refute
⟨*proof*⟩

lemma *U-rec-1 e c d nil cons* (*E x*) = *e x* (*U-rec-2 e c d nil cons x*)
refute [*maxsize=1*]
⟨*proof*⟩

lemma *U-rec-2 e c d nil cons* (*C x*) = *c x*
refute [*maxsize=1*]
⟨*proof*⟩

lemma *U-rec-2 e c d nil cons* (*D x*) = *d x* (*U-rec-3 e c d nil cons x*)
refute [*maxsize=1*]
⟨*proof*⟩

lemma *U-rec-3 e c d nil cons* [] = *nil*
refute [*maxsize=2*]
⟨*proof*⟩

lemma *U-rec-3 e c d nil cons* (*x*#*xs*) = *cons x xs* (*U-rec-1 e c d nil cons x*)
(*U-rec-3 e c d nil cons xs*)
refute [*maxsize=1*]
⟨*proof*⟩

lemma *P* (*U-rec-1 e c d nil cons x*)
refute
⟨*proof*⟩

lemma *P* (*U-rec-2 e c d nil cons x*)
refute

⟨*proof*⟩

lemma *P* (*U-rec-3 e c d nil cons x*)
refute
⟨*proof*⟩

49.1.15 Records

record ('*a*, '*b*) *point* =
xpos :: '*a*
ypos :: '*b*

lemma (*x*::('*a*, '*b*) *point*) = *y*
refute
⟨*proof*⟩

record ('*a*, '*b*, '*c*) *extpoint* = ('*a*, '*b*) *point* +
ext :: '*c*

lemma (*x*::('*a*, '*b*, '*c*) *extpoint*) = *y*
refute
⟨*proof*⟩

49.1.16 Inductively defined sets

inductive-set *arbitrarySet* :: '*a set*
where
undefined : *arbitrarySet*

lemma *x* : *arbitrarySet*
refute
⟨*proof*⟩

inductive-set *evenCard* :: '*a set set*
where
{} : *evenCard*
| ⟦ *S* : *evenCard*; *x* ∉ *S*; *y* ∉ *S*; *x* ≠ *y* ⟧ ⟹ *S* ∪ {*x*, *y*} : *evenCard*

lemma *S* : *evenCard*
refute
⟨*proof*⟩

inductive-set
even :: *nat set*
and *odd* :: *nat set*
where
0 : *even*
| *n* : *even* ⟹ *Suc n* : *odd*
| *n* : *odd* ⟹ *Suc n* : *even*

lemma $n : odd$

⟨*proof*⟩

consts $f :: 'a \Rightarrow 'a$

inductive-set
  a-even :: $'a$ *set*
  and *a-odd* :: $'a$ *set*
where
  undefined : *a-even*
| x : *a-even* $\Longrightarrow$ $f\ x$: *a-odd*
| x : *a-odd* $\Longrightarrow$ $f\ x$: *a-even*

lemma x : *a-odd*
  refute — finds a model of size 2, as expected
⟨*proof*⟩

49.1.17 Examples involving special functions

lemma *card* $x = 0$
  refute
⟨*proof*⟩

lemma *finite* x
  refute — no finite countermodel exists
⟨*proof*⟩

lemma $(x{::}nat) + y = 0$
  refute
⟨*proof*⟩

lemma $(x{::}nat) = x + x$
  refute
⟨*proof*⟩

lemma $(x{::}nat) - y + y = x$
  refute
⟨*proof*⟩

lemma $(x{::}nat) = x * x$
  refute
⟨*proof*⟩

lemma $(x{::}nat) < x + y$
  refute
⟨*proof*⟩

lemma $xs\ @\ [] = ys\ @\ []$

refute
⟨*proof*⟩

lemma *xs* @ *ys* = *ys* @ *xs*
refute
⟨*proof*⟩

lemma *f* (*lfp f*) = *lfp f*
refute
⟨*proof*⟩

lemma *f* (*gfp f*) = *gfp f*
refute
⟨*proof*⟩

lemma *lfp f* = *gfp f*
refute
⟨*proof*⟩

49.1.18 Axiomatic type classes and overloading

A type class without axioms:

axclass *classA*

lemma *P* (*x*::'*a*::*classA*)
refute
⟨*proof*⟩

The axiom of this type class does not contain any type variables:

axclass *classB*
classB-ax: *P* | ~ *P*

lemma *P* (*x*::'*a*::*classB*)
refute
⟨*proof*⟩

An axiom with a type variable (denoting types which have at least two elements):

axclass *classC* < *type*
classC-ax: ∃ *x y*. *x* ≠ *y*

lemma *P* (*x*::'*a*::*classC*)
refute
⟨*proof*⟩

lemma ∃ *x y*. (*x*::'*a*::*classC*) ≠ *y*
refute — no countermodel exists
⟨*proof*⟩

A type class for which a constant is defined:

consts
  classD-const :: $'a \Rightarrow 'a$

axclass *classD* < *type*
  classD-ax: *classD-const* (*classD-const x*) = *classD-const x*

lemma *P* (*x*::$'a$::*classD*)
  refute
⟨*proof*⟩

A type class with multiple superclasses:

axclass *classE* < *classC*, *classD*

lemma *P* (*x*::$'a$::*classE*)
  refute
⟨*proof*⟩

lemma *P* (*x*::$'a$::{*classB*, *classE*})
  refute
⟨*proof*⟩

OFCLASS:

lemma *OFCLASS*($'a$::*type*, *type-class*)
  refute — no countermodel exists
  ⟨*proof*⟩

lemma *OFCLASS*($'a$::*classC*, *type-class*)
  refute — no countermodel exists
  ⟨*proof*⟩

lemma *OFCLASS*($'a$, *classB-class*)
  refute — no countermodel exists
  ⟨*proof*⟩

lemma *OFCLASS*($'a$::*type*, *classC-class*)
  refute
⟨*proof*⟩

Overloading:

consts *inverse* :: $'a \Rightarrow 'a$

defs (**overloaded**)
  inverse-bool: *inverse* (*b*::*bool*) == ~ *b*
  inverse-set : *inverse* (*S*::$'a$ *set*) == $-S$
  inverse-pair: *inverse p* == (*inverse* (*fst p*), *inverse* (*snd p*))

lemma *inverse b*

refute
⟨*proof*⟩

lemma *P* (*inverse* (*S*::′*a set*))
refute
⟨*proof*⟩

lemma *P* (*inverse* (*p*::′*a*×′*b*))
refute
⟨*proof*⟩

refute-params [*satsolver*=*auto*]

end

50 Examples for the 'quickcheck' command

theory *Quickcheck-Examples*
imports *Main*
begin

The 'quickcheck' command allows to find counterexamples by evaluating formulae under an assignment of free variables to random values. In contrast to 'refute', it can deal with inductive datatypes, but cannot handle quantifiers.

50.1 Lists

theorem *map g* (*map f xs*) = *map* (*g o f*) *xs*
quickcheck
⟨*proof*⟩

theorem *map g* (*map f xs*) = *map* (*f o g*) *xs*
quickcheck
⟨*proof*⟩

theorem *rev* (*xs* @ *ys*) = *rev ys* @ *rev xs*
quickcheck
⟨*proof*⟩

theorem *rev* (*xs* @ *ys*) = *rev xs* @ *rev ys*
quickcheck
⟨*proof*⟩

theorem *rev* (*rev xs*) = *xs*
quickcheck
⟨*proof*⟩

theorem *rev xs = xs*
quickcheck
⟨*proof*⟩

An example involving functions inside other data structures

primrec *app* :: (*'a* ⇒ *'a*) *list* ⇒ *'a* ⇒ *'a* **where**
app [] *x* = *x*
| *app* (*f* # *fs*) *x* = *app fs* (*f x*)

lemma *app* (*fs* @ *gs*) *x* = *app gs* (*app fs x*)
quickcheck
⟨*proof*⟩

lemma *app* (*fs* @ *gs*) *x* = *app fs* (*app gs x*)
quickcheck
⟨*proof*⟩

primrec *occurs* :: *'a* ⇒ *'a list* ⇒ *nat* **where**
occurs a [] = *0*
| *occurs a* (*x*#*xs*) = (*if* (*x*=*a*) *then Suc*(*occurs a xs*) *else occurs a xs*)

primrec *del1* :: *'a* ⇒ *'a list* ⇒ *'a list* **where**
del1 a [] = []
| *del1 a* (*x*#*xs*) = (*if* (*x*=*a*) *then xs else* (*x*#*del1 a xs*))

A lemma, you'd think to be true from our experience with delAll

lemma *Suc* (*occurs a* (*del1 a xs*)) = *occurs a xs*
— Wrong. Precondition needed.
quickcheck
⟨*proof*⟩

lemma *xs* ~= [] ⟶ *Suc* (*occurs a* (*del1 a xs*)) = *occurs a xs*
quickcheck
— Also wrong.
⟨*proof*⟩

lemma *0* < *occurs a xs* ⟶ *Suc* (*occurs a* (*del1 a xs*)) = *occurs a xs*
quickcheck
⟨*proof*⟩

primrec *replace* :: *'a* ⇒ *'a* ⇒ *'a list* ⇒ *'a list* **where**
replace a b [] = []
| *replace a b* (*x*#*xs*) = (*if* (*x*=*a*) *then* (*b*#(*replace a b xs*))
else (*x*#(*replace a b xs*)))

lemma *occurs a xs* = *occurs b* (*replace a b xs*)
quickcheck
— Wrong. Precondition needed.
⟨*proof*⟩

lemma *occurs b xs = 0 ∨ a=b ⟶ occurs a xs = occurs b (replace a b xs)*
quickcheck
⟨*proof*⟩

50.2 Trees

datatype *'a tree = Twig | Leaf 'a | Branch 'a tree 'a tree*

primrec *leaves :: 'a tree ⇒ 'a list* **where**
leaves Twig = []
| leaves (Leaf a) = [a]
| leaves (Branch l r) = (leaves l) @ (leaves r)

primrec *plant :: 'a list ⇒ 'a tree* **where**
plant [] = Twig
| plant (x#xs) = Branch (Leaf x) (plant xs)

primrec *mirror :: 'a tree ⇒ 'a tree* **where**
mirror (Twig) = Twig
| mirror (Leaf a) = Leaf a
| mirror (Branch l r) = Branch (mirror r) (mirror l)

theorem *plant (rev (leaves xt)) = mirror xt*
quickcheck
— Wrong!
⟨*proof*⟩

theorem *plant((leaves xt) @ (leaves yt)) = Branch xt yt*
quickcheck
— Wrong!
⟨*proof*⟩

datatype *'a ntree = Tip 'a | Node 'a 'a ntree 'a ntree*

primrec *inOrder :: 'a ntree ⇒ 'a list* **where**
inOrder (Tip a)= [a]
| inOrder (Node f x y) = (inOrder x)@[f]@(inOrder y)

primrec *root :: 'a ntree ⇒ 'a* **where**
root (Tip a) = a
| root (Node f x y) = f

theorem *hd (inOrder xt) = root xt*
quickcheck
— Wrong!
⟨*proof*⟩

end

51 A formalization of formal power series

theory *Formal-Power-Series*
imports *Main Fact Parity*
begin

51.1 The type of formal power series

typedef (**open**) *'a fps* = $\{f :: nat \Rightarrow 'a.\ True\}$
  morphisms *fps-nth Abs-fps*
  ⟨*proof*⟩

notation *fps-nth* (**infixl** $ 75)

lemma *expand-fps-eq*: $p = q \longleftrightarrow (\forall n.\ p\ \$\ n = q\ \$\ n)$
  ⟨*proof*⟩

lemma *fps-ext*: $(\bigwedge n.\ p\ \$\ n = q\ \$\ n) \Longrightarrow p = q$
  ⟨*proof*⟩

lemma *fps-nth-Abs-fps* [*simp*]: *Abs-fps f* $ *n* = *f n*
  ⟨*proof*⟩

Definition of the basic elements 0 and 1 and the basic operations of addition, negation and multiplication

instantiation *fps* :: (*zero*) *zero*
begin

definition *fps-zero-def*:
  0 = *Abs-fps* (λ*n*. *0*)

instance ⟨*proof*⟩
end

lemma *fps-zero-nth* [*simp*]: *0* $ *n* = *0*
  ⟨*proof*⟩

instantiation *fps* :: ({*one*,*zero*}) *one*
begin

definition *fps-one-def*:
  1 = *Abs-fps* (λ*n*. *if n* = *0 then 1 else 0*)

instance ⟨*proof*⟩
end

lemma *fps-one-nth* [*simp*]: *1* $ *n* = (*if n* = *0 then 1 else 0*)

⟨*proof*⟩

instantiation *fps* :: (*plus*) *plus*
begin

definition *fps-plus-def*:
op + = (λ*f g*. *Abs-fps* (λ*n*. *f* $ *n* + *g* $ *n*))

instance ⟨*proof*⟩
end

lemma *fps-add-nth* [*simp*]: (*f* + *g*) $ *n* = *f* $ *n* + *g* $ *n*
⟨*proof*⟩

instantiation *fps* :: (*minus*) *minus*
begin

definition *fps-minus-def*:
op − = (λ*f g*. *Abs-fps* (λ*n*. *f* $ *n* − *g* $ *n*))

instance ⟨*proof*⟩
end

lemma *fps-sub-nth* [*simp*]: (*f* − *g*) $ *n* = *f* $ *n* − *g* $ *n*
⟨*proof*⟩

instantiation *fps* :: (*uminus*) *uminus*
begin

definition *fps-uminus-def*:
uminus = (λ*f*. *Abs-fps* (λ*n*. − (*f* $ *n*)))

instance ⟨*proof*⟩
end

lemma *fps-neg-nth* [*simp*]: (− *f*) $ *n* = − (*f* $ *n*)
⟨*proof*⟩

instantiation *fps* :: ({*comm-monoid-add*, *times*}) *times*
begin

definition *fps-times-def*:
op ∗ = (λ*f g*. *Abs-fps* (λ*n*. ∑ *i*=*0*..*n*. *f* $ *i* ∗ *g* $ (*n* − *i*)))

instance ⟨*proof*⟩
end

lemma *fps-mult-nth*: (*f* ∗ *g*) $ *n* = (∑ *i*=*0*..*n*. *f*$*i* ∗ *g*$(*n* − *i*))
⟨*proof*⟩

declare *atLeastAtMost-iff*[*presburger*]
declare *Bex-def*[*presburger*]
declare *Ball-def*[*presburger*]

lemma *mult-delta-left*:
fixes *x y* :: *'a*::*mult-zero*
shows (*if b then x else 0*) ∗ *y* = (*if b then x* ∗ *y else 0*)
⟨*proof*⟩

lemma *mult-delta-right*:
fixes *x y* :: *'a*::*mult-zero*
shows *x* ∗ (*if b then y else 0*) = (*if b then x* ∗ *y else 0*)
⟨*proof*⟩

lemma *cond-value-iff*: *f* (*if b then x else y*) = (*if b then f x else f y*)
⟨*proof*⟩
lemma *cond-application-beta*: (*if b then f else g*) *x* = (*if b then f x else g x*)
⟨*proof*⟩

51.2 Formal power series form a commutative ring with unity, if the range of sequences they represent is a commutative ring with unity

instance *fps* :: (*semigroup-add*) *semigroup-add*
⟨*proof*⟩

instance *fps* :: (*ab-semigroup-add*) *ab-semigroup-add*
⟨*proof*⟩

lemma *fps-mult-assoc-lemma*:
fixes *k* :: *nat* **and** *f* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ *'a*::*comm-monoid-add*
shows (∑ *j*=*0*..*k*. ∑ *i*=*0*..*j*. *f i* (*j* − *i*) (*n* − *j*)) =
(∑ *j*=*0*..*k*. ∑ *i*=*0*..*k* − *j*. *f j i* (*n* − *j* − *i*))
⟨*proof*⟩

instance *fps* :: (*semiring-0*) *semigroup-mult*
⟨*proof*⟩

lemma *fps-mult-commute-lemma*:
fixes *n* :: *nat* **and** *f* :: *nat* ⇒ *nat* ⇒ *'a*::*comm-monoid-add*
shows (∑ *i*=*0*..*n*. *f i* (*n* − *i*)) = (∑ *i*=*0*..*n*. *f* (*n* − *i*) *i*)
⟨*proof*⟩

instance *fps* :: (*comm-semiring-0*) *ab-semigroup-mult*
⟨*proof*⟩

instance *fps* :: (*monoid-add*) *monoid-add*
⟨*proof*⟩

instance *fps* :: (*comm-monoid-add*) *comm-monoid-add*
⟨*proof*⟩

instance *fps* :: (*semiring-1*) *monoid-mult*
⟨*proof*⟩

instance *fps* :: (*cancel-semigroup-add*) *cancel-semigroup-add*
⟨*proof*⟩

instance *fps* :: (*cancel-ab-semigroup-add*) *cancel-ab-semigroup-add*
⟨*proof*⟩

instance *fps* :: (*cancel-comm-monoid-add*) *cancel-comm-monoid-add* ⟨*proof*⟩

instance *fps* :: (*group-add*) *group-add*
⟨*proof*⟩

instance *fps* :: (*ab-group-add*) *ab-group-add*
⟨*proof*⟩

instance *fps* :: (*zero-neq-one*) *zero-neq-one*
⟨*proof*⟩

instance *fps* :: (*semiring-0*) *semiring*
⟨*proof*⟩

instance *fps* :: (*semiring-0*) *semiring-0*
⟨*proof*⟩

instance *fps* :: (*semiring-0-cancel*) *semiring-0-cancel* ⟨*proof*⟩

51.3 Selection of the nth power of the implicit variable in the infinite sum

lemma *fps-nonzero-nth*: $f \neq 0 \longleftrightarrow (\exists\ n.\ f\ \$n \neq 0)$
⟨*proof*⟩

lemma *fps-nonzero-nth-minimal*:
$f \neq 0 \longleftrightarrow (\exists n.\ f\ \$\ n \neq 0 \wedge (\forall m{<}n.\ f\ \$\ m = 0))$
⟨*proof*⟩

lemma *fps-eq-iff*: $f = g \longleftrightarrow (\forall n.\ f\ \$\ n = g\ \$n)$
⟨*proof*⟩

lemma *fps-setsum-nth*: $(setsum\ f\ S)\ \$\ n = setsum\ (\lambda k.\ (f\ k)\ \$\ n)\ S$
⟨*proof*⟩

51.4 Injection of the basic ring elements and multiplication by scalars

definition
fps-const c = Abs-fps (λn. if n = 0 then c else 0)

lemma *fps-nth-fps-const* [*simp*]: *fps-const c $ n = (if n = 0 then c else 0)*
⟨*proof*⟩

lemma *fps-const-0-eq-0* [*simp*]: *fps-const 0 = 0*
⟨*proof*⟩

lemma *fps-const-1-eq-1* [*simp*]: *fps-const 1 = 1*
⟨*proof*⟩

lemma *fps-const-neg* [*simp*]: *− (fps-const (c::'a::ring)) = fps-const (− c)*
⟨*proof*⟩

lemma *fps-const-add* [*simp*]: *fps-const (c::'a::monoid-add) + fps-const d = fps-const (c + d)*
⟨*proof*⟩

lemma *fps-const-mult*[*simp*]: *fps-const (c::'a::ring) ∗ fps-const d = fps-const (c ∗ d)*
⟨*proof*⟩

lemma *fps-const-add-left*: *fps-const (c::'a::monoid-add) + f = Abs-fps (λn. if n = 0 then c + f$0 else f$n)*
⟨*proof*⟩

lemma *fps-const-add-right*: *f + fps-const (c::'a::monoid-add) = Abs-fps (λn. if n = 0 then f$0 + c else f$n)*
⟨*proof*⟩

lemma *fps-const-mult-left*: *fps-const (c::'a::semiring-0) ∗ f = Abs-fps (λn. c ∗ f$n)*
⟨*proof*⟩

lemma *fps-const-mult-right*: *f ∗ fps-const (c::'a::semiring-0) = Abs-fps (λn. f$n ∗ c)*
⟨*proof*⟩

lemma *fps-mult-left-const-nth* [*simp*]: *(fps-const (c::'a::semiring-1) ∗ f)$n = c∗ f$n*
⟨*proof*⟩

lemma *fps-mult-right-const-nth* [*simp*]: *(f ∗ fps-const (c::'a::semiring-1))$n = f$n ∗ c*
⟨*proof*⟩

51.5 Formal power series form an integral domain

instance *fps* :: (*ring*) *ring* ⟨*proof*⟩

instance *fps* :: (*ring-1*) *ring-1*
⟨*proof*⟩

instance *fps* :: (*comm-ring-1*) *comm-ring-1*
⟨*proof*⟩

instance *fps* :: (*ring-no-zero-divisors*) *ring-no-zero-divisors*
⟨*proof*⟩

instance *fps* :: (*idom*) *idom* ⟨*proof*⟩

instantiation *fps* :: (*comm-ring-1*) *number-ring*
begin
definition *number-of-fps-def*: (*number-of k*::'*a fps*) = *of-int k*

instance
⟨*proof*⟩
end

51.6 Inverses of formal power series

declare *setsum-cong*[*fundef-cong*]

instantiation *fps* :: ({*comm-monoid-add*,*inverse*, *times*, *uminus*}) *inverse*
begin

fun *natfun-inverse*:: '*a fps* ⇒ *nat* ⇒ '*a* **where**
natfun-inverse f 0 = *inverse* (*f*$*0*)
| *natfun-inverse f n* = − *inverse* (*f*$*0*) ∗ *setsum* (λ*i*. *f*$*i* ∗ *natfun-inverse f* (*n* − *i*)) {*1*..*n*}

definition *fps-inverse-def*:
inverse f = (*if f*$*0* = *0 then 0 else Abs-fps* (*natfun-inverse f*))
definition *fps-divide-def*: *divide* = (λ(*f*::'*a fps*) *g*. *f* ∗ *inverse g*)
instance ⟨*proof*⟩
end

lemma *fps-inverse-zero*[*simp*]:
inverse (*0* :: '*a*::{*comm-monoid-add*,*inverse*, *times*, *uminus*} *fps*) = *0*
⟨*proof*⟩

lemma *fps-inverse-one*[*simp*]: *inverse* (*1* :: '*a*::{*division-ring*,*zero-neq-one*} *fps*) = *1*
⟨*proof*⟩

instance *fps* :: ({*comm-monoid-add*,*inverse*, *times*, *uminus*}) *division-by-zero*
⟨*proof*⟩

lemma *inverse-mult-eq-1*[*intro*]: **assumes** *f0*: $f\$0 \neq (0::'a::field)$
shows *inverse f* ∗ *f* = *1*
⟨*proof*⟩

lemma *fps-inverse-0-iff*[*simp*]: $(inverse\ f)\$0 = (0::'a::division\text{-}ring) \longleftrightarrow f\$0 = 0$
⟨*proof*⟩

lemma *fps-inverse-eq-0-iff*[*simp*]: *inverse f* = (*0*:: ('*a*::*field*) *fps*) ⟷ *f* $0 = *0*
⟨*proof*⟩

lemma *fps-inverse-idempotent*[*intro*]: **assumes** *f0*: $f\$0 \neq (0::'a::field)$
shows *inverse* (*inverse f*) = *f*
⟨*proof*⟩

lemma *fps-inverse-unique*: **assumes** *f0*: $f\$0 \neq (0::'a::field)$ **and** *fg*: *f*∗*g* = *1*
shows *inverse f* = *g*
⟨*proof*⟩

lemma *fps-inverse-gp*: *inverse* (*Abs-fps*(λ*n*. (*1*::'*a*::*field*)))
= *Abs-fps* (λ*n*. *if n*= *0 then 1 else if n*=*1 then* − *1 else 0*)
⟨*proof*⟩

51.7 Formal Derivatives, and the MacLaurin theorem around 0

definition *fps-deriv f* = *Abs-fps* (λ*n*. *of-nat* (*n* + *1*) ∗ *f* $ (*n* + *1*))

lemma *fps-deriv-nth*[*simp*]: *fps-deriv f* $ *n* = *of-nat* (*n* +*1*) ∗ *f* $ (*n*+*1*) ⟨*proof*⟩

lemma *fps-deriv-linear*[*simp*]: *fps-deriv* (*fps-const* (*a*::'*a*::*comm-semiring-1*) ∗ *f* + *fps-const b* ∗ *g*) = *fps-const a* ∗ *fps-deriv f* + *fps-const b* ∗ *fps-deriv g*
⟨*proof*⟩

lemma *fps-deriv-mult*[*simp*]:
fixes *f* :: ('*a* :: *comm-ring-1*) *fps*
shows *fps-deriv* (*f* ∗ *g*) = *f* ∗ *fps-deriv g* + *fps-deriv f* ∗ *g*
⟨*proof*⟩

lemma *fps-deriv-neg*[*simp*]: *fps-deriv* (− (*f*:: ('*a*:: *comm-ring-1*) *fps*)) = − (*fps-deriv f*)
⟨*proof*⟩
lemma *fps-deriv-add*[*simp*]: *fps-deriv* ((*f*:: ('*a*::*comm-ring-1*) *fps*) + *g*) = *fps-deriv f* + *fps-deriv g*
⟨*proof*⟩

lemma *fps-deriv-sub*[*simp*]: *fps-deriv* ((*f*:: ('*a*::*comm-ring-1*) *fps*) − *g*) = *fps-deriv* *f* − *fps-deriv* *g*
⟨*proof*⟩

lemma *fps-deriv-const*[*simp*]: *fps-deriv* (*fps-const* *c*) = *0*
⟨*proof*⟩

lemma *fps-deriv-mult-const-left*[*simp*]: *fps-deriv* (*fps-const* (*c*::'*a*::*comm-ring-1*) ∗ *f*) = *fps-const* *c* ∗ *fps-deriv* *f*
⟨*proof*⟩

lemma *fps-deriv-0*[*simp*]: *fps-deriv* *0* = *0*
⟨*proof*⟩

lemma *fps-deriv-1*[*simp*]: *fps-deriv* *1* = *0*
⟨*proof*⟩

lemma *fps-deriv-mult-const-right*[*simp*]: *fps-deriv* (*f* ∗ *fps-const* (*c*::'*a*::*comm-ring-1*)) = *fps-deriv* *f* ∗ *fps-const* *c*
⟨*proof*⟩

lemma *fps-deriv-setsum*: *fps-deriv* (*setsum* *f* *S*) = *setsum* (λ*i*. *fps-deriv* (*f* *i* :: ('*a*::*comm-ring-1*) *fps*)) *S*
⟨*proof*⟩

lemma *fps-deriv-eq-0-iff*[*simp*]: *fps-deriv* *f* = *0* ⟷ (*f* = *fps-const* (*f*$*0* :: '*a*::{*idom*,*semiring-char-0*}))
⟨*proof*⟩

lemma *fps-deriv-eq-iff*:
fixes *f*:: ('*a*::{*idom*,*semiring-char-0*}) *fps*
shows *fps-deriv* *f* = *fps-deriv* *g* ⟷ (*f* = *fps-const*(*f*$*0* − *g*$*0*) + *g*)
⟨*proof*⟩

lemma *fps-deriv-eq-iff-ex*: (*fps-deriv* *f* = *fps-deriv* *g*) ⟷ (∃ (*c*::'*a*::{*idom*,*semiring-char-0*}). *f* = *fps-const* *c* + *g*)
⟨*proof*⟩

fun *fps-nth-deriv* :: *nat* ⇒ ('*a*::*semiring-1*) *fps* ⇒ '*a* *fps* **where**
fps-nth-deriv *0* *f* = *f*
| *fps-nth-deriv* (*Suc* *n*) *f* = *fps-nth-deriv* *n* (*fps-deriv* *f*)

lemma *fps-nth-deriv-commute*: *fps-nth-deriv* (*Suc* *n*) *f* = *fps-deriv* (*fps-nth-deriv* *n* *f*)
⟨*proof*⟩

lemma *fps-nth-deriv-linear*[*simp*]: *fps-nth-deriv* *n* (*fps-const* (*a*::'*a*::*comm-semiring-1*) ∗ *f* + *fps-const* *b* ∗ *g*) = *fps-const* *a* ∗ *fps-nth-deriv* *n* *f* + *fps-const* *b* ∗ *fps-nth-deriv* *n* *g*

⟨*proof*⟩

lemma *fps-nth-deriv-neg*[*simp*]: *fps-nth-deriv n* (− (*f*:: ('*a*:: *comm-ring-1*) *fps*)) = − (*fps-nth-deriv n f*)
⟨*proof*⟩

lemma *fps-nth-deriv-add*[*simp*]: *fps-nth-deriv n* ((*f*:: ('*a*::*comm-ring-1*) *fps*) + *g*) = *fps-nth-deriv n f* + *fps-nth-deriv n g*
⟨*proof*⟩

lemma *fps-nth-deriv-sub*[*simp*]: *fps-nth-deriv n* ((*f*:: ('*a*::*comm-ring-1*) *fps*) − *g*) = *fps-nth-deriv n f* − *fps-nth-deriv n g*
⟨*proof*⟩

lemma *fps-nth-deriv-0*[*simp*]: *fps-nth-deriv n 0* = *0*
⟨*proof*⟩

lemma *fps-nth-deriv-1*[*simp*]: *fps-nth-deriv n 1* = (*if n* = *0 then 1 else 0*)
⟨*proof*⟩

lemma *fps-nth-deriv-const*[*simp*]: *fps-nth-deriv n* (*fps-const c*) = (*if n* = *0 then fps-const c else 0*)
⟨*proof*⟩

lemma *fps-nth-deriv-mult-const-left*[*simp*]: *fps-nth-deriv n* (*fps-const* (*c*::'*a*::*comm-ring-1*) ∗ *f*) = *fps-const c* ∗ *fps-nth-deriv n f*
⟨*proof*⟩

lemma *fps-nth-deriv-mult-const-right*[*simp*]: *fps-nth-deriv n* (*f* ∗ *fps-const* (*c*::'*a*::*comm-ring-1*)) = *fps-nth-deriv n f* ∗ *fps-const c*
⟨*proof*⟩

lemma *fps-nth-deriv-setsum*: *fps-nth-deriv n* (*setsum f S*) = *setsum* (λ*i*. *fps-nth-deriv n* (*f i* :: ('*a*::*comm-ring-1*) *fps*)) *S*
⟨*proof*⟩

lemma *fps-deriv-maclauren-0*: (*fps-nth-deriv k* (*f*:: ('*a*::*comm-semiring-1*) *fps*)) $ *0* = *of-nat* (*fact k*) ∗ *f*$(*k*)
⟨*proof*⟩

51.8 Powers

instantiation *fps* :: (*semiring-1*) *power*
begin

fun *fps-pow* :: *nat* ⇒ '*a fps* ⇒ '*a fps* **where**
fps-pow 0 f = *1*
| *fps-pow* (*Suc n*) *f* = *f* ∗ *fps-pow n f*

definition *fps-power-def*: *power* (*f*::*'a fps*) *n* = *fps-pow n f*
instance ⟨*proof*⟩
end

instantiation *fps* :: (*comm-ring-1*) *recpower*
begin
instance
⟨*proof*⟩
end

lemma *fps-power-zeroth-eq-one*: *a*$*0* =*1* ⟹ *a*^*n* $ *0* = (*1*::*'a*::*semiring-1*)
⟨*proof*⟩

lemma *fps-power-first-eq*: (*a*:: *'a*::*comm-ring-1 fps*)$*0* =*1* ⟹ *a*^*n* $ *1* = *of-nat n* ∗ *a*$*1*
⟨*proof*⟩

lemma *startsby-one-power*:*a* $ *0* = (*1*::*'a*::*comm-ring-1*) ⟹ *a*^*n* $ *0* = *1*
⟨*proof*⟩

lemma *startsby-zero-power*:*a* $*0* = (*0*::*'a*::*comm-ring-1*) ⟹ *n* > *0* ⟹ *a*^*n* $*0* = *0*
⟨*proof*⟩

lemma *startsby-power*:*a* $*0* = (*v*::*'a*::{*comm-ring-1*, *recpower*}) ⟹ *a*^*n* $*0* = *v*^*n*
⟨*proof*⟩

lemma *startsby-zero-power-iff* [*simp*]:
a^*n* $*0* = (*0*::*'a*::{*idom*, *recpower*}) ⟷ (*n* ≠ *0* ∧ *a*$*0* = *0*)
⟨*proof*⟩

lemma *startsby-zero-power-prefix*:
assumes *a0*: *a* $*0* = (*0*::*'a*::*idom*)
shows ∀ *n* < *k*. *a* ^ *k* $ *n* = *0*
⟨*proof*⟩

lemma *startsby-zero-setsum-depends*:
assumes *a0*: *a* $*0* = (*0*::*'a*::*idom*) **and** *kn*: *n* ≥ *k*
shows *setsum* (λ*i*. (*a* ^ *i*)$*k*) {*0* .. *n*} = *setsum* (λ*i*. (*a* ^ *i*)$*k*) {*0* .. *k*}
⟨*proof*⟩

lemma *startsby-zero-power-nth-same*: **assumes** *a0*: *a*$*0* = (*0*::*'a*::{*recpower*, *idom*})
shows *a*^*n* $ *n* = (*a*$*1*) ^ *n*
⟨*proof*⟩

lemma *fps-inverse-power*:
fixes *a* :: (*'a*::{*field*, *recpower*}) *fps*
shows *inverse* (*a*^*n*) = *inverse a* ^ *n*

⟨*proof*⟩

lemma *fps-deriv-power*: *fps-deriv* (*a* ^ *n*) = *fps-const* (*of-nat n* :: *'a*:: *comm-ring-1*) ∗ *fps-deriv a* ∗ *a* ^ (*n* − *1*)
⟨*proof*⟩

lemma *fps-inverse-deriv*:
fixes *a*:: (*'a* :: *field*) *fps*
assumes *a0*: *a*$*0* ≠ *0*
shows *fps-deriv* (*inverse a*) = − *fps-deriv a* ∗ *inverse a* ^ *2*
⟨*proof*⟩

lemma *fps-inverse-mult*:
fixes *a*::(*'a* :: *field*) *fps*
shows *inverse* (*a* ∗ *b*) = *inverse a* ∗ *inverse b*
⟨*proof*⟩

lemma *fps-inverse-deriv'*:
fixes *a*:: (*'a* :: *field*) *fps*
assumes *a0*: *a*$*0* ≠ *0*
shows *fps-deriv* (*inverse a*) = − *fps-deriv a* / *a* ^ *2*
⟨*proof*⟩

lemma *inverse-mult-eq-1'*: **assumes** *f0*: *f*$*0* ≠ (*0*::*'a*::*field*)
shows *f* ∗ *inverse f* = *1*
⟨*proof*⟩

lemma *fps-divide-deriv*: **fixes** *a*:: (*'a* :: *field*) *fps*
assumes *a0*: *b*$*0* ≠ *0*
shows *fps-deriv* (*a* / *b*) = (*fps-deriv a* ∗ *b* − *a* ∗ *fps-deriv b*) / *b* ^ *2*
⟨*proof*⟩

51.9 The eXtractor series X

lemma *minus-one-power-iff*: (− (*1*::*'a* :: {*recpower*, *comm-ring-1*})) ^ *n* = (*if even n then 1 else* − *1*)
⟨*proof*⟩

definition *X* = *Abs-fps* (*λn*. *if n* = *1 then 1 else 0*)

lemma *fps-inverse-gp'*: *inverse* (*Abs-fps*(*λn*. (*1*::*'a*::*field*)))
= *1* − *X*
⟨*proof*⟩

lemma *X-mult-nth*[*simp*]: (*X* ∗ (*f* :: (*'a*::*semiring-1*) *fps*)) $*n* = (*if n* = *0 then 0 else f* $ (*n* − *1*))
⟨*proof*⟩

lemma *X-mult-right-nth*[*simp*]: ((*f* :: (*'a*::*comm-semiring-1*) *fps*) ∗ *X*) $*n* = (*if n*

$= 0$ *then* 0 *else* $f\ \$\ (n - 1))$
⟨*proof*⟩

lemma *X-power-iff*: $X\hat{}k = Abs\text{-}fps\ (\lambda n.\ if\ n = k\ then\ (1::'a::comm\text{-}ring\text{-}1)\ else\ 0)$
⟨*proof*⟩

lemma *X-power-mult-nth*: $(X\hat{}k * (f :: ('a::comm\text{-}ring\text{-}1)\ fps))\ \$n = (if\ n < k\ then\ 0\ else\ f\ \$\ (n - k))$
⟨*proof*⟩

lemma *X-power-mult-right-nth*: $((f :: ('a::comm\text{-}ring\text{-}1)\ fps) * X\hat{}k)\ \$n = (if\ n < k\ then\ 0\ else\ f\ \$\ (n - k))$
⟨*proof*⟩
lemma *fps-deriv-X*[*simp*]: $fps\text{-}deriv\ X = 1$
⟨*proof*⟩

lemma *fps-nth-deriv-X*[*simp*]: $fps\text{-}nth\text{-}deriv\ n\ X = (if\ n = 0\ then\ X\ else\ if\ n=1\ then\ 1\ else\ 0)$
⟨*proof*⟩

lemma *X-nth*[*simp*]: $X\$n = (if\ n = 1\ then\ 1\ else\ 0)$ ⟨*proof*⟩
lemma *X-power-nth*[*simp*]: $(X\hat{}k)\ \$n = (if\ n = k\ then\ 1\ else\ (0::'a::comm\text{-}ring\text{-}1))$
⟨*proof*⟩

lemma *fps-inverse-X-plus1*:
$inverse\ (1 + X) = Abs\text{-}fps\ (\lambda n.\ (-\ (1::'a::\{recpower, field\}))\ \hat{}\ n)$ (**is** $- = ?r$)
⟨*proof*⟩

51.10 Integration

definition *fps-integral a a0* $= Abs\text{-}fps\ (\lambda n.\ if\ n = 0\ then\ a0\ else\ (a\$(n - 1)\ /\ of\text{-}nat\ n))$

lemma *fps-deriv-fps-integral*: $fps\text{-}deriv\ (fps\text{-}integral\ a\ (a0 :: 'a :: \{field, ring\text{-}char\text{-}0\}))$ $= a$
⟨*proof*⟩

lemma *fps-integral-linear*: $fps\text{-}integral\ (fps\text{-}const\ (a::'a::\{field, ring\text{-}char\text{-}0\}) * f + fps\text{-}const\ b * g)\ (a*a0 + b*b0) = fps\text{-}const\ a * fps\text{-}integral\ f\ a0 + fps\text{-}const\ b * fps\text{-}integral\ g\ b0$ (**is** $?l = ?r$)
⟨*proof*⟩

51.11 Composition of FPSs

definition $fps\text{-}compose :: ('a::semiring\text{-}1)\ fps \Rightarrow 'a\ fps \Rightarrow 'a\ fps$ (**infixl** *oo* 55)
where
fps-compose-def: $a\ oo\ b = Abs\text{-}fps\ (\lambda n.\ setsum\ (\lambda i.\ a\$i * (b\hat{}i\$n))\ \{0..n\})$

lemma *fps-compose-nth*: $(a\ oo\ b)\$n = setsum\ (\lambda i.\ a\$i * (b\hat{}i\$n))\ \{0..n\}$ ⟨*proof*⟩

lemma *fps-compose-X*[*simp*]: *a oo X* = (*a* :: (*'a* :: *comm-ring-1*) *fps*)
⟨*proof*⟩

lemma *fps-const-compose*[*simp*]:
fps-const (*a*::*'a*::{*comm-ring-1*}) *oo b* = *fps-const* (*a*)
⟨*proof*⟩

lemma *X-fps-compose-startby0*[*simp*]: *a*$*0* = *0* ⟹ *X oo a* = (*a* :: (*'a* :: *comm-ring-1*) *fps*)
⟨*proof*⟩

51.12 Rules from Herbert Wilf's Generatingfunctionology

51.12.1 Rule 1

lemma *fps-power-mult-eq-shift*:
X^*Suc k* ∗ *Abs-fps* (λ*n*. *a* (*n* + *Suc k*)) = *Abs-fps a* − *setsum* (λ*i*. *fps-const* (*a i* :: *'a*:: *field*) ∗ *X*^*i*) {*0* .. *k*} (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

51.12.2 Rule 2

definition *XD* = *op* ∗ *X o fps-deriv*

lemma *XD-add*[*simp*]:*XD* (*a* + *b*) = *XD a* + *XD* (*b* :: (*'a*::*comm-ring-1*) *fps*)
⟨*proof*⟩

lemma *XD-mult-const*[*simp*]:*XD* (*fps-const* (*c*::*'a*::*comm-ring-1*) ∗ *a*) = *fps-const c* ∗ *XD a*
⟨*proof*⟩

lemma *XD-linear*[*simp*]: *XD* (*fps-const c* ∗ *a* + *fps-const d* ∗ *b*) = *fps-const c* ∗ *XD a* + *fps-const d* ∗ *XD* (*b* :: (*'a*::*comm-ring-1*) *fps*)
⟨*proof*⟩

lemma *XDN-linear*: (*XD*^*n*) (*fps-const c* ∗ *a* + *fps-const d* ∗ *b*) = *fps-const c* ∗ (*XD*^*n*) *a* + *fps-const d* ∗ (*XD*^*n*) (*b* :: (*'a*::*comm-ring-1*) *fps*)
⟨*proof*⟩

lemma *fps-mult-X-deriv-shift*: *X*∗ *fps-deriv a* = *Abs-fps* (λ*n*. *of-nat n*∗ *a*$*n*)
⟨*proof*⟩

lemma *fps-mult-XD-shift*: (*XD* ^*k*) (*a*:: (*'a*::{*comm-ring-1*, *recpower*, *ring-char-0*}) *fps*) = *Abs-fps* (λ*n*. (*of-nat n* ^ *k*) ∗ *a*$*n*)
⟨*proof*⟩

51.12.3 Rule 3 is trivial and is given by *fps-times-def*

51.12.4 Rule 5 — summation and "division" by (1 - X)

lemma *fps-divide-X-minus1-setsum-lemma*:
$a = ((1::('a::comm\text{-}ring\text{-}1)\ fps) - X) * Abs\text{-}fps\ (\lambda n.\ setsum\ (\lambda i.\ a\ \$\ i)\ \{0..n\})$
⟨*proof*⟩

lemma *fps-divide-X-minus1-setsum*:
$a\ /((1::('a::field)\ fps) - X) = Abs\text{-}fps\ (\lambda n.\ setsum\ (\lambda i.\ a\ \$\ i)\ \{0..n\})$
⟨*proof*⟩

51.12.5 Rule 4 in its more general form: generalizes Rule 3 for an arbitrary finite product of FPS, also the relvant instance of powers of a FPS

definition $natpermute\ n\ k = \{l::\ nat\ list.\ length\ l = k \wedge foldl\ op\ +\ 0\ l = n\}$

lemma *natlist-trivial-1*: $natpermute\ n\ 1 = \{[n]\}$
⟨*proof*⟩

lemma *foldl-add-start0*:
$foldl\ op\ +\ x\ xs = x + foldl\ op\ +\ (0::nat)\ xs$
⟨*proof*⟩

lemma *foldl-add-append*: $foldl\ op\ +\ (x::nat)\ (xs@ys) = foldl\ op\ +\ x\ xs + foldl\ op\ +\ 0\ ys$
⟨*proof*⟩

lemma *foldl-add-setsum*: $foldl\ op\ +\ (x::nat)\ xs = x + setsum\ (nth\ xs)\ \{0..<length\ xs\}$
⟨*proof*⟩

lemma *append-natpermute-less-eq*:
assumes h: $xs@ys \in natpermute\ n\ k$ **shows** $foldl\ op\ +\ 0\ xs \leq n$ **and** $foldl\ op\ +\ 0\ ys \leq n$
⟨*proof*⟩

lemma *natpermute-split*:
assumes mn: $h \leq k$
shows $natpermute\ n\ k = (\bigcup m \in\{0..n\}.\ \{l1\ @\ l2\ |l1\ l2.\ l1 \in natpermute\ m\ h \wedge l2 \in natpermute\ (n - m)\ (k - h)\})$ (**is** $?L = ?R$ **is** $?L = (\bigcup m \in\{0..n\}.\ ?S\ m)$)
⟨*proof*⟩

lemma *natpermute-0*: $natpermute\ n\ 0 = (if\ n = 0\ then\ \{[]\}\ else\ \{\})$
⟨*proof*⟩
lemma *natpermute-0'*[*simp*]: $natpermute\ 0\ k = (if\ k = 0\ then\ \{[]\}\ else\ \{replicate\ k\ 0\})$
⟨*proof*⟩

lemma *natpermute-finite*: *finite* (*natpermute n k*)
⟨*proof*⟩

lemma *natpermute-contain-maximal*:
{*xs* ∈ *natpermute n* (*k*+*1*). *n* ∈ *set xs*} = *UNION* {*0* .. *k*} (λ*i*. {(*replicate* (*k*+*1*) *0*) [*i*:=*n*]})
(**is** *?A* = *?B*)
⟨*proof*⟩

lemma *fps-setprod-nth*:
fixes *m* :: *nat* **and** *a* :: *nat* ⇒ ('*a*::*comm-ring-1*) *fps*
shows (*setprod a* {*0* .. *m*})$*n* = *setsum* (λ*v*. *setprod* (λ*j*. (*a j*) $ (*v*!*j*)) {*0*..*m*}) (*natpermute n* (*m*+*1*))
(**is** *?P m n*)
⟨*proof*⟩

The special form for powers

lemma *fps-power-nth-Suc*:
fixes *m* :: *nat* **and** *a* :: ('*a*::*comm-ring-1*) *fps*
shows (*a* ^ *Suc m*)$*n* = *setsum* (λ*v*. *setprod* (λ*j*. *a* $ (*v*!*j*)) {*0*..*m*}) (*natpermute n* (*m*+*1*))
⟨*proof*⟩
lemma *fps-power-nth*:
fixes *m* :: *nat* **and** *a* :: ('*a*::*comm-ring-1*) *fps*
shows (*a* ^*m*)$*n* = (*if m*=*0 then 1*$*n else setsum* (λ*v*. *setprod* (λ*j*. *a* $ (*v*!*j*)) {*0*..*m* − *1*}) (*natpermute n m*))
⟨*proof*⟩

lemma *fps-nth-power-0*:
fixes *m* :: *nat* **and** *a* :: ('*a*::{*comm-ring-1*, *recpower*}) *fps*
shows (*a* ^*m*)$*0* = (*a*$*0*) ^ *m*
⟨*proof*⟩

lemma *fps-compose-inj-right*:
assumes *a0*: *a*$*0* = (*0*::'*a*::{*recpower*,*idom*})
and *a1*: *a*$*1* ≠ *0*
shows (*b oo a* = *c oo a*) ⟷ *b* = *c* (**is** *?lhs* ⟷*?rhs*)
⟨*proof*⟩

51.13 Radicals

declare *setprod-cong*[*fundef-cong*]
function *radical* :: (*nat* ⇒ '*a* ⇒ '*a*) ⇒ *nat* ⇒ ('*a*::{*field*, *recpower*}) *fps* ⇒ *nat* ⇒ '*a* **where**
radical r 0 a 0 = *1*
| *radical r 0 a* (*Suc n*) = *0*
| *radical r* (*Suc k*) *a 0* = *r* (*Suc k*) (*a*$*0*)

| *radical r (Suc k) a (Suc n) = (a$ Suc n − setsum (λxs. setprod (λj. radical r (Suc k) a (xs ! j)) {0..k}) {xs. xs ∈ natpermute (Suc n) (Suc k) ∧ Suc n ∉ set xs}) / (of-nat (Suc k) ∗ (radical r (Suc k) a 0) ^k)*
⟨*proof*⟩

termination *radical*
⟨*proof*⟩

definition *fps-radical r n a = Abs-fps (radical r n a)*

lemma *fps-radical0*[*simp*]: *fps-radical r 0 a = 1*
⟨*proof*⟩

lemma *fps-radical-nth-0*[*simp*]: *fps-radical r n a $ 0 = (if n=0 then 1 else r n (a$0))*
⟨*proof*⟩

lemma *fps-radical-power-nth*[*simp*]:
assumes *r*: *(r k (a$0)) ^ k = a$0*
shows *fps-radical r k a ^ k $ 0 = (if k = 0 then 1 else a$0)*
⟨*proof*⟩

lemma *natpermute-max-card*: **assumes** *n0*: *n≠0*
shows *card {xs ∈ natpermute n (k+1). n ∈ set xs} = k+1*
⟨*proof*⟩

lemma *power-radical*:
fixes *a*:: *'a* ::{*field*, *ring-char-0*, *recpower*} *fps*
assumes *r0*: *(r (Suc k) (a$0)) ^ Suc k = a$0* **and** *a0*: *a$0 ≠ 0*
shows *(fps-radical r (Suc k) a) ^ (Suc k) = a*
⟨*proof*⟩

lemma *eq-divide-imp'*: **assumes** *c0*: *(c::'a::field) ~= 0* **and** *eq*: *a ∗ c = b*
shows *a = b / c*
⟨*proof*⟩

lemma *radical-unique*:
assumes *r0*: *(r (Suc k) (b$0)) ^ Suc k = b$0*
and *a0*: *r (Suc k) (b$0 ::'a::{field, ring-char-0, recpower}) = a$0* **and** *b0*: *b$0 ≠ 0*
shows *a^(Suc k) = b ⟷ a = fps-radical r (Suc k) b*
⟨*proof*⟩

lemma *radical-power*:
assumes *r0*: *r (Suc k) ((a$0) ^ Suc k) = a$0*
and *a0*: *(a$0 ::'a::{field, ring-char-0, recpower}) ≠ 0*
shows *(fps-radical r (Suc k) (a ^ Suc k)) = a*
⟨*proof*⟩

lemma *fps-deriv-radical*:
fixes *a*:: *'a* ::{*field*, *ring-char-0*, *recpower*} *fps*
assumes *r0*: (*r* (*Suc k*) (*a$0*)) ^ *Suc k* = *a$0* **and** *a0*: *a$0* ≠ *0*
shows *fps-deriv* (*fps-radical r* (*Suc k*) *a*) = *fps-deriv a* / (*fps-const* (*of-nat* (*Suc k*)) * (*fps-radical r* (*Suc k*) *a*) ^ *k*)
⟨*proof*⟩

lemma *radical-mult-distrib*:
fixes *a*:: *'a* ::{*field*, *ring-char-0*, *recpower*} *fps*
assumes
ra0: *r* (*k*) (*a $ 0*) ^ *k* = *a $ 0*
and *rb0*: *r* (*k*) (*b $ 0*) ^ *k* = *b $ 0*
and *r0'*: *r* (*k*) ((*a* * *b*) $ *0*) = *r* (*k*) (*a $ 0*) * *r* (*k*) (*b $ 0*)
and *a0*: *a$0* ≠ *0*
and *b0*: *b$0* ≠ *0*
shows *fps-radical r* (*k*) (*a*b*) = *fps-radical r* (*k*) *a* * *fps-radical r* (*k*) (*b*)
⟨*proof*⟩

lemma *radical-inverse*:
fixes *a*:: *'a* ::{*field*, *ring-char-0*, *recpower*} *fps*
assumes
ra0: *r* (*k*) (*a $ 0*) ^ *k* = *a $ 0*
and *ria0*: *r* (*k*) (*inverse* (*a $ 0*)) = *inverse* (*r* (*k*) (*a $ 0*))
and *r1*: (*r* (*k*) *1*) = *1*
and *a0*: *a$0* ≠ *0*
shows *fps-radical r* (*k*) (*inverse a*) = *inverse* (*fps-radical r* (*k*) *a*)
⟨*proof*⟩

lemma *fps-divide-inverse*: (*a*::(*'a*::*field*) *fps*) / *b* = *a* * *inverse b*
⟨*proof*⟩

lemma *radical-divide*:
fixes *a*:: *'a* ::{*field*, *ring-char-0*, *recpower*} *fps*
assumes
ra0: *r k* (*a $ 0*) ^ *k* = *a $ 0*
and *rb0*: *r k* (*b $ 0*) ^ *k* = *b $ 0*
and *r1*: *r k 1* = *1*
and *rb0'*: *r k* (*inverse* (*b $ 0*)) = *inverse* (*r k* (*b $ 0*))
and *raib'*: *r k* (*a$0* / (*b$0*)) = *r k* (*a$0*) / *r k* (*b$0*)
and *a0*: *a$0* ≠ *0*
and *b0*: *b$0* ≠ *0*
shows *fps-radical r k* (*a*/*b*) = *fps-radical r k a* / *fps-radical r k b*
⟨*proof*⟩

51.14 Derivative of composition

lemma *fps-compose-deriv*:
fixes *a*:: (*'a*::*idom*) *fps*

assumes *b0*: *b$0 = 0*
shows *fps-deriv (a oo b) = ((fps-deriv a) oo b) * (fps-deriv b)*
⟨*proof*⟩

lemma *fps-mult-X-plus-1-nth*:
*((1+X)*a) $n = (if n = 0 then (a$n :: 'a::comm-ring-1) else a$n + a$(n − 1))*
⟨*proof*⟩

51.15 Finite FPS (i.e. polynomials) and X

lemma *fps-poly-sum-X*:
assumes *z*: *∀ i > n. a$i = (0::'a::comm-ring-1)*
shows *a = setsum (λi. fps-const (a$i) * X^i) {0..n}* (**is** *a = ?r*)
⟨*proof*⟩

51.16 Compositional inverses

fun *compinv* :: *'a fps ⇒ nat ⇒ 'a::{recpower,field}* **where**
compinv a 0 = X$0
| *compinv a (Suc n) = (X$ Suc n − setsum (λi. (compinv a i) * (a^i)$Suc n) {0 .. n}) / (a$1) ^ Suc n*

definition *fps-inv a = Abs-fps (compinv a)*

lemma *fps-inv*: **assumes** *a0*: *a$0 = 0* **and** *a1*: *a$1 ≠ 0*
shows *fps-inv a oo a = X*
⟨*proof*⟩

fun *gcompinv* :: *'a fps ⇒ 'a fps ⇒ nat ⇒ 'a::{recpower,field}* **where**
gcompinv b a 0 = b$0
| *gcompinv b a (Suc n) = (b$ Suc n − setsum (λi. (gcompinv b a i) * (a^i)$Suc n) {0 .. n}) / (a$1) ^ Suc n*

definition *fps-ginv b a = Abs-fps (gcompinv b a)*

lemma *fps-ginv*: **assumes** *a0*: *a$0 = 0* **and** *a1*: *a$1 ≠ 0*
shows *fps-ginv b a oo a = b*
⟨*proof*⟩

lemma *fps-inv-ginv*: *fps-inv = fps-ginv X*
⟨*proof*⟩

lemma *fps-compose-1*[*simp*]: *1 oo a = 1*
⟨*proof*⟩

lemma *fps-compose-0*[*simp*]: *0 oo a = 0*
⟨*proof*⟩

lemma *fps-pow-0*: *fps-pow n 0 = (if n = 0 then 1 else 0)*

⟨*proof*⟩

lemma *fps-compose-0-right*[*simp*]: *a oo 0 = fps-const* (*a*$*0*)
⟨*proof*⟩

lemma *fps-compose-add-distrib*: (*a* + *b*) *oo c* = (*a oo c*) + (*b oo c*)
⟨*proof*⟩

lemma *fps-compose-setsum-distrib*: (*setsum f S*) *oo a* = *setsum* (λ*i*. *f i oo a*) *S*
⟨*proof*⟩

lemma *convolution-eq*:
setsum (%*i*. *a* (*i* :: *nat*) ∗ *b* (*n* − *i*)) {*0* .. *n*} = *setsum* (%(*i*,*j*). *a i* ∗ *b j*) {(*i*,*j*). *i* <= *n* ∧ *j* ≤ *n* ∧ *i* + *j* = *n*}
⟨*proof*⟩

lemma *product-composition-lemma*:
assumes *c0*: *c*$*0* = (*0*::′*a*::*idom*) **and** *d0*: *d*$*0* = *0*
shows ((*a oo c*) ∗ (*b oo d*))$*n* = *setsum* (%(*k*,*m*). *a*$*k* ∗ *b*$*m* ∗ (*c* ^ *k* ∗ *d* ^ *m*) $ *n*) {(*k*,*m*). *k* + *m* ≤ *n*} (**is** *?l* = *?r*)
⟨*proof*⟩

lemma *product-composition-lemma*′:
assumes *c0*: *c*$*0* = (*0*::′*a*::*idom*) **and** *d0*: *d*$*0* = *0*
shows ((*a oo c*) ∗ (*b oo d*))$*n* = *setsum* (%*k*. *setsum* (%*m*. *a*$*k* ∗ *b*$*m* ∗ (*c* ^ *k* ∗ *d* ^ *m*) $ *n*) {*0*..*n*}) {*0*..*n*} (**is** *?l* = *?r*)
⟨*proof*⟩

lemma *setsum-pair-less-iff*:
setsum (%((*k*::*nat*),*m*). *a k* ∗ *b m* ∗ *c* (*k* + *m*)) {(*k*,*m*). *k* + *m* ≤ *n*} = *setsum* (%*s*. *setsum* (%*i*. *a i* ∗ *b* (*s* − *i*) ∗ *c s*) {*0*..*s*}) {*0*..*n*} (**is** *?l* = *?r*)
⟨*proof*⟩

lemma *fps-compose-mult-distrib-lemma*:
assumes *c0*: *c*$*0* = (*0*::′*a*::*idom*)
shows ((*a oo c*) ∗ (*b oo c*))$*n* = *setsum* (%*s*. *setsum* (%*i*. *a*$*i* ∗ *b*$(*s* − *i*) ∗ (*c* ^ *s*) $ *n*) {*0*..*s*}) {*0*..*n*} (**is** *?l* = *?r*)
⟨*proof*⟩

lemma *fps-compose-mult-distrib*:
assumes *c0*: *c*$*0* = (*0*::′*a*::*idom*)
shows (*a* ∗ *b*) *oo c* = (*a oo c*) ∗ (*b oo c*) (**is** *?l* = *?r*)
⟨*proof*⟩
lemma *fps-compose-setprod-distrib*:
assumes *c0*: *c*$*0* = (*0*::′*a*::*idom*)
shows (*setprod a S*) *oo c* = *setprod* (%*k*. *a k oo c*) *S* (**is** *?l* = *?r*)
⟨*proof*⟩

lemma *fps-compose-power*: **assumes** *c0*: $c\$0 = (0::'a::idom)$
shows $(a\ oo\ c)\hat{}n = a\hat{}n\ oo\ c$ (**is** *?l* = *?r*)
⟨*proof*⟩

lemma *fps-const-mult-apply-left*:
$fps\text{-}const\ c * (a\ oo\ b) = (fps\text{-}const\ c * a)\ oo\ b$
⟨*proof*⟩

lemma *fps-const-mult-apply-right*:
$(a\ oo\ b) * fps\text{-}const\ (c::'a::comm\text{-}semiring\text{-}1) = (fps\text{-}const\ c * a)\ oo\ b$
⟨*proof*⟩

lemma *fps-compose-assoc*:
assumes *c0*: $c\$0 = (0::'a::idom)$ **and** *b0*: $b\$0 = 0$
shows $a\ oo\ (b\ oo\ c) = a\ oo\ b\ oo\ c$ (**is** *?l* = *?r*)
⟨*proof*⟩

lemma *fps-X-power-compose*:
assumes *a0*: $a\$0=0$ **shows** $X\hat{}k\ oo\ a = (a::('a::idom\ fps))\hat{}k$ (**is** *?l* = *?r*)
⟨*proof*⟩

lemma *fps-inv-right*: **assumes** *a0*: $a\$0 = 0$ **and** *a1*: $a\$1 \neq 0$
shows $a\ oo\ fps\text{-}inv\ a = X$
⟨*proof*⟩

lemma *fps-inv-deriv*:
assumes *a0*:$a\$0 = (0::'a::\{recpower,field\})$ **and** *a1*: $a\$1 \neq 0$
shows $fps\text{-}deriv\ (fps\text{-}inv\ a) = inverse\ (fps\text{-}deriv\ a\ oo\ fps\text{-}inv\ a)$
⟨*proof*⟩

51.17 Elementary series

51.17.1 Exponential series

definition $E\ x = Abs\text{-}fps\ (\lambda n.\ x\hat{}n\ /\ of\text{-}nat\ (fact\ n))$

lemma *E-deriv*[*simp*]: $fps\text{-}deriv\ (E\ a) = fps\text{-}const\ (a::'a::\{field, recpower, ring\text{-}char\text{-}0\}) * E\ a$ (**is** *?l* = *?r*)
⟨*proof*⟩

lemma *E-unique-ODE*:
$fps\text{-}deriv\ a = fps\text{-}const\ c * a \longleftrightarrow a = fps\text{-}const\ (a\$0) * E\ (c :: 'a::\{field, ring\text{-}char\text{-}0, recpower\})$
(**is** *?lhs* ⟷ *?rhs*)
⟨*proof*⟩

lemma *E-add-mult*: $E\ (a + b) = E\ (a::'a::\{ring\text{-}char\text{-}0, field, recpower\}) * E\ b$ (**is** *?l* = *?r*)

⟨*proof*⟩

lemma *E-nth*[*simp*]: *E a $ n = a^n / of-nat (fact n)*
⟨*proof*⟩

lemma *E0*[*simp*]: *E (0::'a::{field, recpower}) = 1*
⟨*proof*⟩

lemma *E-neg*: *E (− a) = inverse (E (a::'a::{ring-char-0, field, recpower}))*
⟨*proof*⟩

lemma *E-nth-deriv*[*simp*]: *fps-nth-deriv n (E (a::'a::{field, recpower, ring-char-0})) = (fps-const a)^n * (E a)*
⟨*proof*⟩

lemma *fps-compose-uminus*: *− (a::'a::ring-1 fps) oo c = − (a oo c)*
⟨*proof*⟩

lemma *fps-compose-sub-distrib*:
shows *(a − b) oo (c::'a::ring-1 fps) = (a oo c) − (b oo c)*
⟨*proof*⟩

lemma *X-fps-compose*:*X oo a = Abs-fps (λn. if n = 0 then (0::'a::comm-ring-1) else a$n)*
⟨*proof*⟩

lemma *X-compose-E*[*simp*]: *X oo E (a::'a::{field, recpower}) = E a − 1*
⟨*proof*⟩

lemma *LE-compose*:
assumes *a*: *a≠0*
shows *fps-inv (E a − 1) oo (E a − 1) = X*
and *(E a − 1) oo fps-inv (E a − 1) = X*
⟨*proof*⟩

lemma *fps-const-inverse*:
inverse (fps-const (a::'a::{field, division-by-zero})) = fps-const (inverse a)
⟨*proof*⟩

lemma *inverse-one-plus-X*:
inverse (1 + X) = Abs-fps (λn. (− 1 ::'a::{field, recpower})^n)
(**is** *inverse ?l = ?r*)
⟨*proof*⟩

lemma *E-power-mult*: *(E (c::'a::{field,recpower,ring-char-0}))^n = E (of-nat n * c)*
⟨*proof*⟩

51.17.2 Logarithmic series

definition (*L*::'*a*::{*field*, *ring-char-0*,*recpower*} *fps*)
= *Abs-fps* (λ*n*. (− *1*) ^ *Suc n* / *of-nat n*)

lemma *fps-deriv-L*: *fps-deriv L* = *inverse* (*1* + *X*)
⟨*proof*⟩

lemma *L-nth*: *L* $ *n* = (− *1*) ^ *Suc n* / *of-nat n*
⟨*proof*⟩

lemma *L-E-inv*:
assumes *a*: *a*≠ (*0*::'*a*::{*field*,*division-by-zero*,*ring-char-0*,*recpower*})
shows *L* = *fps-const a* ∗ *fps-inv* (*E a* − *1*) (**is** *?l* = *?r*)
⟨*proof*⟩

51.17.3 Formal trigonometric functions

definition *fps-sin* (*c*::'*a*::{*field*, *recpower*, *ring-char-0*}) =
Abs-fps (λ*n*. *if even n then 0 else* (− *1*) ^((*n* − *1*) *div 2*) ∗ *c*^*n* /(*of-nat* (*fact n*)))

definition *fps-cos* (*c*::'*a*::{*field*, *recpower*, *ring-char-0*}) = *Abs-fps* (λ*n*. *if even n then* (− *1*) ^ (*n div 2*) ∗ *c*^*n* / (*of-nat* (*fact n*)) *else 0*)

lemma *fps-sin-deriv*:
fps-deriv (*fps-sin c*) = *fps-const c* ∗ *fps-cos c*
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *fps-cos-deriv*:
fps-deriv (*fps-cos c*) = *fps-const* (− *c*)∗ (*fps-sin c*)
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *fps-sin-cos-sum-of-squares*:
fps-cos c ^ *2* + *fps-sin c* ^ *2* = *1* (**is** *?lhs* = *1*)
⟨*proof*⟩

definition *fps-tan c* = *fps-sin c* / *fps-cos c*

lemma *fps-tan-deriv*: *fps-deriv*(*fps-tan c*) = *fps-const c*/ (*fps-cos c* ^ *2*)
⟨*proof*⟩

end

52 Some applications of formal power series and some properties over complex numbers

theory *Formal-Power-Series-Examples*
imports *Formal-Power-Series Binomial Complex*
begin

53 The generalized binomial theorem

lemma *gbinomial-theorem*:
((*a*::'*a*::{*ring-char-0*, *field*, *division-by-zero*, *recpower*})+*b*) ^ *n* = (∑ *k*=*0*..*n*. *of-nat* (*n choose k*) * *a*^*k* * *b*^(*n*−*k*))
⟨*proof*⟩

And the nat-form – also available from Binomial.thy

lemma *binomial-theorem*: (*a*+*b*) ^ *n* = (∑ *k*=*0*..*n*. (*n choose k*) * *a*^*k* * *b*^(*n*−*k*))
⟨*proof*⟩

54 The binomial series and Vandermonde's identity

definition *fps-binomial a* = *Abs-fps* (λ*n*. *a gchoose n*)

lemma *fps-binomial-nth*[*simp*]: *fps-binomial a* $ *n* = *a gchoose n*
⟨*proof*⟩

lemma *fps-binomial-ODE-unique*:
fixes *c* :: '*a*::{*field*, *recpower*,*ring-char-0*}
shows *fps-deriv a* = (*fps-const c* * *a*) / (*1* + *X*) ⟷ *a* = *fps-const* (*a*$*0*) * *fps-binomial c*
(**is** *?lhs* ⟷ *?rhs*)
⟨*proof*⟩

lemma *fps-binomial-deriv*: *fps-deriv* (*fps-binomial c*) = *fps-const c* * *fps-binomial c* / (*1* + *X*)
⟨*proof*⟩

lemma *fps-binomial-add-mult*: *fps-binomial* (*c*+*d*) = *fps-binomial c* * *fps-binomial d* (**is** *?l* = *?r*)
⟨*proof*⟩

lemma *fps-minomial-minus-one*: *fps-binomial* (− *1*) = *inverse* (*1* + *X*)
(**is** *?l* = *inverse ?r*)
⟨*proof*⟩

lemma *gbinomial-Vandermond*: *setsum* (λ*k*. (*a gchoose k*) * (*b gchoose* (*n* − *k*))) {*0*..*n*} = (*a* + *b*) *gchoose n*

⟨*proof*⟩

lemma *binomial-Vandermond*: *setsum* (λ*k*. (*a choose k*) ∗ (*b choose* (*n* − *k*))) {*0*..*n*} = (*a* + *b*) *choose n*
⟨*proof*⟩

lemma *binomial-symmetric*: **assumes** *kn*: *k* ≤ *n*
shows *n choose k* = *n choose* (*n* − *k*)
⟨*proof*⟩

lemma *binomial-Vandermond-same*: *setsum* (λ*k*. (*n choose k*) ^*2*) {*0*..*n*} = (*2*∗*n*) *choose n*
⟨*proof*⟩

55 Relation between formal sine/cosine and the exponential FPS

lemma *Eii-sin-cos*:
E (*ii* ∗ *c*) = *fps-cos c* + *fps-const ii* ∗ *fps-sin c*
(**is** *?l* = *?r*)
⟨*proof*⟩

lemma *fps-sin-neg*[*simp*]: *fps-sin* (− *c*) = − *fps-sin c*
⟨*proof*⟩

lemma *fps-cos-neg*[*simp*]: *fps-cos* (− *c*) = *fps-cos c*
⟨*proof*⟩
lemma *E-minus-ii-sin-cos*: *E* (− (*ii* ∗ *c*)) = *fps-cos c* − *fps-const ii* ∗ *fps-sin c*
⟨*proof*⟩

lemma *fps-const-minus*: *fps-const* (*c*::′*a*::*group-add*) − *fps-const d* = *fps-const* (*c* − *d*) ⟨*proof*⟩

lemma *fps-number-of-fps-const*: *number-of i* = *fps-const* (*number-of i* :: ′*a*:: {*comm-ring-1*, *number-ring*})
⟨*proof*⟩

lemma *fps-cos-Eii*:
fps-cos c = (*E* (*ii* ∗ *c*) + *E* (− *ii* ∗ *c*)) / *fps-const 2*
⟨*proof*⟩

lemma *fps-sin-Eii*:
fps-sin c = (*E* (*ii* ∗ *c*) − *E* (− *ii* ∗ *c*)) / *fps-const* (*2*∗*ii*)
⟨*proof*⟩

lemma *fps-const-mult-2*: *fps-const* (*2*::′*a*::*number-ring*) ∗ *a* = *a* +*a*
⟨*proof*⟩

lemma *fps-const-mult-2-right*: $a * fps\text{-}const\ (2::'a::number\text{-}ring) = a + a$
⟨*proof*⟩

lemma *fps-tan-Eii*:
$fps\text{-}tan\ c = (E\ (ii * c) - E\ (-\ ii * c))\ /\ (fps\text{-}const\ ii * (E\ (ii * c) + E\ (-\ ii * c)))$
⟨*proof*⟩

lemma *fps-demoivre*: $(fps\text{-}cos\ a + fps\text{-}const\ ii * fps\text{-}sin\ a)\hat{}n = fps\text{-}cos\ (of\text{-}nat\ n * a) + fps\text{-}const\ ii * fps\text{-}sin\ (of\text{-}nat\ n * a)$
⟨*proof*⟩

Now some trigonometric identities

lemma *fps-sin-add*:
$fps\text{-}sin\ (a+b) = fps\text{-}sin\ (a::complex) * fps\text{-}cos\ b + fps\text{-}cos\ a * fps\text{-}sin\ b$
⟨*proof*⟩

lemma *fps-cos-add*:
$fps\text{-}cos\ (a+b) = fps\text{-}cos\ (a::complex) * fps\text{-}cos\ b - fps\text{-}sin\ a * fps\text{-}sin\ b$
⟨*proof*⟩

end

56 Hilbert's choice and classical logic

theory *Hilbert-Classical* **imports** *Main* **begin**

Derivation of the classical law of tertium-non-datur by means of Hilbert's choice operator (due to M. J. Beeson and J. Harrison).

56.1 Proof text

theorem *tnd*: $A \vee \neg A$
⟨*proof*⟩

56.2 Proof term of text

```
disjE · - · - · - · - · •
 (thm.Hilbert-Choice.someI · (λX. X = False ∨ X = True ∧ ?A) · - •
   (disjI1 · - · - · - · (thm.HOL.refl · -))) •
 (λH: -.
   disjE · - · - · - · - · •
    (thm.Hilbert-Choice.someI · (λX. X = False ∧ ?A ∨ X = True) · - •
      (disjI2 · - · - · - · (thm.HOL.refl · -))) •
    (λH: -. disjI1 · - · - · - · (conjE · - · - · - · - · - · H • (λ(H: -) H: -. H))) •
    (λHa: -.
      disjI2 · - · - · - ·
       (notI · - •
```

```
(λHb: -.
  notE · - · - · •
   (notI · - · •
     (λHb: -.
       False-neq-True · - · •
        (order-trans-rules-29 · - · - · - · - · •
          (order-trans-rules-14 ·
            (λa. a = (SOME X. X = False ∧ ?A ∨ X = True)) ·
            - ·
            - •
            (arg-cong · (λX. X = False ∨ X = True ∧ ?A) ·
              (λX. X = False ∧ ?A ∨ X = True) ·
              Eps •
              Hb) •
            H) •
          Ha))) •
     (thm.HOL.ext · - · - · •
       (λX. iffI · - · - · •
              (λH: -.
                disjE · - · - · - · - · H •
                 (λH: -. disjI1 · - · - · (conjI · - · - · H • Hb)) •
                 (λH: -.
                   disjI2 · - · - · •
                    (conjE · - · - · - · - · H • (λ(H: -) Ha: -. H)))) •
              (λH: -.
                disjE · - · - · - · - · H •
                 (λH: -.
                   disjI1 · - · - · •
                    (conjE · - · - · - · - · H • (λ(H: -) Ha: -. H))) •
                 (λH: -.
                   disjI2 · - · - · (conjI · - · - · H • Hb)))))))))) •
(λH: -. disjI1 · - · - · (conjE · - · - · - · - · H • (λ(H: -) H: -. H)))
```

56.3 Proof script

theorem *tnd′*: $A \vee \neg A$
⟨*proof*⟩

56.4 Proof term of script

```
conjE · - · - · - · - · •
 (conjI · - · - · •
   (thm.Hilbert-Choice.someI · (λx. x = False ∨ x = True ∧ ?A) · - · •
     (disjI1 · - · - · (thm.HOL.refl · -))) •
   (thm.Hilbert-Choice.someI · (λx. x = False ∧ ?A ∨ x = True) · - · •
     (disjI2 · - · - · (thm.HOL.refl · -)))) •
 (λ(H: -) Ha: -.
   disjE · - · - · - · - · H •
```

```
(λH: -.
  disjE · - · - · - · - · Ha ·
   (λH: -. conjE · - · - · - · - · H · (λH: -. disjI1 · - · - ·)) ·
   (λHa: -.
     disjI2 · - · - · - ·
      (notI · - ·
        (λHb: -.
          notE · - · - · - ·
           (notI · - ·
             (λHb: -.
               False-neq-True · - ·
                (HOL.trans · - · - · - · - · (HOL.sym · - · - · - · H) ·
                 (HOL.trans · - · - · - · - · - ·
                   (arg-cong · (λx. x = False ∨ x = True ∧ ?A) ·
                     (λx. x = False ∧ ?A ∨ x = True) ·
                     Eps ·
                     Hb) ·
                    Ha)))) ·
           (thm.HOL.ext · - · - · - ·
             (λx. iffI · - · - · - ·
                    (λH: -.
                      disjE · - · - · - · - · - · H ·
                       (λH: -. disjI1 · - · - · - · (conjI · - · - · - · - · H · Hb)) ·
                       (λH: -.
                         conjE · - · - · - · - · - · H ·
                          (λ(H: -) Ha: -. disjI2 · - · - · - · - · H))) ·
                    (λH: -.
                      disjE · - · - · - · - · - · H ·
                       (λH: -.
                         conjE · - · - · - · - · - · H ·
                          (λ(H: -) Ha: -. disjI1 · - · - · - · - · H)) ·
                       (λH: -.
                         disjI2 · - · - · - · - · (conjI · - · - · - · - · H · Hb))))))))) ·
(λH: -. conjE · - · - · - · - · - · H · (λH: -. disjI1 · - · - ·)))
```

end

57 Installing an oracle for SVC (Stanford Validity Checker)

theory *SVC-Oracle*
imports *Main*
uses *svc-funcs.ML*
begin

consts
  iff-keep :: [*bool*, *bool*] => *bool*

iff-unfold :: [*bool*, *bool*] => *bool*

hide *const iff-keep iff-unfold*

⟨*ML*⟩

end

References

[1] M. J. C. Gordon. HOL: A machine oriented formulation of higher order logic. Technical Report 68, University of Cambridge Computer Laboratory, 1985.

[2] K. McMillan. Lecture notes on verification of digital and hybrid systems. NATO summer school, http://www-cad.eecs.berkeley.edu/~kenmcmil/tutorial/toc.html.

[3] K. McMillan. *Symbolic Model Checking: an approach to the state explosion problem*. PhD thesis, Carnegie Mellon University, 1992.