Abstract

This document presents the formalization of introductory material from recursion theory — definitions and basic properties of primitive recursive functions, Cantor pairing function and computably enumerable sets (including a proof of existence of a one-complete computably enumerable set and a proof of the Rice’s theorem).

Contents

1 Cantor pairing function 2
   1.1 Pairing function 2
   1.2 Inverse mapping 7

2 Primitive recursive functions 12
   2.1 Basic definitions 12
   2.2 Bounded least operator 22
   2.3 Examples 29

3 Primitive recursive coding of lists of natural numbers 35

4 Primitive recursive functions of one variable 55
   4.1 Alternative definition of primitive recursive functions of one variable 55
   4.2 The scheme datatype 60
   4.3 Indexes of primitive recursive functions of one variables 66
   4.4 s-1-1 theorem for primitive recursive functions of one variable 69

5 Finite sets 75

6 The function which is universal for primitive recursive functions of one variable 98
7 Computably enumerable sets of natural numbers

7.1 Basic definitions ................................................. 128
7.2 Basic properties of computably enumerable sets .......... 128
7.3 Enumeration of computably enumerable sets ............. 133
7.4 Characteristic functions ........................................ 134
7.5 Computably enumerable relations .............................. 135
7.6 Total computable functions .................................... 147
7.7 Computable sets, Post’s theorem ............................ 151
7.8 Universal computably enumerable set ...................... 153
7.9 s-1-1 theorem, one-one and many-one reducibilities ... 156
7.10 One-complete sets .............................................. 163
7.11 Index sets, Rice’s theorem .................................... 163

1 Cantor pairing function

theory CPair imports Main begin

We introduce a particular coding c-pair from ordered pairs of natural
numbers to natural numbers. See [1] and the Isabelle documentation for more
information.

1.1 Pairing function

definition
sf :: nat ⇒ nat where
sf-def: sf x = x * (x+1) div 2

definition
c-pair :: nat ⇒ nat ⇒ nat where
c-pair x y = sf (x+y) + x

lemma sf-at-0: sf 0 = 0 by (simp add: sf-def)

lemma sf-at-1: sf 1 = 1 by (simp add: sf-def)

lemma sf-at-Suc: sf (x+1) = sf x + x + 1

proof –
  have S1: sf(x+1) = ((x+1)*(x+2)) div 2 by (simp add: sf-def)
  have S2: (x+1)*(x+2) = x*(x+1) + 2*(x+1) by (auto)
  have S2-1: \ x y. x=y ⇒ x div 2 = y div 2 by auto
  from S2 have S3: (x+1)*(x+2) div 2 = (x*(x+1) + 2*(x+1)) div 2 by (rule S2-1)
  have S4: (0::nat) < 2 by (auto)
  from S4 have S5: (x*(x+1) + 2*(x+1)) div 2 = (x+1) + x*(x+1) div 2 by simp
lemma arg-le-sf: \( x \leq \text{sf } x \)
proof
  have \( x + x \leq x \cdot (x + 1) \) by simp
  hence \( (x + x) \div 2 \leq x \cdot (x + 1) \div 2 \) by (rule div-le-mono)
  hence \( x \leq x \cdot (x + 1) \div 2 \) by simp
  thus \( \text{thesis by (simp add: sf-def)} \)
qed

lemma sf-mono: \( x \leq y \Rightarrow \text{sf } x \leq \text{sf } y \)
proof
  assume \( A1: x \leq y \)
  then have \( x + 1 \leq y + 1 \) by (auto)
  with \( A1 \) have \( x \cdot (x + 1) \leq y \cdot (y + 1) \) by (rule mult-le-mono)
  then have \( x \cdot (x + 1) \div 2 \leq y \cdot (y + 1) \div 2 \) by (rule div-le-mono)
  thus \( \text{thesis by (simp add: sf-def)} \)
qed

lemma sf-strict-mono: \( x < y \Rightarrow \text{sf } x < \text{sf } y \)
proof
  assume \( A1: x < y \)
  from \( A1 \) have \( S1: x + 1 \leq y \) by simp
  from \( A1 \) have \( S2: \text{sf } (x + 1) \leq \text{sf } y \) by (auto)
  from sf-at-Suc have \( S3: \text{sf } x < \text{sf } (x + 1) \) by (auto)
  from \( S2 \) have \( S3 \) show \( \text{thesis by (auto)} \)
qed

lemma sf-posI: \( x > 0 \Rightarrow \text{sf } (x) > 0 \)
proof
  assume \( A1: x > 0 \)
  then have \( \text{sf} (0) < \text{sf} (x) \) by (rule sf-strict mono)
  then show \( \text{thesis by simp} \)
qed

lemma arg-less-sf: \( x > 1 \Rightarrow x < \text{sf } x \)
proof
  assume \( A1: x > 1 \)
  let \( y = x - (1 :: \text{nat}) \)
  from \( A1 \) have \( S1: x = \text{sf } y + 1 \) by simp
  from \( A1 \) have \( \textsf{?y > 0} \) by simp
  then have \( S2: \text{sf} (\textsf{?y}) > 0 \) by (rule sf-posI)
  have \( \text{sf} (\textsf{?y} + 1) = \text{sf} (\textsf{?y}) + \textsf{?y} + 1 \) by (rule sf-at-Suc)
  with \( S1 \) have \( \text{sf} (x) = \text{sf} (\textsf{?y}) + x \) by simp
  with \( S2 \) show \( \text{thesis by simp} \)
qed

lemma sf-eq-arg: \( \text{sf } x = x \Rightarrow x \leq 1 \)
proof

assume \( sf(x) = x \)
then have \( \neg (x < sf(x)) \) by simp
then have \( \neg (x > 1) \) by (auto simp add: arg-less-sf)
then show \( \text{thesis} \) by simp
qed

lemma sf-le-sfD: \( sf x \leq sf y \implies x \leq y \)
proof

assume A1: \( sf x \leq sf y \)
have S1: \( y < x \implies sf y < sf x \) by (rule sf-strict-mono)
have S2: \( y < x \lor x \leq y \) by (auto)
from A1 S1 S2 show \( \text{thesis} \) by (auto)
qed

lemma sf-less-sfD: \( sf x < sf y \implies x < y \)
proof

assume A1: \( sf x < sf y \)
have S1: \( y \leq x \implies sf y \leq sf x \) by (rule sf-mono)
have S2: \( y \leq x \lor x < y \) by (auto)
from A1 S1 S2 show \( \text{thesis} \) by (auto)
qed

lemma sf-inj: \( sf x = sf y \implies x = y \)
proof

assume A1: \( sf x = sf y \)
have S1: \( sf x \leq sf y \implies x \leq y \) by (rule sf-le-sfD)
have S2: \( y \leq sf x \implies y \leq x \) by (rule sf-le-sfD)
from A1 have S3: \( sf x \leq sf y \land sf y \leq sf x \) by (auto)
from S3 S1 S2 have S4: \( x \leq y \land y \leq x \) by (auto)
from S4 show \( \text{thesis} \) by (auto)
qed

Auxiliary lemmas

lemma sf-aux1: \( x + y < z \implies sf(x+y) + x < sf(z) \)
proof

assume A1: \( x + y < z \)
from A1 have S1: \( x+y+1 \leq z \) by (auto)
from S1 have S2: \( sf(x+y+1) \leq sf(z) \) by (rule sf-monotone)
have S3: \( sf(x+y+1) = sf(x+y) + (x+y) + 1 \) by (rule at-Suc)
from S3 S2 have S4: \( sf(x+y) + (x+y) + 1 \leq sf(z) \) by (auto)
from S4 show \( \text{thesis} \) by (auto)
qed

lemma sf-aux2: \( sf(z) \leq sf(x+y) + x \implies z \leq x+y \)
proof

assume A1: \( sf(z) \leq sf(x+y) + x \)
from A1 have S1: \( sf(x+y) + x < sf(z) \) by (auto)
from S1 sf-aux1 have S2: \( \neg x+y < z \) by (auto)
from S2 show \(?thesis\) by \?(auto)
qed

lemma sf-aux3: \(sf(z) + m < sf(z+1) \implies m \leq z\)
proof –
  assume \(A1: sf(z) + m < sf(z+1)\)
  have \(S1: sf(z+1) = sf(z) + z + 1\) by \?(rule sf-at-Suc)
  from \(A1 S1\) have \(S2: sf(z) + m < sf(z) + z + 1\) by \?(auto)
  from \(S2\) have \(S3: m < z + 1\) by \?(auto)
  from \(S3\) show \(?thesis\) by \?(auto)
qed

lemma sf-aux4: \((s::nat) < t \implies (sf s) + s < sf t\)
proof –
  assume \(A1: (s::nat) < t\)
  have \(s*(s + 1) + 2*(s+1) \leq t*(t+1)\)
  proof –
    from \(A1\) have \(S1: (s::nat) + 1 \leq t\) by \?(auto)
    from \(A1\) have \((s::nat) + 2 \leq t+1\) by \?(auto)
    with \(S1\) have \((s::nat)+1)*(s+2) \leq t*(t+1)\) by \?(rule mult-le-mono)
    thus \(?thesis\) by \?(auto)
  qed
  then have \(S1: (s*(s+1) + 2*(s+1)) \div 2 \leq t*(t+1) \div 2\) by \?(rule div-le-mono)
  have \((0::nat) < 2\) by \?(auto)
  then have \((s*(s+1) + 2*(s+1)) \div 2 = (s+1) + (s*(s+1)) \div 2\) by \?simp
  with \(S1\) have \((s*(s+1)) \div 2 + (s+1) \leq t*(t+1) \div 2\) by \?(auto)
  then have \((s*(s+1)) \div 2 + s < t*(t+1) \div 2\) by \?(auto)
  thus \(?thesis\) by \?(simp add: sf-def)
qed

Basic properties of \(c\)-pair function

lemma sum-le-c-pair: \(x + y \leq c\)-pair \(x y\)
proof –
  have \(x+y \leq sf(x+y)\) by \?(rule arg-le-sf)
  thus \(?thesis\) by \?(simp add: c-pair-def)
qed

lemma arg1-le-c-pair: \(x \leq c\)-pair \(x y\)
proof –
  have \((x::nat) \leq x + y\) by \?(simp)
  moreover have \(x + y \leq c\)-pair \(x y\) by \?(rule sum-le-c-pair)
  ultimately show \(?thesis\) by \?(simp)
qed

lemma arg2-le-c-pair: \(y \leq c\)-pair \(x y\)
proof –
  have \((y::nat) \leq x + y\) by \?(simp)
  moreover have \(x + y \leq c\)-pair \(x y\) by \?(rule sum-le-c-pair)
  ultimately show \(?thesis\) by \?(simp)
lemma c-pair-sum-mono: \((x1::nat) + y1 < x2 + y2 \Rightarrow c\text{-}pair x1 y1 < c\text{-}pair x2 y2\)
proof
  - assume \((x1::nat) + y1 < x2 + y2\)
  hence \(sf (x1+y1) + (x1+y1) < sf(x2+y2)\) by (rule sf-aux4)
  hence \(sf (x1+y1) + x1 < sf(x2+y2) + x2\) by (auto)
  thus \(?thesis\) by (simp add: c-pair-def)
qed

lemma c-pair-sum-inj: \(c\text{-}pair x1 y1 = c\text{-}pair x2 y2 \Rightarrow x1 + y1 = x2 + y2\)
proof
  - assume \(A1: c\text{-}pair x1 y1 = c\text{-}pair x2 y2\)
  have \(S1: (x1::nat) + y1 < x2 + y2 \Rightarrow c\text{-}pair x1 y1 \neq c\text{-}pair x2 y2\) by (rule less-not-refl3, rule c-pair-sum-mono, auto)
  have \(S2: (x2::nat) + y2 < x1 + y1 \Rightarrow c\text{-}pair x1 y1 \neq c\text{-}pair x2 y2\) by (rule less-not-refl2, rule c-pair-sum-mono, auto)
  from \(S1\) \(S2\) have \((x1::nat) + y1 \neq x2 + y2 \Rightarrow c\text{-}pair x1 y1 \neq c\text{-}pair x2 y2\) by (arith)
  with \(A1\) show \(?thesis\) by (auto)
qed

lemma c-pair-inj: \(c\text{-}pair x1 y1 = c\text{-}pair x2 y2 \Rightarrow x1 = x2 \land y1 = y2\)
proof
  - assume \(A1: c\text{-}pair x1 y1 = c\text{-}pair x2 y2\)
  from \(A1\) have \(S1: x1 + y1 = x2 + y2\) by (rule c-pair-sum-inj)
  from \(A1\) have \(S2: sf (x1+y1) + x1 = sf (x2+y2) + x2\) by (unfold c-pair-def)
  from \(S1\) \(S2\) have \(S3: x1 = x2\) by (simp)
  from \(S1\) \(S3\) have \(S4: y1 = y2\) by (simp)
  from \(S3\) \(S4\) show \(?thesis\) by (auto)
qed

lemma c-pair-inj1: \(c\text{-}pair x1 y1 = c\text{-}pair x2 y2 \Rightarrow x1 = x2\) by (frule c-pair-inj, drule conjunct1)

lemma c-pair-inj2: \(c\text{-}pair x1 y1 = c\text{-}pair x2 y2 \Rightarrow y1 = y2\) by (frule c-pair-inj, drule conjunct2)

lemma c-pair-strict-mono1: \(x1 < x2 \Rightarrow c\text{-}pair x1 y < c\text{-}pair x2 y\)
proof
  - assume \(x1 < x2\)
  then have \(x1 + y < x2 + y\) by simp
  then show \(?thesis\) by (rule c-pair-sum-mono)
qed

lemma c-pair-mono1: \(x1 \leq x2 \Rightarrow c\text{-}pair x1 y \leq c\text{-}pair x2 y\)
proof
  - assume \(A1: x1 \leq x2\)
show \(?thesis
proof cases
  assume \(x_1 < x_2\)
  then have \(\text{c-pair } x_1 y < \text{c-pair } x_2 y\) by (rule \text{c-pair-strict-mono1})
  then show \(?thesis\) by simp
next
  assume \(\neg x_1 < x_2\)
  with A1 have \(x_1 = x_2\) by simp
  then show \(?thesis\) by simp
qed

lemma \text{c-pair-strict-mono2}: \(y_1 < y_2 \implies \text{c-pair } x y_1 < \text{c-pair } x y_2\)
proof –
  assume A1: \(y_1 < y_2\)
  from A1 have S1: \(x + y_1 < x + y_2\) by simp
  then show \(?thesis\) by (rule \text{c-pair-sum-mono})
qed

lemma \text{c-pair-mono2}: \(y_1 \leq y_2 \implies \text{c-pair } x y_1 \leq \text{c-pair } x y_2\)
proof –
  assume A1: \(y_1 \leq y_2\)
  show \(?thesis\)
    proof cases
      assume \(y_1 < y_2\)
      then have \(\text{c-pair } x y_1 < \text{c-pair } x y_2\) by (rule \text{c-pair-strict-mono2})
      then show \(?thesis\) by simp
    next
      assume \(\neg y_1 < y_2\)
      with A1 have \(y_1 = y_2\) by simp
      then show \(?thesis\) by simp
    qed
  qed

1.2 Inverse mapping

\text{c-fst} and \text{c-snd} are the functions which yield the inverse mapping to \text{c-pair}.

definition
\text{c-sum} :: \text{n}at \Rightarrow \text{n}at where
\text{c-sum } u = (\text{LEAST } z. \ u < \text{s}f (z+1))

definition
\text{c-fst} :: \text{n}at \Rightarrow \text{n}at where
\text{c-fst } u = \ u - \text{s}f (\text{c-sum } u)

definition
\text{c-snd} :: \text{n}at \Rightarrow \text{n}at where
\text{c-snd } u = \text{c-sum } u - \text{c-fst } u
**Lemma** arg-less-sf-at-Suc-of-c-sum: $u < sf((c\text{-}sum\ u) + 1)$

**Proof**
- have $u+1 \leq sf(u+1)$ by (rule arg-le-sf)
- hence $u < sf(u+1)$ by simp
- thus ?thesis by (unfold c-sum-def, rule LeastI)

qed

**Lemma** arg-less-sf-imp-c-sum-less-arg: $u < sf(x) \Rightarrow c\text{-}sum\ u < x$

**Proof**
- assume $A1: u < sf(x)$
  - then show ?thesis
    proof (cases $x$)
    assume $x=0$
    with $A1$ show ?thesis by (simp add: sf-def)
  - next
    fix $y$
    assume $A2: x = Suc\ y$
    show ?thesis
    proof
      from $A1\ A2$ have $u < sf(y+1)$ by simp
      hence $(Least (\%z. u < sf(z+1))) \leq y$ by (rule Least-le)
      hence $c\text{-}sum\ u \leq y$ by (fold c-sum-def)
      with $A2$ show ?thesis by simp
    qed
  qed

qed

**Lemma** sf-c-sum-le-arg: $u \geq sf(c\text{-}sum\ u)$

**Proof**
- let $?z = c\text{-}sum\ u$
  - from arg-less-sf-at-Suc-of-c-sum have $S1: u < sf(?z+1)$ by (auto)
  - have $S2: \neg c\text{-}sum\ u < c\text{-}sum\ u$ by (auto)
  - from arg-less-sf-imp-c-sum-less-arg $S2$ have $S3: \neg u < sf(c\text{-}sum\ u)$ by (auto)
  - from $S3$ show ?thesis by (auto)

qed

**Lemma** c-sum-le-arg: $c\text{-}sum\ u \leq u$

**Proof**
- have $c\text{-}sum\ u \leq sf(c\text{-}sum\ u)$ by (rule arg-le-sf)
- moreover have $sf(c\text{-}sum\ u) \leq u$ by (rule sf-c-sum-le-arg)
- ultimately show ?thesis by simp

qed

**Lemma** c-sum-of-c-pair [simp]: $c\text{-}sum\ (c\text{-}pair\ x\ y) = x + y$

**Proof**
- let $?u = c\text{-}pair\ x\ y$
- let $?z = c\text{-}sum\ ?u$
  - have $S1: ?u < sf(?z+1)$ by (rule arg-less-sf-at-Suc-of-c-sum)
  - have $S2: sf(?z) \leq ?u$ by (rule sf-c-sum-le-arg)
from S1 have S3: \( sf(x+y)+x < sf(?z+1) \) by (simp add: c-pair-def)
from S2 have S4: \( sf(?z) \leq sf(x+y) + x \) by (simp add: c-pair-def)
from S3 have S5: \( sf(x+y) < sf(?z+1) \) by (auto)
from S5 have S6: \( x+y < ?z+1 \) by (rule sf-less-sfD)
from S6 have S7: \( x+y \leq ?z \) by (auto)
from S7 S8 have S9: \( ?z = x+y \) by (auto)
from S9 show \( \text{thesis} \) by (simp)
qed

lemma c-fst-of-c-pair[simp]: \( c\text{-fst} (c\text{-pair } x \ y) = x \)
proof
  let \(?u = c\text{-pair } x \ y\)
  have \( c\text{-sum } ?u = x + y \) by simp
  hence \( c\text{-fst } ?u = ?u - sf(x+y) \) by (simp add: c-fst-def)
  moreover have \( ?u = sf(x+y) + x \) by (simp add: c-pair-def)
  ultimately show \( \text{thesis} \) by (simp)
qed

lemma c-snd-of-c-pair[simp]: \( c\text{-snd} (c\text{-pair } x \ y) = y \)
proof
  let \(?u = c\text{-pair } x \ y\)
  have \( c\text{-sum } ?u = x + y \) by simp
  moreover have \( c\text{-fst } ?u = x \) by simp
  ultimately show \( \text{thesis} \) by (simp add: c-snd-def)
qed

lemma c-pair-at-0: \( c\text{-pair } 0 \ 0 = 0 \) by (simp add: sf-def c-pair-def)

lemma c-fst-at-0: \( c\text{-fst } 0 = 0 \)
proof
  have \( c\text{-pair } 0 \ 0 = 0 \) by (rule c-pair-at-0)
  hence \( c\text{-fst } 0 = c\text{-fst} (c\text{-pair } 0 \ 0) \) by simp
  thus \( \text{thesis} \) by simp
qed

lemma c-snd-at-0: \( c\text{-snd } 0 = 0 \)
proof
  have \( c\text{-pair } 0 \ 0 = 0 \) by (rule c-pair-at-0)
  hence \( c\text{-snd } 0 = c\text{-snd} (c\text{-pair } 0 \ 0) \) by simp
  thus \( \text{thesis} \) by simp
qed

lemma sf-c-sum-plus-c-fst: \( sf(c\text{-sum } u) + c\text{-fst } u = u \)
proof
  have S1: \( sf(c\text{-sum } u) \leq u \) by (rule sf-c-sum-le-arg)
  have S2: \( c\text{-fst } u = u - sf(c\text{-sum } u) \) by (simp add: c-fst-def)
  from S1 S2 show \( \text{thesis} \) by (auto)
qed
lemma c-fst-le-c-sum: c-fst u ≤ c-sum u
proof –
  have S1: sf(c-sum u) + c-fst u = u by (rule sf-c-sum-plus-c-fst)
  have S2: u < sf((c-sum u) + 1) by (rule arg-less-sf-at-Suc-of-c-sum)
  from S1 S2 sf-aux3 show ?thesis by (auto)
qed

lemma c-snd-le-c-sum: c-snd u ≤ c-sum u by (simp add: c-snd-def)

lemma c-fst-le-arg: c-fst u ≤ u
proof –
  assume A1: c-fst u ≤ c-sum u by (rule c-fst-le-c-sum)
  moreover have c-sum u ≤ u by (rule c-sum-le-arg)
  ultimately show ?thesis by simp
qed

lemma c-snd-le-arg: c-snd u ≤ u
proof –
  assume A1: c-snd u ≤ c-sum u by (rule c-snd-le-c-sum)
  moreover have c-sum u ≤ u by (rule c-sum-le-arg)
  ultimately show ?thesis by simp
qed

lemma c-sum-is-sum: c-sum u = c-fst u + c-snd u by (simp add: c-snd-def c-fst-le-c-sum)

lemma proj-eq-imp-arg-eq: [ c-fst u = c-fst v; c-snd u = c-snd v ] ⟹ u = v
proof –
  assume A1: c-fst u = c-fst v
  assume A2: c-snd u = c-snd v
  from A1 A2 c-sum-is-sum have S1: c-sum u = c-sum v by (auto)
  have S2: sf(c-sum u) + c-fst u = u by (rule sf-c-sum-plus-c-fst)
  from A1 S1 S2 have S3: sf(c-sum v) + c-fst v = u by (auto)
  from S3 sf-c-sum-plus-c-fst show ?thesis by (auto)
qed

lemma c-pair-of-c-fst-c-snd[simp]: c-pair (c-fst u) (c-snd u) = u
proof –
  let ?x = c-fst u
  let ?y = c-snd u
  have S1: c-pair ?x ?y = sf(?x + ?y) + ?x by (simp add: c-pair-def)
  have S2: c-sum u = ?x + ?y by (rule c-sum-is-sum)
  from S1 S2 have c-pair ?x ?y = sf(c-sum u) + c-fst u by (auto)
  thus ?thesis by (simp add: sf-c-sum-plus-c-fst)
qed

lemma c-sum-eq-arg: c-sum x = x ⟹ x ≤ 1
proof –
assume \( A1: \) \( c \sum x = x \)

have \( S1: \) \( sf(c \sum x) + c \text{fst} x = x \) by (rule \( sf \cdot c \sum + c \text{fst} \))

from \( A1 \) \( S1 \) have \( S2: \) \( sf x + c \text{fst} x = x \) by simp

have \( S3: \) \( x \leq sf x \) by (rule \( \text{arg-le-sf} \))

from \( S2 \) \( S3 \) have \( \text{sf}(x) = x \) by simp

thus \( \text{thesis} \) by (rule \( sf \cdot \text{eq-arg} \))

qed

lemma \( c \sum \cdot \text{eq-arg-2} \): \( c \sum x = x \implies c \text{fst} x = 0 \)

proof -

assume \( A1: \) \( c \sum x = x \)

have \( S1: \) \( sf(c \sum x) + c \text{fst} x = x \) by (rule \( sf \cdot c \sum + c \text{fst} \))

from \( A1 \) \( S1 \) have \( S2: \) \( sf x + c \text{fst} x = x \) by simp

have \( S3: \) \( x \leq sf x \) by (rule \( \text{arg-le-sf} \))

from \( S2 \) \( S3 \) show \( \text{thesis} \) by simp

qed

lemma \( c \text{fst} \cdot \text{eq-arg} \): \( c \text{fst} x = x \implies x = 0 \)

proof -

assume \( A1: \) \( c \text{fst} x = x \)

have \( S1: \) \( c \text{fst} x \leq c \sum x \) by (rule \( c \text{fst} \cdot c \sum \))

have \( S2: \) \( c \sum x \leq x \) by (rule \( c \sum \cdot \text{le-arg} \))

from \( A1 \) \( S1 \) \( S2 \) have \( \text{c-sum} x = x \) by simp

then have \( c \text{fst} x = 0 \) by (rule \( \text{c-sum-eq-arg-2} \))

with \( A1 \) show \( \text{thesis} \) by simp

qed

lemma \( c \text{fst} \cdot \text{less-arg} \): \( x > 0 \implies c \text{fst} x < x \)

proof -

assume \( A1: \) \( x > 0 \)

show \( \text{thesis} \)

proof cases

assume \( c \text{fst} x < x \)

then show \( \text{thesis} \) by simp

next

assume \( \neg c \text{fst} x < x \)

then have \( S1: \) \( c \text{fst} x \geq x \) by simp

have \( c \text{fst} x \leq x \) by (rule \( c \text{fst} \cdot \text{le-arg} \))

with \( S1 \) have \( c \text{fst} x = x \) by simp

then have \( x = 0 \) by (rule \( c \text{fst} \cdot \text{eq-arg} \))

with \( A1 \) show \( \text{thesis} \) by simp

qed

qed

lemma \( c \text{snd} \cdot \text{eq-arg} \): \( c \text{snd} x = x \implies x \leq 1 \)

proof -

assume \( A1: \) \( c \text{snd} x = x \)

have \( S1: \) \( c \text{snd} x \leq c \sum x \) by (rule \( c \text{snd} \cdot \text{le-arg} \))

have \( S2: \) \( c \sum x \leq x \) by (rule \( c \sum \cdot \text{le-arg} \))

11
from $A1 \ S1 \ S2$ have $c$-sum $x = x$ by simp 
then show $?thesis$ by (rule $c$-sum-eq-arg)
qed

lemma $c$-snd-less-arg: $x > 1 \implies c$-snd $x < x$
proof
assume $A1$: $x > 1$
show $?thesis$
proof cases
assume $c$-snd $x < x$
then show $?thesis$.
next
assume $\neg c$-snd $x < x$
then have $S1$: $c$-snd $x \geq x$ by auto
have $c$-snd $x \leq x$ by (rule $c$-snd-le-arg)
with $S1$ have $c$-snd $x = x$ by simp
then have $x \leq 1$ by (rule $c$-snd-eq-arg)
with $A1$ show $?thesis$ by simp
qed
qed
end

2 Primitive recursive functions

theory $PRecFun$ imports $CPair$
begin

This theory contains definition of the primitive recursive functions.

2.1 Basic definitions

primrec
$PrimRecOp :: (nat \Rightarrow nat) \Rightarrow (nat \Rightarrow nat \Rightarrow nat \Rightarrow nat) \Rightarrow (nat \Rightarrow nat \Rightarrow nat)$
where
$PrimRecOp g h 0 x = g x$
$\mid PrimRecOp g h \ (Suc \ y) \ x = h \ y \ (PrimRecOp g \ h \ y \ x) \ x$

primrec
$PrimRecOp\_last :: (nat \Rightarrow nat) \Rightarrow (nat \Rightarrow nat \Rightarrow nat \Rightarrow nat) \Rightarrow (nat \Rightarrow nat \Rightarrow nat)$
where
$PrimRecOp\_last g h x \ 0 = g x$
$\mid PrimRecOp\_last g h x \ (Suc \ y)\ x = h \ x \ (PrimRecOp\_last g \ h \ x \ y) \ y$

primrec
$PrimRecOp1 :: nat \Rightarrow (nat \Rightarrow nat \Rightarrow nat) \Rightarrow (nat \Rightarrow nat)$
where
$PrimRecOp1 a h \ 0 = a$
| PrimRecOp1 a h (Suc y) = h y (PrimRecOp1 a h y)

**inductive-set**

PrimRec1 :: (nat ⇒ nat) set and
PrimRec2 :: (nat ⇒ nat ⇒ nat) set and
PrimRec3 :: (nat ⇒ nat ⇒ nat ⇒ nat) set

**where**

zero: (λ x. 0) ∈ PrimRec1
| suc: Suc ∈ PrimRec1
| id1-1: (λ x, x) ∈ PrimRec1
| id2-1: (λ x y, x) ∈ PrimRec2
| id2-2: (λ x y, y) ∈ PrimRec2
| id3-1: (λ x y z, x) ∈ PrimRec3
| id3-2: (λ x y z, y) ∈ PrimRec3
| id3-3: (λ x y z, z) ∈ PrimRec3
| comp1-1: [f ∈ PrimRec1; g ∈ PrimRec1] ⇒ (λ x. f (g x)) ∈ PrimRec1
| comp1-2: [f ∈ PrimRec1; g ∈ PrimRec2] ⇒ (λ x y. f (g x y)) ∈ PrimRec2
| comp1-3: [f ∈ PrimRec1; g ∈ PrimRec3] ⇒ (λ x y z. f (g x y z)) ∈ PrimRec3
| comp2-1: [f ∈ PrimRec2; g ∈ PrimRec1; h ∈ PrimRec1] ⇒ (λ x. f (g x) (h x)) ∈ PrimRec1
| comp3-1: [f ∈ PrimRec3; g ∈ PrimRec1; h ∈ PrimRec1; k ∈ PrimRec1] ⇒
  (λ x. f (g x) (h x) (k x)) ∈ PrimRec1
| comp2-2: [f ∈ PrimRec2; g ∈ PrimRec2; h ∈ PrimRec2] ⇒ (λ x y. f (g x y) (h x y)) ∈ PrimRec2
| comp2-3: [f ∈ PrimRec2; g ∈ PrimRec3; h ∈ PrimRec3] ⇒ (λ x y z. f (g x y z) (h x y z) (k x y z)) ∈ PrimRec3
| comp3-2: [f ∈ PrimRec3; g ∈ PrimRec3; h ∈ PrimRec3; k ∈ PrimRec3] ⇒
  (λ x y z. f (g x y z) (h x y z) (k x y z) (k x y z)) ∈ PrimRec3
| prim-rec: [g ∈ PrimRec1; h ∈ PrimRec3] ⇒ PrimRecOp g h ∈ PrimRec2

**lemmas pr-zero** = PrimRec1-PrimRec2-PrimRec3.zero
**lemmas pr-suc** = PrimRec1-PrimRec2-PrimRec3.suc
**lemmas pr-id1-1** = PrimRec1-PrimRec2-PrimRec3.id1-1
**lemmas pr-id2-1** = PrimRec1-PrimRec2-PrimRec3.id2-1
**lemmas pr-id2-2** = PrimRec1-PrimRec2-PrimRec3.id2-2
**lemmas pr-id3-1** = PrimRec1-PrimRec2-PrimRec3.id3-1
**lemmas pr-id3-2** = PrimRec1-PrimRec2-PrimRec3.id3-2
**lemmas pr-id3-3** = PrimRec1-PrimRec2-PrimRec3.id3-3
**lemmas pr-comp1-1** = PrimRec1-PrimRec2-PrimRec3.comp1-1
**lemmas pr-comp1-2** = PrimRec1-PrimRec2-PrimRec3.comp1-2
**lemmas pr-comp1-3** = PrimRec1-PrimRec2-PrimRec3.comp1-3
**lemmas pr-comp2-1** = PrimRec1-PrimRec2-PrimRec3.comp2-1
**lemmas pr-comp2-2** = PrimRec1-PrimRec2-PrimRec3.comp2-2
**lemmas pr-comp2-3** = PrimRec1-PrimRec2-PrimRec3.comp2-3
**lemmas pr-comp3-1** = PrimRec1-PrimRec2-PrimRec3.comp3-1
**lemmas pr-comp3-2** = PrimRec1-PrimRec2-PrimRec3.comp3-2
**lemmas pr-comp3-3** = PrimRec1-PrimRec2-PrimRec3.comp3-3
lemmas \(pr-rec = \text{PrimRec1-PrimRec2-PrimRec3.prim-rec}\)

ML-file \texttt{Utils.ML}

named-theorems \prec

method-setup \preeq = \left<\!
\begin{array}{l}
\text{Attrib.thms} >> (\text{fn ths} => \text{fn ctxt} => \text{Method.METHOD (fn facts} => \\
\text{HEADGOAL (preeq-tac ctxt (facts @ Named-Theorems.get ctxt @(\{named-theorems \preeq\})))}})
\end{array}\!ight>

apply primitive recursive functions

lemmas \[\preeq\] = \text{pr-zero pr-suc pr-id1-1 pr-id2-1 pr-id3-1 pr-id3-2 pr-id3-3}

lemma \(pr-swap\): \(f \in \text{PrimRec2} \implies (\lambda x y. f y x) \in \text{PrimRec2 \ by \ preeq}\)

theorem \(pr-rec-scheme\): \[ \begin{array}{l}
g \in \text{PrimRec1}; \ h \in \text{PrimRec3}; \ \forall x. f \ 0 \ x = g x; \ \forall x y. f (\text{Suc} \ y) \ x = h y (f y x) \ x \ \end{array}\] \implies f \in \text{PrimRec2}

proof –
  assume \(g\text{-is-pr}: g \in \text{PrimRec1}\)
  assume \(h\text{-is-pr}: h \in \text{PrimRec3}\)
  assume \(f\text{-at-0}: \forall x. f \ 0 \ x = g x\)
  assume \(f\text{-at-Suc}: \forall x y. f (\text{Suc} \ y) \ x = h y (f y x) \ x\)
  from \(f\text{-at-0} f\text{-at-Suc} \\text{have} \ \forall x y. f y x = \text{PrimRecOp} \ g \ h \ y \ x \ \text{by (induct-tac y, simp-all)}\)
  then \text{have} \(f = \text{PrimRecOp} \ g \ h \ \text{by (simp add: ext)}\)
  with \(g\text{-is-pr} h\text{-is-pr} \text{show} \ ?\text{thesis by (simp add: pr-rec)}\)
qed

lemma \text{op-plus-is-pr} [\preeq]: \((\lambda x y. x + y) \in \text{PrimRec2}\)

proof \(\text{rule pr-swap}\)
show \((\lambda x y. y+x) \in \text{PrimRec2}\)
proof –
  have \(S1: \text{PrimRecOp} (\lambda x. x) (\lambda x y z. \text{Suc} \ y) \in \text{PrimRec2}\)
  proof \(\text{rule pr-rec}\)
    show \((\lambda x. x) \in \text{PrimRec1} \ \text{by (rule pr-id1-1)}\)
next
  show \((\lambda x y z. \text{Suc} \ y) \in \text{PrimRec3} \ \text{by preeq}\)
qed
  have \((\lambda x y. y+x) = \text{PrimRecOp} (\lambda x. x) (\lambda x y z. \text{Suc} \ y) \ (\text{is - = ?f})\)
  proof –
    have \(\forall x y. (\text{if} \ x = y + x) \ \text{by (induct-tac y, auto)}\)
    thus \(\text{thesis by (simp add: ext)}\)
qed
  with \(S1\) show \(\text{thesis by simp}\)
qed
lemma op-mult-is-pr \[\text{prec} \]: \((\lambda \, x \, y \, . \, x \, \ast \, y) \in \text{PrimRec2}\)
proof (rule pr-swap)
show \((\lambda \, x \, y \, . \, y \, \ast \, x) \in \text{PrimRec2}\)
proof
next
have S1: \(\text{PrimRecOp} \, (\lambda \, x \, . \, 0) \, (\lambda \, x \, y \, z \, . \, y \, + \, z) \in \text{PrimRec2}\)
proof (rule pr-rec)
show \((\lambda \, x \, . \, 0) \in \text{PrimRec1}\) by \(\text{rule pr-zero}\)
next
show \((\lambda \, x \, y \, z \, . \, y \, + \, z) \in \text{PrimRec3}\) by \(\text{prec0}\)
qed
have \((\lambda \, x \, y \, . \, y \, \ast \, x) = \text{PrimRecOp} \, (\lambda \, x \, . \, 0) \, (\lambda \, x \, y \, z \, . \, y \, + \, z) \) by \(\text{induct-tac} \, y \, , \, \text{auto}\)
thus \(?\text{thesis}\) by \(\text{simp add: ext}\)
qed
with S1 show \(?\text{thesis}\) by \(\text{simp}\)
qed

lemma const-is-pr: \((\lambda \, x \, . \, (n :: \text{nat})) \in \text{PrimRec1}\)
proof (induct n)
show \((\lambda \, x \, . \, 0) \in \text{PrimRec1}\) by \(\text{rule pr-zero}\)
next
fix n assume \((\lambda \, x \, n) \in \text{PrimRec1}\)
then show \((\lambda \, x \, . \, \text{Suc} \, n) \in \text{PrimRec1}\) by \(\text{prec0}\)
qed

lemma const-is-pr-2: \((\lambda \, x \, y \, . \, (n :: \text{nat})) \in \text{PrimRec2}\)
proof (rule pr-comp1-2 \[\text{where} \, ?f = \% \, x \, . \, (n :: \text{nat}) \, \text{and} \, ?g = \% \, x \, y \, . \, x\])
show \((\lambda \, x \, n) \in \text{PrimRec1}\) by \(\text{rule const-is-pr}\)
next
show \((\lambda \, x \, y \, . \, x) \in \text{PrimRec2}\) by \(\text{rule pr-id2-1}\)
qed

lemma const-is-pr-3: \((\lambda \, x \, y \, z \, . \, (n :: \text{nat})) \in \text{PrimRec3}\)
proof (rule pr-comp1-3 \[\text{where} \, ?f = \% \, x \, (n :: \text{nat}) \, \text{and} \, ?g = \% \, x \, y \, z \, . \, x\])
show \((\lambda \, x \, n) \in \text{PrimRec1}\) by \(\text{rule const-is-pr}\)
next
show \((\lambda \, x \, y \, z \, . \, x) \in \text{PrimRec3}\) by \(\text{rule pr-id3-1}\)
qed

theorem pr-rec-last: \([g \in \text{PrimRec1}; \, h \in \text{PrimRec3}] \Longrightarrow \text{PrimRecOp-last} \, g \, h \in \text{PrimRec2}\)
proof
assume A1: \(g \in \text{PrimRec1}\)
assume A2: \(h \in \text{PrimRec3}\)
let \(?h1 = \lambda \, x \, y \, z \, . \, h \, z \, y \, x\)
from A2 pr-id3-3 pr-id3-2 pr-id3-1 have \(?h1 \in \text{PrimRec3}\) by \(\text{rule pr-comp3-3}\)

let \( f_1 = \text{PrimRecOp} \ g \ ?h_1 \)
from \( A1 \) \( h1 \)-is-pr have \( f1 \)-is-pr: \( f_1 \in \text{PrimRec2} \) by (rule pr-rec)
let \( f = \lambda x \ y. \ f_1 \ y \ x \)
from \( f1 \)-is-pr have \( f \)-is-pr: \( f \in \text{PrimRec2} \) by (rule pr-swap)
then have \( \forall x \ y. \ f x y = \text{PrimRecOp}-last \ g \ x \ y \) by (induct-tac \( y \), simp-all)
with \( f \)-is-pr show \( \text{thesis} \) by simp
qed

theorem \( \text{pr-rec1} \): \( h \in \text{PrimRec2} \rightarrow \text{PrimRecOp1} \ (a :: \text{nat}) \ h \in \text{PrimRec1} \)
proof –
  assume \( A1 \): \( h \in \text{PrimRec2} \)
  let \( g = (\lambda x. \ a) \)
  have \( g \)-is-pr: \( g \in \text{PrimRec1} \) by (rule const-is-pr)
  let \( h_1 = (\lambda x \ y \ z. \ h x y) \)
  from \( A1 \) have \( h1 \)-is-pr: \( h_1 \in \text{PrimRec3} \) by prec0
  let \( f_1 = \text{PrimRecOp} \ g \ ?h_1 \)
  from \( g \)-is-pr \( h1 \)-is-pr have \( f1 \)-is-pr: \( f_1 \in \text{PrimRec2} \) by (rule pr-rec)
  let \( f = (\lambda x. \ f_1 \ x \ 0) \)
  from \( f1 \)-is-pr \( \text{idl-1} \) \( \text{pr-zero} \) have \( f \)-is-pr: \( f \in \text{PrimRec1} \) by (rule pr-comp2-1)
  have \( \forall y. \ f y = \text{PrimRecOp1} \ a \ h y \) by (induct-tac \( y \), auto)
  then have \( \forall f = \text{PrimRecOp1} \ a \ h \) by (simp add: ext)
  with \( f \)-is-pr show \( \text{thesis} \) by (auto)
qed

theorem \( \text{pr-rec1-scheme} \): \[ [ h \in \text{PrimRec2}; \ f \ 0 = a; \ \forall y. \ f \ (\text{Suc} \ y) = h \ y \ (f \ y) ] \]
\( \rightarrow f \in \text{PrimRec1} \)
proof –
  assume \( h \)-is-pr: \( h \in \text{PrimRec2} \)
  assume \( f \)-at-0: \( f \ 0 = a \)
  assume \( f \)-at-Suc: \( \forall y. \ f \ (\text{Suc} \ y) = h \ y \ (f \ y) \)
  from \( f \)-at-0 \( f \)-at-Suc have \( \forall y. \ f y = \text{PrimRecOp1} \ a \ h y \) by (induct-tac \( y \), simp-all)
  then have \( f = \text{PrimRecOp1} \ a \ h \) by (simp add: ext)
  with \( h \)-is-pr show \( \text{thesis} \) by (simp add: pr-rec)
qed

lemma \( \text{pred-is-pr} \): \( (\lambda x. \ x - (1 :: \text{nat})) \in \text{PrimRec1} \)
proof –
  have \( S1 \): \( \text{PrimRecOp1} \ 0 (\lambda x \ y. \ x) \in \text{PrimRec1} \)
    by (rule pr-rec)
  show \( (\lambda x \ y. \ x) \in \text{PrimRec2} \) by (rule pr-id2-1)
  qed

have \( (\lambda x. \ x - (1 :: \text{nat})) = \text{PrimRecOp1} \ 0 (\lambda x \ y. \ x) \) (is - = \(?f\) \)
proof –
  have \( \lambda x. \ (f f x = x - (1 :: \text{nat})) \) by (induct-tac \( x \), auto)
  thus \( \text{thesis} \) by (simp add: ext)
  qed
  with \( S1 \) show \( \text{thesis} \) by simp
lemma \textbf{op-sub-is-pr} \textbf{[prec]}: $(\lambda \ x \ y. \ x - y) \in \text{PrimRec2}$
\begin{proof} \textbf{(rule pr-swap)}
\begin{align*}
\text{show} \ (\lambda \ x \ y. \ y - x) & \in \text{PrimRec2} \\
\text{proof} \\
\text{next} \\
\text{from} \ \text{pred-is-pr \ pr-id3-2} \ \text{show} \ (\lambda \ x \ y \ z. \ y - (1::\text{nat})) & \in \text{PrimRec3} \ \text{by} \ (\text{rule pr-comp1-3}) \\
\end{align*}
\end{proof}
\begin{proof}
\begin{align*}
\text{have} \ S1 & : \text{PrimRecOp} \ (\lambda \ x . \ x) \ \ (\lambda \ x \ y \ z . \ y - (1::\text{nat})) \in \text{PrimRec2} \\
\text{proof} \ \textbf{(rule pr-rec)} \\
\text{thus} \ \text{thesis} & \ \text{by} \ \text{simp} \\
\text{qed} \\
\text{with} \ S1 \ \text{show} \ \text{thesis} & \ \text{by} \ \text{simp} \\
\text{qed} \\
\text{lemmas} \ \textbf{[prec]} & = \\
\text{const-is-pr} \ \text{[of 0]} \ \text{const-is-pr-2} \ \text{[of 0]} \ \text{const-is-pr-3} \ \text{[of 0]} \\
\text{const-is-pr} \ \text{[of 1]} \ \text{const-is-pr-2} \ \text{[of 1]} \ \text{const-is-pr-3} \ \text{[of 1]} \\
\text{const-is-pr} \ \text{[of 2]} \ \text{const-is-pr-2} \ \text{[of 2]} \ \text{const-is-pr-3} \ \text{[of 2]} \\
\end{align*}
\end{proof}
definition \textbf{sgn1 :: nat \Rightarrow nat \ where}
sgn1 x = (case x of 0 ⇒ 0 | Suc y ⇒ 1)
definition \textbf{sgn2 :: nat \Rightarrow nat \ where}
sgn2 x ≡ (case x of 0 ⇒ 1 | Suc y ⇒ 0)
definition \textbf{abs-of-diff :: nat \Rightarrow nat \Rightarrow nat \ where}
abs-of-diff = $(\lambda \ x \ y. \ (x - y) + (y - x))$
\begin{lemma} \textbf{[simp]}: \textbf{sgn1 0 = 0} \ \textbf{by} \ \textbf{(simp add: sgn1-def)}
\end{lemma}
\begin{lemma} \textbf{[simp]}: \textbf{sgn1 (Suc y) = 1} \ \textbf{by} \ \textbf{(simp add: sgn1-def)}
\end{lemma}
\begin{lemma} \textbf{[simp]}: \textbf{sgn2 0 = 1} \ \textbf{by} \ \textbf{(simp add: sgn2-def)}
\end{lemma}
\begin{lemma} \textbf{[simp]}: \textbf{sgn2 (Suc y) = 0} \ \textbf{by} \ \textbf{(simp add: sgn2-def)}
\end{lemma}
\begin{lemma} \textbf{[simp]}: \textbf{x \neq 0 \Rightarrow sgn1 x = 1} \ \textbf{by} \ \textbf{(simp add: sgn1-def, cases x, auto)}
\end{lemma}
\begin{lemma} \textbf{[simp]}: \textbf{x \neq 0 \Rightarrow sgn2 x = 0} \ \textbf{by} \ \textbf{(simp add: sgn2-def, cases x, auto)}
\end{lemma}
\begin{lemma} \textbf{sgn1-nz-impl-arg-pos: sgn1 x \neq 0 \Rightarrow x > 0} \ \textbf{by} \ \textbf{(cases x) auto}
\end{lemma}
\begin{lemma} \textbf{sgn1-zero-impl-arg-zero: sgn1 x = 0 \Rightarrow x = 0} \ \textbf{by} \ \textbf{(cases x) auto}
\end{lemma}
\begin{lemma} \textbf{sgn2-nz-impl-arg-zero: sgn2 x \neq 0 \Rightarrow x = 0} \ \textbf{by} \ \textbf{(cases x) auto}
\end{lemma}
lemma \text{sgn2-zero-impl-arg-pos}: \text{sgn2 }x = 0 \implies x > 0 \text{ by (cases } x\text{) auto}

lemma \text{sgn1-nz-eq-arg-pos}: (\text{sgn1 }x \neq 0) = (x > 0) \text{ by (cases } x\text{) auto}

lemma \text{sgn1-zero-eq-arg-zero}: (\text{sgn1 }x = 0) = (x = 0) \text{ by (cases } x\text{) auto}

lemma \text{sgn2-nz-eq-arg-pos}: (\text{sgn2 }x \neq 0) = (x = 0) \text{ by (cases } x\text{) auto}

lemma \text{sgn2-zero-eq-arg-zero}: (\text{sgn2 }x = 0) = (x > 0) \text{ by (cases } x\text{) auto}

lemma \text{sgn1-pos-eq-one}: \text{sgn1 }x > 0 = \implies \text{sgn1 }x = 1 \text{ by (cases } x\text{) auto}

lemma \text{sgn2-pos-eq-one}: \text{sgn2 }x > 0 = \implies \text{sgn2 }x = 1 \text{ by (cases } x\text{) auto}

lemma \text{sgn2-eq-1-sub-arg}: \text{sgn2 }= (\lambda x. 1 - x)

proof (rule ext)
\begin{align*}
\text{fix } x \text{ show } \text{sgn2 }x = 1 - x \text{ by (cases } x\text{) auto}
\end{align*}
qed

lemma \text{sgn1-eq-1-sub-sgn2}: \text{sgn1 }= (\lambda x. 1 - (\text{sgn2 }x))

proof
\begin{align*}
\text{fix } x \text{ show } \text{sgn1 }x = 1 - \text{sgn2 }x
\end{align*}
proof –
\begin{align*}
\text{have } 1 - \text{sgn2 }x = 1 - (1 - x) \text{ by (simp add: sgn2-eq-1-sub-arg)}
\end{align*}
  then show ?thesis by (simp add: sgn1-def, cases x, auto)
qed

lemma \text{sgn2-is-pr [prec]}: \text{sgn2 }\in \text{PrimRec1}

proof –
\begin{align*}
\text{have } (\lambda x. 1 - x) \in \text{PrimRec1 by prec0}
\end{align*}
  then ?thesis by (simp add: sgn2-eq-1-sub-arg)
qed

lemma \text{sgn1-is-pr [prec]}: \text{sgn1 }\in \text{PrimRec1}

proof –
\begin{align*}
\text{from sgn2-is-pr have } (\lambda x. 1 - (\text{sgn2 }x)) \in \text{PrimRec1 by prec0}
\end{align*}
  then ?thesis by (simp add: sgn1-eq-1-sub-sgn2)
qed

lemma \text{abs-of-diff-is-pr [prec]}: \text{abs-of-diff }\in \text{PrimRec2 unfolding abs-of-diff-def by prec0}

lemma \text{abs-of-diff-eq}: (\text{abs-of-diff }x y = 0) = (x = y) \text{ by (simp add: abs-of-diff-def, arith)}

lemma \text{sf-is-pr [prec]}: \text{sf }\in \text{PrimRec1}

proof –
\begin{align*}
\text{have } S1: \text{PrimRecOp1 0 } (\lambda x y. y + x + 1) \in \text{PrimRec1}
\end{align*}
  proof (rule pr-rec1)
  \begin{align*}
  \text{show } (\lambda x y. y + x + 1) \in \text{PrimRec2 by prec0}
\end{align*}
  qed
\begin{align*}
\text{have } (\lambda x. \text{sf }x) = \text{PrimRecOp1 0 } (\lambda x y. y + x + 1) \text{ (is - = } ?f)
\end{align*}

18
proof –
  have \( \forall x. (\text{if } x = sf x) \)
proof (induct-tac x)
  show \( \text{if } 0 = sf 0 \) by (simp add: sf-at-0)
next
  fix \( x \) assume \( \text{if } x = sf x \)
  with sf-at-Suc show \( \text{if } (\text{Suc } x) = sf (\text{Suc } x) \) by auto
qed
thus \( ?\text{thesis} \) by (simp add: ext)
qed

lemma c-pair-is-pr [prec]: \( \text{c-pair } \in \text{PrimRec2} \)
proof –
  have \( \text{c-pair } = (\lambda x y. sf (x+y) + x) \) by (simp add: c-pair-def ext)
moreover from sf-is-pr have \( (\lambda x y. sf (x+y) + x) \in \text{PrimRec2} \) by prec0
ultimately show \( ?\text{thesis} \) by simp
qed

lemma if-is-pr: \( \[ p \in \text{PrimRec1}; q1 \in \text{PrimRec1}; q2 \in \text{PrimRec1} \] \implies (\lambda x. \text{if } (p x = 0) \text{ then } (q1 x) \text{ else } (q2 x)) \in \text{PrimRec1} \)
proof –
  have if-as-pr: \( (\lambda x. \text{if } (p x = 0) \text{ then } (q1 x) \text{ else } (q2 x)) = (\lambda x. (\text{sgn2 } (p x)) \ast (q1 x)) + (\text{sgn1 } (p x) \ast (q2 x)) \)
proof (rule ext)
    fix \( x \) show \( \text{if } (p x = 0) \text{ then } (q1 x) \text{ else } (q2 x)) = (\text{sgn2 } (p x)) \ast (q1 x) + (\text{sgn1 } (p x)) \ast (q2 x) \) (is \( ?\text{left} = ?\text{right} \))
    proof cases
      assume A1: \( p x = 0 \)
      then have S1: \( ?\text{left} = q1 x \) by simp
      from A1 have S2: \( ?\text{right} = q1 x \) by simp
      from S1 S2 show \( ?\text{thesis} \) by simp
    next
      assume A2: \( p x \neq 0 \)
      then have S3: \( p x > 0 \) by simp
      then show \( ?\text{thesis} \) by simp
    qed
    assume \( p \in \text{PrimRec1} \) and \( q1 \in \text{PrimRec1} \) and \( q2 \in \text{PrimRec1} \)
    then have \( (\lambda x. (\text{sgn2 } (p x)) \ast (q1 x)) + (\text{sgn1 } (p x)) \ast (q2 x)) \in \text{PrimRec1} \) by prec0
    with if-as-pr show \( ?\text{thesis} \) by simp
    qed

lemma if-eq-is-pr [prec]: \( \[ p1 \in \text{PrimRec1}; p2 \in \text{PrimRec1}; q1 \in \text{PrimRec1}; q2 \in \text{PrimRec1} \] \implies (\lambda x. \text{if } (p1 x = p2 x) \text{ then } (q1 x) \text{ else } (q2 x)) \in \text{PrimRec1} \)
proof –
  have S1: \( (\lambda x. \text{if } (p1 x = p2 x) \text{ then } (q1 x) \text{ else } (q2 x)) = (\lambda x. (\text{abs-of-diff } (p1
x) \( (p2 \ x) = 0 \) then \((q1 \ x) \ else \ (q2 \ x)) \((\textit{if-is-pr2})\) \textbf{by} (simp add: abs-of-diff-eq)

\textbf{assume} A1: \( p1 \in \text{PrimRec1} \) \textbf{and} A2: \( p2 \in \text{PrimRec1} \)

\textbf{with} abs-of-diff-is-pr \textbf{have} S2: \((\lambda \ x. \ \text{abs-of-diff} \ (p1 \ x) \ (p2 \ x)) \in \text{PrimRec1} \) \textbf{by} prec0

\textbf{assume} q1 \in \text{PrimRec1} \textbf{and} q2 \in \text{PrimRec1}

\textbf{with} S2 \textbf{have} \(?R \in \text{PrimRec1} \) \textbf{by} (rule if-is-pr)

\textbf{with} S1 \textbf{show} \(?thesis \ by simp}

\textbf{qed}

\textbf{lemma} if-is-pr2 [prec]: \[ p \in \text{PrimRec2}; \ q1 \in \text{PrimRec2}; \ q2 \in \text{PrimRec2} \] \(\implies (\lambda \ x \ y. \ (\text{abs-of-diff} \ (p1 \ x) \ (p2 \ y)) \land \text{q1} = \text{q2}) \in \text{PrimRec2} \)

\textbf{proof} –

\textbf{have} if-as-pr: \((\lambda \ x \ y. \ (p1 \ x) \ (p2 \ y)) = (\lambda \ x \ y. \ (\text{sgn2} \ (p1 \ x) \ (p2 \ y)) \land \text{q1} = \text{q2}) \)

\textbf{proof} (rule ext, rule ext)

\textbf{fix} x \textbf{fix} y \textbf{show} \((\lambda \ x \ y. \ (p1 \ x) \ (p2 \ y)) = (\lambda \ x \ y. \ (\text{sgn2} \ (p1 \ x) \ (p2 \ y)) \land \text{q1} = \text{q2}) \)

\textbf{proof} \textbf{cases}

\textbf{assume} A1: \( p \ x \ y = 0 \)

\textbf{then} \textbf{have} S1: \(?left = q1 \ x \ y \) \textbf{by simp}

\textbf{from} A1 \textbf{have} S2: \(?right = q1 \ x \ y \) \textbf{by simp}

\textbf{from} S1 \textbf{and} S2 \textbf{show} \(?thesis \ by simp}

next

\textbf{assume} A2: \( p \ x \ y \neq 0 \)

\textbf{then} \textbf{have} S3: \( p \ x \ y > 0 \) \textbf{by simp}

\textbf{then} \textbf{show} \(?thesis \ by simp}

\textbf{qed}

\textbf{assume} p \in \text{PrimRec2} \textbf{and} q1 \in \text{PrimRec2} \textbf{and} q2 \in \text{PrimRec2}

\textbf{then} \textbf{have} \(\lambda \ x \ y. \ (\text{abs-of-diff} \ (p1 \ x) \ (p2 \ y)) \land \text{q1} = \text{q2}) \in \text{PrimRec2} \)

\textbf{by} \textit{prec0}

\textbf{with} \textit{if-as-pr} \textbf{show} \(?thesis \ by simp}

\textbf{qed}

\textbf{lemma} if-eq-is-pr2: \[ p1 \in \text{PrimRec2}; \ p2 \in \text{PrimRec2}; q1 \in \text{PrimRec2}; q2 \in \text{PrimRec2} \] \(\implies (\lambda \ x \ y. \ (p1 \ x \ y = p2 \ x \ y) \land \text{q1} = \text{q2}) \in \text{PrimRec2} \)

\textbf{proof} –

\textbf{have} S1: \((\lambda \ x \ y. \ (p1 \ x \ y = p2 \ x \ y) \land \text{q1} = \text{q2}) \in \text{PrimRec2} \)

\textbf{assume} A1: \( p1 \in \text{PrimRec2} \) \textbf{and} A2: \( p2 \in \text{PrimRec2} \)

\textbf{with} abs-of-diff-is-pr \textbf{have} S2: \((\lambda \ x \ y. \ (p1 \ x \ y = p2 \ x \ y) \land \text{q1} = \text{q2}) \in \text{PrimRec2} \)

\textbf{by} \textit{prec0}

\textbf{assume} q1 \in \text{PrimRec2} \textbf{and} q2 \in \text{PrimRec2}

\textbf{with} S2 \textbf{have} \(?R \in \text{PrimRec2} \) \textbf{by} (rule if-is-pr2)

\textbf{with} S1 \textbf{show} \(?thesis \ by simp}

\textbf{qed}
lemma if-is-pr3 [prec]: \[ p \in \text{PrimRec3}; q1 \in \text{PrimRec3}; q2 \in \text{PrimRec3} \] \implies (\lambda x y z. \text{if } (p x y z = 0) \text{ then } (q1 x y z) \text{ else } (q2 x y z)) \in \text{PrimRec3}

proof

have if-as-pr: \( (\lambda x y z. \text{if } (p x y z = 0) \text{ then } (q1 x y z) \text{ else } (q2 x y z)) = (\lambda x y z. (\text{sgn2}(p x y z)) \ast (q1 x y z) + (\text{sgn1}(p x y z)) \ast (q2 x y z)) \)

proof (rule ext, rule ext, rule ext)

fix x fix y fix z show \((\lambda x y z. \text{if } (p x y z = 0) \text{ then } (q1 x y z) \text{ else } (q2 x y z)) = (\lambda x y z. (\text{sgn2}(p x y z)) \ast (q1 x y z) + (\text{sgn1}(p x y z)) \ast (q2 x y z)) \)

proof cases

assume A1: \( p x y z = 0 \)
then have S1: \(?left = q1 x y z \) by simp
from A1 have S2: \(?right = q1 x y z \) by simp
from S1 S2 show \(?thesis \) by simp
next

assume A2: \( p x y z \neq 0 \)
then have S3: \( p x y z > 0 \) by simp
then show \(?thesis \) by simp
qed

assume p \in \text{PrimRec3} and q1 \in \text{PrimRec3} and q2 \in \text{PrimRec3}

then have \((\lambda x y z. (\text{sgn2}(p x y z)) \ast (q1 x y z) + (\text{sgn1}(p x y z)) \ast (q2 x y z)) \) \in \text{PrimRec3}

by prec0

with if-as-pr show \(?thesis \) by simp
qed

lemma if-eq-is-pr3: \[ p1 \in \text{PrimRec3}; p2 \in \text{PrimRec3}; q1 \in \text{PrimRec3}; q2 \in \text{PrimRec3} \] \implies (\lambda x y z. \text{if } (p1 x y z = p2 x y z) \text{ then } (q1 x y z) \text{ else } (q2 x y z)) \in \text{PrimRec3}

proof

have S1: \( (\lambda x y z. (\text{abs-of-diff}(p1 x y z) \ast (q1 x y z) + (\text{sgn1}(p2 x y z)) \ast (q2 x y z)) \) \in \text{PrimRec3}

by prec0

assume q1 \in \text{PrimRec3} and q2 \in \text{PrimRec3}

with abs-of-diff-is-pr have S2: \( (\lambda x y z. \text{abs-of-diff}(p1 x y z) \ast (q1 x y z) \ast (q2 x y z)) \) \in \text{PrimRec3}

by prec0

assume q1 \in \text{PrimRec3} and q2 \in \text{PrimRec3}

with S2 have \(?R \) \in \text{PrimRec3} by (rule if-is-pr3)

with S1 show \(?thesis \) by simp
qed

ML

fun get-if-by-index 1 = @{thm if-eq-is-pr}
| get-if-by-index 2 = @{thm if-eq-is-pr2}
| get-if-by-index 3 = @{thm if-eq-is-pr3}
| get-if-by-index - = raise BadArgument

fun if-comp-tac ctxt = SUBGOAL (fn (t, i) =>
let 
  val t = extract-trueprop-arg (Logic.strip-imp-concl t)
  val (t1, t2) = extract-set-args t
  val n2 = 
    let 
      val Const(s, _) = t2
    in 
      get-num-by-set s
    end
  val (name, _, n1) = extract-free-arg t1
in 
  if name = @\{const-name If\} then 
    resolve-tac ctxt [get-if-by-index n2] i 
  else 
    let 
      val comp = get-comp-by-indexes (n1, n2)
    in 
      Rule-Insts.res-inst-tac ctxt 
        [(((f, 0), Position.none), Variable.revert-fixed ctxt name) [] comp] i 
    end 
  end 
handle BadArgument => no-tac)

fun prec-tac ctxt facts i = 
  Method.insert-tac ctxt facts i THEN 
  REPEAT (resolve-tac ctxt [@{thm const-is-pr}, @{thm const-is-pr-2}, @{thm const-is-pr-3}] i ORELSE 
    assume-tac ctxt i ORELSE if-comp-tac ctxt i)

method-setup prec = ⟨⟨ 
  Attrib.thms >>= (fn ths => fn ctxt => Method.METHOD (fn facts => 
    HEADGOAL (prec-tac ctxt [facts @ Named-Theorems.get ctxt @{thm named-theorems prec}])))) 
⟩⟩

apply primitive recursive functions 2.2 Bounded least operator
definition 
b-least :: (nat ⇒ nat ⇒ nat) ⇒ (nat ⇒ nat) where 
b-least f x ≡ (Least (%y. y = x ∨ (y < x ∧ (f x y) ≠ 0)))
definition 
b-least2 :: (nat ⇒ nat ⇒ nat) ⇒ (nat ⇒ nat ⇒ nat) where 
b-least2 f x y ≡ (Least (%z. z = y ∨ (z < y ∧ (f x z) ≠ 0)))

lemma b-least-aux1: b-least f x = x ∨ (b-least f x < x ∧ (f x (b-least f x)) ≠ 0)
proof 
  let □P = %y. y = x ∨ (y < x ∧ (f x y) ≠ 0)
have \(?P\ x\) by simp
then have \(?P\ (\text{Least } ?P)\) by (rule LeastI)
thus \(?\text{thesis}\) by (simp add: b-least-def)
qed

lemma b-least-le-arg: b-least \(f\ x \leq x\)
proof –
  have b-least \(f\ x = x \lor (\text{b-least } f\ x < x \land (f\ x \ (\text{b-least } f\ x)) \neq 0)\) by (rule b-least-aux1)
  from this show \(?\text{thesis}\) by (arith)
qed

lemma less-b-least-impl-zero: \(y < \text{b-least } f\ x\) \(\implies f\ x\ y = 0\)
proof (rule ccontr)
  assume A1: \(f\ x\ y \neq 0\)
  assume ¬ b-least \(f\ x \leq y\)
  then have \(y < \text{b-least } f\ x\) by simp
  with A1 show False by (simp add: less-b-least-impl-zero)
qed

lemma nz-impl-b-least-le: \((f\ x\ y) \neq 0 \implies (\text{b-least } f\ x) \leq y\)
proof
  assume A1: \((f\ x\ y) \neq 0\)
  assume ¬ b-least \(f\ x \leq y\)
  then have \(y < \text{b-least } f\ x\) by simp
  with A1 show False by (simp add: nz-impl-b-least-le)
qed

lemma b-least-less-impl-nz: \(\text{b-least } f\ x < x\) \(\implies f\ x\ (\text{b-least } f\ x) \neq 0\)
proof –
  assume A1: \(\text{b-least } f\ x < x\)
  have b-least \(f\ x = x \lor (\text{b-least } f\ x < x \land (f\ x \ (\text{b-least } f\ x)) \neq 0)\) by (rule b-least-aux1)
  from A1 this show \(?\text{thesis}\) by simp
qed

lemma b-least-less-impl-eq: \(\text{b-least } f\ x < x\) \(\implies (\text{b-least } f\ x) = (\text{Least } (\%y. (f\ x\ y) \neq 0))\)
proof –
  assume A1: \(\text{b-least } f\ x < x\ (\text{is } ?b < -)\)
  let \(?B = (\text{Least } (\%y. (f\ x\ y) \neq 0))\)
  from A1 have S1: \(f\ x\ ?b \neq 0\) by (rule b-least-less-impl-nz)
  from S1 have S2: \(?B \leq ?b\) by (rule Least-le)
  from S1 have S3: \(f\ x\ ?B \neq 0\) by (rule LeastI)
  from S3 have S4: \(?b \leq ?B\) by (rule nz-impl-b-least-le)
  from S2 S4 show \(?\text{thesis}\) by simp
lemma less-b-least-impl-zero2: \([y < x; b\text{-}\text{least}\ f\ x = x] \implies f\ y = 0\) by \((\text{simp add: less-b-least-impl-zero})\)

lemma nz-impl-b-least-less: \([y < x; (f\ x\ y) \neq 0] \implies (b\text{-}\text{least}\ f\ x) < x\)
proof -
  assume \(A1: y < x\)
  assume \(f\ x\ y \neq 0\)
  then have \((b\text{-}\text{least}\ f\ x) \leq y\) by \((\text{rule nz-impl-b-least-le})\)
  with \(A1\) show \(?\text{thesis}\) by \(\text{simp}\)
qed

lemma b-least-aux2: \([y < x; (f\ x\ y) \neq 0] \implies (b\text{-}\text{least}\ f\ x) = (\text{Least} (z. z = y \lor (z < y \land (f\ x\ z) \neq 0)))\)
proof -
  let \(?P = \%z. z = y \lor (z < y \land (f\ x\ z) \neq 0)\)
  have \(?P\ y\) by \(\text{simp}\)
  then have \(?P\ (\text{Least } ?P)\) by \((\text{rule LeastI})\)
  thus \(?\text{thesis}\) by \((\text{simp add: b-least2-def})\)
qed

lemma b-least2-aux1: \(b\text{-}\text{least2}\ f\ x\ y = y \lor (b\text{-}\text{least2}\ f\ x\ y < y \land (f\ x\ (b\text{-}\text{least2}\ f\ x\ y)) \neq 0)\)
proof -
  let \(?B = b\text{-}\text{least2}\ f\ x\ y\)
  have \(?B = y \lor (?B < y \land (f\ x\ ?B) \neq 0)\) by \((\text{rule b-least2-aux1})\)
  from this show \(?\text{thesis}\) by \(\text{arith}\)
qed

lemma b-least2-le-arg: \(b\text{-}\text{least2}\ f\ x\ y \leq y\)
proof -
  let \(?B = b\text{-}\text{least2}\ f\ x\ y\)
  have \(?B = y \lor (?B < y \land (f\ x\ ?B) \neq 0)\) by \((\text{rule b-least2-aux1})\)
  from this show \(?\text{thesis}\) by \(\text{arith}\)
qed

lemma less-b-least2-impl-zero: \(z < b\text{-}\text{least2}\ f\ x\ y \implies f\ x\ z = 0\)
proof -
  assume \(A1: z < b\text{-}\text{least2}\ f\ x\ y\) (is \(<\ ?b)\)
  have \(b\text{-}\text{least2}\ f\ x\ y \leq y\) by \((\text{rule b-least2-le-arg})\)
  with \(A1\) have \(S1: z < y\) by \(\text{simp}\)
  with \(A1\) have \(z < (\text{Least} (z. z = y \lor (z < y \land (f\ x\ z) \neq 0)))\) by \((\text{simp add: b-least2-def})\)
  then have \(\neg (z = y \lor (z < y \land (f\ x\ z) \neq 0))\) by \((\text{rule not-less-Least})\)
  with \(S1\) show \(?\text{thesis}\) by \(\text{simp}\)
qed

lemma nz-impl-b-least2-le: \((f\ x\ z) \neq 0 \implies (b\text{-}\text{least2}\ f\ x\ y) \leq z\)

\[
\]
proof
  assume A1: \( f \ x \ z \neq 0 \)
  have S1: \( z < b\text{-least2} \ f \ x \ y \implies f \ x \ z = 0 \)
  by (rule less-b\text{-least2-impl-zero})
  from A1 S1 show ?thesis by arith
qed

lemma b\text{-least2-less-impl-nz}: \( b\text{-least2} \ f \ x \ y < y \implies f \ x \ (b\text{-least2} \ f \ x \ y) \neq 0 \)
proof
  assume A1: \( b\text{-least2} \ f \ x \ y < y \)
  have b\text{-least2} \ f \ x \ y = y \lor (b\text{-least2} \ f \ x \ y < y \land (f \ x \ (b\text{-least2} \ f \ x \ y)) \neq 0) \)
  by (rule b\text{-least2-aux1})
  with A1 show ?thesis by simp
qed

lemma b\text{-least2-less-impl-eq}: \( b\text{-least2} \ f \ x \ y < y \implies (b\text{-least2} \ f \ x \ y) = (\text{Least} (%z. \ (f \ x \ z) \neq 0)) \)
proof
  assume A1: \( z < y \) and \( A2: \ f \ x \ z \neq 0 \)
  from A1 A2 have S1: \( b\text{-least2} \ f \ x \ y = y \)
  by (rule nz\text{-impl-b\text{-least2-le}})
with A1 show ?thesis by simp
qed

lemma less-b\text{-least2-impl-zero2}: \( \[ z < y; \ b\text{-least2} \ f \ x \ y = y \] \implies f \ x \ z = 0 \)
proof
  assume \( z < y \) and \( b\text{-least2} \ f \ x \ y = y \)
  hence \( z < b\text{-least2} \ f \ x \ y \)
  by simp
  thus ?thesis by (rule less-b\text{-least2-impl-zero})
qed

lemma nz-b\text{-least2-impl-less}: \( \[ z < y; \ (f \ x \ z) \neq 0 \] \implies (b\text{-least2} \ f \ x \ y) < y \)
proof (rule ccontr)
  assume A1: \( z < y \)
  assume A2: \( f \ x \ z \neq 0 \)
  assume \( \neg (b\text{-least2} \ f \ x \ y) < y \) then have A3: \( y \leq (b\text{-least2} \ f \ x \ y) \)
  by simp
  have b\text{-least2} \ f \ x \ y \leq y \)
  by (rule b\text{-least2-le-arg})
  with A3 have b\text{-least2} \ f \ x \ y = y \)
  by simp
  with A1 have f \ x \ z = 0 \)
  by (rule less-b\text{-least2-impl-zero2})
  with A2 show False by simp
qed

lemma b\text{-least2-less-impl-eq2}: \( \[ z < y; \ (f \ x \ z) \neq 0 \] \implies (b\text{-least2} \ f \ x \ y) = (\text{Least} (%z. \ (f \ x \ z) \neq 0)) \)
proof
  assume A1: \( z < y \) and \( A2: \ f \ x \ z \neq 0 \)
  from A1 A2 have S1: \( b\text{-least2} \ f \ x \ y < y \)
  by (rule nz\text{-impl-b\text{-least2-le}})
thus \( \text{thesis by (rule b-least2-less-impl-eq)} \)

qed

lemma b-least2-aux2: \( b\text{-}\text{least2 } f \ x \ y < y \implies b\text{-}\text{least2 } f \ x \ (\text{Suc} \ y) = b\text{-}\text{least2 } f \ x \ y \)
proof –
let \( ?B = b\text{-}\text{least2 } f \ x \ y \)
assume \( A1: ?B < y \)
from \(A1\) have \( S1: f \ x \ ?B \neq 0 \) by (rule b-least2-less-impl-nz)
from \(S1\) have \( S2: b\text{-}\text{least2 } f \ x \ (\text{Suc} \ y) \leq ?B \) by (simp add: nz-impl-b-least2-le)
from \(A1 \ S2\) have \( S3: b\text{-}\text{least2 } f \ x \ (\text{Suc} \ y) < ?B \) by (rule nz-b-least2-impl-less)
from \(S3\) have \( \neg \ (b\text{-}\text{least2 } f \ x \ (\text{Suc} \ y) < y) \) by auto
from \(S3\) show \( \text{thesis by simp} \)
qed

lemma b-least2-aux3: \( b\text{-}\text{least2 } f \ x \ y = y; f \ x \ y \neq 0 \implies b\text{-}\text{least2 } f \ x \ (\text{Suc} \ y) = y \)
proof –
assume \( A1: b\text{-}\text{least2 } f \ x \ y = y \)
assume \( A2: f \ x \ y \neq 0 \)
from \(A2\) have \( S1: b\text{-}\text{least2 } f \ x \ (\text{Suc} \ y) \leq y \) by (rule nz-impl-b-least2-le)
from \(S1\) have \( S2: b\text{-}\text{least2 } f \ x \ (\text{Suc} \ y) < y \implies False \)
proof –
assume \( A2-1: b\text{-}\text{least2 } f \ x \ (\text{Suc} \ y) < y \) (is \( ?z < - \))
from \(A2-1\) have \( S2-1: ?z < \text{Suc} \ y \) by simp
from \(S2-1\) have \( S2-2: f \ x \ ?z \neq 0 \) by (rule b-least2-less-impl-nz)
from \(A2-1 \ S2-2\) have \( S2-3: b\text{-}\text{least2 } f \ x \ y < y \) by (rule nz-b-least2-impl-less)
from \(S2-3\) show \( \text{thesis by simp} \)
qed

from \(S2\) have \( S3: \neg \ (b\text{-}\text{least2 } f \ x \ (\text{Suc} \ y) < y) \) by auto
from \(S1 \ S3\) show \( \text{thesis by simp} \)
qed

lemma b-least2-mono: \( y1 \leq y2 \implies b\text{-}\text{least2 } f \ x \ y1 \leq b\text{-}\text{least2 } f \ x \ y2 \)
proof (rule ccontr)
assume \( A1: y1 \leq y2 \)
let \( ?b1 = b\text{-}\text{least2 } f \ x \ y1 \) and \( ?b2 = b\text{-}\text{least2 } f \ x \ y2 \)
assume \( \neg \ ?b1 < ?b2 \) then have \( A2: \ ?b2 < \ ?b1 \) by simp
have \( S1: \ ?b1 \leq y1 \) by (rule b-least2-le-arg)
have \( S2: \ ?b2 \leq y2 \) by (rule b-least2-le-arg)
from \(A1 \ A2 \ S1 \ S2\) have \( S3: \ ?b2 < y2 \) by simp
then have \( S4: f \ x \ ?b2 \neq 0 \) by (rule b-least2-less-impl-nz)
from \(A2\) have \( S5: f \ x \ ?b2 = 0 \) by (rule less-b-least2-impl-zero)
from \(S4 \ S5\) show \( False \) by simp
qed

lemma b-least2-aux4: \( b\text{-}\text{least2 } f \ x \ y = y; f \ x \ y = 0 \implies b\text{-}\text{least2 } f \ x \ (\text{Suc} \ y) = \text{Suc} \ y \)
proof –
assume \( A1: b\text{-}\text{least2 } f \ x \ y = y \)
assume $A2: f \cdot x \cdot y = 0$

have $S1: b\text{-}\text{least}2 f \cdot x \cdot (\text{Suc} \cdot y) \leq \text{Suc} \cdot y$ by (rule $b\text{-}\text{least}2\text{-le-arg}$)

have $S2: y \leq b\text{-}\text{least}2 f \cdot x \cdot (\text{Suc} \cdot y)$

proof –
  have $y \leq \text{Suc} \cdot y$ by simp
  then have $b\text{-}\text{least}2 f \cdot x \cdot y \leq b\text{-}\text{least}2 f \cdot x \cdot (\text{Suc} \cdot y)$ by (rule $b\text{-}\text{least}2\text{-mono}$)
  with $A1$ show $\text{thesis by simp}$

qed

from $S1$ $S2$ have $b\text{-}\text{least}2 f \cdot x \cdot (\text{Suc} \cdot y) = y \lor b\text{-}\text{least}2 f \cdot x \cdot (\text{Suc} \cdot y) = \text{Suc} \cdot y$ by arith

moreover

{  
  assume $A3: b\text{-}\text{least}2 f \cdot x \cdot (\text{Suc} \cdot y) = y$
  have $f \cdot x \cdot y \neq 0$
  proof –
    have $y < \text{Suc} \cdot y$ by simp
    with $A3$ have $b\text{-}\text{least}2 f \cdot x \cdot (\text{Suc} \cdot y) < \text{Suc} \cdot y$ by simp
    from this have $f \cdot x \cdot (b\text{-}\text{least}2 f \cdot x \cdot (\text{Suc} \cdot y)) \neq 0$ by (simp add: $b\text{-}\text{least}2\text{-less-impl-nz}$)
    with $A3$ show $f \cdot x \cdot y \neq 0$ by simp
  qed
  with $A2$ have $\text{thesis by simp}$
}

moreover

{  
  assume $b\text{-}\text{least}2 f \cdot x \cdot (\text{Suc} \cdot y) = \text{Suc} \cdot y$
  then have $\text{thesis by simp}$
}

ultimately show $\text{thesis by blast}$

qed

lemma $b\text{-}\text{least}2\text{-at-zero}: b\text{-}\text{least}2 f \cdot x \cdot 0 = 0$

proof –
  have $S1: b\text{-}\text{least}2 f \cdot x \cdot 0 \leq 0$ by (rule $b\text{-}\text{least}2\text{-le-arg}$)
  from $S1$ show $\text{thesis by auto}$

qed

theorem pr-$b\text{-}\text{least}2: f \in \text{PrimRec}2 \implies b\text{-}\text{least}2 f \in \text{PrimRec2}$

proof –
  def $\text{loc-Op1} = (\lambda (f::\text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat}) \cdot x \cdot y \cdot z. \cdot (\text{sgn}1 \cdot (z - y)) \cdot y + (\text{sgn}2 \cdot (z - y))\cdot((\text{sgn}1 \cdot (f \cdot x \cdot z))\cdot z + (\text{sgn}2 \cdot (f \cdot x \cdot z))\cdot (\text{Suc} \cdot z)))$
  def $\text{loc-Op2} = (\lambda f. \text{PrimRecOp-last} \cdot (\lambda x. \cdot 0) \cdot (\text{loc-Op1} \cdot f))$
  have $\text{loc-op2-lm-1:} \land f \cdot x \cdot y. \cdot \text{loc-Op2} f \cdot x \cdot y < y \implies \text{loc-Op2} f \cdot x \cdot (\text{Suc} \cdot y) = \text{loc-Op2} f \cdot x \cdot y$
  proof –
    fix $f \cdot x \cdot y$
    let $?b = \text{loc-Op2} f \cdot x \cdot y$
    have $S1: \text{loc-Op2} f \cdot x \cdot (\text{Suc} \cdot y) = (\text{loc-Op1} \cdot f) \cdot x \cdot ?b \cdot y$ by (simp add: $\text{loc-Op2}\text{-def}$)
    assume $?b < y$
    then have $y - ?b > 0$ by simp
then have \( \text{loc-Op1 } f \ x \ ?b \ y = \ ?b \) by (simp add: \( \text{loc-Op1-def} \))
with \( S1 \) show \( \text{loc-Op2 } f \ x \ y \ < \ y \Rightarrow \text{loc-Op2 } f \ x \ (\text{Suc } y) = \text{loc-Op2 } f \ x \ y \) by simp
qed
have \( \text{loc-op2-lm-2} \): \( \forall f \ x \ y. \left[ \neg(\text{loc-Op2 } f \ x \ y < y); f \ x \ y \neq 0 \right] \Rightarrow \text{loc-Op2 } f \ x \ (\text{Suc } y) = y \)
proof -
  fix \( f \ x \ y \)
  let \( ?b = \text{loc-Op2 } f \ x \ y \) and \( ?h = \text{loc-Op1 } f \)
  have \( S1 \): \( \text{loc-Op2 } f \ x \ (\text{Suc } y) = ?h \ ?b \ y \) by (simp add: \( \text{loc-Op2-def} \))
  assume \( \neg(?b < y) \)
  then have \( S2 \): \( y - ?b = 0 \) by simp
  assume \( f \ x \ y \neq 0 \)
  with \( S2 \) have \( ?h \ ?b \ y = y \) by (simp add: \( \text{loc-Op1-def} \))
  with \( S1 \) show \( \text{loc-Op2 } f \ x \ (\text{Suc } y) = y \) by simp
qed
have \( \text{loc-op2-lm-3} \): \( \forall f \ x \ y. \left[ \neg(\text{loc-Op2 } f \ x \ y < y); f \ x \ y = 0 \right] \Rightarrow \text{loc-Op2 } f \ x \ (\text{Suc } y) = \text{Suc } y \)
proof -
  fix \( f \ x \ y \)
  let \( ?b = \text{loc-Op2 } f \ x \ y \) and \( ?h = \text{loc-Op1 } f \)
  have \( S1 \): \( \text{loc-Op2 } f \ x \ (\text{Suc } y) = ?h \ ?b \ y \) by (simp add: \( \text{loc-Op2-def} \))
  assume \( \neg(?b < y) \)
  then have \( S2 \): \( y - ?b = 0 \) by simp
  assume \( f \ x \ y = 0 \)
  with \( S2 \) have \( ?h \ ?b \ y = \text{Suc } y \) by (simp add: \( \text{loc-Op1-def} \))
  with \( S1 \) show \( \text{loc-Op2 } f \ x \ (\text{Suc } y) = \text{Suc } y \) by simp
qed
have \( \text{Op2-eq-b-least2-at-point} \): \( \forall f \ x \ y. \ \text{loc-Op2 } f \ x \ y = \text{b-least2 } f \ x \ y \)
proof -
  fix \( f \ x \ y \) show \( \forall y. \ \text{loc-Op2 } f \ x \ y = \text{b-least2 } f \ x \ y \)
proof (induct-tac \( y \))
  show \( \text{loc-Op2 } f \ x \ 0 = \text{b-least2 } f \ x \ 0 \) by (simp add: \( \text{loc-Op2-def} \text{ b-least2-at-zero} \))
next
  fix \( y \)
  assume \( A1 \): \( \text{loc-Op2 } f \ x \ y = \text{b-least2 } f \ x \ y \)
  then show \( \text{loc-Op2 } f \ x \ (\text{Suc } y) = \text{b-least2 } f \ x \ (\text{Suc } y) \)
proof cases
  assume \( A2 \): \( \text{loc-Op2 } f \ x \ y < y \)
  then have \( S1 \): \( \text{loc-Op2 } f \ x \ (\text{Suc } y) = \text{loc-Op2 } f \ x \ y \) by (rule \( \text{loc-op2-bm-1} \))
  from \( A1 \) \( A2 \) have \( \text{b-least2 } f \ x \ y < y \) by simp
  then have \( S2 \): \( \text{b-least2 } f \ x \ (\text{Suc } y) = \text{b-least2 } f \ x \ y \) by (rule \( \text{b-least2-aux2} \))
  from \( A1 \) \( S1 \) \( S2 \) show \( \?thesis \) by simp
next
  assume \( A3 \): \( \neg \text{loc-Op2 } f \ x \ y < y \)
  have \( A3': \text{b-least2 } f \ x \ y = y \)
  proof -
    have \( \text{b-least2 } f \ x \ y \leq y \) by (rule \( \text{b-least2-le-arg} \))
    from \( A1 \) \( A3 \) this show \( \?thesis \) by simp
  qed
then show \textit{thesis}

proof cases

assume \(A_4\): \(f \, x \, y \neq 0\)

with \(A_3\) have \(S_3\): \textit{loc-Op2} \(f \, x \) \((\text{Suc} \, y) = y \) by (rule \textit{loc-op2-lm-2})

from \(A_3' \, A_4\) have \(S_4\): \textit{b-least2} \(f \, x \) \((\text{Suc} \, y) = y \) by (rule \textit{b-least2-aux3})

from \(S_3 \, S_4\) show \textit{thesis} by simp

next

assume \(\neg \, f \, x \, y \neq 0\)

then have \(A_5\): \(f \, x \, y = 0\) by simp

with \(A_3\) have \(S_5\): \textit{loc-Op2} \(f \, x \) \((\text{Suc} \, y) = \text{Suc} \, y \) by (rule \textit{loc-op2-lm-3})

from \(A_3' \, A_5\) have \(S_6\): \textit{b-least2} \(f \, x \) \((\text{Suc} \, y) = \text{Suc} \, y \) by (rule \textit{b-least2-aux4})

from \(S_5 \, S_6\) show \textit{thesis} by simp

qed

have \(\textit{Op2-eq-b-least2}\): \textit{loc-Op2} = \textit{b-least2} by (simp add: \textit{Op2-eq-b-least2-at-point ext})

assume \(A_1\): \(f \in \textit{PrimRec2}\)

have \(\textit{pr-loc-Op2}\): \textit{loc-Op2} \(f \in \textit{PrimRec2}\)

proof -

from \(A_1\) have \(S_1\): \textit{loc-Op1} \(f \in \textit{PrimRec3}\) by (simp add: \textit{loc-Op1-def}, \textit{prec})

from \(\textit{pr-zero} \, S_1\) have \(S_2\): \textit{PrimRecOp-last} \((\lambda \, x.0) \) \((\text{loc-Op1} \, f) \) \(\in \textit{PrimRec2}\) by (rule \textit{pr-rec-last})

from this show \textit{thesis} by (simp add: \textit{loc-Op2-def})

qed

from \(\textit{Op2-eq-b-least2} \, this\) show \textit{b-least2} \(f \in \textit{PrimRec2}\) by simp

qed

lemma \(\textit{b-least-def1}\): \textit{b-least} \(f = (\lambda \, x. \textit{b-least2} \, f \, x \, x)\) by (simp add: \textit{b-least2-def b-least-def ext})

theorem \(\textit{pr-b-least}\): \(f \in \textit{PrimRec2} \Rightarrow \textit{b-least} \, f \in \textit{PrimRec1}\)

proof -

assume \(f \in \textit{PrimRec2}\)

then have \(\textit{b-least2} \, f \in \textit{PrimRec2}\) by (rule \textit{pr-b-least2})

from \(\textit{pr-id1-1} \, \textit{pr-id1-1}\) have \((\lambda \, x. \textit{b-least2} \, f \, x \, x) \in \textit{PrimRec1}\) by (rule \textit{pr-comp2-1})

then show \textit{thesis} by (simp add: \textit{b-least-def1})

qed

2.3 Examples

theorem \(\textit{c-sum-as-b-least}\): \(\textit{c-sum} = (\lambda \, u. \textit{b-least2} \, (\lambda \, u \, z. \, (\textit{sgn1} \, (\textit{sf} \,(z+1) - u)))) \, u \) \((\text{Suc} \, u)\)

proof (rule \textit{ext})

fix \(u\) show \(\textit{c-sum} \, u = \textit{b-least2} \, (\lambda \, u \, z. \, (\textit{sgn1} \, (\textit{sf} \,(z+1) - u)))) \, u \) \((\text{Suc} \, u)\)

proof -

have \(\textit{lm-1}: \, (\lambda \, x \, y. \, (\textit{sgn1} \, (\textit{sf} \,(y+1) - x) \neq 0)) = (\lambda \, x \, y. \, (x \, < \, \textit{sf} \,(y+1)))\)

\end{verbatim}
proof (rule ext, rule ext)
  fix x y show (sgn1 (sf(y+1) - x) ≠ 0) = (x < sf(y+1))
proof -
  have (sgn1 (sf(y+1) - x) ≠ 0) = (sf(y+1) - x > 0) by (rule sgn1-nz-eq-arg-pos)
  thus (sgn1 (sf(y+1) - x) ≠ 0) = (x < sf(y+1)) by auto
qed

proof
  have S1: ?f u u ≠ 0
proof -
  have S1-1: u+1 ≤ sf(u+1) by (rule arg-le-sf)
  have S1-2: u < u+1 by simp
  from S1-1 S1-2 have S1-3: u < sf(u+1) by simp
  from S1-3 have S1-4: sf(u+1) - u > 0 by simp
  from S1-4 have S1-5: sgn1 (sf(u+1)-u) = 1 by simp
  from S1-5 show ?thesis by simp
qed

proof
  have S3: u < Suc u by simp
  from S3 S1 have S4: b-least2 ?f u (Suc u) = (Least (%z. (?f u z) ≠ 0)) by (rule b-least2-less-impl-eq2)
  let ?P = λ u z. ?f u z ≠ 0
  let ?Q = λ u z. u < sf(z+1)
  from lm-1 have S6: ?P = ?Q by simp
  from S6 have S7: (%z. ?P u z) = (%z. ?Q u z) by (rule fun-cong)
  from S7 have S8: (Least (%z. ?P u z)) = (Least (%z. ?Q u z)) by auto
  from S4 S8 have S9: b-least2 ?f u (Suc u) = (Least (%z. u < sf(z+1))) by (rule trans)
  thus ?thesis by (simp add: c-sum-def)
qed

proof
  let $?f = λ u z. (sgn1 (sf(z+1) - u))
  have S1: (λ u z. sgn1 ((sf(z+1) - u))) ∈ PrimRec2 by prec
  def D1: g == b-least2 $?f
  from D1 S1 have g ∈ PrimRec2 by (simp add: pr-b-least2)
  then have S2: (λ u. g u (Suc u)) ∈ PrimRec1 by prec
  from D1 have c-sum = (λ u. g u (Suc u)) by (simp add: c-sum-as-b-least-ext)
  with S2 show ?thesis by simp
qed

proof [prec]: c-fst ∈ PrimRec1
proof -
  have S1: (λ u. c-fst u) = (λ u. (u - sf (c-sum u))) by (simp add: c-fst-def ext)
  from c-sum-is-pr have (λ u. (u - sf (c-sum u))) ∈ PrimRec1 by prec
  with S1 show ?thesis by simp
qed
theorem c-snd-is-pr [precc]: c-snd ∈ PrimRec1
proof
  have S1: c-snd = (λ u. (c-sum u) − (c-fst u)) by (simp add: c-snd-def ext)
  from c-sum-is-pr c-fst-is-pr have S2: (λ u. (c-sum u) − (c-fst u)) ∈ PrimRec1
  by preceq
  from S1 this show ?thesis by simp
qed

theorem pr-1-to-2: f ∈ PrimRec1 ⇒ (λ x y. f (c-pair x y)) ∈ PrimRec2 by preceq

theorem pr-2-to-1: f ∈ PrimRec2 ⇒ (λ x. f (c-fst x) (c-snd x)) ∈ PrimRec1 by preceq

definition pr-conv-1-to-2 = (λ f x y. f (c-pair x y))
definition pr-conv-1-to-3 = (λ f x y z. f (c-pair x y) z)
definition pr-conv-2-to-1 = (λ f x. f (c-fst x) (c-snd x))
definition pr-conv-3-to-1 = (λ f x. f (c-fst x) (c-snd x) (c-snd x))
definition pr-conv-3-to-2 = (λ f. pr-conv-1-to-2 (pr-conv-3-to-1 f))
definition pr-conv-3-to-3 = (λ f. pr-conv-1-to-3 (pr-conv-2-to-1 f))

lemma [simp]: pr-conv-1-to-2 (pr-conv-2-to-1 f) = f by (simp add: pr-conv-1-to-2-def pr-conv-2-to-1-def)
lemma [simp]: pr-conv-2-to-1 (pr-conv-1-to-2 f) = f by (simp add: pr-conv-1-to-2-def pr-conv-2-to-1-def)
lemma [simp]: pr-conv-1-to-3 (pr-conv-3-to-1 f) = f by (simp add: pr-conv-1-to-3-def pr-conv-3-to-1-def)
lemma [simp]: pr-conv-3-to-1 (pr-conv-1-to-3 f) = f by (simp add: pr-conv-1-to-3-def pr-conv-3-to-1-def)
lemma [simp]: pr-conv-3-to-2 (pr-conv-2-to-3 f) = f by (simp add: pr-conv-3-to-2-def pr-conv-2-to-3-def)
lemma [simp]: pr-conv-2-to-3 (pr-conv-3-to-2 f) = f by (simp add: pr-conv-3-to-2-def pr-conv-2-to-3-def)

lemma pr-conv-1-to-2-lm: f ∈ PrimRec1 ⇒ pr-conv-1-to-2 f ∈ PrimRec2 by (simp add: pr-conv-1-to-2-def, precc)
lemma pr-conv-1-to-3-lm: f ∈ PrimRec1 ⇒ pr-conv-1-to-3 f ∈ PrimRec3 by (simp add: pr-conv-1-to-3-def, precc)
lemma pr-conv-2-to-1-lm: f ∈ PrimRec2 ⇒ pr-conv-2-to-1 f ∈ PrimRec1 by (simp add: pr-conv-2-to-1-def, precc)
lemma pr-conv-3-to-1-lm: f ∈ PrimRec3 ⇒ pr-conv-3-to-1 f ∈ PrimRec1 by (simp add: pr-conv-3-to-1-def, precc)
lemma pr-conv-3-to-2-lm: f ∈ PrimRec3 ⇒ pr-conv-3-to-2 f ∈ PrimRec2
proof
  assume f ∈ PrimRec3
  then have pr-conv-3-to-1 f ∈ PrimRec1 by (rule pr-conv-3-to-1-lm)
  thus ?thesis by (simp add: pr-conv-3-to-2-def pr-conv-1-to-2-lm)
qed

lemma pr-conv-2-to-3-lm: f ∈ PrimRec2 ⇒ pr-conv-2-to-3 f ∈ PrimRec3
proof –

assume \( f \in \text{PrimRec}_2 \)
then have \( \text{pr-conv-2-to-1} \ f \in \text{PrimRec}_1 \) by (rule \( \text{pr-conv-2-to-1-lm} \))
thus \( \text{?thesis} \) by (simp add: \( \text{pr-conv-2-to-3-def} \) \( \text{pr-conv-1-to-3-lm} \))

qed

theorem \( \text{b-least2-scheme}: \) \\
\[
\begin{array}{l}
f \in \text{PrimRec}_2; \ g \in \text{PrimRec}_1; \ \forall \ x. \ h \ x < g \ x; \ \forall \ x. \ f \ x \ (h \ x) \neq 0; \ \forall \ z \ x. \ z < h \ x \ \rightarrow f \ x \ z = 0 \end{array}
\implies
h \in \text{PrimRec}_1
\]

proof –

assume \( f\text{-is-pr}: f \in \text{PrimRec}_2 \)
assume \( g\text{-is-pr}: g \in \text{PrimRec}_1 \)
assume \( h\text{-lt-g}: \forall \ x. \ h \ x < g \ x \)
assume \( f\text{-at-h-nz}: \forall \ x. \ f \ x \ (h \ x) \neq 0 \)
assume \( h\text{-is-min}: \forall \ z \ x. \ z < h \ x \ \rightarrow f \ x \ z = 0 \)

have \( \text{h-def}: h = (\lambda \ x. \ \text{b-least2} \ f \ x \ (g \ x)) \)

proof
fix \( x \)
show \( h \ x = \text{b-least2} \ f \ x \ (g \ x) \)

proof –

from \( \text{f-at-h-nz} \) have \( S1: \text{b-least2} \ f \ x \ (g \ x) \leq h \ x \) by (simp add: \( \text{nz-impl-b-least2-le} \))

from \( h\text{-lt-g} \) have \( h \ x < g \ x \) by auto

with \( S1 \) have \( \text{b-least2} \ f \ x \ (g \ x) \neq 0 \) by (rule \( \text{b-least2-less-impl-nz} \))

have \( S3: h \ x \leq \text{b-least2} \ f \ x \ (g \ x) \)

proof (rule \( \text{contr} \))

assume \( \neg h \ x \leq \text{b-least2} \ f \ x \ (g \ x) \) then have \( \text{b-least2} \ f \ x \ (g \ x) < h \ x \) by auto

with \( \text{h-is-min} \) have \( f \ x \ (\text{b-least2} \ f \ x \ (g \ x)) = 0 \) by simp

with \( S2 \) show False by auto

qed
from \( S1 \) \( S3 \) show \( \text{?thesis} \) by auto

qed

qed

def \( f1\text{-def}: f1 \equiv \text{b-least2} \ f \)

from \( f\text{-is-pr} \) \( f1\text{-def} \) have \( f1\text{-is-pr}: f1 \in \text{PrimRec}_2 \) by (simp add: \( \text{pr-b-least2} \))

with \( g\text{-is-pr} \) have \( (\lambda \ x. \ f1 \ x \ (g \ x)) \in \text{PrimRec}_1 \) by prec

with \( \text{h-def} \) \( f1\text{-def} \) show \( h \in \text{PrimRec}_1 \) by auto

qed

theorem \( \text{b-least2-scheme2}: \) \\
\[
\begin{array}{l}
f \in \text{PrimRec}_3; \ g \in \text{PrimRec}_2; \ \forall \ x \ y. \ h \ x \ y < g \ x \ y; \ \forall \ x \ y. \ f \ x \ y \ (h \ x \ y) \neq 0; \\ \forall \ z \ x \ y. \ z < h \ x \ y \ \rightarrow f \ x \ y \ z = 0 \end{array}
\implies
h \in \text{PrimRec}_2
\]

proof –

assume \( f\text{-is-pr}: f \in \text{PrimRec}_3 \)
assume \( g\text{-is-pr}: g \in \text{PrimRec}_2 \)
assume \( h\text{-lt-g}: \forall \ x \ y. \ h \ x \ y < g \ x \ y \)
assume \( f\text{-at-h-nz}: \forall \ x \ y. \ f \ x \ y \ (h \ x \ y) \neq 0 \)
assume \( h\text{-is-min}: \forall \ z \ x \ y. \ z < h \ x \ y \ \rightarrow f \ x \ y \ z = 0 \)
def ff-def: f1 ≡ pr-conv-3-to-2 f
def g1-def: g1 ≡ pr-conv-2-to-1 g
def h1-def: h1 ≡ pr-conv-2-to-1 h
from f-is-pr ff-def have f1-is-pr: f1 \in PrimRec2 by (simp add: pr-conv-3-to-2-lm)
from g-is-pr g1-def have g1-is-pr: g1 \in PrimRec1 by (simp add: pr-conv-2-to-1-lm)
from h-lt-g h1-def g1-def have h1-lt-g1: \forall x. h1 x < g1 x by (simp add: pr-conv-2-to-1-def)
from f-at-h-nz f1-def h1-def have f1-at-h1-nz by (simp add: pr-conv-3-to-2-lm)
from h-is-min f1-def h1-def have h1-is-min: \forall z x. z < h1 x \rightarrow f1 x z = 0 by (simp add: pr-conv-2-to-1-def pr-conv-3-to-2-def pr-conv-3-to-1-def pr-conv-1-to-2-def)
from f1-is-pr g1-is-pr h1-lt-g1 f1-at-h1-nz f1-is-pr: h1 \in PrimRec1 by (rule b-least2-scheme)
from h1-def have h = pr-conv-1-to-2 h1 by simp
with h1-is-pr show h \in PrimRec2 by (simp add: pr-conv-1-to-2-lm)
qed

theorem div-is-pr: (\lambda a b. a \ div b) \in PrimRec2
proof
  def f-def: f \equiv \lambda a b z. (sgn1 b) * (sgn1 (b*(z+1)-a)) + (sgn2 b)*(sgn2 z)
  have f-is-pr: f \in PrimRec3 unfolding f-def by prec
  def h-def: h \equiv \lambda (a::nat) (b::nat). a \ div b
  def g-def: g \equiv \lambda (a::nat) (b::nat). a + 1
  have g-is-pr: g \in PrimRec2 unfolding g-def by prec
  have h-lt-g: \forall a b. h a b < g a b
  proof (rule allI, rule allI)
    fix a b
    from h-def have h a b \leq a by simp
    also from g-def have a < g a b by simp
    ultimately show h a b < g a b by simp
  qed
  have f-at-h-nz: \forall a b. f a b (h a b) \neq 0
  proof (rule allI, rule allI)
    fix a b show f a b (h a b) \neq 0
  proof cases
    assume A: b = 0
    with h-def have h a b = 0 by simp
  with f-def A show ?thesis by simp
  next
  assume A: b \neq 0
  then have S1: b > 0 by auto
  from A f-def have S2: f a b (h a b) = sgn1 (b * (h a b + 1) - a) by simp
  then have ?thesis = (sgn1(b * (h a b + 1) - a) \neq 0) by auto
  also have \ldots = (b * (h a b + 1) - a > 0) by (rule sgn1-nz-eq-arg-pos)
  also have \ldots = (a < b * (h a b + 1)) by auto
  also have \ldots = (a < b * (h a b + b)) by auto
  also from h-def have \ldots = (a < b * (a \ div b) + b) by simp
  finally have S3: ?thesis = (a < b * (a \ div b) + b) by auto
  have S4: a < b * (a \ div b) + b

33
proof –
from S1 have S4-1: a mod b < b by (rule mod-less-divisor)
also have S4-2: b * (a div b) + a mod b = a by (rule mult-div-mod-eq)
from S4-1 have S4-3: b * (a div b) + a mod b < b * (a div b) + b by arith
from S4-2 S4-3 show thesis by auto
qed
from S3 S4 show thesis by auto
qed

have h-is-min: \( \forall z \ a \ b. \ z < h a b \rightarrow f a b z = 0 \)
proof (rule allI, rule allI, rule allI, rule impI)
fix a b z assume A: \( z < h a b \) show \( f a b z = 0 \)
proof –
from A h-def have S1: \( z < a \div b \) by simp
then have S2: \( a \div b > 0 \) by simp
have S3: \( b \neq 0 \)
proof (rule ccontr)
assume \( \neg b \neq 0 \) then have \( b = 0 \) by auto
with S2 show False by auto
qed
from S3 have b-pos: \( 0 < b \) by auto
from S1 have S4: \( z+1 \leq a \div b \) by auto
from b-pos have \( (b \cdot (z+1)) \leq b \cdot (a \div b) \) = \( (z+1 \leq a \div b) \) by (rule nat-mult-le-cancel1)
with S4 have S5: \( b \cdot (z+1) \leq b \cdot (a \div b) \) by simp
moreover have \( b \cdot (a \div b) \leq a \)
proof –
have \( b \cdot (a \div b) + (a \mod b) = a \) by (rule mult-div-mod-eq)
moreover have \( 0 \leq a \mod b \) by auto
ultimately show thesis by arith
qed
ultimately have S6: \( b \cdot (z+1) \leq a \) by auto
then have \( b \cdot (z+1) - a = 0 \) by auto
with S3 f-def show thesis by simp
qed
qed

from f-is-pr g-is-pr h-lt-g f-at-h-nz h-is-min have h-is-pr: \( h \in \text{PrimRec2} \) by (rule b-least2-scheme2)
with h-def show thesis by simp
qed

theorem mod-is-pr: \( \lambda a b. \ a \mod b \in \text{PrimRec2} \)
proof –
have \( \lambda (a::nat) (b::nat). \ a \mod b = (\lambda a b. \ a - (a \div b) \cdot b) \)
proof (rule ext, rule ext)
fix a b show \( (a::nat) \mod b = a - (a \div b) \cdot b \) by (rule minus-div-mult-eq-mod [symmetric])
qed
also from div-is-pr have \((\lambda\ a\ b.\ a - (a\ div\ b) * b) \in\ PrimRec2\) by prec
ultimately show \(?thesis\) by auto
qed

\begin{enumerate}
\item \textbf{pr-rec-last-scheme:} \(g \in PrimRec1; h \in PrimRec2; \forall x. f \ x \ 0 = g \ x;\)
\(\forall x\ y. f \ x \ (Suc \ y) = h \ x \ (f \ x \ y) \ y\) \implies f \in PrimRec2
\item \textbf{pr-rec-last-scheme:} \(g \in PrimRec1; h \in PrimRec3; \forall x. f \ x \ 0 = g \ x;\)
\(\forall x\ y. f \ x \ (Suc \ y) = h \ x \ (f \ x \ y) \ y\) \implies f \in PrimRec2
\end{enumerate}

\begin{enumerate}
\item \textbf{power-is-pr:} \((\lambda\ x::nat\ (n::nat).\ x ^ n) \in PrimRec2\)
\item \textbf{power-is-pr:} \((\lambda\ x::nat\ (n::nat).\ x ^ n) \in PrimRec2\)
\end{enumerate}

3 Primitive recursive coding of lists of natural numbers

\begin{enumerate}
\item \textbf{PRecList}
\item \textbf{PRecFun}
\item \textbf{begin}
\item We introduce a particular coding \(list-to-nat\) from lists of natural numbers
\item to natural numbers.
\item \textbf{definition}
\item \textbf{c-len :: nat \Rightarrow nat where}
\end{enumerate}
\[\text{c-len} = (\lambda (u::\text{nat}). (\text{sgn1 } u) \ast (\text{c-fst}(u-(1::\text{nat}))+1))\]

**Lemma c-len-1:** \(\text{c-len } u = (\text{case } u \text{ of } 0 \Rightarrow 0 \mid \text{Suc } v \Rightarrow \text{c-fst}(v)+1)\) by (unfold c-len-def, cases u, auto)

**Lemma c-len-is-pr:** \(\text{c-len} \in \text{PrimRec1}\) unfolding c-len-def by prec

**Lemma simp:** \(\text{c-len } 0 = 0\) by (simp add: c-len-def)

**Lemma c-len-2:** \(u \neq 0 \Rightarrow \text{c-len } u = \text{c-fst}(u-(1::\text{nat}))+1\) by (simp add: c-len-def)

**Lemma c-len-3:** \(u > 0 \Rightarrow \text{c-len } u > 0\) by (simp add: c-len-2)

**Lemma c-len-4:** \(\text{c-len } u = 0 \Rightarrow u = 0\)

**Proof:**

- Assume \(A1: u = 0\)
- Thus ?thesis by simp

**Next:**

- Assume \(A1: \text{c-len } u = 0\) and \(A2: u \neq 0\)
- From \(A2\) have \(\text{c-len } u > 0\) by (simp add: c-len-3)
- From \(A1\) this show \(u=0\) by simp

**QED**

**Lemma c-len-5:** \(\text{c-len } u > 0 \Rightarrow u > 0\)

**Proof:**

- Assume \(A1: \text{c-len } u > 0\) and \(A2: u=0\)
- From \(A2\) have \(\text{c-len } u = 0\) by simp
- From \(A1\) this show ?thesis by simp

**Next:**

- Assume \(A1: u \neq 0\)
- From \(A1\) show \(u>0\) by simp

**QED**

**Fun c-fold :: nat list \Rightarrow nat where**

- \(c-fold [] = 0\)
- \(c-fold [x] = x\)
- \(c-fold (x#ls) = \text{c-pair } x \ (c-fold \ ls)\)

**Lemma c-fold-0:** \(ls \neq [] \Rightarrow c-fold (x#ls) = \text{c-pair } x \ (c-fold \ ls)\)

**Proof:**

- Assume \(A1: ls \neq []\)
  - Then have \(S1: ls = (\text{hd } ls)#(tl \ ls)\) by simp
  - Then have \(S2: x#ls = x#(\text{hd } ls)#(tl \ ls)\) by simp
  - Have \(S3: c-fold (x#(\text{hd } ls)#(tl \ ls)) = \text{c-pair } x \ (c-fold \ ((\text{hd } ls)#(tl \ ls)))\) by simp
  - From \(S1 \ S2 \ S3\) show ?thesis by simp

**QED**

**Primrec**

- \(c-unfold :: \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat list}\)
where
c-unfold 0 u = []
| c-unfold (Suc k) u = (if k = 0 then [u] else ((c-fst u) # (c-unfold k (c-snd u)))))

lemma c-fold-1: c-unfold 1 (c-fold [x]) = [x] by simp

lemma c-fold-2: c-fold (c-unfold 1 u) = u by simp

lemma c-unfold-1: c-unfold 1 u = [u] by simp

lemma c-unfold-2: c-unfold (Suc 1) u = (c-fst u) # (c-unfold 1 (c-snd u)) by simp

lemma c-unfold-3: c-unfold (Suc 1) u = [c-fst u, c-snd u] by simp

lemma c-unfold-4: k > 0 =⇒ c-unfold (Suc k) u ≠ [] by simp

lemma c-unfold-4-1: k > 0 =⇒ c-unfold (Suc k) u ≠ [] by (simp add: c-unfold-4)

lemma two: (2::nat) = Suc 1 by simp

lemma c-unfold-5: c-unfold 2 u = [c-fst u, c-snd u] by (simp add: two)

lemma c-unfold-6: k > 0 =⇒ c-unfold k u ≠ []

proof
assume A1: k > 0
let ?k1 = k - (1::nat)
from A1 have S1: k = Suc ?k1 by simp
have S2: ?k1 = 0 =⇒ ¬thesis
proof
assume A2-1: ?k1 = 0
from A1 A2-1 have S2-1: k = 1 by simp
from S2-1 show ¬thesis by (simp add: c-unfold-1)
qed
have S3: ?k1 > 0 =⇒ ¬thesis
proof
assume A3-1: ?k1 > 0
from A1 A3-1 have S3-1: c-unfold (Suc ?k1) u ≠ [] by (rule c-unfold-4-1)
from S1 S3-1 show ¬thesis by simp
qed
from S2 S3 show ¬thesis by arith
qed

lemma th-lm-1: k = 1 =⇒ (∀ u. c-fold (c-unfold k u) = u) by (simp add: c-fold-2)

lemma th-lm-2: [k > 0; (∀ u. c-fold (c-unfold k u) = u)] =⇒ (∀ u. c-fold (c-unfold (Suc k) u) = u)

proof
assume A1: \( k > 0 \)
assume A2: \( \forall u. \) c-fold \((\text{c-unfold } k \ u) = u \)
fix \( u \)
from A1 have S1: \( \text{c-unfold} \ (\text{Suc } k) \ u = (\text{c-fst } u) \# (\text{c-unfold } k \ (\text{c-snd } u)) \) by (rule c-unfold-4)
let \( ?ls = \text{c-unfold } k \ (\text{c-snd } u) \)
from A1 have S2: \( ?ls \neq [] \) by (rule c-unfold-6)
from S2 have S3: \( \text{c-fold} \ ( (\text{c-fst } u) \# ?ls) = \text{c-pair} \ (\text{c-fst } u) \ (\text{c-fold} \ ?ls) \) by (rule c-fold-0)
from A2 have S4: \( \text{c-fold} \ ?ls = \text{c-snd } u \) by simp
from S3 S4 have S5: \( \text{c-fold} \ ( (\text{c-fst } u) \# ?ls) = \text{c-pair} \ (\text{c-fst } u) \ (\text{c-snd } u) \) by simp
from S5 have S6: \( \text{c-fold} \ ( (\text{c-fst } u) \# ?ls) = u \) by simp
from S1 S6 have S7: \( \text{c-fold} \ (\text{c-unfold} \ (\text{Suc } k) \ u) = u \) by simp
thus \( \text{thesis} \) by simp
qed

lemma th-lm-3: \( (\forall u. \) c-fold \((\text{c-unfold} \ (\text{Suc } k) \ u) = u) \Rightarrow (\forall u. \) c-fold \((\text{c-unfold} \ (\text{Suc } (\text{Suc } k)) \ u) = u) \)
proof –
  assume A1: \( \forall u. \) c-fold \((\text{c-unfold} \ (\text{Suc } k) \ u) = u \)
  let \( ?k1 = \text{Suc } k \)
  have S1: \( ?k1 > 0 \) by simp
  from S1 A1 have S2: \( \forall u. \) c-fold \((\text{c-unfold} \ (\text{Suc } ?k1) \ u) = u \) by (rule th-lm-2)
  thus \( \text{thesis} \) by simp
qed

theorem th-1: \( \forall u. \) c-fold \((\text{c-unfold} \ (\text{Suc } k) \ u) = u \)
apply(induct k)
apply(simp add: c-fold-2)
apply(rule th-lm-3)
apply(assumption)
done

theorem th-2: \( k > 0 \Rightarrow (\forall u. \) c-fold \((\text{c-unfold} \ k \ u) = u \)
proof –
  assume A1: \( k > 0 \)
  let \( ?k1 = k - (1::nat) \)
  from A1 have S1: \( \text{Suc } ?k1 = k \) by simp
  have S2: \( \forall u. \) c-fold \((\text{c-unfold} \ (\text{Suc } ?k1) \ u) = u \) by (rule th-1)
  from S1 S2 show \( \text{thesis} \) by simp
qed

lemma c-fold-3: \( \text{c-unfold } 2 \ (\text{c-fold} \ [x, y]) = [x, y] \) by (simp add: two)

theorem c-unfold-len: \( \forall u. \) length \((\text{c-unfold} \ k \ u) = k \)
apply(induct k)
apply(simp)
apply(subgoal-tac n=(0::nat) \& n>0)

38
apply (drule disjE)
prefer 3
apply (simp-all)
apply (auto)
done

lemma th-3-lm-0: [c-unfold (length ls) (c-fold ls) = ls; ls = a # ls1; ls1 = aa # list] \implies c-unfold (length (x # ls)) (c-fold (x # ls)) = x # ls

proof –
  assume A1: c-unfold (length ls) (c-fold ls) = ls
  assume A2: ls = a # ls1
  assume A3: ls1 = aa # list
  from A2 have S1: ls ≠ [] by simp
  from S1 have S2: c-fold (x#ls) = c-pair x (c-fold ls) by (rule c-fold-0)
  have S3: length (x#ls) = Suc (length ls) by simp
  from S3 have S4: c-unfold (length (x # ls)) (c-fold (x # ls)) = c-unfold (Suc (length ls)) (c-fold (x # ls)) by simp
  from A2 have S5: length ls > 0 by simp
  from S5 have S6: c-unfold (Suc (length ls)) (c-fold (x # ls)) = c-fst (c-fold (x # ls)) (c-unfold (length ls)) (c-fold (x # ls)) by simp
  from S2 have S7: c-fst (c-fold (x#ls)) = x by simp
  from S2 have S8: c-snd (c-fold (x#ls)) = c-fold ls by simp
  from S6 S7 S8 have S9: c-unfold (Suc (length ls)) (c-fold (x # ls)) = x # (c-unfold (length ls)) (c-fold ls) by simp
  from A1 have S10: x # (c-unfold (length ls)) (c-fold ls) = x # ls by simp
  from S9 S10 have S11: c-unfold (Suc (length ls)) (c-fold (x # ls)) = (x # ls) by simp
  thus ?thesis by simp
qed

lemma th-3-lm-1: [c-unfold (length ls) (c-fold ls) = ls; ls = a # ls1] \implies c-unfold (length (x # ls)) (c-fold (x # ls)) = x # ls
apply (cases ls1)
apply (simp add: c-fold-1)
apply (simp)
done

lemma th-3-lm-2: c-unfold (length ls) (c-fold ls) = ls \implies c-unfold (length (x # ls)) (c-fold (x # ls)) = x # ls
apply (cases ls)
apply (simp add: c-fold-1)
apply (rule th-3-lm-1)
apply (assumption+)
done

theorem th-3: c-unfold (length ls) (c-fold ls) = ls
apply (induct ls)
apply (simp)
apply (rule th-3-lm-2)
apply (assumption)
done

definition
list-to-nat :: nat list ⇒ nat where
list-to-nat = (λ ls. if ls=[] then 0 else (c-pair ((length ls) - 1) (c-fold ls))+1)

definition
nat-to-list :: nat ⇒ nat list where
nat-to-list = (λ u. if u=0 then [] else (c-unfold (c-len u) (c-snd (u-(1::nat))))))

lemma nat-to-list-of-pos: u>0 ⇒ nat-to-list u = c-unfold (c-len u) (c-snd (u-(1::nat)))
by (simp add: nat-to-list-def)

theorem list-to-nat-th [simp]: list-to-nat (nat-to-list u) = u
proof –
  have S1: u=0 ⇒ ?thesis by (simp add: list-to-nat-def nat-to-list-def)
  have S2: u>0 ⇒ ?thesis
    proof –
      assume A1: u>0
      def D1: ls == nat-to-list u
      from D1 A1 have S2-1: ls = c-unfold (c-len u) (c-snd (u-(1::nat))) by (simp add: nat-to-list-def)
      let ?k = c-len u
      from A1 have S2-2: ?k > 0 by (rule c-len-3)
      from S2-1 have S2-3: length ls = ?k by (simp add: c-unfold-len)
      from S2-2 S2-3 have S2-4: length ls > 0 by simp
      from S2-4 have S2-5: ls ≠ [] by simp
      from S2-5 have S2-6: list-to-nat ls = c-pair ((length ls)-(1::nat)) (c-fold ls)+1 by (simp add: list-to-nat-def)
      have S2-7: c-fold ls = c-snd(u-(1::nat))
      proof –
        from S2-1 have S2-7-1: c-fold ls = c-fold (c-unfold (c-len u) (c-snd (u-(1::nat))))
        by simp
        from S2-2 S2-7-1 show ?thesis by (simp add: th-2)
      qed
      have S2-8: (length ls)-(1::nat) = c-fst (u-(1::nat))
      proof –
        from S2-3 have S2-8-1: length ls = c-len u by simp
        from A1 S2-8-1 have S2-8-2: length ls = c-fst(u-(1::nat)) + 1 by (simp add: c-len-2)
      qed
      from S2-8-2 show ?thesis by simp
      from S2-7 S2-8 have S2-9: c-pair ((length ls)-(1::nat)) (c-fold ls) = c-pair (c-fst (u-(1::nat))) (c-snd (u-(1::nat))) by simp
      from S2-9 have S2-10: c-pair ((length ls)-(1::nat)) (c-fold ls) = u - (1::nat)
      by simp
      from S2-6 S2-10 have S2-11: list-to-nat ls = (u - (1::nat))+1 by simp
      from A1 have S2-12: (u - (1::nat))+1 = u by simp
  qed

40
from D1 S2-11 S2-12 show ?thesis by simp
qed
from S1 S2 show ?thesis by arith
qed

theorem nat-to-list th [simp]: nat-to-list (list-to-nat ls) = ls
proof –
  have S1: ls=[] ⇒ ?thesis by (simp add: list-to-nat-def list-to-nat-def)
  have S2: ls ≠ [] ⇒ ?thesis
proof –
  assume A1: ls ≠ []
  def D1: u == list-to-nat ls
  from D1 A1 have S2-1: u = (c-pair ((length ls)−(1::nat)) (c-fold ls)+1 by (simp add: list-to-nat-def)
    let ?k = length ls
    from A1 have S2-2: ?k > 0 by simp
    from S2-1 have S2-3: u>0 by simp
    from S2-3 have S2-4: nat-to-list u = c-unfold (c-len u) (c-snd (u−(1::nat)))
  by (simp add: nat-to-list-def)
  have S2-5: c-len u = length ls
proof –
  from S2-1 have S2-5-1: u−(1::nat) = (c-pair ((length ls)−(1::nat)) (c-fold ls)) by simp
  from S2-5-1 have S2-5-2: c-fst (u−(1::nat)) = (length ls)−(1::nat) by simp
  from S2-2 S2-5-2 have c-fst (u−(1::nat))+1 = length ls by simp
  from S2-3 this show ?thesis by (simp add: c-len-2)
  have S2-6: c-snd (u−(1::nat)) = c-fold ls
proof –
  from S2-1 have S2-6-1: u−(1::nat) = c-pair ((length ls)−(1::nat)) (c-fold ls) by simp
  from S2-6-1 show ?thesis by simp
  have S2-4 S2-5 S2-6 have S2-7: nat-to-list u = c-unfold (length ls) (c-fold ls)
  by simp
  from S2-7 have nat-to-list u = ls by (simp add: th-3)
  from D1 this show ?thesis by simp
  have S3: ls = [] ∨ ls ≠ [] by simp
  from S1 S2 S3 show ?thesis by auto
  qed

lemma [simp]: list-to-nat [] = 0 by (simp add: list-to-nat-def)

lemma [simp]: nat-to-list 0 = [] by (simp add: nat-to-list-def)

theorem c-len-th-1: c-len (list-to-nat ls) = length ls
proof (cases)
  assume ls=[]
from this show \( \text{thesis} \) by simp

next

assume \( S1 : \mathrm{ls} \neq [] \)
then have \( S2 : \mathrm{list-to-nat} \ \mathrm{ls} = \mathrm{c-pair} \ ((\mathrm{length} \ \mathrm{ls})-(1::\mathrm{nat})) \ (\mathrm{c-fold} \ \mathrm{ls})+1 \) by (simp add: list-to-nat-def)
let \( ?u = \mathrm{list-to-nat} \ \mathrm{ls} \)
from \( S2 \) have \( u \text{-not-zero} : ?u > 0 \) by simp
from \( S2 \) have \( S3 : \ ?u-(1::\mathrm{nat}) = \mathrm{c-pair} \ ((\mathrm{length} \ \mathrm{ls})-(1::\mathrm{nat})) \ (\mathrm{c-fold} \ \mathrm{ls}) \) by simp
then have \( S4 : \mathrm{c-fst} \ (?u-(1::\mathrm{nat})) = \mathrm{c-fst} \ ((\mathrm{length} \ \mathrm{ls})-(1::\mathrm{nat})+1=\mathrm{length} \ \mathrm{ls}) \) by simp
from \( u \text{-not-zero} \ S4 \) have \( S5 : \mathrm{c-fst} \ (?u-(1::\mathrm{nat})) = (\mathrm{length} \ \mathrm{ls})-(1::\mathrm{nat}) \) by simp
from \( S1 \ \mathrm{this} \) have \( S6 : \mathrm{c-fst} \ (?u-(1::\mathrm{nat}))+1=\mathrm{length} \ \mathrm{ls} \) by (simp add: c-len-5)
from \( S1 \ S6 \) show \( \text{thesis} \) by simp

qed

theorem \( \mathrm{length} \ (\mathrm{nat-to-list} \ \mathrm{u}) = \mathrm{c-len} \ \mathrm{u} \)
proof
let \( ?\mathrm{ls} = \mathrm{nat-to-list} \ \mathrm{u} \)
have \( S1 : \ ?u = \mathrm{list-to-nat} \ ?\mathrm{ls} \) by (rule list-to-nat-th [THEN sym])
from \( \mathrm{c-len-th-1} \) have \( S2 : \mathrm{length} \ ?\mathrm{ls} = \mathrm{c-len} \ (\mathrm{list-to-nat} \ ?\mathrm{ls}) \) by (rule sym)
from \( S1 \ \mathrm{S2} \) show \( \text{thesis} \) by (rule ssubst)
qed

definition \( \mathrm{c-hd} :: \mathrm{nat} \Rightarrow \mathrm{nat} \) where
\( \mathrm{c-hd} = (\lambda \ u. \ \mathrm{if} \ u=0 \ \mathrm{then} \ 0 \ \mathrm{else} \ \mathrm{hd} \ (\mathrm{nat-to-list} \ \mathrm{u})) \)

definition \( \mathrm{c-tl} :: \mathrm{nat} \Rightarrow \mathrm{nat} \) where
\( \mathrm{c-tl} = (\lambda \ u. \ \mathrm{list-to-nat} \ (\mathrm{tl} \ (\mathrm{nat-to-list} \ \mathrm{u}))) \)

definition \( \mathrm{c-cons} :: \mathrm{nat} \Rightarrow \mathrm{nat} \Rightarrow \mathrm{nat} \) where
\( \mathrm{c-cons} = (\lambda \ x \ u. \ \mathrm{list-to-nat} \ (x \ # \ (\mathrm{nat-to-list} \ \mathrm{u}))) \)

lemma \[\mathrm{simp}] : \( \mathrm{c-hd} \ 0 = 0 \) by (simp add: c-hd-def)

lemma \( \mathrm{c-hd-aux0} : \mathrm{c-len} \ \mathrm{u} = 1 \Longrightarrow \mathrm{nat-to-list} \ \mathrm{u} = [\mathrm{c-snd} \ (\mathrm{u}-(1::\mathrm{nat}))] \) by (simp add: nat-to-list-def c-len-5)

lemma \( \mathrm{c-hd-aux1} : \mathrm{c-len} \ \mathrm{u} = 1 \Longrightarrow \mathrm{c-hd} \ \mathrm{u} = \mathrm{c-snd} \ (\mathrm{u}-(1::\mathrm{nat})) \)
proof
assume \( A1 : \mathrm{c-len} \ \mathrm{u} = 1 \)
then have \( S1 : \mathrm{nat-to-list} \ \mathrm{u} = [\mathrm{c-snd} \ (\mathrm{u}-(1::\mathrm{nat}))] \) by (simp add: nat-to-list-def c-len-5)
from \( A1 \) have \( u > 0 \) by (simp add: c-len-5)
with \( S1 \) show \( \text{thesis} \) by (simp add: c-hd-def)

qed
lemma c-hd-aux2: \( \text{c-len } u > 1 \implies \text{c-hd } u = \text{c-fst } (\text{c-snd } (u-1)) \)

proof –

assume \( A1: \text{c-len } u > 1 \)

let \( ?k = (\text{c-len } u) - 1 \)

from \( A1 \) have \( S1: \text{c-len } u = \text{Suc } ?k \) by simp

from \( A1 \) have \( S2: \text{c-len } u > 0 \) by simp

from \( S2 \) have \( S3: u > 0 \) by (rule c-len-5)

from \( S3 \) have \( S4: \text{c-hd } u = \text{hd } (\text{nat-to-list } u) \) by (simp add: c-hd-def)

from \( S3 \) have \( S5: \text{nat-to-list } u = \text{c-unfold } (\text{c-len } u) (\text{c-snd } (u-1)) \) by (rule nat-to-list-of-pos)

from \( S1 \) \( S5 \) have \( S6: \text{nat-to-list } u = \text{c-unfold } (\text{Suc } ?k) (\text{c-snd } (u-1)) \) by simp

from \( A1 \) have \( S7: ?k > 0 \) by simp

from \( S7 \) have \( S8: \text{c-unfold } (\text{Suc } ?k) (\text{c-snd } (u-1)) = (\text{c-fst } (\text{c-snd } (u-1))) \) by (rule c-unfold-4)

from \( S6 \) \( S8 \) have \( S9: \text{nat-to-list } u = (\text{c-fst } (\text{c-snd } (u-1))) \) by simp

from \( S4 \) \( S10 \) show \( ?thesis \) by simp

qed

lemma c-hd-aux3: \( u > 0 \implies \text{c-hd } u = (\text{if } (\text{c-len } u) = 1 \text{ then } \text{c-snd } (u-1) \text{ else } \text{c-fst } (\text{c-snd } (u-1))) \)

proof –

assume \( A1: u > 0 \)

from \( A1 \) have \( \text{c-len } u > 0 \) by (rule c-len-3)

then have \( S1: \text{c-len } u = 1 \lor \text{c-len } u > 1 \) by arith

let \( ?tmp = (\text{if } (\text{c-len } u) = 1 \text{ then } \text{c-snd } (u-1) \text{ else } \text{c-fst } (\text{c-snd } (u-1))) \)

have \( S2: \text{c-len } u = 1 \implies ?thesis \)

proof –

assume \( A2-1: \text{c-len } u = 1 \)

then have \( S2-1: \text{c-hd } u = \text{c-snd } (u-1) \) by (rule c-hd-aux1)

from \( A2-1 \) have \( S2-2: ?tmp = \text{c-snd } (u-1) \) by simp

from \( S2-1 \) this show \( ?thesis \) by simp

qed

have \( S3: \text{c-len } u > 1 \implies ?thesis \)

proof –

assume \( A3-1: \text{c-len } u > 1 \)

from \( A3-1 \) have \( S3-1: \text{c-hd } u = \text{c-fst } (\text{c-snd } (u-1)) \) by (rule c-hd-aux2)

from \( A3-1 \) have \( S3-2: ?tmp = \text{c-fst } (\text{c-snd } (u-1)) \) by simp

from \( S3-1 \) this show \( ?thesis \) by simp

qed

from \( S1 \) \( S2 \) \( S3 \) show \( ?thesis \) by auto

qed

lemma c-hd-aux4: \( \text{c-hd } u = (\text{if } u=0 \text{ then } 0 \text{ else } (\text{if } (\text{c-len } u) = 1 \text{ then } \text{c-snd } (u-1) \text{ else } \text{c-fst } (\text{c-snd } (u-1)))) \)

proof cases
assume \( u = 0 \) then show \( \)thesis by simp

next

assume \( u \neq 0 \) then have \( A1: u > 0 \) by simp

then show \( \)thesis by (simp add: c-hd-aux3)

qed

lemma c-hd-is-pr: \( c\cdot hd \in \text{PrimRec1} \)

proof –

have \( c\cdot hd = (\%u. \text{if } u = 0 \text{ then } 0 \text{ else } (c\cdot len u = 1 \text{ then } c\cdot snd (u - (1::nat))) \) else \( c\cdot fst (c\cdot snd (u - (1::nat)))) \) \( \)is \( = \) ?R \) by (simp add: c-hd-aux4 ext)

moreover have \( ?R \in \text{PrimRec1} \)

proof (rule if-is-pr)

show \( (\lambda x. x) \in \text{PrimRec1} \) by (rule pr-id1-1)

next show \( (\lambda x. 0) \in \text{PrimRec1} \) by (rule pr-zero)

next show \( (\lambda x. \text{if } c\cdot len x = 1 \text{ then } c\cdot snd (x - 1) \text{ else } c\cdot fst (c\cdot snd (x - 1))) \) \( \in \text{PrimRec1} \)

proof (rule if-eq-is-pr)

show \( c\cdot len \in \text{PrimRec1} \) by (rule c-len-is-pr)

next show \( (\lambda x. 1) \in \text{PrimRec1} \) by (rule const-is-pr)

next show \( (\lambda x. c\cdot fst (c\cdot snd (x - 1))) \in \text{PrimRec1} \) by prec

qed

qed

ultimately show \( \)thesis by simp

qed

lemma \( \)simp\): \( c\cdot tl \ 0 = 0 \) by (simp add: c-tl-def)

lemma c-tl-eq-tl: \( c\cdot tl \ (\text{list-to-nat } ls) = \text{list-to-nat } (\text{tl } ls) \) by (simp add: c-tl-def)

lemma tl-eq-c-tl: \( \text{tl } (\text{nat-to-list } x) = \text{nat-to-list } (c\cdot tl x) \) by (simp add: c-tl-def)

lemma c-tl-aux1: \( c\cdot len u = 1 \implies c\cdot tl u = 0 \) by (unfold c-tl-def, simp add: c-hd-aux0)

lemma c-tl-aux2: \( c\cdot len u > 1 \implies c\cdot tl u = (c\cdot pair \ (c\cdot len u - (2::nat))) (c\cdot snd (c\cdot snd (u - (1::nat)))) + 1 \)

proof –

assume \( A1: c\cdot len u > 1 \)

let \( ?k = (c\cdot len u) - 1 \)

from \( A1 \) have \( S1: c\cdot len u = \text{Suc } ?k \) by simp

from \( A1 \) have \( S2: c\cdot len u > 0 \) by simp

from \( S2 \) have \( S3: u > 0 \) by (rule c-len-5)

from \( S3 \) have \( S4: \text{nat-to-list } u = c\cdot unfold (c\cdot len u) (c\cdot snd (u - (1::nat))) \) by (rule nat-to-list-of-pos)

from \( A1 \) have \( S5: ?k > 0 \) by simp

from \( S5 \) have \( S6: c\cdot unfold (\text{Suc } ?k) (c\cdot snd (u - (1::nat))) = (c\cdot fst (c\cdot snd (u - (1::nat)))) \# (c\cdot unfold ?k (c\cdot snd (c\cdot snd (u - (1::nat)))) \) by (rule c-unfold-4)

from \( S6 \) have \( S7: \text{tl } (c\cdot unfold (\text{Suc } ?k) (c\cdot snd (u - (1::nat)))) = c\cdot unfold ?k \text{ by simp} \)
\[(\text{c-snd } (\text{c-snd } (u-(1::\text{nat}))))\] by simp

from S2 S4 S7 have S8: \(tl (\text{nat-to-list } u) = \text{c-unfold } ?k (\text{c-snd } (\text{c-snd } (u-(1::\text{nat}))))\) by simp

def D1: \(ls == tl (\text{nat-to-list } u)\)

from D1 S8 have S9: \(\text{length } ls = ?k\) by (simp add: c-unfold-len)

from D1 have S10: \(\text{c-tl } u = \text{list-to-nat } ls\) by (simp add: c-tl-def)

from S5 S9 have S11: \(\text{length } ls > 0\) by simp

from S11 have S12: \(ls \neq []\) by simp

from S12 have S13: \(\text{list-to-nat } ls = (\text{c-pair } ((\text{length } ls) - 1) (\text{c-fold } ls)) + 1\) by (simp add: list-to-nat-def)

from S10 S13 have S14: \(\text{c-tl } u = (\text{c-pair } ((\text{length } ls) - 1) (\text{c-fold } ls)) + 1\) by simp

from S9 have S15: \((\text{length } ls) -(1::\text{nat}) = ?k-(1::\text{nat})\) by simp

from A1 have S16: \(?k-(1::\text{nat}) = \text{c-len } u -(2::\text{nat})\) by arith

from S15 S16 have S17: \((\text{length } ls) -(1::\text{nat}) = \text{c-len } u -(2::\text{nat})\) by simp

from D1 S8 have S18: \(ls = \text{c-unfold } ?k (\text{c-snd } (\text{c-snd } (u-(1::\text{nat}))))\) by simp

from S5 have S19: \(\text{c-fold } (\text{c-unfold } ?k (\text{c-snd } (\text{c-snd } (u-(1::\text{nat})))))) = \text{c-snd } (\text{c-snd } (u-(1::\text{nat}))))\) by (simp add: th-2)

from S18 S19 have S20: \(\text{c-fold } ls = \text{c-snd } (\text{c-snd } (u-(1::\text{nat}))))\) by simp

from S14 S17 S20 show \(\text{thesis}\) by simp

qed

lemma c-tl-aux3: \(\text{c-tl } u = (\text{sgn1 } ((\text{c-len } u) - 1))*((\text{c-pair } (\text{c-len } u -(2::\text{nat}))) (\text{c-snd } (\text{c-snd } (u-(1::\text{nat})))) + 1) (\text{ls } = ?R)\)

proof –

  have S1: \(u=0\) \(\Longrightarrow\) \(\text{thesis}\) by simp

  have S2: \(u>0\) \(\Longrightarrow\) \(\text{thesis}\)

  proof –

    assume A1: \(u>0\)

    have S2-1: \(\text{c-len } u = 1\) \(\Longrightarrow\) \(\text{thesis}\) by (simp add: c-tl-aux1)

    have S2-2: \(\text{c-len } u \neq 1\) \(\Longrightarrow\) \(\text{thesis}\)

    proof –

      assume A2-2-1: \(\text{c-len } u \neq 1\)

      from A1 have S2-2-1: \(\text{c-len } u > 0\) by (rule c-len-3)

      from A2-2-1 S2-2-1 have S2-2-2: \(\text{c-len } u > 1\) by arith

      from this have S2-2-3: \(\text{c-len } u - 1 > 0\) by simp

      from this have S2-2-4: \(\text{sgn1 } (\text{c-len } u - 1) = 1\) by simp

      from S2-2-4 have S2-2-5: \(?R = (\text{c-pair } (\text{c-len } u -(2::\text{nat}))) (\text{c-snd } (\text{c-snd } (u-(1::\text{nat})))) + 1\) by simp

      from S2-2-2 have S2-2-6: \(\text{c-tl } u = (\text{c-pair } (\text{c-len } u -(2::\text{nat}))) (\text{c-snd } (\text{c-snd } (u-(1::\text{nat})))) + 1\) by (rule c-tl-aux2)

      from S2-2-5 S2-2-6 show \(\text{thesis}\) by simp

      qed

      from S2-1 S2-2 show \(\text{thesis}\) by blast

    qed

    from S1 S2 show \(\text{thesis}\) by arith

  qed

lemma c-tl-less: \(u > 0\) \(\Longrightarrow\) \(\text{c-tl } u < u\)
proof –  
assume \( A1: u > 0 \)  
then show \( S1: c\text{-}len\ u > 0 \) by (rule c\text{-}len-3)  
then show \( \neg \)thesis  
proof cases  
assume \( c\text{-}len\ u = 1 \)  
from this \( A1 \) show \( \neg \)thesis by (simp add: c\text{-}tl-aux1)  
next  
assume \( \neg \)c\text{-}len\ u = 1 \) with \( S1 \) have \( A2: c\text{-}len\ u > 1 \) by simp  
then have \( S2: c\text{-}tl\ u = (c\text{-}pair\ (c\text{-}len\ u - (2::nat))\ (c\text{-}snd\ (c\text{-}snd\ (u-(1::nat))))) + 1 \) by (rule c\text{-}tl-aux2)  
from \( A1 \) have \( S3: c\text{-}len\ u = c\text{-}fst}(u-(1::nat)) + 1 \) by (simp add: c\text{-}len-def)  
from \( A2\ S3 \) have \( S4: c\text{-}len\ u - (2::nat) < c\text{-}fst}(u-(1::nat)) \) by simp  
then have \( S5: (c\text{-}pair\ (c\text{-}len\ u - (2::nat))\ (c\text{-}snd\ (c\text{-}snd\ (u-(1::nat))))) < (c\text{-}pair\ (c\text{-}fst}(u-(1::nat)))\ (c\text{-}snd\ (c\text{-}snd\ (u-(1::nat)))) \) by (rule c\text{-}pair-strict-mono1)  
have \( S6: c\text{-}snd\ (c\text{-}snd\ (u-(1::nat))) < c\text{-}snd\ (u-(1::nat)) \) by (rule c\text{-}snd-le-arg)  
then have \( S7: (c\text{-}pair\ (c\text{-}fst}(u-(1::nat)))\ (c\text{-}snd\ (c\text{-}snd\ (u-(1::nat)))) < (c\text{-}pair\ (c\text{-}fst}(u-(1::nat)))\ (c\text{-}snd\ (c\text{-}snd\ (u-(1::nat)))) \) by (rule c\text{-}pair-mono2)  
then have \( S8: (c\text{-}pair\ (c\text{-}fst}(u-(1::nat)))\ (c\text{-}snd\ (c\text{-}snd\ (u-(1::nat)))) \leq\ c\text{-}snd\ (u-(1::nat)) \) by simp  
with \( S5 \) have \( (c\text{-}pair\ (c\text{-}len\ u - (2::nat))\ (c\text{-}snd\ (c\text{-}snd\ (u-(1::nat))))) < u - (1::nat) \) by simp  
with \( S2 \) have \( c\text{-}tl\ u < (u-(1::nat)) + 1 \) by simp  
with \( A1 \) show \( \neg \)thesis by simp  
qed

lemma c\text{-}tl-le: c\text{-}tl\ u \leq\ u  
proof (cases u)  
assume \( u=0 \)  
then show \( \neg \)thesis by simp  
next  
fix \( v \) assume \( A1: u = \text{Suc}\ v \)  
then have \( S1: u > 0 \) by simp  
then have \( S2: c\text{-}tl\ u < u \) by (rule c\text{-}tl-less)  
with \( A1 \) show \( c\text{-}tl\ u \leq\ u \) by simp  
qed

theorem c\text{-}tl-is-pr: c\text{-}tl\ \in\ \text{PrimRec1}  
proof –  
have \( c\text{-}tl = (\lambda\ u. (\text{snr}\ 1\ ((c\text{-}len\ u) - 1))\ ((c\text{-}pair\ (c\text{-}len\ u - (2::nat))\ (c\text{-}snd\ (c\text{-}snd\ (u-(1::nat)))) + 1))\) \( \text{is -} \ ?R \) by (simp add: c\text{-}tl-aux3 ext)  
moreover from c\text{-}len-is-pr c\text{-}pair-is-pr have \( ?R \in\ \text{PrimRec1} \) by prec  
ultimately show \( \neg \)thesis by simp  
qed

lemma c\text{-}cons-aux1: c\text{-}cons\ x\ 0 = (c\text{-}pair\ 0\ x) + 1  
apply(unfold c\text{-}cons-def)  
apply(simp)
apply\(\text{unfold list-to-nat-def}\)
apply\(\text{simp}\)
done

\textbf{lemma} \(\text{c-cons-aux2}: u > 0 \implies \text{c-cons } x \ u = (\text{c-pair } (\text{c-len } u) \ (\text{c-snd } (u - (1::\text{nat}))) + 1\)

\textbf{proof} –

\textbf{assume} \(A1: u > 0\)

\textbf{from} \(A1\) \textbf{have} \(S1: \text{c-len } u > 0\) \textbf{by} (rule \text{c-len-3})

\textbf{from} \(A1\) \textbf{have} \(S2: \text{nat-to-list } u = \text{c-unfold } (\text{c-len } u) \ (\text{c-snd } (u - (1::\text{nat})))\) \textbf{by} (rule \text{nat-to-list-of-pos})

def \(D1\): \(ls == \text{nat-to-list } u\)

\textbf{from} \(D1\) \textbf{have} \(S3: ls = \text{c-unfold } (\text{c-len } u) \ (\text{c-snd } (u - (1::\text{nat})))\) \textbf{by} simp

\textbf{from} \(S3\) \textbf{have} \(S4: \text{length } ls = \text{c-len } u\) \textbf{by} (simp add: \text{c-unfold-len})

\textbf{from} \(S4\) \textbf{have} \(S5: \text{length } ls > 0\) \textbf{by} simp

\textbf{from} \(S5\) \textbf{have} \(S6: \text{ls } \neq []\) \textbf{by} simp

\textbf{from} \(D1\) \textbf{have} \(S7: \text{c-cons } x \ u = \text{list-to-nat } (x \# ls)\) \textbf{by} (simp add: \text{c-cons-def})

\textbf{have} \(S8: \text{list-to-nat } (x \# ls) = (\text{c-pair } ((\text{length } (x\#ls)) - (1::\text{nat})) \ (\text{c-fold } (x\#ls))) + 1\)

\textbf{by} (simp add: \text{list-to-nat-def})

\textbf{have} \(S9: (\text{length } (x\#ls)) - (1::\text{nat}) = \text{length } ls\) \textbf{by} simp

\textbf{from} \(S9\) \(S4\) \(S8\) \textbf{have} \(S10: \text{list-to-nat } (x \# ls) = (\text{c-pair } (\text{c-len } u) \ (\text{c-fold } (x\#ls))) + 1\)

\textbf{by} simp

\textbf{have} \(S11: \text{c-fold } (x\#ls) = \text{c-pair } (\text{c-snd } (u - (1::\text{nat})))\)

\textbf{proof} –

\textbf{from} \(S6\) \textbf{have} \(S11-1: \text{c-fold } (x\#ls) = \text{c-pair } (\text{c-fold } ls)\) \textbf{by} (rule \text{c-fold-0})

\textbf{from} \(S3\) \textbf{have} \(S11-2: \text{c-fold } ls = \text{c-fold } (\text{c-unfold } (\text{c-len } u) \ (\text{c-snd } (u - (1::\text{nat})))\))

\textbf{by} simp

\textbf{from} \(S1\) \(S11-2\) \textbf{have} \(S11-3: \text{c-fold } ls = \text{c-snd } (u - (1::\text{nat}))\) \textbf{by} (simp add: \text{th-2})

\textbf{from} \(S11-1\) \(S11-3\) \textbf{show} \(\text{thesis}\) \textbf{by} simp

\textbf{qed}

\textbf{from} \(S7\) \(S10\) \(S11\) \textbf{show} \(\text{thesis}\) \textbf{by} simp

\textbf{qed}

\textbf{lemma} \(\text{c-cons-aux3}: \text{c-cons } = (\lambda x u. \ (\text{sgn2 } u) * ((\text{c-pair } 0 x) + 1) + (\text{sgn1 } u) * ((\text{c-pair } (\text{c-len } u) \ (\text{c-snd } (u - (1::\text{nat})))) + 1))\)

\textbf{proof} (rule ext, rule ext)

\textbf{fix} \(x u\) \textbf{show} \(\text{c-cons } x \ u = (\text{sgn2 } u) * ((\text{c-pair } 0 x) + 1) + (\text{sgn1 } u) * ((\text{c-pair } (\text{c-len } u) \ (\text{c-snd } (u - (1::\text{nat})))) + 1)\) \(\text{if } R = ?R\)

\textbf{proof cases}

\textbf{assume} \(A1: u = 0\)

\textbf{then} \textbf{have} \(?R = (\text{c-pair } 0 x) + 1\) \textbf{by} simp

\textbf{moreover from} \(A1\) \textbf{have} \(\text{c-cons } x \ u = (\text{c-pair } 0 x) + 1\) \textbf{by} (simp add: \text{c-cons-aux1})

\textbf{ultimately show} \(\text{thesis}\) \textbf{by} simp

\textbf{next}

\textbf{assume} \(A1: u \neq 0\)

\textbf{then} \textbf{have} \(?R = (\text{c-pair } (\text{c-len } u) \ (\text{c-snd } (u - (1::\text{nat})))) + 1\)

\textbf{ultimately show} \(\text{thesis}\) \textbf{by} simp

\textbf{next}

\textbf{from} \(A1\) \textbf{have} \(S2: \text{c-cons } x \ u = (\text{c-pair } (\text{c-len } u) \ (\text{c-snd } (u - (1::\text{nat})))) + 1\) \textbf{by} (simp add: \text{c-cons-aux2})

47
from S1 S2 have c-cons x u = ?R by simp
then show ?thesis .
qed

lemma c-cons-pos: c-cons x u > 0
proof cases
assume u=0
then show c-cons x u > 0 by (simp add: c-cons-aux1)
next
assume ¬ u=0 then have u>0 by simp
then show c-cons x u > 0 by (simp add: c-cons-aux2)
qed

theorem c-cons-is-pr: c-cons ∈ PrimRec2
proof
have c-cons = (λ x u. (sgn2 u)*((c-pair 0 x)+1) + (sgn1 u)*((c-pair (c-len u) (c-pair x (c-snd (u-(1::nat)))))+1) (is - = ?R) by (simp add: c-cons-aux3)
moreover from c-pair-is-pr c-len-is-pr have ?R ∈ PrimRec2 by prec
ultimately show ?thesis by simp
qed

definition
    c-drop :: nat ⇒ nat ⇒ nat where
    c-drop = PrimRecOp (λ x. x) (λ x y z. c-tl y)

lemma c-drop-at-0 [simp]: c-drop 0 x = x by (simp add: c-drop-def)
lemma c-drop-at-Suc: c-drop (Suc y) x = c-tl (c-drop y x) by (simp add: c-drop-def)

theorem c-drop-is-pr: c-drop ∈ PrimRec2
proof
have (λ x. x) ∈ PrimRec1 by (rule pr-id1-1)
moreover from c-tl-is-pr have (λ x y z. c-tl y) ∈ PrimRec3 by prec
ultimately show ?thesis by (simp add: c-drop-def pr-rec)
qed

lemma c-tl-c-drop: c-tl (c-drop y x) = c-drop y (c-tl x)
apply(induct y)
apply(simp)
apply(simp add: c-drop-at-Suc)
done

lemma c-drop-at-Suc1: c-drop (Suc y) x = c-drop y (c-tl x)
apply(simp add: c-drop-at-Suc c-tl-c-drop)
done

lemma c-drop-df: ∀ ls. drop n ls = nat-to-list (c-drop n (list-to-nat ls))
proof (induct n)

48
show \( \forall \; \text{ls}. \; \text{drop} \; 0 \; \text{ls} = \text{nat-to-list} \; (\text{c-drop} \; 0 \; (\text{list-to-nat} \; \text{ls})) \) by (simp add: c-drop-def)

next

fix \( n \) assume \( A1: \forall \; \text{ls}. \; \text{drop} \; n \; \text{ls} = \text{nat-to-list} \; (\text{c-drop} \; n \; (\text{list-to-nat} \; \text{ls})) \)
then show \( \forall \; \text{ls}. \; \text{drop} \; (\text{Suc} \; n) \; \text{ls} = \text{nat-to-list} \; (\text{c-drop} \; (\text{Suc} \; n) \; (\text{list-to-nat} \; \text{ls})) \)
proof –
{
fix \( \text{ls}: \text{nat list} \)
  have \( S1: \text{drop} \; (\text{Suc} \; n) \; \text{ls} = \text{drop} \; n \; (\text{tl} \; \text{ls}) \) by (rule drop-Suc)
  from \( A1 \) have \( S2: \text{drop} \; n \; (\text{tl} \; \text{ls}) = \text{nat-to-list} \; (\text{c-drop} \; n \; (\text{list-to-nat} \; (\text{tl} \; \text{ls}))) \)
  by simp
also have \( \ldots = \text{nat-to-list} \; (\text{c-drop} \; n \; (\text{c-tl} \; (\text{list-to-nat} \; \text{ls}))) \) by (simp add: c-tl-eq-tl)
also have \( \ldots = \text{nat-to-list} \; (\text{c-drop} \; (\text{Suc} \; n) \; (\text{list-to-nat} \; \text{ls})) \) by (simp add: c-drop-at-Suc1)
finally have \( \text{drop} \; n \; (\text{tl} \; \text{ls}) = \text{nat-to-list} \; (\text{c-drop} \; (\text{Suc} \; n) \; (\text{list-to-nat} \; \text{ls})) \) by simp
with \( S1 \) have \( \text{drop} \; (\text{Suc} \; n) \; \text{ls} = \text{nat-to-list} \; (\text{c-drop} \; (\text{Suc} \; n) \; (\text{list-to-nat} \; \text{ls})) \) by simp
}
then show \( ?\text{thesis} \) by blast
qed

definition
\( \text{c-nth} :: \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \) where
\( \text{c-nth} = (\lambda \; x \; n. \; \text{c-hd} \; (\text{c-drop} \; n \; x)) \)

lemma \( \text{c-nth-is-pr}:: \text{c-nth} \in \text{PrimRec2} \)
proof (unfold \( \text{c-nth-def} \))
  from \( \text{c-hd-is-pr} \; \text{c-drop-is-pr} \) show \( (\lambda x \; n. \; \text{c-hd} \; (\text{c-drop} \; n \; x)) \in \text{PrimRec2} \) by prec
qed

lemma \( \text{c-nth-at-0}:: \text{c-nth} \; x \; 0 = \text{c-hd} \; x \) by (simp add: c-nth-def)

lemma \( \text{c-hd-c-cons} \) [simp]: \( \text{c-hd} \; (\text{c-cons} \; x \; y) = x \)
proof –
  have \( \text{c-cons} \; x \; y > 0 \) by (rule c-cons-pos)
  then show \( ?\text{thesis} \) by (simp add: c-hd-def c-cons-def)
qed

lemma \( \text{c-tl-c-cons} \) [simp]: \( \text{c-tl} \; (\text{c-cons} \; x \; y) = y \) by (simp add: c-tl-def c-cons-def)

definition
\( \text{c-f-list} :: (\text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat}) \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \) where
\( \text{c-f-list} = (\lambda \; f. \; \text{let} \; g = (\% x. \; \text{c-cons} \; (f \; 0 \; x) \; 0); \; h = (\% \; a \; b \; c. \; \text{c-cons} \; (f \; (\text{Suc} \; a) \; c) \; b)) \in \text{PrimRecOp} \; g \; h) \)
lemma c-f-list-at-0: c-f-list f 0 x = c-cons (f 0 x) 0 by simp add: c-f-list-def Let-def

lemma c-f-list-at-Suc: c-f-list f (Suc y) x = c-cons (f (Suc y) x) (c-f-list f y x) by (simp add: c-f-list-def Let-def)

lemma c-f-list-is-pr: f ∈ PrimRec2 ⟹ c-f-list f ∈ PrimRec2
proof –
  assume A1: f ∈ PrimRec2
  let ?g = (%x. c-cons (f 0 x) 0) from A1 c-cons-is-pr have S1: ?g ∈ PrimRec1 by prec
  let ?h = (%a b c. c-cons (f (Suc a) c) b) from A1 c-cons-is-pr have S2: ?h ∈ PrimRec3 by prec
  from S1 S2 show ?thesis by simp add: pr-rec c-f-list-def Let-def
qed

lemma c-f-list-to-f-0: f y x = c-hd (c-f-list f y x)
apply (induct y)
apply (simp add: c-f-list-at-0)
apply (simp add: c-f-list-at-Suc)
done

lemma c-f-list-to-f: f = (λ y x. c-hd (c-f-list f y x))
apply (rule ext, rule ext)
apply (rule c-f-list-to-f-0)
done

lemma c-f-list-f-is-pr: c-f-list f ∈ PrimRec2 ⟹ f ∈ PrimRec2
proof –
  assume A1: c-f-list f ∈ PrimRec2
  have S1: f = (λ y x. c-hd (c-f-list f y x)) by (rule c-f-list-to-f)
  from A1 c-hd-is-pr have S2: (λ y x. c-hd (c-f-list f y x)) ∈ PrimRec2 by prec
  with S1 show ?thesis by simp
qed

lemma c-f-list-lm-1: c-nth (c-cons x y) (Suc z) = c-nth y z by simp add: c-nth-def c-drop-at-Suc1

lemma c-f-list-lm-2: z < Suc n ⟹ c-nth (c-f-list f (Suc n) x) (Suc n - z) = c-nth (c-f-list f n x) (n - z)
proof –
  assume z < Suc n
  then have Suc n - z = Suc (n - z) by arith
  then have c-nth (c-f-list f (Suc n) x) (Suc n - z) = c-nth (c-f-list f (Suc n) x) (Suc (n - z)) by simp
  also have ... = c-nth (c-cons (f (Suc n) x) (c-f-list f n x)) (Suc (n - z)) by simp add: c-f-list-at-Suc
  also have ... = c-nth (c-f-list f n x) (n - z) by simp add: c-f-list-lm-1

50
finally show \( \text{thesis by simp} \)

qed

lemma \( \text{c-f-list-nth: } z \leq y \rightarrow \text{c-nth (c-f-list f y x)} (y-z) = f z x \)

proof (induct y)
  show \( z \leq 0 \rightarrow \text{c-nth (c-f-list f 0 x)} (0 - z) = f z x \)
  proof
    assume \( z \leq 0 \) then have \( A1: z = 0 \) by simp
    then have \( \text{c-nth (c-f-list f 0 x)} (0 - z) = \text{c-nth (c-f-list f 0 x)} 0 \) by simp
    also have \( \ldots = \text{c-hd (c-f-list f 0 x)} \) by (simp add: c-nth-at-0)
    also have \( \ldots = \text{c-hd (c-cons (f 0 x) 0)} \) by (simp add: c-f-list-at-0)
    also have \( \ldots = f 0 x \) by simp
    finally show \( \text{c-nth (c-f-list f 0 x)} (0 - z) = f z x \) by (simp add: A1)
  qed
next
  fix \( n \) assume \( A2: z \leq n \rightarrow \text{c-nth (c-f-list f n x)} (n - z) = f z x \) show \( z \leq \text{Suc n} \rightarrow \text{c-nth (c-f-list f (Suc n) x)} (\text{Suc n} - z) = f z x \)
  proof
    assume \( A3: z \leq \text{Suc n} \)
    show \( z \leq \text{Suc n} \rightarrow \text{c-nth (c-f-list f (Suc n) x)} (\text{Suc n} - z) = f z x \)
    proof cases
      assume \( A A 1: z \leq n \)
      then have \( A A 2: z < \text{Suc n} \) by simp
      from \( A 2 \) this have \( S 1: \text{c-nth (c-f-list f n x)} (n - z) = f z x \) by auto
      from \( A A 2 \) have \( \text{c-nth (c-f-list f (Suc n) x)} (\text{Suc n} - z) = \text{c-nth (c-f-list f n x)} (n - z) \) by (rule c-f-list-lm-2)
      with \( S 1 \) show \( \text{c-nth (c-f-list f (Suc n) x)} (\text{Suc n} - z) = f z x \) by simp
    next
      assume \( \neg z \leq n \)
      from \( A 3 \) this have \( S 1: z = \text{Suc n} \) by simp
      then have \( S 2: \text{Suc n} - z = 0 \) by simp
      then have \( \text{c-nth (c-f-list f (Suc n) x)} (\text{Suc n} - z) = \text{c-nth (c-f-list f (Suc n) x)} 0 \) by simp
      also have \( \ldots = \text{c-hd (c-f-list f (Suc n) x)} \) by (simp add: c-nth-at-0)
      also have \( \ldots = \text{c-hd (c-cons (f (Suc n) x) (c-f-list f n x))} \) by (simp add: c-f-list-at-Suc)
      also have \( \ldots = f (\text{Suc n} x) \) by simp
      finally show \( \text{c-nth (c-f-list f (Suc n) x)} (\text{Suc n} - z) = f z x \) by (simp add: S1)
    qed
  qed
  qed

theorem \( \text{th-pr-rec: } [ g \in \text{PrimRec1}; h \in \text{PrimRec3}; (\forall x. f 0 x = (g x)); (\forall x g. (f (\text{Suc y}) x) = h y (f y x) x) ] \rightarrow f \in \text{PrimRec2} \)

proof
  assume \( g \)-is-pr: \( g \in \text{PrimRec1} \)
  assume \( h \)-is-pr: \( h \in \text{PrimRec3} \)
  assume \( f\)-0: \( \forall x. f 0 x = g x \)
**Theorem** th-rec: \( g \in \text{PrimRec}1; \alpha \in \text{PrimRec}2; h \in \text{PrimRec}3; (\forall \ x y. \alpha \ y x \leq y); (\forall x. (f \ 0 x) = (g x)); (\forall \ x y. (f (\text{Suc} y) x) = h y (f (\alpha y x) x)) \) \( \implies f \in \text{PrimRec}2 \)

**Proof** –

**assume** \( g \text{-is-pr}; g \in \text{PrimRec}1 \)

**assume** \( \alpha \text{-is-pr}; \alpha \in \text{PrimRec}2 \)

**assume** \( h \text{-is-pr}; h \in \text{PrimRec}3 \)

**assume** \( \alpha \text{-le}; (\forall \ x y. \alpha \ y x \leq y) \)

**assume** \( f \text{-0}; \forall x. f 0 x = g x \)

**assume** \( f \text{-1}; \forall x y. (f (\text{Suc} y) x) = h y (f (\alpha y x) x) \)

**let** \( ?g' = \lambda x. \text{c-cons} (g x) 0 \)

**let** \( ?h' = \lambda a b c. \text{c-cons} (h a (\text{c-nth} b (a - (\alpha a c))) c) b \)

**let** \( ?r = \text{c-f-list} f \)

**from** \( g \text{-is-pr} \text{ c-cons-is-pr} \text{ have} \ g' \text{-is-pr}; ?g' \in \text{PrimRec}1 \text{ by prec} \)

**from** \( h \text{-is-pr} \text{ c-cons-is-pr} \text{ c-nth-is-pr} a \text{-is-pr} \text{ have} \ h' \text{-is-pr}; ?h' \in \text{PrimRec}3 \text{ by prec} \)

**have** \( S1; \forall x. ?r 0 x = ?g' x \)

**proof** –

**fix** \( x \) **have** \( ?r 0 x = \text{c-cons} (f 0 x) 0 \text{ by } \text{rule c-f-list-at-0} \)

**with** \( f \text{-0} \text{ have} \ ?r 0 x = \text{c-cons} (g x) 0 \text{ by simp} \)

**then** **show** \( ?r 0 x = ?g' x \text{ by simp} \)

**qed**

**have** \( S2; \forall x y. ?r (\text{Suc} y) x = ?h' y (\?r y x) x \)

**proof** \( \text{rule allI, rule allII} \)

**fix** \( x y \) **show** \( ?r (\text{Suc} y) x = ?h' y (\?r y x) x \)

**proof** –

**have** \( S2-1; ?r (\text{Suc} y) x = \text{c-cons} (f (\text{Suc} y) x) (?r y x) \text{ by } \text{rule c-f-list-at-Suc} \)

**with** \( f \text{-1} \text{ have} \ S2-2; f (\text{Suc} y) x = h y (f (\alpha y x) x) x \text{ by simp} \)

**from** \( \alpha \text{-le} \text{ have} \ S2-3; \alpha y x \leq y \text{ by simp} \)

**then** **have** \( S2-4; f (\alpha y x) x = \text{c-nth} (?r y x) (y-(\alpha y x)) \text{ by } (\text{simp add: c-f-list-nth}) \)

**from** \( S2-1 \text{ S2-2 S2-4} \text{ show } \text{thesis} \text{ by simp} \)
\textbf{qed}

\textbf{qed}

\textbf{from} \textit{g'-is-pr h'-is-pr S1 S2 have S3: ?r ∈ PrimRec2 by (rule th-pr-rec)}

\textbf{then show} \textit{f ∈ PrimRec2 by (rule e-f-list-f-is-pr)}

\textbf{qed}

\textbf{declare}\textit{ c-tl-less \[termination-simp\]}

\textbf{fun} \textit{c-assoc-have-key :: nat ⇒ nat ⇒ nat where}

\textbf{c-assoc-have-key-df}\textit{ [simp del]: c-assoc-have-key y x = (if y = 0 then 1 else (if c-fst (c-hd y) = x then 0 else c-assoc-have-key (c-tl y) x))}

\textbf{lemma} \textit{c-assoc-have-key-lm-1: y \neq 0 ⇒ c-assoc-have-key y x = (if c-fst (c-hd y) = x then 0 else c-assoc-have-key (c-tl y) x)} \textbf{by (simp add: c-assoc-have-key-df)}

\textbf{theorem} \textit{c-assoc-have-key-is-pr: c-assoc-have-key ∈ PrimRec2}

\textbf{proof} –

\textbf{let} \textit{?h = \lambda a b c. if c-fst (c-hd (Suc a)) = c then 0 else b}

\textbf{let} \textit{?a = \lambda y x. c-tl (Suc y)}

\textbf{let} \textit{?g = \lambda x. (1::nat) have g-is-pr: ?g ∈ PrimRec1 by (rule const-is-pr) from c-tl-is-pr have a-is-pr: ?a ∈ PrimRec2 by prec have h-is-pr: ?h ∈ PrimRec3 proof (rule if-eq-is-pr3) from c-fst-is-pr c-hd-is-pr show \((λx y z. c-fst (c-hd (Suc x))) ∈ PrimRec3\) by prec next show \((λx y z. Suc y) \in PrimRec3\) by \textit{prec} next show \((λx y z. Suc y) \in PrimRec3\) by \textit{prec} next show \((λx y z. Suc y) \in PrimRec3\) by \textit{prec}TES

\textbf{qed}

\textbf{have} \textit{a-le: \forall x y. ?a y x ≤ y}

\textbf{proof} (\textit{rule allI, rule allI})

\textbf{fix} \textit{x y show ?a y x ≤ y}

\textbf{proof} –

\textbf{have} \textit{Suc y > 0 by simp then have ?a y x < Suc y by (rule c-tl-less)}

\textbf{then show} \textit{?thesis by simp}

\textbf{qed}

\textbf{fun} \textit{c-assoc-value :: nat ⇒ nat ⇒ nat where}
lemma c-assoc-value-lm-1: \( y \neq 0 \Rightarrow c\text{-assoc-value} y x = (\text{if } c\text{-fst} (c\text{-hd} y) = x \text{ then } c\text{-snd} (c\text{-hd} y) \text{ else } c\text{-assoc-value} (c\text{-tl} y) x) \) by (simp add: c-assoc-value-df)

theorem c-assoc-value-is-pr: \( c\text{-assoc-value} \in \text{PrimRec2} \)
proof
  let \( ?h = \lambda a b c. \text{if } c\text{-fst} (\text{c-hd} (\text{Suc} a)) = c \text{ then } c\text{-snd} (\text{c-hd} (\text{Suc} a)) \text{ else } b \)
  let \( ?a = \lambda y x. \text{c-tl} (\text{Suc} y) \)
  let \( ?g = \lambda x. (0::\text{nat}) \)
  have g-is-pr: \( ?g \in \text{PrimRec1} \) by (rule const-is-pr)
  from c-tl-is-pr have a-is-pr: \( ?a \in \text{PrimRec2} \) by prec
  have h-is-pr: \( ?h \in \text{PrimRec3} \)
    proof (rule if-eq-is-pr3)
      from c-fst-is-pr c-hd-is-pr
      show \( (\lambda x y z. c\text{-fst} (c\text{-hd} (\text{Suc} x))) \in \text{PrimRec3} \) by prec
    next
    show \( (\lambda x y z. 0) \in \text{PrimRec3} \) by (rule pr-id3-2)
  qed

  have a-le: \( \forall x y. ?a y x \leq y \)
    proof (rule allI, rule allI)
      fix \( x y \)
      show \( ?a y x \leq y \)
        proof
          have \( \text{Suc} y > 0 \) by simp
          then have \( ?a y x < \text{Suc} y \) by (rule c-tl-less)
          then show \( \text{thesis} \) by simp
        qed
  qed

  have f-0: \( \forall x. c\text{-assoc-value} 0 x = ?g x \) by (simp add: c-assoc-value-df)
  have f-1: \( \forall x y. c\text{-assoc-value} (\text{Suc} y) x = ?h y (c\text{-assoc-value} (?a y x) x) \) by (simp add: c-assoc-value-df)
    from g-is-pr a-is-pr h-is-pr a-le f-0 f-1 show \( \text{thesis} \) by (rule th-rec)
  qed

lemma c-assoc-lm-1: \( c\text{-assoc-have-key} (c\text{-cons} (c\text{-pair} x y) z) x = 0 \)
apply(simp add: c-assoc-have-key-df)
apply(simp add: c-cons-pos)
done

lemma c-assoc-lm-2: \( c\text{-assoc-value} (c\text{-cons} (c\text{-pair} x y) z) x = y \)
apply(simp add: c-assoc-value-df)
apply(rule impl)
apply(insert c-cons-pos [where \( x = (c\text{-pair} x y) \) and \( u = z \)])
apply(auto)
done

lemma c-assoc-lm-3: \( x_1 \neq x \implies c\text{-assoc-have-key} (c\text{-cons} (c\text{-pair} x y) z) x_1 = c\text{-assoc-have-key} z x_1 \)
proof -
  assume A1: \( x_1 \neq x \)
  let \( ?ls = (c\text{-cons} (c\text{-pair} x y) z) \)
  have S1: \( ?ls \neq 0 \) by (simp add: c-cons-pos)
  then have S2: c-assoc-have-key ?ls x_1 = (if c-fst (c-hd ?ls) = x_1 then 0 else c-assoc-have-key (c-tl ?ls) x_1) (is - = ?R) by (rule c-assoc-have-key-lm-1)
  have S3: c-fst (c-hd ?ls) = x by simp
  with A1 have S4: \( (c\text{-fst} (c\text{-hd} ?ls) = x_1) \) by simp
  from S2 S4 have S5: ?R = c-assoc-have-key (c-tl ?ls) x_1 by (rule if-not-P)
  from S2 S5 show ?thesis by simp
qed

lemma c-assoc-lm-4: \( x_1 \neq x \implies c\text{-assoc-value} (c\text{-cons} (c\text{-pair} x y) z) x_1 = c\text{-assoc-value} z x_1 \)
proof -
  assume A1: \( x_1 \neq x \)
  let \( ?ls = (c\text{-cons} (c\text{-pair} x y) z) \)
  have S1: \( ?ls \neq 0 \) by (simp add: c-cons-pos)
  then have S2: c-assoc-value ?ls x_1 = (if c-fst (c-hd ?ls) = x_1 then c-snd (c-hd ?ls) x_1 else c-assoc-value (c-tl ?ls) x_1) (is - = ?R) by (rule c-assoc-value-lm-1)
  have S3: c-fst (c-hd ?ls) = x by simp
  with A1 have S4: \( (c\text{-fst} (c\text{-hd} ?ls) = x_1) \) by simp
  from S2 S4 have S5: ?R = c-assoc-value (c-tl ?ls) x_1 by (rule if-not-P)
  from S2 S5 show ?thesis by simp
qed

4 Primitive recursive functions of one variable

theory PRecFun2
imports PRecFun
begin

4.1 Alternative definition of primitive recursive functions of one variable

definition UnaryRecOp :: \( \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \) where
 UnaryRecOp = (\( \lambda \ g \ h. \ \text{pr-conv-2-to-1} (\text{PrimRecOp} g \ (\text{pr-conv-1-to-3} \ h)) \))

lemma unary-rec-into-pr: \( \forall g \in \text{PrimRec1}; h \in \text{PrimRec1} \quad \Rightarrow \text{UnaryRecOp} g \ h \in \text{PrimRec1} \) by (simp add: UnaryRecOp-def pr-conv-1-to-3-lm pr-conv-2-to-1-lm pr-rec)
definition

\[ c\text{-}f\text{-}pair :: (\text{nat} \Rightarrow \text{nat}) \Rightarrow (\text{nat} \Rightarrow \text{nat}) \Rightarrow (\text{nat} \Rightarrow \text{nat}) \text{ where} \]
\[ c\text{-}f\text{-}pair = (\lambda f g x. \text{c-pair} (f \, x) \, (g \, x)) \]

lemma c-f-pair-to-pr: \[ f \in \text{PrimRec1}; \, g \in \text{PrimRec1} \] \implies c-f-pair \, f \, g \in \text{PrimRec1}

unfolding c-f-pair-def by prec

inductive-set PrimRec1': (\text{nat} \Rightarrow \text{nat}) set

where

\begin{align*}
\text{zero}: \quad & \lambda x. \, 0 \in \text{PrimRec1}' \\
\text{suc}: \quad & \text{Suc} \in \text{PrimRec1}' \\
\text{fst}: \quad & \text{c-fst} \in \text{PrimRec1}' \\
\text{snd}: \quad & \text{c-snd} \in \text{PrimRec1}' \\
\text{comp}: \quad & \lambda f \in \text{PrimRec1}'; \, g \in \text{PrimRec1}' \implies (\lambda x. \, (g \, x)) \in \text{PrimRec1}' \\
\text{pair}: \quad & \lambda f \in \text{PrimRec1}'; \, g \in \text{PrimRec1}' \implies \text{c-f-pair} \, f \, g \in \text{PrimRec1}' \\
\text{un-rec}: \quad & \lambda f \in \text{PrimRec1}'; \, g \in \text{PrimRec1}' \implies \text{UnaryRecOp} \, f \, g \in \text{PrimRec1}' \\
\end{align*}

lemma primrec'-into-primrec: \[ f \in \text{PrimRec1}' \implies f \in \text{PrimRec1} \]

proof (induct f rule: PrimRec1'.induct)

case zero show ?case by (rule pr-zero)
next
case suc show ?case by (rule pr-suc)
next
case fst show ?case by (rule c-fst-is-pr)
next
case snd show ?case by (rule c-snd-is-pr)
next
case comp from comp show ?case by (simp add: pr-comp1-1)
next
case pair from pair show ?case by (simp add: c-f-pair-to-pr)
next
case un-rec from un-rec show ?case by (simp add: unary-rec-into-pr)
qed

lemma pr-id1-1': \( \lambda x. \, x \in \text{PrimRec1}' \)

proof

have c-f-pair c-fst c-snd \in PrimRec1' by (simp add: PrimRec1'.fst PrimRec1'.snd PrimRec1'.pair)

moreover have c-f-pair c-fst c-snd = (\lambda x. \, x) by (simp add: c-f-pair-def)

ultimately show ?thesis by simp
qed

lemma pr-id2-1': pr-conv-2-to-1 (\lambda x y. \, x) \in PrimRec1' by (simp add: pr-conv-2-to-1-def PrimRec1'.fst)

lemma pr-id2-2': pr-conv-2-to-1 (\lambda x y. \, y) \in PrimRec1' by (simp add: pr-conv-2-to-1-def PrimRec1'.snd)

56
lemma \textit{pr-id3-1}: \(\text{pr-conv-3-to-1} \; (\lambda \; x \; y \; z \; . \; x) \in \text{PrimRec1}'\)

proof

\begin{itemize}
\item have \(\text{pr-conv-3-to-1} \; (\lambda \; x \; y \; z \; . \; x) = (\lambda x \cdot \text{c-fst} \; (\text{c-fst} \; x))\) by (simp add: \text{pr-conv-3-to-1-def})
\item moreover from \text{PrimRec1}'.\text{fst} \text{PrimRec1}'.\text{fst} \; (\lambda x \cdot \text{c-fst} \; (\text{c-fst} \; x)) \in \text{PrimRec1}' \; \text{by} \; (\text{rule PrimRec1}'.\text{comp})
\item ultimately show \(?\text{thesis} \; \text{by simp}\)
\end{itemize}

qed

lemma \textit{pr-id3-2}: \(\text{pr-conv-3-to-1} \; (\lambda x \; y \; z \; . \; y) \in \text{PrimRec1}'\)

proof

\begin{itemize}
\item have \(\text{pr-conv-3-to-1} \; (\lambda x \; y \; z \; . \; y) = (\lambda x \cdot \text{c-snd} \; (\text{c-fst} \; x))\) by (simp add: \text{pr-conv-3-to-1-def})
\item moreover from \text{PrimRec1}'.\text{fst} \text{PrimRec1}'.\text{fst} \; (\lambda x \cdot \text{c-snd} \; (\text{c-fst} \; x)) \in \text{PrimRec1}' \; \text{by} \; (\text{rule PrimRec1}'.\text{comp})
\item ultimately show \(?\text{thesis} \; \text{by simp}\)
\end{itemize}

qed

lemma \textit{pr-id3-3}: \(\text{pr-conv-3-to-1} \; (\lambda x \; y \; z \; . \; z) \in \text{PrimRec1}'\)

proof

\begin{itemize}
\item have \(\text{pr-conv-3-to-1} \; (\lambda x \; y \; z \; . \; z) = (\lambda x \cdot \text{c-snd} \; x)\) by (simp add: \text{pr-conv-3-to-1-def})
\item thus \(?\text{thesis} \; \text{by simp add: PrimRec1}'.\text{snd}\)
\end{itemize}

qed

lemma \textit{pr-comp2-1}: \(\text{pr-conv-2-to-1} \; f \in \text{PrimRec1}' \; ; \; g \in \text{PrimRec1}' \; ; \; h \in \text{PrimRec1}' \implies (\lambda x \cdot f \; (g \; (h \; x)) \; (h \; x)) \in \text{PrimRec1}'\)

proof

\begin{itemize}
\item assume \(A1: \text{pr-conv-2-to-1} \; f \in \text{PrimRec1}'\)
\item assume \(A2: \; g \in \text{PrimRec1}'\)
\item assume \(A3: \; h \in \text{PrimRec1}'\)
\item let \(?f1 = \text{pr-conv-2-to-1} \; f\)
\item have \(S1: \; \forall x. \; ?f1 \; ((\text{c-f-pair} \; g \; h) \; x)) = (\lambda x \cdot f \; (g \; (h \; x))\) by \((\text{simp add: c-f-pair-def pr-conv-2-to-1-def})\)
\item from \(A2 \; A3 \text{ have } S2: \; \text{c-f-pair} \; g \; h \in \text{PrimRec1}' \; \text{by} \; (\text{rule PrimRec1}'.\text{pair})\)
\item from \(A1 \; S2 \text{ have } S3: \; \forall x. \; ?f1 \; ((\text{c-f-pair} \; g \; h) \; x)) \in \text{PrimRec1}' \; \text{by} \; (\text{rule PrimRec1}'.\text{comp})\)
\item with \(S1\) show \(?\text{thesis} \; \text{by simp}\)
\end{itemize}

qed

lemma \textit{pr-comp3-1}: \(\text{pr-conv-3-to-1} \; f \in \text{PrimRec1}' \; ; \; g \in \text{PrimRec1}' \; ; \; h \in \text{PrimRec1}' \; ; \; k \in \text{PrimRec1}' \implies (\lambda x \cdot f \; (g \; (h \; x)) \; (h \; x) \; (k \; x)) \in \text{PrimRec1}'\)

proof

\begin{itemize}
\item assume \(A1: \text{pr-conv-3-to-1} \; f \in \text{PrimRec1}'\)
\item assume \(A2: \; g \in \text{PrimRec1}'\)
\item assume \(A3: \; h \in \text{PrimRec1}'\)
\item assume \(A4: \; k \in \text{PrimRec1}'\)
\item from \(A2 \; A3 \text{ have } \text{c-f-pair} \; g \; h \in \text{PrimRec1}' \; \text{by} \; (\text{rule PrimRec1}'.\text{pair})\)
\item from \(\text{this} \; A4 \text{ have } \text{c-f-pair} \; (\text{c-f-pair} \; g \; h) \; k \in \text{PrimRec1}' \; \text{by} \; (\text{rule PrimRec1}'.\text{pair})\)
\item from \(A1 \; \text{this} \text{ have } \forall x. \; \text{(pr-conv-3-to-1} \; f) \; ((\text{c-f-pair} \; (\text{c-f-pair} \; g \; h) \; k) \; x) \in \text{PrimRec1}' \; \text{by} \; (\text{rule PrimRec1}'.\text{comp})\)
\item then show \(?\text{thesis} \; \text{by simp add: c-f-pair-def pr-conv-3-to-1-def})\)
\end{itemize}
lemma pr-comp1-2′: \[ f \in \text{PrimRec1}' \land \text{pr-conv-2-to-1} \ g \in \text{PrimRec1}' \] \implies \text{pr-conv-2-to-1} \ ((\lambda \ y. \ f \ (g \ y)) \ x) \in \text{PrimRec1}'
proof
  assume \( f \in \text{PrimRec1}' \)
  and \( \text{pr-conv-2-to-1} \ g \in \text{PrimRec1}' \)
  then have \( (\lambda \ x. \ f \ (g \ x)) \in \text{PrimRec1}' \) by \( \text{rule PrimRec1',comp} \)
  then show \( ?\text{thesis} \) by \( \text{simp add: pr-conv-2-to-1-def} \)
qed

lemma pr-comp1-3′: \[ f \in \text{PrimRec1}' \land \text{pr-conv-3-to-1} \ g \in \text{PrimRec1}' \] \implies \text{pr-conv-3-to-1} \ ((\lambda \ x z. \ f \ (g \ x)) \ y z) \in \text{PrimRec1}'
proof
  assume \( f \in \text{PrimRec1}' \)
  and \( \text{pr-conv-3-to-1} \ g \in \text{PrimRec1}' \)
  then have \( (\lambda \ x. \ f \ (g \ x)) \in \text{PrimRec1}' \) by \( \text{rule PrimRec1',comp} \)
  then show \( ?\text{thesis} \) by \( \text{simp add: pr-conv-3-to-1-def} \)
qed

lemma pr-comp2-2′: \[ \text{pr-conv-2-to-1} \ f \in \text{PrimRec1}' \land \text{pr-conv-2-to-1} \ g \in \text{PrimRec1}' \land \text{pr-conv-2-to-1} \ h \in \text{PrimRec1}' \] \implies \text{pr-conv-2-to-1} \ ((\lambda \ x y. \ f \ (g \ x)) \ (h \ y)) \in \text{PrimRec1}'
proof
  assume \( \text{pr-conv-2-to-1} \ f \in \text{PrimRec1}' \)
  and \( \text{pr-conv-2-to-1} \ g \in \text{PrimRec1}' \)
  and \( \text{pr-conv-2-to-1} \ h \in \text{PrimRec1}' \)
  then have \( (\lambda \ x. \ f \ (g \ x)) \ (h \ x) \ y) \in \text{PrimRec1}' \) by \( \text{rule pr-comp2-1'} \)
  then show \( ?\text{thesis} \) by \( \text{simp add: pr-conv-2-to-1-def} \)
qed

lemma pr-comp2-3′: \[ \text{pr-conv-2-to-1} \ f \in \text{PrimRec1}' \land \text{pr-conv-3-to-1} \ g \in \text{PrimRec1}' \land \text{pr-conv-2-to-1} \ h \in \text{PrimRec1}' \land \text{pr-conv-2-to-1} \ k \in \text{PrimRec1}' \] \implies \text{pr-conv-2-to-1} \ ((\lambda \ x y. \ f \ (g \ x)) \ (h \ y) \ (k \ y)) \in \text{PrimRec1}'
proof
  assume \( \text{pr-conv-2-to-1} \ f \in \text{PrimRec1}' \)
  and \( \text{pr-conv-3-to-1} \ g \in \text{PrimRec1}' \)
  and \( \text{pr-conv-2-to-1} \ h \in \text{PrimRec1}' \)
  and \( \text{pr-conv-2-to-1} \ k \in \text{PrimRec1}' \)
  then have \( (\lambda \ x. \ f \ (g \ x)) \ (h \ x) \ (k \ x) \ y) \in \text{PrimRec1}' \) by \( \text{rule pr-comp2-1'} \)
  then show \( ?\text{thesis} \) by \( \text{simp add: pr-conv-2-to-1-def} \)
qed
and \( \text{pr-conv-2-to-1} \ k \in \text{PrimRec1}' (\text{is} \ ?k1 \in \text{PrimRec1}') \)
then have \( (\lambda \ x. \ f \ (\exists g1 \ x) \ (\exists h1 \ x) \ (\exists k1 \ x)) \in \text{PrimRec1}' \) by (rule \( \text{pr-comp3-1}' \))
then show \( ?\text{thesis} \) by (simp add: \( \text{pr-conv-2-to-1-def} \))
qed

**Lemma pr-comp3-3':** \[ \text{pr-conv-2-to-1} \ f \in \text{PrimRec1}' ; \text{pr-conv-3-to-1} \ g \in \text{PrimRec1}' ; \text{pr-conv-3-to-1} \ h \in \text{PrimRec1}' ; \text{pr-conv-3-to-1} \ k \in \text{PrimRec1}' \Rightarrow \text{pr-conv-3-to-1} \]
(\( \lambda \ x \ y \ z \ f \ (g \ x \ y \ z) \ (h \ x \ y \ z) \ (k \ x \ y \ z) \)) \in \text{PrimRec1}'

**Proof**
assume \( \text{pr-conv-3-to-1} \ f \in \text{PrimRec1}' \)
and \( \text{pr-conv-3-to-1} \ g \in \text{PrimRec1}' (\text{is} \ ?g1 \in \text{PrimRec1}') \)
and \( \text{pr-conv-3-to-1} \ h \in \text{PrimRec1}' (\text{is} \ ?h1 \in \text{PrimRec1}') \)
and \( \text{pr-conv-3-to-1} \ k \in \text{PrimRec1}' (\text{is} \ ?k1 \in \text{PrimRec1}') \)
then have \( (\lambda \ x. \ f \ (\exists g1 \ x) \ (\exists h1 \ x) \ (\exists k1 \ x)) \in \text{PrimRec1}' (\text{by} (\text{rule} \ \text{pr-comp3-1}')) \)
then show \( ?\text{thesis} \) by (simp add: \( \text{pr-conv-3-to-1-def} \))
qed

**Lemma \( \text{bn}' \):** \( (f1 \in \text{PrimRec1} \Rightarrow f1 \in \text{PrimRec1}') \land (g1 \in \text{PrimRec2} \Rightarrow \text{pr-conv-2-to-1} \ g1 \in \text{PrimRec1}') \land (h1 \in \text{PrimRec3} \Rightarrow \text{pr-conv-3-to-1} \ h1 \in \text{PrimRec1}') \)

**Proof** (induct rule: \( \text{PrimRec1-PrimRec2-PrimRec3.induct} \))
case **zero** show \( ?\text{case} \) by (rule \( \text{PrimRec1}'.\text{zero} \))
next case **suc** show \( ?\text{case} \) by (rule \( \text{PrimRec1}'.\text{suc} \))
next case **id1-1** show \( ?\text{case} \) by (rule \( \text{pr-id1-1}' \))
next case **id2-1** show \( ?\text{case} \) by (rule \( \text{pr-id2-1}' \))
next case **id2-2** show \( ?\text{case} \) by (rule \( \text{pr-id2-2}' \))
next case **id3-1** show \( ?\text{case} \) by (rule \( \text{pr-id3-1}' \))
next case **id3-2** show \( ?\text{case} \) by (rule \( \text{pr-id3-2}' \))
next case **id3-3** show \( ?\text{case} \) by (rule \( \text{pr-id3-3}' \))
next case **comp1-1** from **comp1-1** show \( ?\text{case} \) by (simp add: \( \text{PrimRec1}'.\text{comp} \))
next case **comp1-2** from **comp1-2** show \( ?\text{case} \) by (simp add: \( \text{pr-comp1-2}' \))
next case **comp1-3** from **comp1-3** show \( ?\text{case} \) by (simp add: \( \text{pr-comp1-3}' \))
next case **comp2-1** from **comp2-1** show \( ?\text{case} \) by (simp add: \( \text{pr-comp2-1}' \))
next case **comp2-2** from **comp2-2** show \( ?\text{case} \) by (simp add: \( \text{pr-comp2-2}' \))
next case **comp2-3** from **comp2-3** show \( ?\text{case} \) by (simp add: \( \text{pr-comp2-3}' \))
next case **comp3-1** from **comp3-1** show \( ?\text{case} \) by (simp add: \( \text{pr-comp3-1}' \))
next case **comp3-2** from **comp3-2** show \( ?\text{case} \) by (simp add: \( \text{pr-comp3-2}' \))
next case **comp3-3** from **comp3-3** show \( ?\text{case} \) by (simp add: \( \text{pr-comp3-3}' \))
next case **prim-rec**
fix \( g \ h \) assume **A1**: \( g \in \text{PrimRec1}' \) and \( \text{pr-conv-3-to-1} \ h \in \text{PrimRec1}' \)
then have \( \text{UnaryRecOp} \ g \ (\text{pr-conv-3-to-1} \ h) \in \text{PrimRec1}' (\text{by} (\text{rule} \ \text{PrimRec1}'.\text{un-rec})) \)
moreover have \( \text{UnaryRecOp} \ g \ (\text{pr-conv-3-to-1} \ h) = \text{pr-conv-2-to-1} (\text{PrimRecOp} \ g \ h) (\text{by} (\text{simp add: \( \text{UnaryRecOp-def} \))) \)
ultimately show \( \text{pr-conv-2-to-1} (\text{PrimRecOp} \ g \ h) \in \text{PrimRec1}' (\text{by} \ \text{simp}) \)
qed

**Theorem pr-1-eq-1':** \( \text{PrimRec1} = \text{PrimRec1}' \)

**Proof**
have **S1**: \( \land f. \ f \in \text{PrimRec1} \Rightarrow f \in \text{PrimRec1}' \) by (simp add: \( \text{bn}' \))
have S2: \( \forall f, \bar{f} \in \text{PrimRec}^1 \rightarrow \bar{f} \in \text{PrimRec}^1 \) by (simp add: primrec'-into-primrec)

from S1 S2 show \(?thesis\) by blast

qed

4.2 The scheme datatype

datatype PrimScheme = Base-zero | Base-suc | Base-fst | Base-snd
| Comp-op PrimScheme PrimScheme
| Pair-op PrimScheme PrimScheme
| Rec-op PrimScheme PrimScheme

primrec
sch-to-pr :: PrimScheme \( \Rightarrow \) (nat \( \Rightarrow \) nat)
where
\[
\begin{align*}
sch-to-pr\ Base-zero &= (\lambda x. 0) \\
sch-to-pr\ Base-suc &= Suc \\
sch-to-pr\ Base-fst &= c-fst \\
sch-to-pr\ Base-snd &= c-snd \\
sch-to-pr (\text{Comp-op}\ t1\ t2) &= (\lambda x. (sch-to-pr t1) ((sch-to-pr t2) x)) \\
sch-to-pr (\text{Pair-op}\ t1\ t2) &= c-f-pair (sch-to-pr t1) (sch-to-pr t2) \\
sch-to-pr (\text{Rec-op}\ t1\ t2) &= \text{UnaryRecOp} (sch-to-pr t1) (sch-to-pr t2)
\end{align*}
\]

lemma sch-to-pr-into-pr: sch-to-pr \( \bar{sch} \) \( \in \) PrimRec1 by (simp add: pr-1-eq-1', induct sch, simp-all add: PrimRec1'.intras)

lemma sch-to-pr-srj: \( f \in \text{PrimRec}^1 \Rightarrow (\exists \ bar{sch}. \ f = \text{sch-to-pr} \ bar{sch}) \)

proof –
assume \( f \in \text{PrimRec}^1 \) then have A1: \( f \in \text{PrimRec}^1^' \) by (simp add: pr-1-eq-1')
from A1 show \(?thesis\)
proof (induct f rule: PrimRec1'.induct)
have \((\lambda x. 0) = \text{sch-to-pr}\ Base-zero\) by simp
then show \( \exists sch. (\lambda u. 0) = \text{sch-to-pr} \ bar{sch} \) by (rule exI)
next
have Suc = sch-to-pr Base-suc by simp
then show \( \exists sch. \ Suc = \text{sch-to-pr} \ bar{sch} \) by (rule exI)
next
have c-fst = sch-to-pr Base-fst by simp
then show \( \exists sch. \ c-fst = \text{sch-to-pr} \ bar{sch} \) by (rule exI)
next
have c-snd = sch-to-pr Base-snd by simp
then show \( \exists sch. \ c-snd = \text{sch-to-pr} \ bar{sch} \) by (rule exI)
next
fix \( f1\ f2 \) assume B1: \( \exists sch. \ f1 = \text{sch-to-pr} \ bar{sch} \) and B2: \( \exists sch. \ f2 = \text{sch-to-pr} \ bar{sch} \)
from B1 obtain \( \bar{sch}1 \) where S1: \( f1 = \text{sch-to-pr} \ bar{sch}1 \)
from B2 obtain \( \bar{sch}2 \) where S2: \( f2 = \text{sch-to-pr} \ bar{sch}2 \)
from S1 S2 have \((\lambda x. f1 (f2 x)) = \text{sch-to-pr} (\text{Comp-op} \ bar{sch}1 \ bar{sch}2)\) by simp
then show \( \exists sch. (\lambda x. f1 (f2 x)) = \text{sch-to-pr} \ bar{sch} \) by (rule exI)
next
\begin{verbatim}
fix f1 f2 assume B1: \exists sch. f1 = sch-to-pr sch and B2: \exists sch. f2 = sch-to-pr sch
from B1 obtain sch1 where S1: f1 = sch-to-pr sch1 ..
from B2 obtain sch2 where S2: f2 = sch-to-pr sch2 ..
from S1 S2 have c-f-pair f1 f2 = sch-to-pr (Pair-op sch1 sch2) by simp
then show \exists sch. c-f-pair f1 f2 = sch-to-pr sch by (rule exI)
next
fix f1 f2 assume B1: \exists sch. f1 = sch-to-pr sch and B2: \exists sch. f2 = sch-to-pr sch
from B1 obtain sch1 where S1: f1 = sch-to-pr sch1 ..
from B2 obtain sch2 where S2: f2 = sch-to-pr sch2 ..
from S1 S2 have UnaryRecOp f1 f2 = sch-to-pr (Rec-op sch1 sch2) by simp
then show \exists sch. UnaryRecOp f1 f2 = sch-to-pr sch by (rule exI)
qed

definition
loc-f :: nat ⇒ PrimScheme ⇒ PrimScheme ⇒ PrimScheme where
loc-f n sch1 sch2 =
(if n=0 then Base-zero else
 if n=1 then Base-suc else
 if n=2 then Base-fst else
 if n=3 then Base-snd else
 if n=4 then (Comp-op sch1 sch2) else
 if n=5 then (Pair-op sch1 sch2) else
 if n=6 then (Rec-op sch1 sch2) else
 Base-zero)

definition
mod7 :: nat ⇒ nat where
mod7 = (λ x. x mod 7)

lemma c-snd-snd-lt [termination-simp]: c-snd (c-snd (Suc (Suc x))) < Suc (Suc x)
proof –
let ?y = Suc (Suc x)
have ?y > 1 by simp
then have c-snd ?y < ?y by (rule c-snd-less-arg)
moreover have c-snd (c-snd ?y) ≤ c-snd ?y by (rule c-snd-le-arg)
ultimately show ?thesis by simp
qed

lemma c-fst-snd-snd-lt [termination-simp]: c-fst (c-snd (Suc (Suc x))) < Suc (Suc x)
proof –
let ?y = Suc (Suc x)
have ?y > 1 by simp
then have c-snd ?y < ?y by (rule c-snd-less-arg)
moreover have c-fst (c-snd ?y) ≤ c-snd ?y by (rule c-fst-le-arg)
ultimately show ?thesis by simp

\end{verbatim}
fun nat-to-sch :: nat ⇒ PrimScheme where
  nat-to-sch 0 = Base-zero
| nat-to-sch (Suc 0) = Base-zero
| nat-to-sch x = (let u=mod7 (c-fst x); v=c-snd x; v1=c-fst v; v2 = c-snd v;
  sch1=nat-to-sch v1; sch2=nat-to-sch v2 in loc-f u sch1 sch2)

primrec sch-to-nat :: PrimScheme ⇒ nat where
  sch-to-nat Base-zero = 0
| sch-to-nat Base-suc = c-pair 1 0
| sch-to-nat Base-fst = c-pair 2 0
| sch-to-nat Base-snd = c-pair 3 0
| sch-to-nat (Comp-op t1 t2) = c-pair 4 (c-pair (sch-to-nat t1) (sch-to-nat t2))
| sch-to-nat (Pair-op t1 t2) = c-pair 5 (c-pair (sch-to-nat t1) (sch-to-nat t2))
| sch-to-nat (Rec-op t1 t2) = c-pair 6 (c-pair (sch-to-nat t1) (sch-to-nat t2))

lemma loc-srj-lm-1: nat-to-sch ((Suc (Suc x))) = (let u=mod7 (c-fst (Suc (Suc x))); v=c-snd (Suc (Suc x)); v1=c-fst v; v2 = c-snd v; sch1=nat-to-sch v1; sch2=nat-to-sch v2 in loc-f u sch1 sch2) by simp

lemma loc-srj-lm-2: x > 1 ⇒ nat-to-sch x = (let u=mod7 (c-fst x); v=c-snd x; v1=c-fst v; v2 = c-snd v; sch1=nat-to-sch v1; sch2=nat-to-sch v2 in loc-f u sch1 sch2)

proof
  assume A1: x > 1
  let ?y = x-(2::nat)
  from A1 have S1: x = Suc (Suc ?y) by arith
  have S2: nat-to-sch (Suc (Suc ?y)) = (let u=mod7 (c-fst (Suc (Suc ?y))); v=c-snd (Suc (Suc ?y)); v1=c-fst v; v2 = c-snd v; sch1=nat-to-sch v1; sch2=nat-to-sch v2 in loc-f u sch1 sch2) by (rule loc-srj-lm-1)
  from S1 S2 show ?thesis by simp
qed

lemma loc-srj-0: nat-to-sch (c-pair 1 0) = Base-suc

proof
  let ?x = c-pair 1 0
  have S1: ?x = 2 by (simp add: c-pair-def sf-def)
  then have S2: ?x = Suc (Suc 0) by simp
  let ?y = Suc (Suc 0)
  have S3: nat-to-sch ?y = (let u=mod7 (c-fst ?y); v=c-snd ?y; v1=c-fst v; v2 = c-snd v; sch1=nat-to-sch v1; sch2=nat-to-sch v2 in loc-f u sch1 sch2) (ls - = ?R) by (rule loc-srj-lm-1)
  have S4: c-fst ?y = 1
  proof
    from S2 have c-fst ?y = c-fst ?x by simp
    then show ?thesis by simp
  qed
  have S5: c-snd ?y = 0

qed
proof
  from S2 have c-snd ?y = c-snd ?x by simp
  then show ?thesis by simp
qed
from S4 have S6: mod7 (c-fst ?y) = 1 by (simp add: mod7-def)
from S3 S5 S6 have S9: ?R = loc-f 1 Base-zero Base-zero by (simp add: Let-def c-fst-at-0 c-snd-at-0)
  then have S10: ?R = Base-suc by (simp add: loc-f-def)
with S3 have S11: nat-to-sch ?y = Base-suc by simp
from S2 this show ?thesis by simp
qed

lemma nat-to-sch-at-2: nat-to-sch 2 = Base-suc
proof
  have S1: c-pair 1 0 = 2 by (simp add: c-pair-def sf-def)
  have S2: nat-to-sch (c-pair 1 0) = Base-suc by (rule loc-srj-0)
  from S1 S2 show ?thesis by simp
qed

lemma loc-srj-1: nat-to-sch (c-pair 2 0) = Base-fst
proof
  let ?x = c-pair 2 0
  have S1: ?x = 5 by (simp add: c-pair-def sf-def)
  then have S2: ?x = Suc (Suc 3) by simp
  let ?y = Suc (Suc 3)
  have S3: nat-to-sch ?y = (let u=mod7 (c-fst ?y); v=c-snd ?y; v1=c-fst v; v2 = c-snd v; sch1=nat-to-sch v1; sch2=nat-to-sch v2 in loc-f u sch1 sch2) (is = ?R) by (rule loc-srj-lm-1)
  have S4: c-fst ?y = 2
  proof
    from S2 have c-fst ?y = c-fst ?x by simp
    then show ?thesis by simp
  qed
  have S5: c-snd ?y = 0
  proof
    from S2 have c-snd ?y = c-snd ?x by simp
    then show ?thesis by simp
  qed
  from S4 have S6: mod7 (c-fst ?y) = 2 by (simp add: mod7-def)
  from S3 S5 S6 have S9: ?R = loc-f 2 Base-zero Base-zero by (simp add: Let-def c-fst-at-0 c-snd-at-0)
  then have S10: ?R = Base-fst by (simp add: loc-f-def)
  with S3 have S11: nat-to-sch ?y = Base-fst by simp
  from S2 this show ?thesis by simp
qed

lemma loc-srj-2: nat-to-sch (c-pair 3 0) = Base-snd
proof
  let ?x = c-pair 3 0
have $\text{S1: } ?x > 1$ by (simp add: c-pair-def sf-def)

from $\text{S1 have S2: } \text{nat-to-sch } ?x = (\text{let } u = \text{mod7 } (\text{c-fst } ?x); v = \text{c-snd } ?x; v1 = \text{c-fst } v; v2 = \text{c-snd } v; \text{sch1 = nat-to-sch } v1; \text{sch2 = nat-to-sch } v2 \text{ in loc-f } u \text{ sch1 sch2})$ (is - = ?R) by (rule loc-srj-lm-2)

have $\text{S3: } \text{c-fst } ?x = 3$ by simp

have $\text{S4: } \text{c-snd } ?x = 0$ by simp

from $\text{S3 have S6: } \text{mod7 } (\text{c-fst } ?x) = 3$ by (simp add: mod7-def)

from $\text{S3 S4 have S7: } \text{?R = Loc-f 3 Base-zero Base-zero}$ by (simp add: Let-def c-fst-at-0 c-snd-at-0)

then have $\text{S8: } \text{?R = Base-snd}$ by (simp add: loc-f-def)

with $\text{S2 have S10: } \text{nat-to-sch } ?x = \text{Base-snd}$ by simp

from $\text{S2 this show } \text{thesis}$ by simp

qed

lemma loc-srj-3: \[
\text{nat-to-sch } \text{(sch-to-nat sch1) = sch1; nat-to-sch } \text{(sch-to-nat sch2) = sch2}\]

\[
\implies \text{nat-to-sch } \text{(c-pair } 4 \text{ (c-pair } \text{(sch-to-nat sch1) (sch-to-nat sch2)}) = \text{Comp-op sch1 sch2}}
\]

proof

- assume $\text{A1: } \text{nat-to-sch } \text{(sch-to-nat sch1) = sch1}$
- assume $\text{A2: } \text{nat-to-sch } \text{(sch-to-nat sch2) = sch2}$
- let $\text{?x = c-pair } 4 \text{ (c-pair } \text{(sch-to-nat sch1) (sch-to-nat sch2)})$

then have $\text{S1: } \text{?x > 1}$ by (simp add: c-pair-def sf-def)

from $\text{S1 have S2: } \text{nat-to-sch } ?x = (\text{let } u = \text{mod7 } (\text{c-fst } ?x); v = \text{c-snd } ?x; v1 = \text{c-fst } v; v2 = \text{c-snd } v; \text{sch1 = nat-to-sch } v1; \text{sch2 = nat-to-sch } v2 \text{ in loc-f } u \text{ sch1 sch2})$ (is - = ?R) by (rule loc-srj-lm-2)

have $\text{S3: } \text{c-fst } ?x = 4$ by simp

have $\text{S4: } \text{c-snd } ?x = \text{c-pair } n1 n2$ by simp

from $\text{S3 have S5: } \text{mod7 } (\text{c-fst } ?x) = 4$ by (simp add: mod7-def)

from $\text{S4 S5 have ?R = Comp-op } \text{(nat-to-sch n1) (nat-to-sch n2)}$ by (simp add: Let-def c-fst-at-0 c-snd-at-0 loc-f-def)

then have $\text{S8: } \text{?thesis}$ by simp

qed

lemma loc-srj-3-1: \[
\text{nat-to-sch } \text{(c-pair } 4 \text{ (c-pair } n1 n2\text{)) = Comp-op } \text{(nat-to-sch n1) (nat-to-sch n2)}
\]

proof

- let $\text{?x = c-pair } 4 \text{ (c-pair } n1 n2\text{)}$

then have $\text{S1: } \text{?x > 1}$ by (simp add: c-pair-def sf-def)

from $\text{S1 have S2: } \text{nat-to-sch } ?x = (\text{let } u = \text{mod7 } (\text{c-fst } ?x); v = \text{c-snd } ?x; v1 = \text{c-fst } v; v2 = \text{c-snd } v; \text{sch1 = nat-to-sch } v1; \text{sch2 = nat-to-sch } v2 \text{ in loc-f } u \text{ sch1 sch2})$ (is - = ?R) by (rule loc-srj-lm-2)

have $\text{S3: } \text{c-fst } ?x = 4$ by simp

have $\text{S4: } \text{c-snd } ?x = \text{c-pair } n1 n2$ by simp

from $\text{S3 have S5: } \text{mod7 } (\text{c-fst } ?x) = 4$ by (simp add: mod7-def)

from $\text{S4 S5 have ?R = Comp-op } \text{(nat-to-sch n1) (nat-to-sch n2)}$ by (simp add: Let-def c-fst-at-0 c-snd-at-0 loc-f-def)

then have $\text{S8: } \text{?thesis}$ by simp

qed
lemma loc-srj-4: \( \text{nat-to-sch (sch-to-nat sch1) = sch1; nat-to-sch (sch-to-nat sch2) = sch2} \) implies \( \text{nat-to-sch (c-pair 5 (c-pair (sch-to-nat sch1) (sch-to-nat sch2))) = Pair-op sch1 sch2} \)

proof -
  assume A1: \( \text{nat-to-sch (sch-to-nat sch1) = sch1} \)
  assume A2: \( \text{nat-to-sch (sch-to-nat sch2) = sch2} \)
  let \( ?x = \text{c-pair 5 (c-pair (sch-to-nat sch1) (sch-to-nat sch2))} \)
  have S1: \( \forall x > 1 \) by (simp add: c-pair-def sf-def)
  from S1 have S2: \( \text{nat-to-sch} \ ?x = (\text{let u}=mod7 (\text{c-fst} \ ?x); v=\text{c-snd} \ ?x; v1=\text{c-fst} v; v2=\text{c-snd} v; \text{sch1}=\text{nat-to-sch} v1; \text{sch2}=\text{nat-to-sch} v2 \ (\text{in loc-f} u \ \text{sch1 sch2}) \ (\text{is - = ?R}) \) by (rule loc-srj-lm-2)
  have S3: \( \text{c-fst} \ ?x = 5 \) by simp
  have S4: \( \text{c-snd} \ ?x = \text{c-pair (sch-to-nat sch1) (sch-to-nat sch2)} \) by simp
  from S3 have S5: \( \text{mod7 (c-fst} \ ?x) = 5 \) by (simp add: mod7-def)
  from A1 A2 S4 S5 have \( \text{?R} = \text{Pair-op sch1 sch2} \) by (simp add: Let-def c-fst-at-0 c-snd-at-0 loc-f-def)
  with S2 show \( \text{thesis} \) by simp
qed

lemma loc-srj-4-1: \( \text{nat-to-sch (c-pair 5 (c-pair n1 n2)) = Pair-op (nat-to-sch n1) (nat-to-sch n2)} \)

proof -
  let \( ?x = \text{c-pair 5 (c-pair n1 n2)} \)
  have S1: \( \forall x > 1 \) by (simp add: c-pair-def sf-def)
  from S1 have S2: \( \text{nat-to-sch} \ ?x = (\text{let u}=\text{mod7} (\text{c-fst} \ ?x); v=\text{c-snd} \ ?x; v1=\text{c-fst} v; v2=\text{c-snd} v; \text{sch1}=\text{nat-to-sch} v1; \text{sch2}=\text{nat-to-sch} v2 \ (\text{in loc-f} u \ \text{sch1 sch2}) \ (\text{is - = ?R}) \) by (rule loc-srj-lm-2)
  have S3: \( \text{c-fst} \ ?x = 5 \) by simp
  have S4: \( \text{c-snd} \ ?x = \text{c-pair n1 n2} \) by simp
  from S3 have S5: \( \text{mod7 (c-fst} \ ?x) = 5 \) by (simp add: mod7-def)
  from S4 S5 have \( \text{?R} = \text{Pair-op (nat-to-sch n1) (nat-to-sch n2)} \) by (simp add: Let-def c-fst-at-0 c-snd-at-0 loc-f-def)
  with S2 show \( \text{thesis} \) by simp
qed

lemma loc-srj-5: \( \text{nat-to-sch (sch-to-nat sch1) = sch1; nat-to-sch (sch-to-nat sch2) = sch2} \) implies \( \text{nat-to-sch (c-pair 6 (c-pair (sch-to-nat sch1) (sch-to-nat sch2))) = Rec-op sch1 sch2} \)

proof -
  assume A1: \( \text{nat-to-sch (sch-to-nat sch1) = sch1} \)
  assume A2: \( \text{nat-to-sch (sch-to-nat sch2) = sch2} \)
  let \( ?x = \text{c-pair 6 (c-pair (sch-to-nat sch1) (sch-to-nat sch2))} \)
  have S1: \( \forall x > 1 \) by (simp add: c-pair-def sf-def)
  from S1 have S2: \( \text{nat-to-sch} \ ?x = (\text{let u}=\text{mod7} (\text{c-fst} \ ?x); v=\text{c-snd} \ ?x; v1=\text{c-fst} v; v2=\text{c-snd} v; \text{sch1}=\text{nat-to-sch} v1; \text{sch2}=\text{nat-to-sch} v2 \ (\text{in loc-f} u \ \text{sch1 sch2}) \ (\text{is - = ?R}) \) by (rule loc-srj-lm-2)
have S3: \( c\text{-fst }?x = 6 \) by simp
have S4: \( c\text{-snd }?x = \text{c-pair } (\text{sch-to-nat } \text{sch1}) (\text{sch-to-nat } \text{sch2}) \) by simp
from S3 have S5: \( \text{mod7 } (c\text{-fst }?x) = 6 \) by (simp add: \text{mod7-def})
from A1 A2 S4 S5 have \( ?R = \text{Rec-op } \text{sch1 } \text{sch2} \) by (simp add: \text{Let-def } c\text{-fst-at-0 } c\text{-snd-at-0 } \text{loc-f-def})
with S2 show \( ?\text{thesis} \) by simp
qed

lemma \text{loc-srj-5-1}: \( \text{nat-to-sch } (\text{c-pair } 6 (\text{c-pair } n1 n2)) = \text{Rec-op } (\text{nat-to-sch } n1) (\text{nat-to-sch } n2) \)
proof –
let \( ?x = \text{c-pair } 6 (\text{c-pair } n1 n2) \)
have S1: \( ?x > 1 \) by (simp add: \text{c-pair-def } \text{sf-def})
from S1 have S2: \( \text{nat-to-sch } ?x = (\text{let } u=\text{mod7 } (c\text{-fst }?x); v=\text{c-snd }?x; v1=\text{c-fst } v; v2 = \text{c-snd } v; \text{sch1}=\text{nat-to-sch } v1; \text{sch2}=\text{nat-to-sch } v2 \text{ in loc-f } u \text{ sch1 sch2}) \) (is \( - = ?R \)) by (rule \text{loc-srj-lm-2})

have S3: \( c\text{-fst }?x = 6 \) by simp
have S4: \( c\text{-snd }?x = \text{c-pair } n1 n2 \) by simp
from S3 have S5: \( \text{mod7 } (c\text{-fst }?x) = 6 \) by (simp add: \text{mod7-def})
from S4 S5 have \( ?R = \text{Rec-op } (\text{nat-to-sch } n1) (\text{nat-to-sch } n2) \) by (simp add: \text{Let-def } c\text{-fst-at-0 } c\text{-snd-at-0 } \text{loc-f-def})
with S2 show \( ?\text{thesis} \) by simp
qed

theorem \text{nat-to-sch-srj}: \( \text{nat-to-sch } (\text{sch-to-nat } \text{sch}) = \text{sch} \)
apply(induct \text{sch}, auto simp add: \text{loc-srj-0 } \text{loc-srj-1 } \text{loc-srj-2 } \text{loc-srj-3 } \text{loc-srj-4 } \text{loc-srj-5})
apply(insert \text{loc-srj-0})
apply(simp)
done

4.3 Indexes of primitive recursive functions of one variables

definition
\( \text{nat-to-pr } :: \text{nat } \Rightarrow (\text{nat } \Rightarrow \text{nat}) \text{ where} \)
\( \text{nat-to-pr } = (\lambda x. \text{sch-to-pr } (\text{nat-to-sch } x)) \)

theorem \text{nat-to-pr-into-pr}: \( \text{nat-to-pr } n \in \text{PrimRec1} \) by (simp add: \text{nat-to-pr-def } \text{sch-to-pr-into-pr})

lemma \text{nat-to-pr-srj}: \( f \in \text{PrimRec1} \Rightarrow (\exists n. f = \text{nat-to-pr } n) \)
proof –
assume \( f \in \text{PrimRec1} \)
then have S1: \( (\exists t. f = \text{sch-to-pr } t) \) by (rule \text{sch-to-pr-srj})
from S1 obtain \( t \) where S2: \( f = \text{sch-to-pr } t \).
let \( ?n = \text{sch-to-nat } t \)
have S3: \( \text{nat-to-pr } ?n = \text{sch-to-pr } (\text{nat-to-sch } ?n) \) by (simp add: \text{nat-to-pr-def})
have S4: \( \text{nat-to-sch } ?n = t \) by (rule \text{nat-to-sch-srj})
from S3 S4 have S5: \( \text{nat-to-pr } ?n = \text{sch-to-pr } t \) by simp
from S2 S5 have nat-to-pr ?n = f by simp
then have f = nat-to-pr ?n by simp
then show ?thesis ..
qed

lemma nat-to-pr-at-0: nat-to-pr 0 = (λ x. 0) by (simp add: nat-to-pr-def)
definition index-of-pr :: (nat ⇒ nat) ⇒ nat where
index-of-pr f = (SOME n. f = nat-to-pr n)
theorem index-of-pr-is-real: f ∈ PrimRec1 ⇒ nat-to-pr (index-of-pr f) = f
proof –
assume f ∈ PrimRec1
hence ∃ n. f = nat-to-pr n by (rule nat-to-pr-srj)
hence f = nat-to-pr (SOME n. f = nat-to-pr n) by (rule someI-ex)
thus ?thesis by (simp add: index-of-pr-def)
qed

definition comp-by-index :: nat ⇒ nat ⇒ nat where
comp-by-index = (λ n1 n2. c-pair 4 (c-pair n1 n2))
definition pair-by-index :: nat ⇒ nat ⇒ nat where
pair-by-index = (λ n1 n2. c-pair 5 (c-pair n1 n2))
definition rec-by-index :: nat ⇒ nat ⇒ nat where
rec-by-index = (λ n1 n2. c-pair 6 (c-pair n1 n2))
lemma comp-by-index-is-pr: comp-by-index ∈ PrimRec2
unfolding comp-by-index-def
using const-is-pr-2 [of 4] by prec
lemma comp-by-index-inj: comp-by-index x1 y1 = comp-by-index x2 y2 ⇒ x1=x2 ∧ y1=y2
proof –
assume comp-by-index x1 y1 = comp-by-index x2 y2
hence c-pair 4 (c-pair x1 y1) = c-pair 4 (c-pair x2 y2) by (unfold comp-by-index-def)
hence c-pair x1 y1 = c-pair x2 y2 by (rule c-pair-inj2)
thus ?thesis by (rule c-pair-inj)
qed

lemma comp-by-index-inj1: comp-by-index x1 y1 = comp-by-index x2 y2 ⇒ x1 = x2 by (frule comp-by-index-inj, drule conjunct1)
lemma comp-by-index-inj2: comp-by-index x1 y1 = comp-by-index x2 y2 ⇒ y1 = y2 by (frule comp-by-index-inj, drule conjunct2)
lemma comp-by-index-main: nat-to-pr (comp-by-index n1 n2) = (λ x. (nat-to-pr n1) ((nat-to-pr n2) x)) by (unfold comp-by-index-def, unfold nat-to-pr-def, simp add: loc-srj-3-1)

lemma pair-by-index-is-pr: pair-by-index ∈ PrimRec2 by (unfold pair-by-index-def, insert const-is-pr-2 [where ?n=5\nat], prec)

lemma pair-by-index-inj: pair-by-index x1 y1 = pair-by-index x2 y2 ⇒ x1=x2 ∧ y1=y2
  proof –
  assume pair-by-index x1 y1 = pair-by-index x2 y2
  hence c-pair 5 (c-pair x1 y1) = c-pair 5 (c-pair x2 y2) by (unfold pair-by-index-def)
  hence c-pair x1 y1 = c-pair x2 y2 by (rule c-pair-inj2)
  thus ?thesis by (rule c-pair-inj)
  qed

lemma pair-by-index-inj1: pair-by-index x1 y1 = pair-by-index x2 y2 ⇒ x1=x2 by (frule pair-by-index-inj, drule conjunct1)

lemma pair-by-index-inj2: pair-by-index x1 y1 = pair-by-index x2 y2 ⇒ y1=y2 by (frule pair-by-index-inj, drule conjunct2)

lemma pair-by-index-main: nat-to-pr (pair-by-index n1 n2) = c-f-pair (nat-to-pr n1) (nat-to-pr n2) by (unfold pair-by-index-def, unfold nat-to-pr-def, simp add: loc-srj-4-1)

lemma nat-to-sch-of-pair-by-index [simp]: nat-to-sch (pair-by-index n1 n2) = Pair-op (nat-to-sch n1) (nat-to-sch n2) by (simp add: pair-by-index-def loc-srj-4-1)

lemma rec-by-index-is-pr: rec-by-index ∈ PrimRec2 by (unfold rec-by-index-def, insert const-is-pr-2 [where ?n=6\nat], prec)

lemma rec-by-index-inj: rec-by-index x1 y1 = rec-by-index x2 y2 ⇒ x1=x2 ∧ y1=y2
  proof –
  assume rec-by-index x1 y1 = rec-by-index x2 y2
  hence c-pair 6 (c-pair x1 y1) = c-pair 6 (c-pair x2 y2) by (unfold rec-by-index-def)
  hence c-pair x1 y1 = c-pair x2 y2 by (rule c-pair-inj2)
  thus ?thesis by (rule c-pair-inj)
  qed

lemma rec-by-index-inj1: rec-by-index x1 y1 = rec-by-index x2 y2 ⇒ x1=x2 by (frule rec-by-index-inj, drule conjunct1)

lemma rec-by-index-inj2: rec-by-index x1 y1 = rec-by-index x2 y2 ⇒ y1=y2 by (frule rec-by-index-inj, drule conjunct2)
**lemma** \( \text{rec-by-index-main}: \text{nat-to-pr} \ (\text{rec-by-index} \ n1 \ n2) = \text{UnaryRecOp} \ (\text{nat-to-pr} \ n1) \ (\text{nat-to-pr} \ n2) \)** by \( \text{unfold rec-by-index-def, unfold nat-to-pr-def, simp add: loc-srj-5-1} \)

4.4 \( \text{s-1-1 theorem for primitive recursive functions of one variable} \)

definition

\( \text{index-of-const} :: \text{nat} \Rightarrow \text{nat} \)

where

\( \text{index-of-const} = \text{PrimRecOp1} 0 (\lambda x. y. \text{c-pair} \ 4 \ (\text{c-pair} \ 2 \ y)) \)

**lemma** \( \text{index-of-const-is-pr}: \text{index-of-const} \in \text{PrimRec1} \)

proof –

\( \text{have} \ (\lambda x. y. \text{c-pair} \ (4 :: \text{nat}) \ (\text{c-pair} \ (2 :: \text{nat}) \ y)) \in \text{PrimRec2} \) by \( \text{insert const-is-pr-2 [where n=(4 :: \text{nat})], prec} \)

then show \( \text{?thesis} \) by \( \text{simp add: index-of-const-def pr-rec1} \)

qed

**lemma** \( \text{index-of-const-at-0}: \text{index-of-const} \ 0 = 0 \) by \( \text{simp add: index-of-const-def} \)

**lemma** \( \text{index-of-const-at-suc}: \text{index-of-const} \ (\text{Suc} \ u) = \text{c-pair} \ 4 \ (\text{c-pair} \ 2 \ (\text{index-of-const} \ u)) \) by \( \text{unfold index-of-const-def, induct u, auto} \)

**lemma** \( \text{index-of-const-main}: \text{nat-to-pr} \ (\text{index-of-const} \ n) = (\lambda x. n) \) (is \?P \( n \))

proof \( \text{induct n} \)

\( \text{show} \ ?P \ 0 \) by \( \text{simp add: index-of-const-at-0 nat-to-pr-at-0} \)

next

\( \text{fix} \ n \ \text{assume} \ ?P \ n \)

then show \( ?P \ (\text{Suc} \ n) \) by \( ((\text{simp add: index-of-const-at-suc nat-to-sch-at-2 nat-to-pr-def loc-srj-3-1}) \)

qed

**lemma** \( \text{index-of-const-lm-1}: \ (\text{nat-to-pr} \ (\text{index-of-const} \ n)) \ 0 = n \) by \( \text{simp add: index-of-const-main} \)

**lemma** \( \text{index-of-const-inj}: \text{index-of-const} \ n1 = \text{index-of-const} \ n2 \Rightarrow n1 = n2 \)

proof –

\( \text{assume} \ \text{index-of-const} \ n1 = \text{index-of-const} \ n2 \)

then have \( \ (\text{nat-to-pr} \ (\text{index-of-const} \ n1)) \ 0 = (\text{nat-to-pr} \ (\text{index-of-const} \ n2)) \)

\( \text{0 by simp} \)

thus \( \text{?thesis} \) by \( \text{simp add: index-of-const-lm-1} \)

qed

**definition** \( \text{index-of-zero} = \text{sch-to-nat} \ Base-zero \)

**definition** \( \text{index-of-suc} = \text{sch-to-nat} \ Base-suc \)

**definition** \( \text{index-of-c-fst} = \text{sch-to-nat} \ Base-fst \)

**definition** \( \text{index-of-c-snd} = \text{sch-to-nat} \ Base-snd \)

**definition** \( \text{index-of-id} = \text{pair-by-index} \ \text{index-of-c-fst} \ \text{index-of-c-snd} \)

69
lemma index-of-zero-main: nat-to-pr index-of-zero = (λ x. 0) by (simp add: index-of-zero-def nat-to-pr-def)

lemma index-of-suc-main: nat-to-pr index-of-suc = Suc
apply (simp add: index-of-suc-def nat-to-pr-def)
apply (insert loc-srj-0)
apply (simp)
done

lemma index-of-c-fst-main: nat-to-pr index-of-c-fst = c-fst by (simp add: index-of-c-fst-def nat-to-pr-def loc-srj-1)

lemma [simp]: nat-to-sch index-of-c-fst = Base-fst by (unfold index-of-c-fst-def, rule nat-to-sch-srj)


lemma [simp]: nat-to-sch index-of-c-snd = Base-snd by (unfold index-of-c-snd-def, rule nat-to-sch-srj)

lemma index-of-id-main: nat-to-pr index-of-id = (λ x. x) by (simp add: index-of-id-def nat-to-pr-def c-f-pair-def)

definition index-of-c-pair-n :: nat ⇒ nat where
index-of-c-pair-n = (λ n. pair-by-index (index-of-const n) index-of-id)

lemma index-of-c-pair-n-is-pr: index-of-c-pair-n ∈ PrimRec1
proof −
  have (λ x. index-of-id) ∈ PrimRec1 by (rule const-is-pr)
  with pair-by-index-is-pr index-of-const-is-pr have (λ n. pair-by-index (index-of-const n) index-of-id) ∈ PrimRec1 by prec
  then show ?thesis by (fold index-of-c-pair-n-def)
qed

lemma index-of-c-pair-n-main: nat-to-pr (index-of-c-pair-n n) = (λ x. c-pair n x)
proof −
  have nat-to-pr (index-of-c-pair-n n) = nat-to-pr (pair-by-index (index-of-const n) index-of-id) by (simp add: index-of-c-pair-n-def)
  also have ... = c-f-pair (nat-to-pr (index-of-const n)) (nat-to-pr index-of-id) by (simp add: pair-by-index-main)
  also have ... = c-f-pair (λ x. n) (λ x. x) by (simp add: index-of-const-main index-of-id-main)
  finally show ?thesis by (simp add: c-f-pair-def)
qed

lemma index-of-c-pair-n-inj: index-of-c-pair-n x1 = index-of-c-pair-n x2 ⇒ x1 = x2
proof −
assume \( \text{index-of-c-pair-n } x_1 = \text{index-of-c-pair-n } x_2 \)

\( \text{hence } \text{pair-by-index } \left( \text{index-of-const } x_1 \right) \text{index-of-id } = \text{pair-by-index } \left( \text{index-of-const } x_2 \right) \text{index-of-id} \) by (unfold \( \text{index-of-c-pair-n-def} \))

\( \text{hence } \text{index-of-const } x_1 = \text{index-of-const } x_2 \) by (rule \text{pair-by-index-inj1})

thus \( \text{?thesis} \) by (rule \text{index-of-const-inj})

qed

definition
s1-I :: \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat}
where
s1-I = (\lambda n x . \text{comp-by-index } n \left( \text{index-of-c-pair-n } x \right))

lemma s1-I-is-pr: \( s1-I \in \text{PrimRec2} \) by (unfold \( s1-I-def \), insert \text{comp-by-index-is-pr} \text{index-of-c-pair-n-is-pr}, \text{prec})

theorem s1-I-th: \( \left( \lambda y . \left( \text{nat-to-pr } n \right) \left( \text{c-pair } x y \right) \right) = \text{nat-to-pr } \left( s1-I n x \right) \)
proof
 have \( \text{nat-to-pr } \left( s1-I n x \right) = \text{nat-to-pr } \left( \text{comp-by-index } n \left( \text{index-of-c-pair-n } x \right) \right) \) by (simp add: \( s1-I-def \))
 also have \( \ldots = \left( \lambda z . \left( \text{nat-to-pr } n \right) \left( \left( \text{nat-to-pr } \text{index-of-c-pair-n } x \right) z \right) \right) \) by (simp add: \text{comp-by-index-main})
 also have \( \ldots = \left( \lambda z . \left( \text{nat-to-pr } n \right) \left( \left( \lambda u . \text{c-pair } x u \right) z \right) \right) \) by (simp add: \text{index-of-c-pair-n-main})
finally show \( \text{?thesis} \) by simp
qed

lemma s1-I-inj: \( s1-I \ x_1 \ y_1 = s1-I \ x_2 \ y_2 \Rightarrow x_1=x_2 \land y_1=y_2 \)
proof
 assume \( s1-I \ x_1 \ y_1 = s1-I \ x_2 \ y_2 \)
 then have \( \text{comp-by-index } x_1 \left( \text{index-of-c-pair-n } y_1 \right) = \text{comp-by-index } x_2 \left( \text{index-of-c-pair-n } y_2 \right) \) by (unfold \( s1-I-def \))
 then have \( S1: x_1=x_2 \land \text{index-of-c-pair-n } y_1 = \text{index-of-c-pair-n } y_2 \) by (rule \text{comp-by-index-inj})
 then have \( S2: x_1=x_2 \ldots \)
 from \( S1 \) have \( \text{index-of-c-pair-n } y_1 = \text{index-of-c-pair-n } y_2 \ldots \)
 then have \( y_1 = y_2 \) by (rule \text{index-of-c-pair-n-inj})
 with \( S2 \) show \( \text{?thesis} \) \ldots 
qed

lemma s1-I-inj1: \( s1-I \ x_1 \ y_1 = s1-I \ x_2 \ y_2 \Rightarrow x_1=x_2 \) by (frule \( s1-I-inj \), drule \text{conjunct1})

lemma s1-I-inj2: \( s1-I \ x_1 \ y_1 = s1-I \ x_2 \ y_2 \Rightarrow y_1=y_2 \) by (frule \( s1-I-inj \), drule \text{conjunct2})

primrec
pr-index-enumerator :: \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat}
where
pr-index-enumerator \( n \ 0 \ = n \)
| pr-index-enumerator \( n \ (\text{Suc } m) \) = \text{comp-by-index } \text{index-of-id} \ (\text{pr-index-enumerator} \ n \ m)
n m)

**Theorem** \( \text{pr-index-enumerator-is-pr} \): \( \text{pr-index-enumerator} \in \text{PrimRec2} \)

**Proof**
- **def** \( \text{g-def} \): \( g \equiv \lambda (x::\text{nat}). x \)
- **have** \( \text{g-is-pr} \): \( g \in \text{PrimRec1} \) by \((\text{unfold g-def, rule pr-id1-1})\)
- **def** \( \text{h-def} \): \( h \equiv \lambda (a::\text{nat}) b (c::\text{nat}). \text{comp-by-index index-of-id } b \)
- **from** \( \text{comp-by-index-is-pr} \) **have** \( \text{h-is-pr} \): \( h \in \text{PrimRec3} \) \((\text{unfolding h-def by prec})\)
- let \( ?f \equiv \text{pr-index-enumerator} \)
- **have** \( \text{f-at-0} \): \( \forall x. ?f x 0 = g x \) by \text{auto}
- **from** \( \text{h-def} \) **have** \( \text{f-at-Suc} \): \( \forall x y. \text{comp-by-index index-of-id } ?f x y \) \( \text{Suc y} \) \( \text{by} \) \text{auto}
- **from** \( \text{g-is-pr} \) \( \text{h-is-pr} \) \( \text{f-at-0} \) \( \text{f-at-Suc} \) **show** \( \text{?thesis} \) by \((\text{rule pr-rec-last-scheme})\)

**Lemma** \( \text{pr-index-enumerator-increase1} \): \( \text{pr-index-enumerator } n \text{ m } < \text{pr-index-enumerator } (n+1) \text{ m} \)

**Proof** \((\text{induct m})\)
- **show** \( \text{pr-index-enumerator } n \text{ 0 } < \text{pr-index-enumerator } (n+1) \text{ 0} \) by \text{simp}
- **next fix** \( \text{na} \) **assume** \( A \): \( \text{pr-index-enumerator } n \text{ na } < \text{pr-index-enumerator } (n + 1) \text{ na} \)
- **show** \( \text{pr-index-enumerator } n \text{ (Suc na) } < \text{pr-index-enumerator } (n + 1) \text{ (Suc na)} \)
- **proof**
  - let \( ?a = \text{pr-index-enumerator } n \text{ na} \)
  - let \( ?b = \text{pr-index-enumerator } (n+1) \text{ na} \)
  - **have** \( S1 \): \( \text{pr-index-enumerator } n \text{ (Suc na) } = \text{comp-by-index index-of-id } ?a \) by \text{simp}
  - **have** \( S2 \): \( \text{comp-by-index index-of-id } ?a < \text{c-pair index-of-id } ?b \) by \((\text{rule c-pair-strict-mono2})\)
  - **then** **have** \( \text{c-pair } 4 \text{ (c-pair index-of-id } ?a) < \text{c-pair } 4 \text{ (c-pair index-of-id } ?b) \) by \((\text{simp add: comp-by-index-def})\)
  - **then** **have** \( \text{comp-by-index index-of-id } ?a < \text{comp-by-index index-of-id } ?b \) by \((\text{simp add: comp-by-index-def})\)
  - **with** \( S1 \) \( L1 \) **show** \( ?\text{thesis} \) by \text{auto}

**Qed**

**Lemma** \( \text{pr-index-enumerator-increase2} \): \( \text{pr-index-enumerator } n \text{ m } < \text{pr-index-enumerator } n \text{ (m + 1)} \)

**Proof**
- let \( ?a = \text{pr-index-enumerator } n \text{ m} \)
- **have** \( S1 \): \( \text{pr-index-enumerator } n \text{ (m + 1) } = \text{comp-by-index index-of-id } ?a \) by \text{simp}
- **have** \( S2 \): \( \text{comp-by-index index-of-id } ?a = \text{c-pair } 4 \text{ (c-pair index-of-id } ?a) \) by \((\text{simp add: comp-by-index-def})\)
- **have** \( S3 \): \( 4 + \text{c-pair index-of-id } ?a \leq \text{c-pair } 4 \text{ (c-pair index-of-id } ?a) \) by \((\text{rule sum-le-c-pair})\)
then have \( S4: \text{c-pair index-of-id } ?a < \text{c-pair } 4 \text{ (c-pair index-of-id } ?a) \text{ by auto} \)

have \( S5: ?a \leq \text{c-pair index-of-id } ?a \text{ by (rule arg2-le-c-pair)} \)

from \( S4 \) \( S5 \) have \( S6: ?a < \text{c-pair } 4 \text{ (c-pair index-of-id } ?a) \text{ by auto} \)

with \( S1 \) \( S2 \) show \( \text{thesis by auto} \)

qed

lemma \( f\text{-inc-mono}: (\forall (x::\text{nat}). (f::\text{nat}\Rightarrow\text{nat}) x < f (x+1)) \implies (\forall (x::\text{nat}) (y::\text{nat}). (x < y \longrightarrow f x < f y)) \)

proof

fix \( x \text{ \( y \) assume } A: \forall (x::\text{nat}). f x < f (x+1) \text{ show } x < y \longrightarrow f x < f y \)

proof

assume \( A1: x < y \)

have \( L1: \forall u v. f u < f (u+(v+1)) \)

proof –

fix \( u \text{ \( v \) show } f u < f (u+(v+1)) \)

proof (induct \( v \))

from \( A \) show \( f u < f (u+(0+1)) \text{ by auto} \)

next

fix \( v \text{ \( n \) assume } A2: f u < f (u+(n+1)) \)

from \( A \) have \( S1: f (u+(n+1)) < f (u+(\text{Suc } n+1)) \text{ by auto} \)

from \( A2 \) \( S1 \) show \( f u < f (u+(\text{Suc } n+1)) \text{ by (rule less-trans)} \)

qed

qed

let \( ?v = (y-x)-1 \)

from \( A1 \) have \( S2: y = x+(?v+1) \text{ by auto} \)

have \( f x < f (x+(?v+1)) \text{ by (rule } L1) \)

with \( S2 \) show \( f x < f y \text{ by auto} \)

qed

qed

lemma \( \text{pr-index-enumerator-mono1}: n1 < n2 \Rightarrow \text{pr-index-enumerator } n1 \text{ \( m \) < \( \text{pr-index-enumerator } n2 \text{ \( m \) \( n \)}} \)

proof –

assume \( A: n1 < n2 \)

def \( f\text{-def}: f \equiv \lambda x. \text{pr-index-enumerator } x \text{ \( m \) \( m \) m \text{ \( m \)}} \)

have \( f\text{-inc}: \forall x. f x < f (x+1) \)

proof

fix \( x \) show \( f x < f (x+1) \text{ by (unfold } f\text{-def, rule } \text{pr-index-enumerator-increase1)} \)

qed

from \( A \) have \( \forall x y. (x < y \longrightarrow f x < f y) \text{ by (rule } f\text{-inc-mono)} \)

with \( A \) \( f\text{-def \) show \( \text{thesis by auto} \)

qed

lemma \( \text{pr-index-enumerator-mono2}: m1 < m2 \Rightarrow \text{pr-index-enumerator } n \text{ \( m1 \text{ \( m2 \) \( m2 \)}} \)

proof –

assume \( A: m1 < m2 \)

def \( f\text{-def}: f \equiv \lambda x. \text{pr-index-enumerator } n \text{ \( x \)}} \)

73
have \( f\text{-inc} : \forall \ x. \ f x < f (x+1) \)

proof

fix \( x \) show \( f x < f (x+1) \) by (unfold f-def, rule pr-index-enumerator-increase2)

qed

from \( f\text{-inc} \) have \( \forall \ x \ y. \ (x < y \rightarrow f x < f y) \) by (rule f-ine mono)

with \( A \) f-def show \( \text{thesis} \) by auto

qed

lemma \( f\text{-mono-inj} : \forall \ (x::nat) \ (y::nat). \ (x < y \rightarrow (f::nat\Rightarrow nat) x < y) \rightarrow \forall \) 
\( (x::nat) \ (y::nat). \ (f x = f y \rightarrow x = y) \)

proof (rule allI, rule allI)

fix \( x \ y \) assume \( A \): \( \forall \ x \ y. \ x < y \rightarrow f x < f y \) show \( f x = f y \rightarrow x = y \)

proof

assume \( A1 : f x = f y \) show \( x = y \)

proof (rule ccontr)

assume \( A2 : x \neq y \) show \( \text{False} \)

proof cases

assume \( A3 : x < y \)

from \( A \ A3 \) have \( f x < f y \) by auto

with \( A1 \) show \( \text{False} \) by auto

next

assume \( \neg x < y \) with \( A2 \) have \( A4 : y < x \) by auto

from \( A \ A4 \) have \( f y < f x \) by auto

with \( A1 \) show \( \text{False} \) by auto

qed

qed

theorem \( \text{pr-index-enumerator-inj1} : \text{pr-index-enumerator} \ n1 \ m = \text{pr-index-enumerator} \ n2 \ m \Rightarrow n1 = n2 \)

proof –

assume \( A \): \( \text{pr-index-enumerator} \ n1 \ m = \text{pr-index-enumerator} \ n2 \ m \)

def \( f\text{-def} : f \equiv \lambda \ x. \ \text{pr-index-enumerator} \ x \ m \)

have \( f\text{-mono} : \forall \ x \ y. \ (x < y \rightarrow f x < f y) \)

proof (rule allI, rule allI)

fix \( x \ y \) show \( x < y \rightarrow f x < f y \) by (unfold f-def, simp add: pr-index-enumerator-mono1)

qed

from \( f\text{-mono} \) have \( \forall \ x \ y. \ (f x = f y \rightarrow x = y) \) by (rule f-mono-inj)

with \( A \) f-def show \( \text{thesis} \) by auto

qed

theorem \( \text{pr-index-enumerator-inj2} : \text{pr-index-enumerator} \ n \ m1 = \text{pr-index-enumerator} \ n \ m2 \Rightarrow m1 = m2 \)

proof –

assume \( A \): \( \text{pr-index-enumerator} \ n \ m1 = \text{pr-index-enumerator} \ n \ m2 \)

def \( f\text{-def} : f \equiv \lambda \ x. \ \text{pr-index-enumerator} \ n \ x \)

have \( f\text{-mono} : \forall \ x \ y. \ (x < y \rightarrow f x < f y) \)

proof (rule allI, rule allI)
fix x y show x < y → f x < f y by (unfold f-def, simp add: pr-index-enumerator-mono2)
qed
from f-mono have ∀ x y. (f x = f y → x = y) by (rule f-mono-inj)
with A f-def show ?thesis by auto
qed

theorem pr-index-enumerator-main: nat-to-pr n = nat-to-pr (pr-index-enumerator n m)
proof (induct m)
  show nat-to-pr n = nat-to-pr (pr-index-enumerator n 0) by simp
next
  fix na assume A: nat-to-pr n = nat-to-pr (pr-index-enumerator n na)
  show nat-to-pr n = nat-to-pr (pr-index-enumerator n (Suc na))
  proof
    let ?a = pr-index-enumerator n na
    have S1: pr-index-enumerator n (Suc na) = comp-by-index index-of-id ?a by simp
    have nat-to-pr (comp-by-index index-of-id ?a) = (λ x. (nat-to-pr index-of-id) (nat-to-pr ?a x)) by (rule comp-by-index-main)
    with index-of-id-main have nat-to-pr (comp-by-index index-of-id ?a) = nat-to-pr ?a by simp
    with A S1 show ?thesis by simp
  qed
qed
end

5 Finite sets

theory PRecFinSet
imports PRecFun
begin

We introduce a particular mapping nat-to-set from natural numbers to finite sets of natural numbers and a particular mapping set-to-nat from finite sets of natural numbers to natural numbers. See [1] and [2] for more information.

definition
c-in :: nat ⇒ nat ⇒ nat where
c-in = (λ x u. (u div (2 ^ x)) mod 2)

lemma c-in-is-pr: c-in ∈ PrimRec2
proof –
  from mod-is-pr powe-is-pr div-is-pr have (λ x u. (u div (2 ^ x)) mod 2) ∈ PrimRec2 by prec
  with c-in-def show ?thesis by auto
qed

definition
\(\text{nat-to-set} :: \text{nat} \rightarrow \text{nat set} \) where 
\(\text{nat-to-set} u \equiv \{x. \ 2 \cdot x \leq u \land \text{c-in } x u = 1\}\)

**lemma** \(c\text{-in-upper-bound} \): \(\text{c-in } x u = 1 \implies 2 \cdot x \leq u\)

**proof** –
assume \(A: \text{c-in } x u = 1\) then have \(S1: (u \div (2 \cdot x)) \mod 2 = 1 \) by (unfold \text{c-in-def}) then have \(S2: u \div (2 \cdot x) > 0 \) by \(\text{arith}\) show \(?\text{thesis}\)
proof (rule \text{ccontr}) assume \(\neg 2 \cdot x \leq u\) then have \(u < 2 \cdot x\) by \(\text{auto}\) then have \(u \div (2 \cdot x) = 0\) by (rule \text{div-less}) with \(S2\) show \(\text{False}\) by \(\text{auto}\) qed

**lemma** \(\text{nat-to-set-upper-bound}\): \(x \in \text{nat-to-set } u \implies 2 \cdot x \leq u\) by \((\text{simp add: nat-to-set-def})\)

**lemma** \(x\text{-lt-2-x}\): \(x < 2 \cdot x\) by (\text{induct } x) \(\text{auto}\)

**lemma** \(\text{nat-to-set-upper-bound1}\): \(x \in \text{nat-to-set } u \implies x < u\)

**proof** –
assume \(x \in \text{nat-to-set } u\) then have \(S1: 2 \cdot x \leq u\) by (simp add: \text{nat-to-set-def}) have \(S2: x < 2 \cdot x\) by (rule \text{x-lt-2-x}) from \(S1\) \(S2\) show \(?\text{thesis}\) by \(\text{auto}\) qed

**lemma** \(\text{nat-to-set-upper-bound2}\): \(\text{nat-to-set } u \subseteq \{i. \ i < u\}\)

**proof** –
from \(\text{nat-to-set-upper-bound1}\) show \(?\text{thesis}\) by \(\text{blast}\) qed

**lemma** \(\text{nat-to-set-is-finite}\): \(\text{finite } (\text{nat-to-set } u)\)

**proof** –
have \(S1: \text{finite } \{i. \ i < u\}\)

**proof** –
let \(?B = \{i. \ i < u\}\) let \(?f = (\lambda (x::\text{nat}). \ x)\) have \(?B = ?f \cdot ?B\) by \(\text{auto}\) then show \(\text{finite } ?B\) by (rule \text{nat-seg-image-imp-finite}) qed

have \(S2: \text{nat-to-set } u \subseteq \{i. \ i < u\}\) by (rule \text{nat-to-set-upper-bound2}) from \(S2\) \(S1\) show \(?\text{thesis}\) by (rule \text{finite-subset}) qed

**lemma** \(x\text{-in-u-eq}\): \((x \in \text{nat-to-set } u) = (\text{c-in } x u = 1)\) by (auto simp add: \text{nat-to-set-def})
definition
\[ \log_2 :: \text{nat} \Rightarrow \text{nat} \quad \text{where} \quad \log_2 = (\lambda x. \text{Least}(\%z. x < 2^z(x+1))) \]

lemma \(\log_2\)-at-0: \(\log_2 0 = 0\)
proof –
let \(?v = \log_2 0\)
have \(S1: 0 \leq ?v\) by auto
have \(S2: ?v = \text{Least}(\%z::\text{nat}. (0::\text{nat}) < 2^z(z+1))\) by (simp add: log2-def)
have \(S3: (0::\text{nat}) < 2^z(0+1)\) by auto
from \(S3\) have \(S4: \text{Least}(\%z::\text{nat}. (0::\text{nat}) < 2^z(z+1)) \leq 0\) by (rule Least-le)
from \(S2 \quad S4\) have \(S5: ?v \leq 0\) by auto
from \(S1 \quad S5\) have \(S6: ?v = 0\) by auto
thus \(?\thesis\) by auto
qed

lemma \(\log_2\)-at-1: \(\log_2 1 = 0\)
proof –
let \(?v = \log_2 1\)
have \(S1: 0 \leq ?v\) by auto
have \(S2: ?v = \text{Least}(\%z::\text{nat}. (1::\text{nat}) < 2^z(z+1))\) by (simp add: log2-def)
have \(S3: (1::\text{nat}) < 2^z(0+1)\) by auto
from \(S3\) have \(S4: \text{Least}(\%z::\text{nat}. (1::\text{nat}) < 2^z(z+1)) \leq 0\) by (rule Least-le)
from \(S2 \quad S4\) have \(S5: ?v \leq 0\) by auto
from \(S1 \quad S5\) have \(S6: ?v = 0\) by auto
thus \(?\thesis\) by auto
qed

lemma \(\log_2\)-le: \(x > 0 \implies 2^\log_2 x \leq x\)
proof –
assume \(A: x > 0\)
show \(?\thesis\)
proof (cases)
assume \(A1: \log_2 x = 0\)
with \(A\) show \(?\thesis\) by auto
next
assume \(A1: \log_2 x \neq 0\)
then have \(S1: \log_2 x > 0\) by auto
def \(y\)-def: \(y \equiv \log_2 x - 1\)
from \(S1\) \(y\)-def have \(S2: \log_2 x = y + 1\) by auto
then have \(S3: y < \log_2 x\) by auto
have \(2^z(y+1) \leq x\)
proof (rule econtr)
assume \(A2: \neg 2^z(y+1) \leq x\) then have \(x < 2^z(y+1)\) by auto
then have \(\log_2 x \leq y\) by (simp add: log2-def Least-le)
with \(S3\) show \(\text{False}\) by auto
qed
with $S2$ show $\text{thesis by auto}$
qed

**lemma** $\text{log2-gt: } x < 2 \cdot (\log2 x + 1)$

**proof**
- have $x < 2 \cdot x$ by (rule x-lt-2-x)
- then have $S1$: $x < 2 \cdot (x+1)$ by simp
- def $\text{y-def: } y \equiv x$
- from $S1$ $\text{y-def}$ have $S2$: $x < 2 \cdot (y+1)$ by auto
- let $?P = \lambda z. x < 2 \cdot (z+1)$
- from $S2$ have $S3$: $?P y$ by simp
- let $?P = \lambda z. x < 2 \cdot (z+1)$
- from $S2$ $\text{y-def}$ have $S5$: $\log2 x = \text{Least } ?P$ by (unfold log2-def, auto)
- from $S4$ $S5$ show $\text{thesis by auto}$
qed

**lemma** $\text{x-div-x: } x > 0 \implies (x::nat) \text{ div } x = 1$ by auto

**lemma** $\text{div-ge: } (k::nat) \leq m \text{ div } n \implies n \cdot k \leq m$

**proof**
- assume $A$: $k \leq m \text{ div } n$
- have $S1$: $n \cdot (m \text{ div } n) + m \text{ mod } n = m$ by (rule mult-div-mod-eq)
- have $S2$: $0 \leq m \text{ mod } n$ by auto
- from $S1$ $S2$ have $S3$: $n \cdot (m \text{ div } n) \leq m$ by arith
- from $A$ have $S4$: $n \cdot k \leq n \cdot (m \text{ div } n)$ by auto
- from $S4$ $S3$ show $\text{thesis by (rule order-trans)}$
qed

**lemma** $\text{div-lt: } m < n \cdot k \implies m \text{ div } n < (k::nat)$

**proof**
- assume $A$: $m < n \cdot k$
- show $\text{thesis}$
- proof (rule ccontr)
- assume $\neg m \text{ div } n < k$
- then have $S1$: $k \leq m \text{ div } n$ by auto
- then have $S2$: $n \cdot k \leq m$ by (rule div-ge)
- with $A$ show False by auto
qed

**lemma** $\text{log2-ln1: } u > 0 \implies u \text{ div } 2 \cdot (\log2 u) = 1$

**proof**
- assume $A$: $u > 0$
- then have $S1$: $2 \cdot (\log2 u) \leq u$ by (rule log2-le)
- have $S2$: $u < 2 \cdot (\log2 u+1)$ by (rule log2-gt)
- then have $S3$: $u < (2 \cdot \log2 u) \cdot 2$ by simp
- have $(2::nat) > 0$ by simp
- then have $(2::nat) \cdot \log2 u > 0$ by simp
- then have $S4$: $(2::nat) \cdot \log2 u \text{ div } 2 \cdot \log2 u = 1$ by auto
- from $S1$ have $S5$: $(2::nat) \cdot \log2 u \text{ div } 2 \cdot \log2 u \leq u \text{ div } 2 \cdot \log2 u$ by (rule
with $S_4$ have $S_6$: $1 \leq u \div 2^\log_2 u$ by auto
from $S_3$ have $S_7$: $u \div 2^\log_2 u < 2$ by (rule div-lt)
from $S_6$ $S_7$ show thesis by auto
qed

lemma log2-lm2: $u > 0 \implies c \in (\log_2 u)$ $u = 1$
proof
  assume $A$: $u > 0$
  then have $S_1$: $u \div 2 ^ \cdot (\log_2 u) = 1$ by (rule log2-lm1)
  have c-in (log2 u) $u = (u \div 2 ^ \cdot (\log_2 u)) \mod 2$ by (simp add: c-in-def)
  also from $S_1$ have ... = $1 \mod 2$ by simp
  also have ... = $0$ by auto
  finally show thesis by auto
qed

lemma log2-lm3: $\log_2 u < x \implies c \in x$ $u = 0$
proof
  assume $A$: $\log_2 u < x$
  then have $S_1$: $(\log_2 u) + 1 \leq x$ by auto
  have $S_2$: $1 \leq (2::nat)$ by auto
  from $S_1$ $S_2$ have $S_3$: $(2::nat) ^ \cdot (\log_2 u+1) \leq 2^x$ by (rule power-increasing)
  have $S_4$: $u < (2::nat) ^ \cdot (\log_2 u+1)$ by (rule log2-gt)
  from $S_3$ $S_4$ have $S_5$: $u < 2^x$ by auto
  then have $S_6$: $u \div 2^x = 0$ by (rule div-less)
  have c-in x $u = (u \div 2^x) \mod 2$ by (simp add: c-in-def)
  also from $S_6$ have ... = $0 \mod 2$ by simp
  also have ... = $0$ by auto
  finally have thesis by auto
  thus thesis by auto
qed

lemma log2-lm4: c-in x $u = 1 \implies x \leq \log_2 u$
proof
  assume $A$: c-in x $u = 1$
  show thesis
  proof (rule ccontr)
    assume \( \neg x \leq \log_2 u \)
    then have $S_1$: $\log_2 u < x$ by auto
    then have $S_2$: c-in x $u = 0$ by (rule log2-lm3)
    with $A$ show False by auto
  qed
qed

lemma nat-to-set-lub: $x \in \text{nat-to-set} u \implies x \leq \log_2 u$
proof
  assume $x \in \text{nat-to-set} u$
  then have $S_1$: c-in x $u = 1$ by (simp add: x-in-u-eq)
  then show thesis by (rule log2-lm4)
lemma log2-lm5: \( u > 0 \implies \log_2 u \in \text{nat-to-set} u \)
proof -
assume A: \( u > 0 \)
then have c-in \((\log_2 u)\) \( u = 1 \) by (rule log2-lm2)
then show ?thesis by (simp add: x-in-u-eq)
qed

lemma pos-imp-ne: \( u > 0 \implies \text{nat-to-set} u \neq \{\} \)
proof -
assume \( u > 0 \)
then have \( \log_2 u \in \text{nat-to-set} u \) by (rule log2-lm5)
thus ?thesis by auto
qed

lemma empty-is-zero: \( \text{nat-to-set} u = \{\} \implies u = 0 \)
proof (rule ccontr)
assume A1: \( \text{nat-to-set} u = \{\} \)
assume A2: \( u \neq 0 \) then have S1: \( u > 0 \) by auto
from S1 have nat-to-set u \(\neq \{\} \) by (rule pos-imp-ne)
with A1 show False by auto
qed

lemma log2-is-max: \( u > 0 \implies \log_2 u = \text{Max} \ (\text{nat-to-set} u) \)
proof -
assume A: \( u > 0 \)
then have S1: \( \log_2 u \in \text{nat-to-set} u \) by (rule log2-lm5)
def max-def: \( \text{max} \equiv \text{Max} \ (\text{nat-to-set} u) \)
from A have ne: \( \text{nat-to-set} u \neq \{\} \) by (rule pos-imp-ne)
from max-def finite ne have max-in: \( \text{max} \in \text{nat-to-set} u \) by simp
from max-in have S2: \( \text{c-in} \text{ max} u = 1 \) by (simp add: x-in-u-eq)
then have S3: \( \text{max} \leq \log_2 u \) by (rule log2-lm4)
from finite ne S1 max-def have S4: \( \log_2 u \leq \text{max} \) by simp
from S3 S4 max-def show ?thesis by auto
qed

lemma zero-is-empty: \( \text{nat-to-set} 0 = \{\} \)
proof -
have S1: \( \{i. \ i<0::\text{nat}\} = \{\} \) by blast
have S2: \( \text{nat-to-set} 0 \subseteq \{i. \ i<0\} \) by (rule nat-to-set-upper-bound2)
from S1 S2 show ?thesis by auto
qed

lemma ne-imp-pos: \( \text{nat-to-set} u \neq \{\} \implies u > 0 \)
proof (rule ccontr)
assume A1: \( \text{nat-to-set} u \neq \{\} \)
assume \( \neg \ 0 < u \) then have \( u = 0 \) by auto
then have nat-to-set $u = \{\}$ by (simp add: zero-is-empty)
with $A1$ show False by auto
qed

lemma div-mod-lm: $y < x \implies ((u + (2::nat) \cdot x) \div (2::nat) \cdot y) \mod 2 = (u \div (2::nat) \cdot y) \mod 2$
proof
  assume y-lt-x: $y < x$
  let $?s = (2::nat) \cdot y$
  have n-pos: $0 < $?s by auto
  let $?n = x - y$
  from y-lt-x have s-pos: $0 < $?s by auto
  from y-lt-x have S3: $x = y + $?s by auto
  moreover have (2::nat) \cdot x = (2::nat) \cdot (y + $?s) by auto
  ultimately have (2::nat) \cdot y = 2^?s by auto
  then have S4: $u + (2::nat) \cdot x = u + (2::nat) \cdot y \cdot 2^?s$ by auto
  from n-pos have S5: $(u + (2::nat) \cdot y \cdot 2^?s) \div 2^y = 2^?s + (u \div 2^y)$ by simp
  from S4 S5 have S6: $(u + (2::nat) \cdot x) \div 2^y = 2^?s + (u \div 2^y)$ by auto
  from s-pos have S8: $?s = (\?? - 1) + 1$ by auto
  have $(2::nat) \cdot ((?s - (1::nat)) + (1::nat)) = (2::nat) \cdot (?s - (1::nat)) + 2^1$ by (rule power-add)
  with S8 have S9: $(2::nat) \cdot ?s = (2::nat) \cdot (?s - (1::nat)) + 2$ by auto
  then have S10: $2^?s + (u \div 2^y) = (u \div 2^y) + (2::nat) \cdot (?s - (1::nat)) + 2^1$ by auto
  have S11: $(u \div 2^y) + (2::nat) \cdot (?s - (1::nat)) + 2) \mod 2 = (u \div 2^y) \mod 2$ by (rule mod-mult-self1)
  from S6 S10 S11 show $\theta$thesis by auto
qed

lemma add-power: $u < 2^x \implies$ nat-to-set $(u + 2^x) =$ nat-to-set $u \cup \{x\}$
proof
  assume A: $u < 2^x$
  have log2-is-x: $\log2 (u+2^x) = x$
  proof (unfold log2-def; rule Least-equality)
    from A show $u+2^x < 2^x(x+1)$ by auto
  next
    fix $z$
    assume A1: $u + 2^z < 2^z(x+1)$
    show $x \leq z$
    proof (rule ccontr)
      assume $\neg x \leq z$
      then have $z < x$ by auto
      then have L1: $z + 1 \leq x$ by auto
      have L2: $1 \leq (2::nat)$ by auto
      from L1 L2 have L3: $(2::nat) \cdot (z+1) \leq (2::nat) \cdot x$ by (rule power-increasing)
      with A1 show False by auto
    qed
    qed

81
qed

show thesis

proof (rule subset-antisym)

show nat-to-set \((u + \text{2}^{\text{2}}x)\) \(\subseteq\) nat-to-set \(u \cup \{x\}\)

proof

fix \(y\)

assume \(A1\): \(y \in\) nat-to-set \((u + \text{2}^{\text{2}}x)\)

show \(y \in\) nat-to-set \(u \cup \{x\}\)

proof

assume \(y \notin \{x\}\) then have \(S1\): \(y \neq x\) by auto

from \(A1\) have \(y \leq \log2\) \((u + \text{2}^{\text{2}}x)\) by (rule nat-to-set-lub)

with \log2-is-x have \(y \leq x\) by auto

with \(S1\) have \(y < x\) by auto

from \(A1\) have \(c\)-in \(y\) \((u + \text{2}^{\text{2}}x)\) = \(1\) by (simp add: x-in-u-eq)

then have \(S2\): \((u + \text{2}^{\text{2}}x)\) div \((\text{2}^{\text{2}})\) mod 2 = \(1\) by (rule div-mod-lm)

from \(y < x\) have \(u\) div \((\text{2}^{\text{2}})\) mod 2 = \(1\) by auto

then have \(c\)-in \(y\) \((u + \text{2}^{\text{2}}x)\) = \(1\) by (simp add: c-in-def)

then show \(y \in\) nat-to-set \(u\) by (simp add: x-in-u-eq)

qed

qed

next

show nat-to-set \(u \cup \{x\}\) \(\subseteq\) nat-to-set \((u + \text{2}^{\text{2}}x)\)

proof

fix \(y\)

assume \(A1\): \(y \in\) nat-to-set \(u \cup \{x\}\)

show \(y \in\) nat-to-set \((u + \text{2}^{\text{2}}x)\)

proof

cases

assume \(y \in \{x\}\)

then have \(y = x\) by auto

then have \(y = \log2\) \((u + \text{2}^{\text{2}}x)\) by (simp add: log2-is-x)

then show \(\text{thesis}\) by (simp add: log2-lm5)

next

assume \(y\)-notin: \(y \notin \{x\}\)

then have \(y\)-ne-x: \(y \neq x\) by auto

from \(A1\) \(y\)-notin have \(y\)-in: \(y \in\) nat-to-set \(u\) by auto

have \(y < x\)

proof (rule ccontr)

assume \(\neg\) \(y < x\)

with \(y\)-ne-x have \(y\)-gt-x: \(x < y\) by auto

have \(1 < (2::nat)\) by auto

from \(y\)-gt-x this have \(L1\): \((2::nat)\) \(\leq\) \(2^y\) by (rule power-strict-increasing)

from \(y\)-in have \(L2\): \(2^y \leq u\) by (rule nat-to-set-upper-bound)

from \(L1\) \(L2\) have \((2::nat)\) \(\leq\) \(u\) by arith

with \(A\) show \(False\) by auto

qed

from \(y\)-in have \(c\)-in \(y\) \(u\) = \(1\) by (simp add: x-in-u-eq)

then have \(S2\): \((u\) div \((2::nat)\) mod 2 = \(1\) by (unfold c-in-def)

from \(y < x\) have \((u + (2::nat)\) \(\leq\) \(2^y\) by (rule div-mod-lm)

82
with $S2$ have $(u + (2::nat) \cdot x) \text{div} 2^y \mod 2 = 1$ by auto
then have c-in y $(u + (2::nat) \cdot x) = 1$ by (simp add: c-in-def)
then show $y \in \text{nat-to-set} \ (u + (2::nat) \cdot x)$ by (simp add: x-in-u-eq)
  qed
  qed
  qed

theorem nat-to-set-inj: nat-to-set u = nat-to-set v $\implies$ u = v
proof
  assume A: nat-to-set u = nat-to-set v
  let ?P = $\lambda$ (n::nat). $(\forall \ (D::nat \ set). \text{finite} \ D \ \land \text{card} \ D \leq n \implies (\forall u \ v. \text{nat-to-set} \ u = D \ \implies \ u = v))$
  have P-at-0: ?P 0
  proof
    assume A1: finite D \land card D \leq 0
    from A1 have S1: finite D by auto
    from A1 have S2: card D = 0 by auto
    show $(\forall u \ v. \text{nat-to-set} \ u = D \ \land \text{nat-to-set} \ v = D \implies u = v)$
      proof
        rule impI
        assume A1: finite D \land card D \leq 0
        from A1 have S1: finite D by auto
        from A1 have S2: card D = 0 by auto
        show $(\forall u \ v. \text{nat-to-set} \ u = D \ \land \text{nat-to-set} \ v = D \implies u = v)$
          proof
            rule allI, rule allI, rule impI
            fix u v show nat-to-set u = D \land nat-to-set v = D \implies u = v
            proof
              assume A2: nat-to-set u = D \land nat-to-set v = D
              from A2 have L1: nat-to-set u = D by auto
              from A2 have L2: nat-to-set v = D by auto
              from L1 S3 have nat-to-set u = {} by auto
              then have u-z: u = 0 by (rule empty-is-zero)
              from L2 S3 have nat-to-set v = {} by auto
              then have v-z: v = 0 by (rule empty-is-zero)
              from u-z v-z show u=v by auto
            qed
            qed
            qed
            have P-at-Suc: \land n. ?P n $\implies$ ?P (Suc n)
            proof
              fix n
              assume A-n: ?P n
              show ?P (Suc n)
                proof
                  rule impI
                  assume A1: finite D \land card D \leq Suc n
                  from A1 have S1: finite D by auto
                  from A1 have S2: card D \leq Suc n by auto
                  show $(\forall u \ v. \text{nat-to-set} \ u = D \ \land \text{nat-to-set} \ v = D \implies u = v)$
                    proof
                      rule allI, rule allI, rule impI
                      assume A1: finite D \land card D \leq Suc n
                      from A1 have S1: finite D by auto
                      from A1 have S2: card D \leq Suc n by auto
                      show $(\forall u \ v. \text{nat-to-set} \ u = D \ \land \text{nat-to-set} \ v = D \implies u = v)$
                        proof
                          rule allI, rule allI, rule impI
                          have P-at-Suc: \land n. ?P n $\implies$ ?P (Suc n)
                          proof
                            fix n
                            assume A-n: ?P n
                            show ?P (Suc n)
                              proof
                                rule impI
                                assume A1: finite D \land card D \leq Suc n
                                from A1 have S1: finite D by auto
                                from A1 have S2: card D \leq Suc n by auto
                                show $(\forall u \ v. \text{nat-to-set} \ u = D \ \land \text{nat-to-set} \ v = D \implies u = v)$
                                  proof
                                    rule allI, rule allI, rule impI
                                    theorem nat-to-set-inj: nat-to-set u = nat-to-set v $\implies$ u = v
                                    proof
                                      assume A: nat-to-set u = nat-to-set v
                                      let ?P = $\lambda$ (n::nat). $(\forall \ (D::nat \ set). \text{finite} \ D \ \land \text{card} \ D \leq n \implies (\forall u \ v. \text{nat-to-set} \ u = D \ \implies \ u = v))$
                                      have P-at-0: ?P 0
                                      proof
                                        assume A1: finite D \land card D \leq 0
                                        from A1 have S1: finite D by auto
                                        from A1 have S2: card D = 0 by auto
                                        show $(\forall u \ v. \text{nat-to-set} \ u = D \ \land \text{nat-to-set} \ v = D \implies u = v)$
                                          proof
                                            rule impI
                                            assume A1: finite D \land card D \leq 0
                                            from A1 have S1: finite D by auto
                                            from A1 have S2: card D = 0 by auto
                                            show $(\forall u \ v. \text{nat-to-set} \ u = D \ \land \text{nat-to-set} \ v = D \implies u = v)$
                                              proof
                                                rule allI, rule allI, rule impI
                                                fix u v show nat-to-set u = D \land nat-to-set v = D \implies u = v
                                                proof
                                                  assume A2: nat-to-set u = D \land nat-to-set v = D
                                                  from A2 have L1: nat-to-set u = D by auto
                                                  from A2 have L2: nat-to-set v = D by auto
                                                  from L1 S3 have nat-to-set u = {} by auto
                                                  then have u-z: u = 0 by (rule empty-is-zero)
                                                  from L2 S3 have nat-to-set v = {} by auto
                                                  then have v-z: v = 0 by (rule empty-is-zero)
                                                  from u-z v-z show u=v by auto
                                                qed
                                              qed
                                            qed
                                          qed
                                        qed
                                     qed
                                  qed
                                qed
                              qed
                            qed
                          qed
                        qed
                    qed
                qed
            qed
          qed
        qed
      qed
    qed
  qed
  qed
  qed
  qed

83
fix \(u\) \(v\)

**assume** \(A2\): \(\text{nat-to-set } u = D \land \text{nat-to-set } v = D\)

**from** \(A2\) **have** \(d-u-d\): \(\text{nat-to-set } u = D\) **by** \text{auto}

**from** \(A2\) **have** \(d-v-d\): \(\text{nat-to-set } v = D\) **by** \text{auto}

**show** \(u = v\)

**proof** (cases)

**assume** \(A3\): \(D = \{\}\)

**from** \(A3\) **d-u-d** have \(\text{nat-to-set } u = \{\}\) **by** \text{auto}

**then have** \(u-z\): \(u = 0\) **by** \((\text{rule empty-is-zero})\)

**from** \(u-z\) **v-z** **show** \(u = v\) **by** \text{auto}

**next**

**assume** \(A3\): \(D \neq \{\}\)

**from** \(A3\) **d-u-d** have \(\text{nat-to-set } u \neq \{\}\) **by** \text{auto}

**then have** \(u-pos\): \(u > 0\) **by** \((\text{rule ne-imp-pos})\)

**from** \(u-pos\) **d-v-d** have \(\text{nat-to-set } v \neq \{\}\) **by** \text{auto}

**then have** \(v-pos\): \(v > 0\) **by** \((\text{rule ne-imp-pos})\)

**def** \(m\)-def: \(m \equiv \text{Max } D\)

**from** \(S1\) **m-def** **A3** have \(\text{m-in}\): \(m \in D\) **by** \text{auto}

**from** \(d-u-d\) **m-def** **have** \(m\)-w: \(m = \text{Max } (\text{nat-to-set } u)\) **by** \text{auto}

**from** \(d-v-d\) **m-def** **have** \(m\)-v: \(m = \text{Max } (\text{nat-to-set } v)\) **by** \text{auto}

**from** \(u-pos\) \(m\)-u **log2-is-max** have \(m\)-log-u: \(m = \log2 u\) **by** \text{auto}

**from** \(v-pos\) \(m\)-v **log2-is-max** have \(m\)-log-v: \(m = \log2 v\) **by** \text{auto}

**def** \(D1\)-def: \(D1 \equiv D - \{m\}\)

**def** \(u1\)-def: \(u1 \equiv u - 2^m\)

**def** \(v1\)-def: \(v1 \equiv v - 2^m\)

**have** \(\text{card-D1}\): \(\text{card } D1 \leq n\)

**proof**

**from** \(D1\)-def **S1** \(\text{m-in}\) **have** \(\text{card } D1 = (\text{card } D) - 1\) **by** \((\text{simp add: card-Diff-singleton})\)

**with** \(S2\) **show** ?thesis **by** \text{auto}

**qed**

**have** \(u\)-u1: \(u = u1 + 2^m\)

**proof**

**from** \(u-pos\) **have** \(L1\): \(2 \cdot \log2 u \leq u\) **by** \((\text{rule log2-le})\)

**with** \(u\)-log-u **have** \(L2\): \(2 \cdot m \leq u\) **by** \text{auto}

**with** \(u1\)-def **show** ?thesis **by** \text{auto}

**qed**

**have** \(u1\)-d1: \(\text{nat-to-set } u1 = D1\)

**proof**

**from** \(m\)-log-u **log2-gt** **have** \(u < 2^*(m+1)\) **by** \text{auto}

**with** \(u\)-u1 **have** \(u1\)-lt-2-m: \(u1 < 2^m\) **by** \text{auto}

**with** \(u\)-u1 **have** \(L1\): \(\text{nat-to-set } u = \text{nat-to-set } u1 \cup \{m\}\) **by** \((\text{simp add: add-power})\)

**have** \(m\)-notin: \(m \notin \text{nat-to-set } u1\)

**proof** \((\text{rule ccontr})\)

**assume** \(\neg m \notin \text{nat-to-set } u1\) **then have** \(m \in \text{nat-to-set } u1\) **by** \text{auto}

**then have** \(2^m \leq u1\) **by** \((\text{rule nat-to-set-upper-bound})\)
with u1-lt-2-m show False by auto

qed

from L1 m-notin have nat-to-set u1 = nat-to-set u - {m} by auto

with d-u-d have nat-to-set u1 = D - {m} by auto

with D1-def show ?thesis by auto

qed

have v-v1: v = v1 + 2^m
proof -
  from v-pos have L1: 2 ^ log2 v ≤ v by (rule log2-le)
  with m-log-v have L2: 2 ^ m ≤ v by auto
  with v1-def show ?thesis by auto

qed

have v1-d1: nat-to-set v1 = D1
proof -
  from m-log-v log2-gt have v < 2^(m+1) by auto
  with v-v1 have v1-lt-2-m: v1 < 2^m by auto
  with v1-def show L1: nat-to-set v = nat-to-set v1 ∪ {m} by (simp add: add-power)
  have m-notin: m ∉ nat-to-set v1
  proof (rule ccontr)
    assume ¬ m ∉ nat-to-set v1 then have m ∈ nat-to-set v1 by auto
    then have 2^m ≤ v by (rule nat-to-set-upper-bound)
    with v1-lt-2-m show False by auto

  qed

  from L1 m-notin have nat-to-set v1 = nat-to-set v - {m} by auto
  with d-v-d have nat-to-set v1 = D - {m} by auto
  with D1-def show ?thesis by auto

  qed

  from S1 D1-def have P1: finite D1 by auto
  with card-D1 have P2: finite D1 ∧ card D1 ≤ n by auto
  from A-n P2 have ( ∀ u. nat-to-set u = D1 ∧ nat-to-set v = D1 → u = v ) by auto
  with u1-d1 v1-d1 have u1 = v1 by auto
  with u-u1 v-v1 show u = v by auto

  qed

  qed

  qed

  qed

  from P-at-0 P-at-Suc have main: ∃ n. ?P n by (rule nat.induct)
  def D-def: D ≡ nat-to-set u
  from n-def D-def have P1: nat-to-set u = D by auto
  from def nat-to-set-is-finite have D-finite: finite D by auto
  def n-def: n ≡ card D
  from n-def D-finite have card-le: card D ≤ n by auto
  from D-finite card-le have P3: finite D ∧ card D ≤ n by auto
  with main have P4: ∃ u v. nat-to-set u = D ∧ nat-to-set v = D → u = v by auto

  qed
with P1 P2 show u = v by auto
qed

definition
set-to-nat :: nat set => nat where
set-to-nat = (λ D. sum (λ x. 2 ^ x) D)

lemma two-power-sum: sum (λ x. (2::nat) ^ x) { i. i < Suc m } = (2 ^ Suc m) - 1
proof (induct m)
  show sum (λ x. (2::nat) ^ x) { i. i < Suc 0 } = (2 ^ Suc 0) - 1 by auto
next
  fix n
  assume A: sum (λ x. (2::nat) ^ x) { i. i < Suc n } = (2 ^ Suc n) - 1
  show sum (λ x. (2::nat) ^ x) { i. i < Suc (Suc n) } = (2 ^ Suc (Suc n)) - 1
  proof
    let f = λ x. (2::nat) ^ x
    have S1: { i. i < Suc (Suc n) } = { i. i < Suc n } by auto
    have S2: { i. i < Suc n } = { i. i < Suc n } ∪ { Suc n } by auto
    from S1 S2 have S3: { i. i < Suc (Suc n) } = { i. i < Suc n } ∪ { Suc n } by auto
    have S4: { i. i < Suc n } = (λ x. x) ^ { i. i < Suc n } by auto
    then have S5: finite { i. i < Suc n } by (rule nat-seg-image-imp-finite)
    have S6: Suc n ≠ { i. i < Suc n } by auto
    from S5 S6 sum.insert have S7: sum f { i. i < Suc n } ∪ { Suc n } = 2 ^ Suc n + sum f { i. i < Suc n } by auto
    from S3 have sum f { i. i < Suc (Suc n) } = sum f { i. i < Suc n } ∪ { Suc n } by auto
    also from S7 have ... = 2 ^ Suc n + sum f { i. i < Suc n } by auto
    also from A have ... = 2 ^ Suc n + (((2::nat) ^ Suc n) - (1::nat)) by auto
    also have ... = (2 ^ Suc (Suc n)) - 1 by auto
    finally show ?thesis by auto
  qed
qed

lemma finite-interval: finite { i. (i::nat)<m }
proof
  have { i. i < m } = (λ x. x) ^ { i. i < m } by auto
  then show ?thesis by (rule nat-seg-image-imp-finite)
qed

lemma set-to-nat-at-empty: set-to-nat {} = θ by (unfold set-to-nat-def, rule sum.empty)

lemma set-to-nat-of-interval: set-to-nat { i. (i::nat)<m } = 2 ^ m - 1
proof (induct m)
  show set-to-nat { i. i < 0 } = 2 ^ 0 - 1
  proof
    have S1: { i. (i::nat) < 0 } = {} by auto
    with set-to-nat-at-empty have set-to-nat { i. i<0 } = 0 by auto
  qed
thus \( ?\text{thesis} \) by auto

qed

next

fix \( n \) show set-to-nat \( \{ i. \, i < \text{Suc} \, n \} = 2 \cdot \text{Suc} \, n - 1 \) by (unfold set-to-nat-def, rule two-power-sum)

qed

lemma set-to-nat-mono: \( \{ \text{finite} \, B; \, A \subseteq B \} \implies \text{set-to-nat} \, A \leq \text{set-to-nat} \, B \)

proof

- assume \( \text{b-finite}: \, \text{finite} \, B \)
- assume \( \text{a-le-b}: \, A \subseteq B \)

let \( \text{?f} = \lambda (x::nat). \, (2::nat) \cdot x \)

have \( S1: \, \text{set-to-nat} \, A = \text{sum} ?A \) by (simp add: set-to-nat-def)

have \( S2: \, \text{set-to-nat} \, B = \text{sum} ?B \) by (simp add: set-to-nat-def)

have \( S3: \, \forall \, x. \, x \in B - A \implies 0 \leq ?x \) by auto

from \( \text{b-finite} \, \text{a-le-b} \, S3 \) have \( \text{sum} \, ?A \leq \text{sum} \, ?B \) by (rule sum-mono2)

with \( S1 \, S2 \) show \( \text{?thesis} \) by auto

qed

theorem nat-to-set-srj: \( \text{finite} \, (D::nat \, \text{set}) \implies \text{nat-to-set} \, (\text{set-to-nat} \, D) = D \)

proof

- assume \( \text{A: finite} \, D \)

let \( \text{?P} = \lambda (n::nat). \, (\forall \, (D::nat \, \text{set}). \, \text{finite} \, D \land \text{card} \, D = n \implies \text{nat-to-set} \, (\text{set-to-nat} \, D) = D) \)

have \( \text{P-at-0}: \, ?P \, 0 \)

proof (rule allI)

fix \( D \)

show \( \text{finite} \, D \land \text{card} \, D = 0 \implies \text{nat-to-set} \, (\text{set-to-nat} \, D) = D \)

proof

- assume \( \text{A1: finite} \, D \land \text{card} \, D = 0 \)

from \( \text{A1} \) have \( S1: \, \text{finite} \, D \) by auto

from \( \text{A1} \) have \( S2: \, \text{card} \, D = 0 \) by auto

from \( S1 \, S2 \) have \( S3: \, D = \{} \) by auto

with \( \text{set-to-nat-def} \) have \( \text{set-to-nat} \, D = \text{sum} \, (\lambda \, x. \, 2 \cdot x) \, D \) by simp

with \( \text{S3 sum.empty} \) have \( \text{set-to-nat} \, D = 0 \) by auto

with \( \text{zero-is-empty} \, S3 \) show \( \text{nat-to-set} \, (\text{set-to-nat} \, D) = D \) by auto

qed

qed

have \( \text{P-at-Suc}: \, \forall \, n. \, ?P \, n \implies ?P \, (\text{Suc} \, n) \)

proof - fix \( n \)

assume \( \text{A-n: ?P} \, n \)

show \( ?P \, (\text{Suc} \, n) \)

proof

fix \( D \) show \( \text{finite} \, D \land \text{card} \, D = \text{Suc} \, n \implies \text{nat-to-set} \, (\text{set-to-nat} \, D) = D \)

proof

- assume \( \text{A1: finite} \, D \land \text{card} \, D = \text{Suc} \, n \)

from \( \text{A1} \) have \( S1: \, \text{finite} \, D \) by auto

from \( \text{A1} \) have \( S2: \, \text{card} \, D = \text{Suc} \, n \) by auto

\( \text{def m-def}: m \equiv \text{Max} \, D \)
from S2 have D-ne: D ≠ {} by auto

with S1 m-def have m-in: m ∈ D by auto
def D1-def: D1 ≡ D – {m}
from S1 D1-def have d1-finite: finite D1 by auto
from D1-def m-in S1 have card D1 = card D – 1 by (simp add: card-Diff-singleton)

with S2 have card-d1: card D1 = n by auto
from d1-finite card-d1 have finite D1 ∧ card D1 = n by auto
with A-n have S3: nat-to-set (set-to-nat D1) = D1 by auto
def u1-def: u1 ≡ set-to-nat D1
from S1 m-in have sum (λ x. 2^n) D = 2^n + sum (λ x. 2^n) (D – {m}) by (rule sum.remove)
with set-to-nat-def have set-to-nat D = 2^n + set-to-nat (D – {m}) by auto

with u1-def u1-def D1-def have u-u1: u = u1 + 2^n m by auto
from S3 u1-def have d1-u1: nat-to-set u1 = D1 by auto
have u1-lt: u1 < 2^n m
proof
have L1: D1 ⊆ {i. i < m}
proof fix x
assume A1: x ∈ D1
show x ∈ {i. i < m}
proof
from A1 D1-def have L1-1: x ∈ D by auto
from S1 D-ne L1-1 m-def have L1-2: x ≤ m by auto
with A1 L1-1 D1-def have x ≠ m by auto
with L1-2 show x < m by auto
qed
qed
have L2: finite {i. i < m} by (rule finite-interval)
from L2 L1 have set-to-nat D1 ≤ set-to-nat {i. i < m} by (rule set-to-nat-mono)
with u1-def have u1 ≤ set-to-nat {i. i < m} by auto
with set-to-nat-of-interval have L3: u1 ≤ 2^n m – 1 by auto
have 0 < (2^n) ^ m by auto
then have (2^n) ^ m – 1 < (2^n) ^ m by auto
with L3 show ?thesis by arith
qed
from u-def have nat-to-set (set-to-nat D) = nat-to-set u by auto
also from u-u1 have ... = nat-to-set (u1 + 2^n m) by auto
also from u1-lt have ... = nat-to-set u1 ∪ {m} by (rule add-power)
also from d1-u1 have ... = D1 ∪ {m} by auto
also from D1-def m-in have ... = D by auto
finally show nat-to-set (set-to-nat D) = D by auto
qed
qed
from P-at-0 P-at-Suc have main: ∃ n. ?P n by (rule nat.induct)
from A main show ?thesis by auto

qed

theorem nat-to-set-srj1: finite (D::nat set) ==> \exists u. nat-to-set u = D

proof

assume A: finite D

show \exists u. nat-to-set u = D

proof

from A show nat-to-set (set-to-nat D) = D by (rule nat-to-set-srj)

qed

qed

lemma sum-of-pr-is-pr: g \in PrimRec1 ==> (\lambda n. sum g \{i. i<n\}) \in PrimRec1

proof

assume g-is-pr: g \in PrimRec1

def f-def: f \equiv \lambda n. sum g \{i. i<n\}

from f-def have f-at-0: f 0 = 0 by auto

def h-def: h \equiv \lambda a (b::nat). (g a) + b

from g-is-pr have h-is-pr: h \in PrimRec2 unfolding h-def by prec

have f-at-Suc: \forall y. f (Suc y) = h y (f y)

proof

fix y show f (Suc y) = h y (f y)

proof

from f-def have S1: f (Suc y) = sum g \{i. i<Suc y\} by auto

have S2: \{i. i<Suc y\} = \{i. i<y\} \cup \{y\} by auto

have S3: finite \{i. i<y\} by (rule finite-interval)

have S4: y \notin \{i. i<y\} by auto

from S1 S2 have f (Suc y) = sum g \{(i. (i::nat)<y) \cup \{y\}\} by auto

also from S3 S4 sum.insert have ... = g y + sum g \{i. i<y\} by auto

also from f-def have ... = g y + f y by auto

also from h-def have ... = h y (f y) by auto

finally show ?thesis by auto

qed

qed

from h-is-pr f-at-0 f-at-Suc have f-is-pr: f \in PrimRec1 by (rule pr-rec1-scheme)

with f-def show ?thesis by auto

qed

lemma sum-of-pr-is-pr2: p \in PrimRec2 ==> (\lambda n m. sum (\lambda x. p x m) \{i. i<n\}) \in PrimRec2

proof

assume p-is-pr: p \in PrimRec2

def f-def: f \equiv \lambda n m. sum (\lambda x. p x m) \{i. i<n\}

def g-def: g \equiv \lambda x::nat. (0::nat)

have g-is-pr: g \in PrimRec1 by (unfold g-def, rule const-is-pr [where ?n=0])

have f-at-0: \forall x. f 0 x = g x

proof

fix x from f-def g-def show f 0 x = g x by auto

qed

89
def h-def: h ≡ λ a (h::nat) e. (p a c) + b
from p-is-pr have h-is-pr: h ∈ PrimRec3 unfolding h-def by prec
have f-at-Suc: ∀ x y. f (Suc y) x = h y (f y x) x
proof (rule allI, rule allI)
  fix x y show f (Suc y) x = h y (f y x) x
proof
  from f-def have S1: f (Suc y) x = sum (λ z. p z x) {i. i < Suc y} by auto
  have S2: {i. i < Suc y} = {i. i < y} ∪ {y} by auto
  have S3: finite {i. i < y} by (rule finite-interval)
  have S4: y ∉ {i. i < y} by auto
  def g1-def: g1 ≡ (λ z. p z x)
  from S1 S2 g1-def have f (Suc y) x = sum g1 (({i. (i::nat) < y} ∪ {y}) by auto
    also from S3 S4 sum.insert have ... = g1 y + sum g1 {i. i < y} by auto
    also from f-def g1-def have ... = h y (f y x) x by auto
    finally show ?thesis by auto
qed
qed
from g-is-pr h-is-pr f-at-0 f-at-Suc have f-is-pr: f ∈ PrimRec2 by (rule pr-rec-scheme)
with f-def show ?thesis by auto
qed

lemma sum-is-pr: g ∈ PrimRec1 ⊢ (λ u. sum g (nat-to-set u)) ∈ PrimRec1
proof
  assume g-is-pr: g ∈ PrimRec1
  def g1-def: g1 ≡ λ x u. if (c-in x u = 1) then (g x) else 0
  have g1-is-pr: g1 ∈ PrimRec2
  proof (unfold g1-def, rule if-eq-is-pr2)
    show c-in ∈ PrimRec2 by (rule c-in-is-pr)
  next
    show (λ x y. 1) ∈ PrimRec2 by (rule const-is-pr-2 [where ?n=1])
  next
  from g-is-pr show (λ x y. g x) ∈ PrimRec2 by prec
next
  show (λ x y. 0) ∈ PrimRec2 by (rule const-is-pr-2 [where ?n=0])
qed
def f-def: f ≡ λ u. sum (λ x. f1 x u) {i. (i::nat) < u}
def f1-def: f1 ≡ λ u. sum (λ x. f1 x u) {i. (i::nat) < u}
from g1-is-pr have (λ u::nat) v. sum (λ x. f1 x v) {i. (i::nat) < u}) ∈ PrimRec2
by (rule sum-of-pr-is-pr)
with f1-def have f1-is-pr: f1 ∈ PrimRec2 by auto
from f-def f1-def have f-f1: f = (λ u. f1 u v) by auto
from f1-is-pr have (λ u. f1 u v) ∈ PrimRec1 by prec
with f-f1 have f-is-pr: f ∈ PrimRec1 by auto
have f-is-result: f = (λ u. sum g (nat-to-set u))
proof
  fix u show f u = sum g (nat-to-set u)
  proof

def U-def: \( U \equiv \{ i. i < u \} \)
def A-def: \( A \equiv \{ x \in U. \text{c-in } x u = 1 \} \)
def B-def: \( B \equiv \{ x \in U. \text{c-in } x u \neq 1 \} \)

have U-finite: finite \( U \) by (unfold U-def, rule finite-interval)
from A-def U-finite have A-finite: finite \( A \) by auto
from B-def U-finite have B-finite: finite \( B \) by auto
from U-def A-def B-def have U-A-B: \( U = A \cup B \) by auto
from A-def B-def have A-B: \( A \cap B = \{ \} \) by auto
from U-def A-def B-def have U-A-B: \( U = A \cup B \) by auto
from A-def B-def have A-B: \( A \cap B = \{ \} \) by auto
from g1-def have B-z: \( \sum (\lambda x. g1 x u) B = 0 \) by auto
have u-in-U: \( \text{nat-to-set } u \subseteq U \) by (unfold U-def, rule nat-to-set-upper-bound2)
from A-def have A: \( A = \text{nat-to-set } u \) by auto
from A-u have A-res: \( \sum (\lambda x. g1 x u) A = \sum g (\text{nat-to-set } u) \) by auto
finally show ?thesis by auto

definition
c-card :: nat \( \Rightarrow \) nat where
c-card = (\( \lambda u. \text{card } (\text{nat-to-set } u) \))

theorem c-card-is-pr: c-card \( \in \) PrimRec1
proof –
def g-def: \( g \equiv \lambda (x::nat). (1::nat) \)
have g-is-pr: \( g \in \text{PrimRec1} \) by (unfold g-def, rule const-is-pr)
have c-card = (\( \lambda u. \sum g (\text{nat-to-set } u) \))
proof
fix \( u \) show c-card u = \( \sum g (\text{nat-to-set } u) \) by (unfold c-card-def, unfold g-def, rule card-eq-sum)
qed
moreover from g-is-pr have \( (\lambda u. \sum g (\text{nat-to-set } u)) \in \text{PrimRec1} \) by (rule sum-is-pr)
ultimately show ?thesis by auto
qed

definition
c-insert :: nat \( \Rightarrow \) nat \( \Rightarrow \) nat where
c-insert = (\( \lambda x u. \text{if c-in } x u = 1 \) then \( u \) else \( u + 2^x \))

lemma c-insert-is-pr: c-insert \( \in \) PrimRec2
proof (unfold c-insert-def, rule if-eq-is-pr2)
  show c-in ∈ PrimRec2 by (rule c-in-is-pr)
next
  show (λx y. 1) ∈ PrimRec2 by (rule const-is-pr-2)
next
  show (λx y. y) ∈ PrimRec2 by (rule pr-id2-2)
next
  from power-is-pr show (λx y. y + 2 ^ x) ∈ PrimRec2 by prec
qed

lemma [simp]: set-to-nat (nat-to-set u) = u
proof –
  def D-def: D ≡ nat-to-set u
  from D-def nat-to-set-is-finite have D-finite: finite D by auto
  with D-def have nat-to-set (set-to-nat D) = D by (rule nat-to-set-srj)
  with D-def show ?thesis by auto
qed

lemma insert-lemma: x /∈ nat-to-set u ⇒ set-to-nat (nat-to-set u ∪ {x}) = u + 2 ^ x
proof –
  assume A: x /∈ nat-to-set u
  def D-def: D ≡ nat-to-set u
  have finite (nat-to-set u) by (rule nat-to-set-is-finite)
  with D-def have D-finite: finite D by auto
  let ?f = λ (x::nat). (2::nat) ^ x
  from set-to-nat-def have set-to-nat (D ∪ {x}) = sum ?f (D ∪ {x}) by auto
  also from D-finite S1 have ... = ?f x + sum ?f D by simp
  also from set-to-nat-def have ... = 2 ^ x + set-to-nat D by auto
  finally have set-to-nat (D ∪ {x}) = set-to-nat D + 2 ^ x by auto
  with D-def show ?thesis by auto
qed

lemma c-insert-df: c-insert = (λ x u. set-to-nat ((nat-to-set u) ∪ {x}))
proof (rule ext, rule ext)
  fix x u show c-insert x u = set-to-nat (nat-to-set u ∪ {x})
  proof (cases)
    assume A: x ∈ nat-to-set u
    then have nat-to-set u ∪ {x} = nat-to-set u by auto
    then have S1: set-to-nat (nat-to-set u ∪ {x}) = u by auto
    from A have c-in x u = 1 by (simp add: x-in-u-eq)
    then have c-insert x u = u by (unfold c-insert-def, simp)
    with S1 show ?thesis by auto
  next
    assume A: x /∈ nat-to-set u
    then have S1: c-in x u ≠ 1 by (simp add: x-in-u-eq)

92
then have S2: c-insert x u = u + 2 ^ x by (unfold c-insert-def, simp)
from A have set-to-nat (nat-to-set u ∪ {x}) = u + 2 ^ x by (rule insert-lemma)
with S2 show ?thesis by auto
qed
qed

definition
c-remove :: nat ⇒ nat ⇒ nat where
c-remove = (λ x u. if c-in x u = 0 then u else u − 2 ^ x)

lemma c-remove-is-pr: c-remove ∈ PrimRec2
proof (unfold c-remove-def, rule if-eq-is-pr2)
  show c-in ∈ PrimRec2 by (rule c-in-is-pr)
next
  show (λ x y. 0) ∈ PrimRec2 by (rule const-is-pr-2)
next
  show (λ x y. y) ∈ PrimRec2 by (rule pr-id2-2)
next
  from power-is-pr show (λ x y. y − 2 ^ x) ∈ PrimRec2 by prec
qed

lemma remove-lemma: x ∈ nat-to-set u ⇒ set-to-nat (nat-to-set u − {x}) = u − 2 ^ x
proof –
  assume A: x ∈ nat-to-set u
  def D-def: D ≡ nat-to-set u − {x}
  from A D-def have S1: x /∈ D by auto
  have finite (nat-to-set u) by (rule nat-to-set-is-finite)
  with D-def have D-finite: finite D by auto
  let ?f = (λ x :: nat). (2 :: nat) ^ x
  from set-to-nat-def have set-to-nat (D ∪ {x}) = sum ?f (D ∪ {x}) by auto
  also from D-finite S1 have . . . = ?f x + sum ?f D by simp
  also from set-to-nat-def have . . . = 2 ^ x + set-to-nat D by auto
  finally have S2: set-to-nat (D ∪ {x}) = set-to-nat D + 2 ^ x by auto
  from A D-def have D ∪ {x} = nat-to-set u by auto
  with S2 have S3: u = set-to-nat D + 2 ^ x by auto
  from A have S4: 2 ^ x ≤ u by (rule nat-to-set-upper-bound)
  with S3 D-def show ?thesis by auto
qed

lemma c-remove-df: c-remove = (λ x u. set-to-nat ((nat-to-set u) − {x}))
proof (rule ext, rule ext)
  fix x u show c-remove x u = set-to-nat (nat-to-set u − {x})
  proof (cases)
    assume A: x ∈ nat-to-set u
    then have S1: c-in x u = 1 by (simp add: x-in-u-eq)
    then have S2: c-remove x u = u − 2 ^ x by (simp add: c-remove-def)
    from A have set-to-nat (nat-to-set u − {x}) = u − 2 ^ x by (rule remove-lemma)
    with S2 show ?thesis by auto
  qed
next

assume A: x \notin nat-to-set u
then have S1: c-in x u \neq 1 by (simp add: x-in-u-eq)
then have S2: c-remove x u = u by (simp add: c-remove-def c-in-def)
from A have nat-to-set u - \{x\} = nat-to-set u by auto
with S2 show ?thesis by auto
qed
qed

definition
c-union :: nat \Rightarrow nat \Rightarrow nat where
c-union = (\lambda u v. set-to-nat (nat-to-set u \cup nat-to-set v))

theorem c-union-is-pr: c-union \in PrimRec2
proof –
def f-def: f \equiv \lambda y x. set-to-nat ((nat-to-set (c-fst x)) \cup \{z \in nat-to-set (c-snd x). z < y\})
have f-is-pr: f \in PrimRec2
proof –
def g-def: g \equiv c-fst
from c-fst-is-pr g-def have g-is-pr: g \in PrimRec1 by auto
def h-def: h \equiv \lambda a b c. if c-in a (c-snd c) = 1 then c-insert a b else b
from c-in-is-pr c-insert-is-pr have h-is-pr: h \in PrimRec3 unfolding h-def by prec
have f-at-0: \forall x. f 0 x = g x
proof
fix x show f 0 x = g x by (unfold f-def, unfold g-def, simp)
qed
have f-at-Suc: \forall x y. f (Suc y) x = h y (f y x) x
proof (rule allI, rule allI)
fix x y show f (Suc y) x = h y (f y x) x
proof (cases)
assume A: c-in y (c-snd x) = 1
then have S1: y \in (nat-to-set (c-snd x)) by (simp add: x-in-u-eq)
from A h-def have S2: h y (f y x) x = c-insert y (f y x) by auto
from S1 have S3: \{z \in nat-to-set (c-snd x). z < Suc y\} = \{z \in nat-to-set (c-snd x). z < y\} \cup \{y\} by auto
from nat-to-set-is-finite have S4: finite ((nat-to-set (c-fst x)) \cup \{z \in nat-to-set (c-snd x). z < y\}) by auto
with nat-to-set-srj f-def have S5: nat-to-set (f y x) = (nat-to-set (c-fst x)) \cup \{z \in nat-to-set (c-snd x). z < y\} by auto
from f-def have S6: f (Suc y) z = set-to-nat ((nat-to-set (c-fst x)) \cup \{z \in nat-to-set (c-snd x). z < Suc y\}) by simp
also from S3 have \ldots = set-to-nat ((nat-to-set (c-fst x)) \cup \{z \in nat-to-set (c-snd x). z < y\} \cup \{y\}) by auto
also from S5 have \ldots = set-to-nat (f y x) by auto
also have \ldots = c-insert y (f y x) by (simp add: c-insert-df)
finally show ?thesis by (simp add: S2)
next

94
assume $A: \neg c\text{-}in \, y \, (c\text{-}snd \, x) = 1$
then have $S1: y \notin (\text{nat-to-set} \, (c\text{-}snd \, x))$ by (simp add: $x\text{-}in\text{-}u\text{-}eq$
from $A$ $h\text{-}def$ have $S2: h \, (f \, y \, x) \, x = f \, y \, x$ by auto
have $S3: \{ z \in \text{nat-to-set} \, (c\text{-}snd \, x), \, z < \text{Suc} \, y \} = \{ z \in \text{nat-to-set} \, (c\text{-}snd \, x), \, z < y \}$
proof
have $\{ z \in \text{nat-to-set} \, (c\text{-}snd \, x), \, z < \text{Suc} \, y \} \cup \{ z \in \text{nat-to-set} \, (c\text{-}snd \, x), \, z = y \}$
by auto
with $S1$ show ?thesis by auto
qed
from $\text{nat-to-set}-\text{is}\text{-}finite$ have $S4: \text{finite} ((\text{nat-to-set} \, (c\text{-}fst \, x)) \cup \{ z \in \text{nat-to-set} \, (c\text{-}snd \, x), \, z < y \})$ by auto
with $c\text{-}union\text{-}def$ have $c\text{-}union = \text{union}$ by simp
finally show ?thesis by (simp add: $S2$)
qed
definition
$c\text{-}diff :: \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat}$
where
$c\text{-}diff = (\lambda u \, v. \, \text{set-to-nat} \, (\text{nat-to-set} \, u - \text{nat-to-set} \, v))$
theorem $c\text{-}diff\text{-}is\text{-}pr: \ c\text{-}diff \in \text{PrimRec2}$
proof –
\begin{verbatim}
def f-def: f \equiv \lambda y \ x. set-to-nat ((\text{nat-to-set} (c-fst x)) - \{ z \in \text{nat-to-set} (c-snd x), z < y \})

have f-is-pr: f \in \text{PrimRec2}
proof -
def g-def: g \equiv c-fst
from c-fst-is-pr g-def have g-is-pr: g \in \text{PrimRec1} by auto
def h-def: h \equiv \lambda a \ b \ c. \text{if } c-in \ a \ (c-snd \ c) = 1 \text{ then } c-remove \ a \ b \ \text{ else } b
from c-in-is-pr c-remove-is-pr have h-is-pr: h \in \text{PrimRec3} unfolding h-def
by prec
have f-at-0: \forall x. f \ 0\ x = \ 0\ x
proof
  fix x show f \ 0\ x = \ 0\ x by (unfold f-def, unfold g-def, simp)
qed
have f-at-Suc: \forall x \ y. f \ Suc \ y \ x = h \ y \ (f \ y \ x) \ x
proof (rule allI, rule allI)
  fix x y show f \ Suc \ y \ x = h \ y \ (f \ y \ x) \ x
proof (cases)
  assume A: c-in \ y \ (c-snd \ x) = 1
  then have S1: y \in \text{nat-to-set} (c-snd \ x) by (simp add: x-in-u-eq)
  from A h-def have S2: h \ y \ (f \ y \ x) \ x = c-remove \ y \ (f \ y \ x) \ by auto
  have (nat-to-set (c-fst x)) - \{ z \in \text{nat-to-set} (c-snd \ x), z < y \} \cup \{ y \} =
    (\text{nat-to-set} (c-fst x)) - (\{ z \in \text{nat-to-set} (c-snd \ x), z < y \}) - \{ y \} by auto
  then have lm1: set-to-nat (nat-to-set (c-fst x)) - (\{ z \in \text{nat-to-set} (c-snd \ x), z < y \} \cup \{ y \}) =
    set-to-nat (nat-to-set (c-fst x)) - \{ z \in \text{nat-to-set} (c-snd \ x), z < y \} \cup \{ y \} by auto
  from S1 have S3: \{ z \in \text{nat-to-set} (c-snd \ x), z < Suc \ y \} = \{ z \in \text{nat-to-set} (c-snd \ x), z < y \} \cup \{ y \} by auto
  from nat-to-set-is-finite have S4: finite ((nat-to-set (c-fst x)) - \{ z \in \text{nat-to-set} (c-snd \ x), z < y \}) by auto
  with nat-to-set-srj f-def have S5: nat-to-set (f \ y \ x) = (nat-to-set (c-fst x)) - \{ z \in \text{nat-to-set} \ (c-snd x), z < y \} by auto
  from f-def have S6: f \ Suc \ y \ x = set-to-nat ((nat-to-set (c-fst x)) - \{ z \in \text{nat-to-set} \ (c-snd x), z < Suc \ y \}) by simp
  also from S3 have \ldots = set-to-nat ((nat-to-set (c-fst x)) - \{ z \in \text{nat-to-set} (c-snd \ x), z < y \} \cup \{ y \}) by auto
  also have \ldots = set-to-nat ((nat-to-set (c-fst x)) - (\{ z \in \text{nat-to-set} (c-snd \ x), z < y \} \cup \{ y \}) by (rule lm1)
  also from S5 have \ldots = set-to-nat (nat-to-set (f \ y \ x) - \{ y \}) by auto
  also have \ldots = c-remove \ y \ (f \ y \ x) by (simp add: c-remove-df)
  finally show \?thesis by (simp add: S2)
next
  assume A: \neg c-in \ y \ (c-snd \ x) = 1
  then have S1: y \notin \text{nat-to-set} (c-snd \ x) by (simp add: x-in-u-eq)
  from A h-def have S2: h \ y \ (f \ y \ x) \ x = f \ y \ x by auto
  have S3: \{ z \in \text{nat-to-set} (c-snd \ x), z < Suc \ y \} = \{ z \in \text{nat-to-set} (c-snd \ x), z < y \}
  proof -

\end{verbatim}
have \{ z \in \text{nat-to-set} \ (c\text{-snd} \ x). \ z < \text{Suc} \ y \} = \{ z \in \text{nat-to-set} \ (c\text{-snd} \ x), \ z = y \}
by auto

with S1 show \textit{thesis} by auto
qed

from \text{nat-to-set-is-finite} have S4: \text{finite} ((\text{nat-to-set} \ (c\text{-fst} \ x)) - \{ z \in \text{nat-to-set} \ (c\text{-snd} \ x), \ z < y \}) by auto

with \text{nat-to-set-srj} f-def have S5: \text{nat-to-set} \ (f \ y \ x) = (\text{nat-to-set} \ (c\text{-fst} \ x)) - \{ z \in \text{nat-to-set} \ (c\text{-snd} \ x), \ z < \text{Suc} \ y \} by simp

also from S3 have \ldots = \text{set-to-nat} (((\text{nat-to-set} \ (c\text{-fst} \ x)) - \{ z \in \text{nat-to-set} \ (c\text{-snd} \ x), \ z < y \})) by auto

also from S5 have \ldots = \text{set-to-nat} (\text{nat-to-set} \ (f \ y \ x)) by auto

also have \ldots = f \ y \ x by simp

finally show \textit{thesis} by (simp add: S2)
qed

def \textit{diff-def}: \textit{diff} \equiv \lambda u v. f \ v \ (c\text{-pair} u v)
from \text{f-is-pr} have \textit{diff-is-pr}: \textit{diff} \in \text{PrimRec2}
unfolding \textit{diff-def} by prec

have \bigwedge u v. \textit{diff} \ u \ v = \text{set-to-nat} (\text{nat-to-set} \ u - \text{nat-to-set} \ v)
proof -
  fix u v show \textit{diff} \ u \ v = \text{set-to-nat} (\text{nat-to-set} \ u - \text{nat-to-set} \ v)
  proof -
    from \text{nat-to-set-upper-bound1} have \{ z \in \text{nat-to-set} \ v, \ z < v \} = \text{nat-to-set} \ v
    by auto
    with \textit{diff-def} \textit{f-def} show \textit{thesis} by auto
  qed
  qed

then have \textit{diff} = (\lambda u v. \text{set-to-nat} (\text{nat-to-set} \ u - \text{nat-to-set} \ v)) by (simp add: ext)

with \textit{c-diff-def} have \textit{c-diff} = \textit{diff} by simp
with \textit{diff-is-pr} show \textit{thesis} by simp
qed

definition \textit{c-intersect} :: \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} where
\textit{c-intersect} = (\lambda u v. \text{set-to-nat} (\text{nat-to-set} \ u \cap \text{nat-to-set} \ v))

theorem \textit{c-intersect-is-pr}: \textit{c-intersect} \in \text{PrimRec2}
proof -
def \textit{f-def}: f \equiv \lambda u v. \textit{c-diff} \ (c\text{-union} u v) \ (c\text{-union} \ (c\text{-diff} \ u \ v) \ (c\text{-diff} \ v \ u))
from \textit{c-diff-is-pr} \textit{c-union-is-pr} have \textit{f-is-pr}: f \in \text{PrimRec2}
unfolding \textit{f-def} by prec

have \bigwedge u v. f \ u \ v = \textit{c-intersect} \ u \ v
proof -
  fix u v show f \ u \ v = \textit{c-intersect} \ u \ v
proof

let ?A = nat-to-set u
let ?B = nat-to-set v

have A-fin: finite ?A by (rule nat-to-set-is-finite)
have B-fin: finite ?B by (rule nat-to-set-is-finite)

have S1: c-union u v = set-to-nat (?A ∪ ?B) by (simp add: c-union-def)
have S2: c-diff u v = set-to-nat (?A - ?B) by (simp add: c-diff-def)
have S3: c-diff v u = set-to-nat (?B - ?A) by (simp add: c-diff-def)

from S2 A-fin B-fin have S4: nat-to-set (c-diff u v) = ?A - ?B by (simp add: nat-to-set-srj)

from S3 A-fin B-fin have S5: nat-to-set (c-diff v u) = ?B - ?A by (simp add: nat-to-set-srj)

from S4 S5 have S6: c-union (c-diff u v) (c-diff v u) = set-to-nat ((?A - ?B) ∪ (?B - ?A)) by (simp add: c-union-def)

from S1 A-fin B-fin have S7: nat-to-set (c-union u v) = ?A ∪ ?B by (simp add: nat-to-set-srj)

from S6 A-fin B-fin have S8: nat-to-set (c-union (c-diff u v) (c-diff v u)) = (?A - ?B) ∪ (?B - ?A) by (simp add: nat-to-set-srj)

from S7 S8 have S9: f u v = set-to-nat ((?A ∪ ?B) - ((?A - ?B) ∪ (?B - ?A)) by auto

with S9 have S11: f u v = set-to-nat (?A ∩ ?B) by auto

have c-intersect u v = set-to-nat (?A ∩ ?B) by (simp add: c-intersect-def)

with S11 show ?thesis by auto

qed

then have f = c-intersect by (simp add: ext)
with f-is-pr show ?thesis by auto

qed

end

6  The function which is universal for primitive recursive functions of one variable

definition g-comp :: nat ⇒ nat ⇒ nat where

g-comp c-ls key = (let n = c-fst key; x = c-snd key; m = c-snd n;
m1 = c-fst m; m2 = c-snd m in
(* We have key = <n, x>; n = <?, m>; m = <m1, m2>. *)
if c-assoc-have-key c-ls (c-pair m2 x) = 0 then
(let y = c-assoc-value c-ls (c-pair m2 x) in
  if c-assoc-have-key c-ls (c-pair m1 y) = 0 then
    (let z = c-assoc-value c-ls (c-pair m1 y) in
      c-cons (c-pair key z) c-ls)
  else c-ls
)
else c-ls
)

definition
g-pair :: nat ⇒ nat ⇒ nat where
g-pair c-ls key = (let n = c-fst key; x = c-snd key; m = c-snd n;
m1 = c-fst m; m2 = c-snd m in
  (* We have key = <n, x>; n = <?, m>; m = <m1, m2>. *)
  if c-assoc-have-key c-ls (c-pair m1 x) = 0 then
    (let y1 = c-assoc-value c-ls (c-pair m1 x) in
      if c-assoc-have-key c-ls (c-pair m2 x) = 0 then
        (let y2 = c-assoc-value c-ls (c-pair m2 x) in
          c-cons (c-pair key (c-pair y1 y2)) c-ls)
        else c-ls
      )
    else c-ls
  )
)

definition
g-rec :: nat ⇒ nat ⇒ nat where
g-rec c-ls key = (let n = c-fst key; x = c-snd key; m = c-snd n;
m1 = c-fst m; m2 = c-snd m; y1 = c-fst x; x1 = c-snd x in
  (* We have key = <n, x>; n = <?, m>; m = <m1, m2>; x = <y1, x1>. *)
  if y1 = 0 then
    (if c-assoc-have-key c-ls (c-pair m1 x1) = 0 then
      c-cons (c-pair key (c-assoc-value c-ls (c-pair m1 x1))) c-ls
    else c-ls
  )
else
  (let y2 = y1 - (1::nat) in
    if c-assoc-have-key c-ls (c-pair n (c-pair y2 x1)) = 0 then
      (let t1 = c-assoc-value c-ls (c-pair n (c-pair y2 x1)); t2 = c-pair (c-pair y2 t1) x1 in
        if c-assoc-have-key c-ls (c-pair m2 t2) = 0 then
          c-cons (c-pair key (c-assoc-value c-ls (c-pair m2 t2))) c-ls
        else c-ls
      )
    else c-ls
  )
)
else c-ls
)
\[
g-step :: \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat}
\]
\[
\text{where}
\]
\[
g-step \text{ c-ls key} = (\text{let } n = \text{c-fst key}; x = \text{c-snd key}; \text{ n1} = (\text{c-fst n}) \mod 7 \text{ in}
\]
\[
\text{if } n1 = 0 \text{ then } \text{c-cons (c-pair key 0) c-ls else}
\]
\[
\text{if } n1 = 1 \text{ then } \text{c-cons (c-pair key (Suc x)) c-ls else}
\]
\[
\text{if } n1 = 2 \text{ then } \text{c-cons (c-pair key (c-fst x)) c-ls else}
\]
\[
\text{if } n1 = 3 \text{ then } \text{c-cons (c-pair key (c-snd x)) c-ls else}
\]
\[
\text{if } n1 = 4 \text{ then } \text{g-comp c-ls key else}
\]
\[
\text{if } n1 = 5 \text{ then } \text{g-pair c-ls key else}
\]
\[
\text{if } n1 = 6 \text{ then } \text{g-rec c-ls key else}
\]
c-ls
\]

\[
definition
pr-gr :: \text{nat} \Rightarrow \text{nat}
\]
\[
\text{where}
\]
\[
pr-gr-def: pr-gr = \text{PrimRecOp1 0 (λ a b. g-step b (c-fst a))}
\]

\[
lemma pr-gr-at-0: pr-gr 0 = 0 \text{ by (simp add: pr-gr-def)}
\]

\[
lemma pr-gr-at-Suc: pr-gr (Suc x) = g-step (pr-gr x) (c-fst x) \text{ by (simp add: pr-gr-def)}
\]

\[
definition
univ-for-pr :: \text{nat} \Rightarrow \text{nat}
\]
\[
\text{where}
\]
\[
univ-for-pr = \text{pr-conv-2-to-1 nat-to-pr}
\]

\[
theorem \text{univ-is-not-pr}: \text{univ-for-pr} / \notin \text{PrimRec1}
\]
\[
\text{proof (rule ccontr)}
\]
\[
\text{assume } \neg \text{univ-for-pr} / \notin \text{PrimRec1 then have A1: univ-for-pr} \in \text{PrimRec1 by simp}
\]
\[
\text{let } \exists f = \lambda n. \text{univ-for-pr (c-pair n n) + 1}
\]
\[
\text{let } ?n0 = \text{index-of-pr } ?f
\]
\[
\text{from A1 have S1: } \exists f \in \text{PrimRec1 by prec}
\]
\[
\text{then have S2: } \text{nat-to-pr } ?n0 = ?f \text{ by (rule index-of-pr-is-real)}
\]
\[
\text{then have S3: } \text{nat-to-pr } ?n0 ?n0 = ?f ?n0 \text{ by simp}
\]
\[
\text{have S4: } ?f ?n0 = \text{univ-for-pr (c-pair } ?n0 ?n0) + 1 \text{ by simp}
\]
\[
\text{from S3 S4 show False by (simp add: univ-for-pr-def pr-conv-2-to-1-def)}
\]

\[
\text{qed}
\]

\[
definition
c-is-sub-fun :: \text{nat} \Rightarrow (\text{nat} \Rightarrow \text{nat}) \Rightarrow \text{bool}
\]
\[
\text{where}
\]
\[
c-is-sub-fun \text{ ls f} \longleftrightarrow (\forall x. \text{c-assoc-have-key ls x = 0 } \longrightarrow \text{c-assoc-value ls x = f x})
\]

\[
lemma \text{c-is-sub-fun-lm-1: } [ \text{c-is-sub-fun ls f; c-assoc-have-key ls x = 0 } ] \implies
\]

100
c-assoc-value ls x = f x
apply(unfold c-is-sub-fun-def)
apply(auto)
done

lemma c-is-sub-fun-lm-2: c-is-sub-fun ls f \implies c-is-sub-fun (c-cons (c-pair x (f x)) ls) f
proof –
assume A1: c-is-sub-fun ls f
show \thesis
proof (unfold c-is-sub-fun-def, rule allI, rule impI)
fix xa assume A2: c-assoc-have-key (c-cons (c-pair x (f x)) ls) xa = 0 show 
c-assoc-value (c-cons (c-pair x (f x)) ls) xa = f xa
proof cases
assume C1: xa = x
  then show c-assoc-value (c-cons (c-pair x (f x)) ls) xa = f xa by (simp add: PRecList.c-assoc-lm-2)
next
assume C2: \neg xa = x
  then have S1: c-assoc-have-key (c-cons (c-pair x (f x)) ls) xa = c-assoc-have-key 
ls xa by (rule c-assoc-lm-3)
  from C2 have S2: c-assoc-value (c-cons (c-pair x (f x)) ls) xa = c-assoc-value 
ls xa by (rule c-assoc-lm-4)
  from A2 S1 have S3: c-assoc-have-key ls xa = 0 by simp
  from A1 S3 have c-assoc-value ls xa = f xa by (rule c-is-sub-fun-lm-1)
  with S2 show \thesis by simp
qed
qed

lemma mod7-lm: (n::nat) mod 7 = 0 \lor 
  (n::nat) mod 7 = 1 \lor 
  (n::nat) mod 7 = 2 \lor 
  (n::nat) mod 7 = 3 \lor 
  (n::nat) mod 7 = 4 \lor 
  (n::nat) mod 7 = 5 \lor 
  (n::nat) mod 7 = 6 by arith

lemma nat-to-sch-at-pos: x > 0 \implies nat-to-sch x = (let u=(c-fst x) mod 7; 
v=c-snd x; v1=c-fst v; v2 = c-snd v; sch1=nat-to-sch v1; sch2=nat-to-sch v2 
in loc-f u sch1 sch2)
proof –
assume A: x > 0
show \thesis
proof cases
assume A1: x = 1
then have S1: c-fst x = 0
proof –
  have 1 = c-pair 0 1 by (simp add: c-pair-def sf-def)
then have \( c\text{-fst} \ 1 = c\text{-fst} \ (c\text{-pair} \ 0 \ 1) \) by simp
then have \( c\text{-fst} \ 1 = 0 \) by simp
with A1 show \(?thesis\) by simp
qed
from A1 have S2: \( \text{nat-to-sch} \ x = \text{Base-zero} \) by simp
from S1 S2 show \( \text{nat-to-sch} \ x = (\text{let} \ u=(c\text{-fst} \ x) \ \text{mod} \ 7; \ v=c\text{-snd} \ x; \ v1=c\text{-fst} \ v; \ v2 = c\text{-snd} \ v; \ sch1=\text{nat-to-sch} \ v1; \ sch2=\text{nat-to-sch} \ v2 \ \text{in} \ \text{loc-f} \ u \ sch1 \ sch2) \) by (rule \text{nat-to-sch-at-pos})
apply(insert S1 S2)
apply(simp add: simp add: \text{Let-def} \ \text{loc-f-def})
done
next
assume \( \neg \ x = 1 \)
from A this have A2: \( x > 1 \) by simp
from this have \( \text{nat-to-sch} \ x = (\text{let} \ u=(c\text{-fst} \ x) \ \text{mod} \ 7; \ v=c\text{-snd} \ x; \ v1=c\text{-fst} \ v; \ v2 = c\text{-snd} \ v; \ sch1=\text{nat-to-sch} \ v1; \ sch2=\text{nat-to-sch} \ v2 \ \text{in} \ \text{loc-f} \ u \ sch1 \ sch2) \) by (simp add: \text{mod7-def})
lemma \text{nat-to-sch-0}: \( c\text{-fst} \ n \ \text{mod} \ 7 = 0 \Rightarrow \text{nat-to-sch} \ n = \text{Base-zero} \)
proof –
assume A: \( c\text{-fst} \ n \ \text{mod} \ 7 = 0 \)
show \(?thesis\)
proof cases
assume \( n=0 \)
then show \( \text{nat-to-sch} \ n = \text{Base-zero} \) by simp
next
assume \( \neg \ n = 0 \) then have \( n > 0 \) by simp
then have \( \text{nat-to-sch} \ n = (\text{let} \ u=(c\text{-fst} \ n) \ \text{mod} \ 7; \ v=c\text{-snd} \ n; \ v1=c\text{-fst} \ v; \ v2 = c\text{-snd} \ v; \ sch1=\text{nat-to-sch} \ v1; \ sch2=\text{nat-to-sch} \ v2 \ \text{in} \ \text{loc-f} \ u \ sch1 \ sch2) \) by (rule \text{nat-to-sch-at-pos})
with A show \( \text{nat-to-sch} \ n = \text{Base-zero} \) by (simp add: simp add: \text{Let-def} \ \text{loc-f-def})
qed
qed

lemma \text{loc-lm-1}: \( c\text{-fst} \ n \ \text{mod} \ 7 \neq 0 \Rightarrow n > 0 \)
proof –
assume A: \( c\text{-fst} \ n \ \text{mod} \ 7 \neq 0 \)
have \( n = 0 \Rightarrow \text{False} \)
proof –
assume \( n = 0 \)
then have \( c\text{-fst} \ n \ \text{mod} \ 7 = 0 \) by (simp add: \text{c-fst-at-0})
with A show \(?thesis\) by simp
qed
then have \( \neg \ n = 0 \) by auto
then show \(?thesis\) by simp
qed

lemma loc-lm-2: c-fst n mod 7 ≠ 0 ⇒ nat-to-sch n = (let u=(c-fst n) mod 7; v=c-snd n; v1=c-fst v; v2 = c-snd v; sch1=nat-to-sch v1; sch2=nat-to-sch v2 in loc-f u sch1 sch2)
proof –
  assume c-fst n mod 7 ≠ 0
  then have n > 0 by (rule loc-lm-1)
  then show ?thesis by (rule nat-to-sch-at-pos)
qed

lemma nat-to-sch-1: c-fst n mod 7 = 1 ⇒ nat-to-sch n = Base-suc
proof –
  assume A1: c-fst n mod 7 = 1
  then have nat-to-sch n = (let u=(c-fst n) mod 7; v=c-snd n; v1=c-fst v; v2 = c-snd v; sch1=nat-to-sch v1; sch2=nat-to-sch v2 in loc-f u sch1 sch2) by (simp add: loc-lm-2)
  with A1 show nat-to-sch n = Base-suc by (simp add: Let-def loc-f-def)
qed

lemma nat-to-sch-2: c-fst n mod 7 = 2 ⇒ nat-to-sch n = Base-fst
proof –
  assume A1: c-fst n mod 7 = 2
  then have nat-to-sch n = (let u=(c-fst n) mod 7; v=c-snd n; v1=c-fst v; v2 = c-snd v; sch1=nat-to-sch v1; sch2=nat-to-sch v2 in loc-f u sch1 sch2) by (simp add: loc-lm-2)
  with A1 show nat-to-sch n = Base-fst by (simp add: Let-def loc-f-def)
qed

lemma nat-to-sch-3: c-fst n mod 7 = 3 ⇒ nat-to-sch n = Base-snd
proof –
  assume A1: c-fst n mod 7 = 3
  then have nat-to-sch n = (let u=(c-fst n) mod 7; v=c-snd n; v1=c-fst v; v2 = c-snd v; sch1=nat-to-sch v1; sch2=nat-to-sch v2 in loc-f u sch1 sch2) by (simp add: loc-lm-2)
  with A1 show nat-to-sch n = Base-snd by (simp add: Let-def loc-f-def)
qed

lemma nat-to-sch-4: c-fst n mod 7 = 4 ⇒ nat-to-sch n = Comp-op (nat-to-sch (c-fst (c-snd n))) (nat-to-sch (c-snd (c-snd n)))
proof –
  assume A1: c-fst n mod 7 = 4
  then have nat-to-sch n = (let u=(c-fst n) mod 7; v=c-snd n; v1=c-fst v; v2 = c-snd v; sch1=nat-to-sch v1; sch2=nat-to-sch v2 in loc-f u sch1 sch2) by (simp add: loc-lm-2)
  with A1 show nat-to-sch n = Comp-op (nat-to-sch (c-fst (c-snd n))) (nat-to-sch (c-snd (c-snd n))) by (simp add: Let-def loc-f-def)
qed

103
lemma nat-to-sch-5: c-fst n mod 7 = 5 ⇒ nat-to-sch n = Pair-op (nat-to-sch (c-fst (c-snd n))) (nat-to-sch (c-snd (c-snd n)))
proof
  assume A1: c-fst n mod 7 = 5
  then have nat-to-sch n = (let u=(c-fst n) mod 7; v=c-snd n; v1=c-fst v; v2 = c-snd v; sch1=nat-to-sch v1; sch2=nat-to-sch v2 in loc-f u sch1 sch2) by (simp add: loc-lm-2)
  with A1 show nat-to-sch n = Pair-op (nat-to-sch (c-fst (c-snd n))) (nat-to-sch (c-snd (c-snd n))) by (simp add: Let-def loc-f-def)
qed

lemma nat-to-sch-6: c-fst n mod 7 = 6 ⇒ nat-to-sch n = Rec-op (nat-to-sch (c-fst (c-snd n))) (nat-to-sch (c-snd (c-snd n)))
proof
  assume A1: c-fst n mod 7 = 6
  then have nat-to-sch n = (let u=(c-fst n) mod 7; v=c-snd n; v1=c-fst v; v2 = c-snd v; sch1=nat-to-sch v1; sch2=nat-to-sch v2 in loc-f u sch1 sch2) by (simp add: loc-lm-2)
  with A1 show nat-to-sch n = Rec-op (nat-to-sch (c-fst (c-snd n))) (nat-to-sch (c-snd (c-snd n))) by (simp add: Let-def loc-f-def)
qed

lemma nat-to-pr-lm-0: c-fst n mod 7 = 0 ⇒ nat-to-pr n x = 0
proof
  assume A: c-fst n mod 7 = 0
  have S1: nat-to-pr n x = sch-to-pr (nat-to-sch n) x by (simp add: nat-to-pr-def)
  from A have S2: nat-to-sch n = Base-zero by (rule nat-to-sch-0)
  from S1 S2 show ?thesis by simp
qed

lemma nat-to-pr-lm-1: c-fst n mod 7 = 1 ⇒ nat-to-pr n x = Suc x
proof
  assume A: c-fst n mod 7 = 1
  have S1: nat-to-pr n x = sch-to-pr (nat-to-sch n) x by (simp add: nat-to-pr-def)
  from A have S2: nat-to-sch n = Base-suc by (rule nat-to-sch-1)
  from S1 S2 show ?thesis by simp
qed

lemma nat-to-pr-lm-2: c-fst n mod 7 = 2 ⇒ nat-to-pr n x = c-fst x
proof
  assume A: c-fst n mod 7 = 2
  have S1: nat-to-pr n x = sch-to-pr (nat-to-sch n) x by (simp add: nat-to-pr-def)
  from A have S2: nat-to-sch n = Base-fst by (rule nat-to-sch-2)
  from S1 S2 show ?thesis by simp
qed

lemma nat-to-pr-lm-3: c-fst n mod 7 = 3 ⇒ nat-to-pr n x = c-snd x
proof
  assume A: c-fst n mod 7 = 3

104
have $S_1$: \(\text{nat-to-pr} \ n \ x = \text{sch-to-pr} \ (\text{nat-to-sch} \ n) \ x\) by (simp add: nat-to-pr-def)
from $A$ have $S_2$: \(\text{nat-to-sch} \ n = \text{Base-snd}\) by (rule nat-to-sch-3)
from $S_1 \ S_2$ show \(?thesis\) by simp

lemma \(\text{nat-to-pr-lm-4}\): \(\text{c-fst} \ n \mod 7 = 4 \implies \text{nat-to-pr} \ n \ x = (\text{nat-to-pr} \ (\text{c-fst} \ (\text{c-snd} \ n))) \ (\text{nat-to-pr} \ (\text{c-snd} \ (\text{c-snd} \ n))) \ x\)
proof –
  assume $A$: \(\text{c-fst} \ n \ mod 7 = 4\)
  have $S_1$: \(\text{nat-to-pr} \ n \ x = \text{sch-to-pr} \ (\text{nat-to-sch} \ n) \ x\) by (simp add: nat-to-pr-def)
  from $A$ have $S_2$: \(\text{nat-to-sch} \ n = \text{Comp-op} \ (\text{nat-to-sch} \ (\text{c-fst} \ (\text{c-snd} \ n))) \ (\text{nat-to-sch} \ (\text{c-snd} \ (\text{c-snd} \ n)))\) by (rule nat-to-sch-4)
  from $S_1 \ S_2$ have $S_3$: \(\text{nat-to-pr} \ n \ x = \text{sch-to-pr} \ (\text{Comp-op} \ (\text{nat-to-sch} \ (\text{c-fst} \ (\text{c-snd} \ n))) \ (\text{nat-to-sch} \ (\text{c-snd} \ (\text{c-snd} \ n)))) \ x\) by simp
  from $S_3$ have $S_4$: \(\text{nat-to-pr} \ n \ x = (\text{sch-to-pr} \ (\text{nat-to-sch} \ (\text{c-fst} \ (\text{c-snd} \ n)))) \ ((\text{sch-to-pr} \ (\text{nat-to-sch} \ (\text{c-snd} \ (\text{c-snd} \ n)))) \ x\) by simp
  from $S_4$ show \(?thesis\) by (simp add: nat-to-pr-def)
qed

lemma \(\text{nat-to-pr-lm-5}\): \(\text{c-fst} \ n \mod 7 = 5 \implies \text{nat-to-pr} \ n \ x = (\text{c-f-pair} \ (\text{nat-to-pr} \ (\text{c-fst} \ (\text{c-snd} \ n))) \ (\text{nat-to-pr} \ (\text{c-snd} \ (\text{c-snd} \ n)))) \ x\)
proof –
  assume $A$: \(\text{c-fst} \ n \ mod 7 = 5\)
  have $S_1$: \(\text{nat-to-pr} \ n \ x = \text{sch-to-pr} \ (\text{nat-to-sch} \ n) \ x\) by (simp add: nat-to-pr-def)
  from $A$ have $S_2$: \(\text{nat-to-sch} \ n = \text{Pair-op} \ (\text{nat-to-sch} \ (\text{c-fst} \ (\text{c-snd} \ n))) \ (\text{nat-to-sch} \ (\text{c-snd} \ (\text{c-snd} \ n)))\) by (rule nat-to-sch-5)
  from $S_1 \ S_2$ have $S_3$: \(\text{nat-to-pr} \ n \ x = \text{sch-to-pr} \ (\text{Pair-op} \ (\text{nat-to-sch} \ (\text{c-fst} \ (\text{c-snd} \ n))) \ (\text{nat-to-sch} \ (\text{c-snd} \ (\text{c-snd} \ n)))) \ x\) by simp
  from $S_3$ show \(?thesis\) by (simp add: nat-to-pr-def)
qed

lemma \(\text{nat-to-pr-lm-6}\): \(\text{c-fst} \ n \mod 7 = 6 \implies \text{nat-to-pr} \ n \ x = (\text{UnaryRecOp} \ (\text{nat-to-pr} \ (\text{c-fst} \ (\text{c-snd} \ n))) \ (\text{nat-to-pr} \ (\text{c-snd} \ (\text{c-snd} \ n)))) \ x\)
proof –
  assume $A$: \(\text{c-fst} \ n \ mod 7 = 6\)
  have $S_1$: \(\text{nat-to-pr} \ n \ x = \text{sch-to-pr} \ (\text{nat-to-sch} \ n) \ x\) by (simp add: nat-to-pr-def)
  from $A$ have $S_2$: \(\text{nat-to-sch} \ n = \text{Rec-op} \ (\text{nat-to-sch} \ (\text{c-fst} \ (\text{c-snd} \ n))) \ (\text{nat-to-sch} \ (\text{c-snd} \ (\text{c-snd} \ n)))\) by (rule nat-to-sch-6)
  from $S_1 \ S_2$ have $S_3$: \(\text{nat-to-pr} \ n \ x = \text{sch-to-pr} \ (\text{Rec-op} \ (\text{nat-to-sch} \ (\text{c-fst} \ (\text{c-snd} \ n))) \ (\text{nat-to-sch} \ (\text{c-snd} \ (\text{c-snd} \ n)))) \ x\) by simp
  from $S_3$ show \(?thesis\) by (simp add: nat-to-pr-def)
qed

lemma \(\text{univ-for-pr-lm-0}\): \(\text{c-fst} \ (\text{c-fst key}) \ mod 7 = 0 \implies \text{univ-for-pr} \ \text{key} = 0\)
proof –
  assume $A$: \(\text{c-fst} \ (\text{c-fst key}) \ mod 7 = 0\)
  have $S_1$: \(\text{univ-for-pr} \ \text{key} = \text{nat-to-pr} \ (\text{c-fst key}) \ (\text{c-snd key})\) by (simp add: univ-for-pr-def pr-conv-2-to-1-def)
  with $A$ show \(?thesis\) by (simp add: nat-to-pr-lm-0)
lemma univ-for-pr-lm-1: c-fst (c-fst key) mod 7 = 1 \implies\ \text{univ-for-pr} \ \text{key} = \text{Suc} (c-snd \ \text{key})

proof –
  assume A: c-fst (c-fst key) mod 7 = 1
  have S1: \text{univ-for-pr} \ \text{key} = \text{nat-to-pr} (c-fst \ (c-fst key)) (c-snd \ \text{key}) by (simp add: univ-for-pr-def pr-conv-2-to-1-def)
  with A show ?thesis by (simp add: nat-to-pr-lm-1)
qed

lemma univ-for-pr-lm-2: c-fst (c-fst key) mod 7 = 2 \implies\ \text{univ-for-pr} \ \text{key} = c-fst (c-snd \ \text{key})

proof –
  assume A: c-fst (c-fst key) mod 7 = 2
  have S1: \text{univ-for-pr} \ \text{key} = \text{nat-to-pr} (c-fst \ (c-fst key)) (c-snd \ \text{key}) by (simp add: univ-for-pr-def pr-conv-2-to-1-def)
  with A show ?thesis by (simp add: nat-to-pr-lm-2)
qed

lemma univ-for-pr-lm-3: c-fst (c-fst key) mod 7 = 3 \implies\ \text{univ-for-pr} \ \text{key} = c-snd (c-snd \ \text{key})

proof –
  assume A: c-fst (c-fst key) mod 7 = 3
  have S1: \text{univ-for-pr} \ \text{key} = \text{nat-to-pr} (c-fst \ (c-fst key)) (c-snd \ \text{key}) by (simp add: univ-for-pr-def pr-conv-2-to-1-def)
  with A show ?thesis by (simp add: nat-to-pr-lm-3)
qed

lemma univ-for-pr-lm-4: c-fst (c-fst key) mod 7 = 4 \implies\ \text{univ-for-pr} \ \text{key} = \text{nat-to-pr} (c-fst \ (c-fst (c-fst \ (c-fst key)))) (c-snd \ (c-snd \ (c-snd \ (c-snd key))))

proof –
  assume A: c-fst (c-fst key) mod 7 = 4
  have S1: \text{univ-for-pr} \ \text{key} = \text{nat-to-pr} (c-fst \ (c-fst key)) (c-snd \ (c-snd \ (c-snd \ (c-snd key)))) by (simp add: univ-for-pr-def pr-conv-2-to-1-def)
  with A show ?thesis by (simp add: nat-to-pr-lm-4)
qed

lemma univ-for-pr-lm-4-1: c-fst (c-fst key) mod 7 = 4 \implies\ \text{univ-for-pr} \ \text{key} = \text{nat-to-pr} (c-fst \ (c-fst \ (c-fst \ (c-fst key)))) (c-snd \ (c-snd \ (c-snd \ (c-snd key))))

proof –
  assume A: c-fst (c-fst key) mod 7 = 4
  have S1: \text{univ-for-pr} \ \text{key} = \text{nat-to-pr} (c-fst \ (c-fst key)) (c-snd \ (c-snd \ (c-snd \ (c-snd key)))) by (simp add: univ-for-pr-def pr-conv-2-to-1-def)
  with A show ?thesis by (simp add: nat-to-pr-lm-4 univ-for-pr-def pr-conv-2-to-1-def)
qed
lemma univ-for-pr-lm-5: \[ \text{c-fst (c-fst key) mod 7} = 5 \implies \text{univ-for-pr key} = \text{c-pair} \]
\[ (\text{univ-for-pr (c-pair (c-fst (c-snd (c-fst key)))) (c-snd key)}) (\text{univ-for-pr (c-pair (c-snd (c-fst key)))) (c-snd key)}) \]
proof –
  assume \( A: \text{c-fst (c-fst key) mod 7} = 5 \)
  have \( S1: \text{univ-for-pr key} = \text{nat-to-pr (c-fst key) (c-snd key)} \) by (simp add: univ-for-pr-def pr-conv-2-to-1-def)
  with \( A \) show \( \text{thesis} \) by (simp add: nat-to-pr-lm-5 c-f-pair-def univ-for-pr-def pr-conv-2-to-1-def)
qed

lemma univ-for-pr-lm-6-1: \[ \[ \text{c-fst (c-fst key) mod 7} = 6; \text{c-fst (c-snd key)} \neq 0 \] \implies \text{univ-for-pr key} = \text{univ-for-pr (c-pair (c-snd (c-fst key))) (c-snd (c-snd key))} \]
proof –
  assume \( A1: \text{c-fst (c-fst key) mod 7} = 6 \)
  assume \( A2: \text{c-fst (c-snd key)} = \text{Suc u} \)
  have \( S1: \text{univ-for-pr key} = \text{nat-to-pr (c-fst key) (c-snd key)} \) by (simp add: univ-for-pr-def pr-conv-2-to-1-def)
  with \( A1 A2 \) show \( \text{thesis} \) by (simp add: nat-to-pr-lm-6 UnaryRecOp-def univ-for-pr-def pr-conv-2-to-1-def)
qed

lemma univ-for-pr-lm-6-2: \[ \[ \text{c-fst (c-fst key) mod 7} = 6; \text{c-fst (c-snd key)} = \text{Suc u} \] \implies \text{univ-for-pr key} = \text{univ-for-pr (c-pair (c-snd (c-fst key))) (c-snd (c-snd key))} \]
proof –
  assume \( A1: \text{c-fst (c-fst key) mod 7} = 6 \)
  assume \( A2: \text{c-fst (c-snd key)} = \text{Suc u} \)
  then have \( A3: \text{c-fst (c-snd key)} > 0 \) by simp
  let \( ?u = \text{c-fst (c-snd key)} - (1::nat) \)
  have \( A4: \text{c-fst (c-snd key)} > 0 \) by simp
  let \( ?u = \text{c-fst (c-snd key)} - (1::nat) \)
  let \( ?u = \text{c-fst (c-snd key)} - (1::nat) \)
  have \( S1: \text{univ-for-pr key} = \text{univ-for-pr (c-pair (c-snd (c-fst key))) (c-snd (c-snd key))} \) by (simp add: nat-to-pr-lm-6 UnaryRecOp-def univ-for-pr-def pr-conv-2-to-1-def)
  with \( A1 A2 A3 \) show \( \text{thesis} \) by (simp add: nat-to-pr-lm-6 UnaryRecOp-def univ-for-pr-def pr-conv-2-to-1-def)
  apply(simp add: nat-to-pr-lm-6 UnaryRecOp-def univ-for-pr-def pr-conv-2-to-1-def)
  done
qed

lemma univ-for-pr-lm-6-3: \[ \[ \text{c-fst (c-fst key) mod 7} = 6; \text{c-fst (c-snd key)} \neq 0 \] \implies \text{univ-for-pr key} = \text{univ-for-pr (c-pair (c-snd (c-fst key))) (c-snd (c-snd key))} \]
proof –
  assume \( A1: \text{c-fst (c-fst key) mod 7} = 6 \)
  assume \( A2: \text{c-fst (c-snd key)} \neq 0 \) then have \( A3: \text{c-fst (c-snd key)} > 0 \) by simp
  let \( ?u = \text{c-fst (c-snd key)} - (1::nat) \)
  have \( S1: \text{univ-for-pr key} = \text{univ-for-pr (c-pair (c-snd (c-fst key))) (c-snd (c-snd key))} \) by (simp add: nat-to-pr-lm-6 UnaryRecOp-def univ-for-pr-def pr-conv-2-to-1-def)
  with \( A1 A2 A3 \) show \( \text{thesis} \) by (simp add: nat-to-pr-lm-6 UnaryRecOp-def univ-for-pr-def pr-conv-2-to-1-def)
  apply(simp add: nat-to-pr-lm-6 UnaryRecOp-def univ-for-pr-def pr-conv-2-to-1-def)
  done
qed
from A3 have S1: c-fst (c-snd key) = Suc ?u by simp 
from A1 S1 have S2: univ-for-pr key = univ-for-pr 
  (c-pair (c-snd (c-fst key))) 
  (c-pair (c-pair ?a (univ-for-pr (c-pair (c-fst key) (c-pair ?a (c-snd (c-snd key)))))) (c-snd (c-snd key))) by (rule univ-for-pr-lm-6-2) 
  thus ?thesis by simp qed 

lemma g-comp-lm-0: \[\left[\begin{array}{l} c-fst (c-fst key) \mod 7 = 4; \ c-is-sub-fun ls \ \text{univ-for-pr}; \ g-comp ls key \neq ls \end{array}\right] \implies \ \text{g-comp ls key = c-cons (c-pair key (univ-for-pr key)) ls} \]
proof – 
assume A1: c-fst (c-fst key) \mod 7 = 4 
assume A2: c-is-sub-fun ls \ \text{univ-for-pr} 
assume A3: g-comp ls key \neq ls 
let ?n = c-fst key 
let ?x = c-snd key 
let ?m = c-snd ?n 
let ?m1 = c-fst ?m 
let ?m2 = c-snd ?m 
let ?k1 = c-pair ?m2 ?x 
have S1: c-assoc-have-key ls ?k1 = 0 
proof (rule ccontr) 
assume A1-1: c-assoc-have-key ls ?k1 \neq 0 
then have g-comp ls key = ls by (simp add: g-comp-def) 
with A3 show False by simp qed 
let ?y = c-assoc-value ls ?k1 
from A2 S1 have S2: ?y = univ-for-pr ?k1 by (rule c-is-sub-fun-lm-1) 
let ?k2 = c-pair ?m1 ?y 
have S3: c-assoc-have-key ls ?k2 = 0 
proof (rule ccontr) 
assume A3-1: c-assoc-have-key ls ?k2 \neq 0 
then have g-comp ls key = ls by (simp add: g-comp-def Let-def) 
with A3 show False by simp qed 
let ?z = c-assoc-value ls ?k2 
from A2 S3 have S4: ?z = univ-for-pr ?k2 by (rule c-is-sub-fun-lm-1) 
from S2 have S5: ?k2 = c-pair ?m1 (univ-for-pr ?k1) by simp 
from S4 S5 have S6: ?z = univ-for-pr (c-pair ?m1 (univ-for-pr ?k1)) by simp 
from A1 S6 have S7: ?z = univ-for-pr key by (simp add: univ-for-pr-lm-4-1) 
from S1 S3 S7 show ?thesis by (simp add: g-comp-def Let-def) qed 

lemma g-comp-lm-1: \[\left[\begin{array}{l} c-fst (c-fst key) \mod 7 = 4; \ c-is-sub-fun ls \ \text{univ-for-pr} \end{array}\right] \implies \ \text{c-is-sub-fun (g-comp ls key) univ-for-pr} \]
proof – 
assume A1: c-fst (c-fst key) \mod 7 = 4 
assume A2: c-is-sub-fun ls \ \text{univ-for-pr} 
show ?thesis
proof cases
  assume $g$-comp $ls$ key = $ls$
  with A2 show $c$-is-sub-fun ($g$-comp $ls$ key) univ-for-pr by simp
next
  assume $g$-comp $ls$ key $\neq$ $ls$
  from A1 A2 this have S1: $g$-comp $ls$ key = $c$-cons ($c$-pair key (univ-for-pr key)) $ls$ by (rule $g$-comp-lm-0)
  with A2 show $c$-is-sub-fun ($g$-comp $ls$ key) univ-for-pr by (simp add: $c$-is-sub-fun-lm-2)
qed

lemma $g$-pair-lm-0: [$c$-fst ($c$-fst key) mod 7 = 5; $c$-is-sub-fun $ls$ univ-for-pr; $g$-pair $ls$ key $\neq$ $ls$] $\implies$ $g$-pair $ls$ key = $c$-cons ($c$-pair key (univ-for-pr key)) $ls$
proof –
  assume A1: $c$-fst ($c$-fst key) mod 7 = 5
  assume A2: $c$-is-sub-fun $ls$ univ-for-pr
  assume A3: $g$-pair $ls$ key $\neq$ $ls$
  let $?n = c$-fst key
  let $?x = c$-snd key
  let $?m = c$-snd $?n
  let $?m1 = c$-fst $?m
  let $?m2 = c$-snd $?m
  let $?k1 = c$-pair $?m1 ??x
  have S1: $c$-assoc-have-key $ls$ $?k1$ = 0
    proof (rule ccontr)
      assume A1-1: $c$-assoc-have-key $ls$ $?k1$ $\neq$ 0
      then have $g$-pair $ls$ key = $ls$ by (simp add: $g$-pair-def Let-def)
      with A3 show False by simp
    qed
  let $?y1 = c$-assoc-value $ls$ $?k1
  from A2 S1 have S2: $?y1 = univ-for-pr $?k1$ by (rule $c$-is-sub-fun-lm-1)
  let $?k2 = c$-pair $?m2 ??x
  have S3: $c$-assoc-have-key $ls$ $?k2$ = 0
    proof (rule ccontr)
      assume A3-1: $c$-assoc-have-key $ls$ $?k2$ $\neq$ 0
      then have $g$-pair $ls$ key = $ls$ by (simp add: $g$-pair-def Let-def)
      with A3 show False by simp
    qed
  let $?y2 = c$-assoc-value $ls$ $?k2
  from A2 S3 have S4: $?y2 = univ-for-pr $?k2$ by (rule $c$-is-sub-fun-lm-1)
  let $?z = c$-pair $?y1 ??y2
  from S2 S4 have S5: $?z = c$-pair (univ-for-pr $?k1$) (univ-for-pr $?k2$) by simp
  from A1 S5 have S6: $?z = univ-for-pr key$ by (simp add: univ-for-pr-lm-5)
  from S1 S3 S6 show ?thesis by (simp add: $g$-pair-def Let-def)
qed

lemma $g$-pair-lm-1: [ $c$-fst ($c$-fst key) mod 7 = 5; $c$-is-sub-fun $ls$ univ-for-pr] $\implies$ $c$-is-sub-fun ($g$-pair $ls$ key) univ-for-pr
proof –
assume $A_1$: $c\text{-}\text{fst} \ (c\text{-}\text{fst} \ \text{key}) \mod 7 = 5$

assume $A_2$: $c\text{-}\text{is-sub-fun} \ \text{ls} \ \text{univ-for-pr}$

show $?\text{thesis}$

proof cases

assume $g\text{-pair} \ \text{ls} \ \text{key} = \text{ls}$

with $A_2$ show $c\text{-}\text{is-sub-fun} \ (g\text{-pair} \ \text{ls} \ \text{key}) \ \text{univ-for-pr}$ by simp

next

assume $g\text{-pair} \ \text{ls} \ \text{key} \neq \text{ls}$

from $A_1 \ A_2$ have $S_1$: $g\text{-pair} \ \text{ls} \ \text{key} = \ c\text{-cons} \ (c\text{-pair} \ \text{key} \ (\text{univ-for-pr} \ \text{key})) \ \text{ls}$ by (rule $g\text{-pair-lm-0}$)

with $A_2$ show $c\text{-}\text{is-sub-fun} \ (g\text{-pair} \ \text{ls} \ \text{key}) \ \text{univ-for-pr}$ by (simp add: $c\text{-}\text{is-sub-fun-lm-2}$)

qed

qed

lemma $g\text{-rec-lm-0}$: $[c\text{-}\text{fst} \ (c\text{-}\text{fst} \ \text{key}) \mod 7 = 6; \ c\text{-}\text{is-sub-fun} \ \text{ls} \ \text{univ-for-pr}; \ g\text{-rec} \ \text{ls} \ \text{key} \neq \ \text{ls}] \implies g\text{-rec} \ \text{ls} \ \text{key} = c\text{-cons} \ (c\text{-pair} \ \text{key} \ (\text{univ-for-pr} \ \text{key})) \ \text{ls}$

proof –

assume $A_1$: $c\text{-}\text{fst} \ (c\text{-}\text{fst} \ \text{key}) \mod 7 = 6$

assume $A_2$: $c\text{-}\text{is-sub-fun} \ \text{ls} \ \text{univ-for-pr}$

assume $A_3$: $g\text{-rec} \ \text{ls} \ \text{key} \neq \ \text{ls}$

let $?n = c\text{-}\text{fst} \ \text{key}$

let $?x = c\text{-snd} \ \text{key}$

let $?m = c\text{-snd} \ ?n$

let $?m1 = c\text{-}\text{fst} \ ?m$

let $?m2 = c\text{-}\text{snd} \ ?m$

let $?y1 = c\text{-}\text{fst} \ ?x$

let $?x1 = c\text{-}\text{snd} \ ?x$

show $?\text{thesis}$

proof cases

assume $A1-1$: $?y1 = 0$

let $?k1 = c\text{-}\text{pair} \ ?m1 \ ?x1$

have $S1-1$: $c\text{-}\text{assoc-have-key} \ \text{ls} \ ?k1 = 0$

proof (rule ccontr)

assume $c\text{-}\text{assoc-have-key} \ \text{ls} \ ?k1 \neq 0$

with $A1-1$ have $g\text{-rec} \ \text{ls} \ \text{key} = \ \text{ls}$ by (simp add: $g\text{-rec-def}$)

with $A3$ show False by simp

qed

let $?v = c\text{-}\text{assoc-value} \ \text{ls} \ ?k1$

from $A2 \ S1-1$ have $S1-2$: $?v = \text{univ-for-pr} \ ?k1$ by (rule $c\text{-}\text{is-sub-fun-lm-1}$)

from $A1 \ A1-1 \ S1-2$ have $S1-3$: $?v = \text{univ-for-pr} \ \text{key}$ by (simp add: $\text{univ-for-pr-lm-6-1}$)

from $A1-1 \ S1-1 \ S1-3$ show $?\text{thesis}$ by (simp add: $g\text{-rec-def} \ \text{Let-def}$)

next

assume $A2-1$: $?y1 \neq 0$ then have $A2-2$: $?y1 > 0$ by simp

let $?y2 = ?y1 - (1::\text{nat})$

let $?k2 = c\text{-}\text{pair} \ ?n \ (c\text{-}\text{pair} \ ?y2 \ ?x1)$

have $S2-1$: $c\text{-}\text{assoc-have-key} \ \text{ls} \ ?k2 = 0$

proof (rule ccontr)

assume $c\text{-}\text{assoc-have-key} \ \text{ls} \ ?k2 \neq 0$

with $A2-1$ have $g\text{-rec} \ \text{ls} \ \text{key} = \ \text{ls}$ by (simp add: $g\text{-rec-def} \ \text{Let-def}$)

110
with A3 show False by simp
qed

let $?t1 = c-assoc-value ls ?k2
from A2 S2-1 have S2-2: $?t1 = univ-for-pr ?k2 by (rule c-is-sub-fun-lm-1)
let $?t2 = c-pair (c-pair ?y2 $?t1) ?x1
let $?k3 = c-pair $?m2 $?t2
have S2-3: c-assoc-have-key ls $?k3 = 0
proof (rule ccontr)
  assume c-assoc-have-key ls $?k3 ≠ 0
  with A2-1 have g-rec ls ?k3 = ls by (simp add: g-rec-def Let-def)
  with A3 show False by simp
qed

let $?u = c-assoc-value ls $?k3
from A2 S2-3 have S2-4: $?u = univ-for-pr (?m2 (c-pair ?y2 (c-pair ?k2))) by simp
from A1 A2-1 S2-5 have S2-6: $?u = univ-for-pr key by (simp add: univ-for-pr-lm-6-3)
from A2-1 S2-1 S2-3 S2-6 show ?thesis by (simp add: g-rec-def Let-def)
qued

lemma g-rec-lm-1: [ c-fst (c-fst key) mod 7 = 6; c-is-sub-fun ls univ-for-pr ] ⟹ c-is-sub-fun (g-rec ls key) univ-for-pr
proof –
  assume A1: c-fst (c-fst key) mod 7 = 6
  assume A2: c-is-sub-fun ls unie-for-pr
  show ?thesis
  proof cases
    assume g-rec ls key = ls
    with A2 show c-is-sub-fun (g-rec ls key) univ-for-pr by simp
  next
    assume g-rec ls key ≠ ls
    from A1 A2 this have S1: g-rec ls key = c-cons (c-pair key (univ-for-pr key))
    ls by (rule g-rec-lm-0)
    with A2 show c-is-sub-fun (g-rec ls key) univ-for-pr by (simp add: c-is-sub-fun-lm-2)
qued

lemma g-step-lm-0: c-fst (c-fst key) mod 7 = 0 ⟹ g-step ls key = c-cons (c-pair key 0) ls by (simp add: g-step-def)

lemma g-step-lm-1: c-fst (c-fst key) mod 7 = 1 ⟹ g-step ls key = c-cons (c-pair key (Suc (c-snd key))) ls by (simp add: g-step-def Let-def)

lemma g-step-lm-2: c-fst (c-fst key) mod 7 = 2 ⟹ g-step ls key = c-cons (c-pair key (c-fst (c-snd key))) ls by (simp add: g-step-def Let-def)

lemma g-step-lm-3: c-fst (c-fst key) mod 7 = 3 ⟹ g-step ls key = c-cons (c-pair key (c-snd (c-snd key))) ls by (simp add: g-step-def Let-def)
lemma g-step-lm-4: c-fst (c-fst key) mod 7 = 4 \implies g-step ls key = g-comp ls key
by (simp add: g-step-def)

lemma g-step-lm-5: c-fst (c-fst key) mod 7 = 5 \implies g-step ls key = g-pair ls key
by (simp add: g-step-def)

lemma g-step-lm-6: c-fst (c-fst key) mod 7 = 6 \implies g-step ls key = g-rec ls key
by (simp add: g-step-def)

lemma g-step-lm-7: c-is-sub-fun ls univ-for-pr \implies c-is-sub-fun (g-step ls key)
univ-for-pr

proof –
  assume A1: c-is-sub-fun ls univ-for-pr
  let ?n = c-fst key
  let ?x = c-snd key
  let ?n1 = (c-fst ?n) mod 7
  have S1: ?n1 = 0 \implies ?thesis
  proof –
    assume A: ?n1 = 0
    then have S1-1: g-step ls key = c-cons (c-pair key 0) ls by (rule g-step-lm-0)
    from A have S1-2: univ-for-pr key = 0 by (rule univ-for-pr-lm-0)
    from A1 have S1-3: c-is-sub-fun (c-cons (c-pair key (univ-for-pr key))) ls
    univ-for-pr by (rule c-is-sub-fun-lm-2)
      from S1-3 S1-1 S1-2 show ?thesis by simp
  qed
  have S2: ?n1 = 1 \implies ?thesis
  proof –
    assume A: ?n1 = 1
    then have S2-1: g-step ls key = c-cons (c-pair key (Suc (c-snd key))) ls by (rule g-step-lm-1)
    from A have S2-2: univ-for-pr key = Suc (c-snd key) by (rule univ-for-pr-lm-1)
    from A1 have S2-3: c-is-sub-fun (c-cons (c-pair key (univ-for-pr key))) ls
    univ-for-pr by (rule c-is-sub-fun-lm-2)
      from S2-3 S2-1 S2-2 show ?thesis by simp
  qed
  have S3: ?n1 = 2 \implies ?thesis
  proof –
    assume A: ?n1 = 2
    then have S2-1: g-step ls key = c-cons (c-pair key (c-fst (c-snd key))) ls by (rule g-step-lm-2)
    from A have S2-2: univ-for-pr key = c-fst (c-snd key) by (rule univ-for-pr-lm-2)
    from A1 have S2-3: c-is-sub-fun (c-cons (c-pair key (univ-for-pr key))) ls
    univ-for-pr by (rule c-is-sub-fun-lm-2)
      from S2-3 S2-1 S2-2 show ?thesis by simp
  qed
  have S4: ?n1 = 3 \implies ?thesis
  proof –
    assume A: ?n1 = 3

112
then have S2-1: \( g\text{-}step \) \( ls \) \( key \) = \( c\)-cons \( (c\text{-}snd \ (c\text{-}snd \ key)) \) \( ls \) by (rule \( g\text{-}step\text{-}lm\text{-}3 \))
from \( A \) have S2-2: \( \text{univ-for-pr} \) \( key \) = \( c\)-snd \( (c\text{-}snd \ key) \) by (rule \( \text{univ-for-pr}\text{-}lm\text{-}3 \))
from \( A1 \) have S2-3: \( c\text{-}is\text{-}sub\text{-}fun \) \( (c\text{-}cons \ (c\text{-}pair \ (\text{univ-for-pr} \ key)) \) \( ls \) \( \text{univ-for-pr} \) by (rule \( c\text{-}is\text{-}sub\text{-}fun\text{-}lm\text{-}2 \))
from S2-3 S2-1 S2-2 show \( \text{thesis} \) by simp
qed
have S5: \( ?n1 = 4 \implies \text{thesis} \)
proof
  assume A: \( ?n1 = 4 \)
  then have S2-1: \( g\text{-}step \) \( ls \) \( key \) = \( g\text{-}comp \) \( ls \) \( key \) by (rule \( g\text{-}step\text{-}lm\text{-}4 \))
  from \( A \ A1 \ S2-1 \) show \( \text{thesis} \) by (simp add: \( g\text{-}comp\text{-}lm\text{-}1 \))
qed
have S6: \( ?n1 = 5 \implies \text{thesis} \)
proof
  assume A: \( ?n1 = 5 \)
  then have S2-1: \( g\text{-}step \) \( ls \) \( key \) = \( g\text{-}pair \) \( ls \) \( key \) by (rule \( g\text{-}step\text{-}lm\text{-}5 \))
  from \( A \ A1 \ S2-1 \) show \( \text{thesis} \) by (simp add: \( g\text{-}pair\text{-}lm\text{-}1 \))
qed
have S7: \( ?n1 = 6 \implies \text{thesis} \)
proof
  assume A: \( ?n1 = 6 \)
  then have S2-1: \( g\text{-}step \) \( ls \) \( key \) = \( g\text{-}rec \) \( ls \) \( key \) by (rule \( g\text{-}step\text{-}lm\text{-}6 \))
  from \( A \ A1 \ S2-1 \) show \( \text{thesis} \) by (simp add: \( g\text{-}rec\text{-}lm\text{-}1 \))
qed
have S8: \( ?n1=0 \lor \ ?n1=1 \lor \ ?n1=2 \lor \ ?n1=3 \lor \ ?n1=4 \lor \ ?n1=5 \lor \ ?n1=6 \)
by (rule \( \text{mod7}\text{-}lm \))
with S1 S2 S3 S4 S5 S6 S7 show \( \text{thesis} \) by fast
qed

theorem pr-gr-1: \( c\text{-}is\text{-}sub\text{-}fun \ (pr\text{-}gr \ x) \) \( \text{univ-for-pr} \)
apply(induct \( x \))
apply(simp add: \( \text{pr-gr-at-0} \ c\text{-}is\text{-}sub\text{-}fun\text{-}def \ c\text{-}assoc\text{-}have\text{-}key\text{-}df \))
apply(simp add: \( \text{pr-gr-at-Suc} \))
apply(simp add: \( g\text{-}step\text{-}lm\text{-}7 \))
done

lemma comp-next: \( g\text{-}comp \) \( ls \) \( key \) = \( ls \lor \ c\text{-}tl \ (g\text{-}comp \ ls \ key) \) = \( ls \) by(simp add: \( g\text{-}comp\text{-}def \ Let\text{-}def \))
lemma pair-next: \( g\text{-}pair \) \( ls \) \( key \) = \( ls \lor \ c\text{-}tl \ (g\text{-}pair \ ls \ key) \) = \( ls \) by(simp add: \( g\text{-}pair\text{-}def \ Let\text{-}def \))
lemma rec-next: \( g\text{-}rec \) \( ls \) \( key \) = \( ls \lor \ c\text{-}tl \ (g\text{-}rec \ ls \ key) \) = \( ls \) by(simp add: \( g\text{-}rec\text{-}def \ Let\text{-}def \))
lemma step-next: \( g\text{-}step \) \( ls \) \( key \) = \( ls \lor \ c\text{-}tl \ (g\text{-}step \ ls \ key) \) = \( ls \)
apply(simp add: \( g\text{-}step\text{-}def \ comp\text{-}next \ pair\text{-}next \ rec\text{-}next \ Let\text{-}def \))
done
lemma lm1: pr-gr (Suc x) = pr-gr x ∨ c-tl (pr-gr (Suc x)) = pr-gr x by (simp add: pr-gr-at-Suc step-next)

lemma c-assoc-have-key-pos: c-assoc-have-key ls x = 0 ⇒ ls > 0
proof -
  assume A1: c-assoc-have-key ls x = 0
  thus ?thesis by (cases)
  assume A2: ls = 0
  then have S1: c-assoc-have-key ls x = 1 by (simp add: c-assoc-have-key-df)
  with A1 have S2: False by auto
  next
  assume A3: ¬ ls = 0
  then show ls > 0 by auto
qed

lemma lm2: c-assoc-have-key (c-tl ls) key = 0 ⇒ c-assoc-have-key ls key = 0
proof -
  assume A1: c-assoc-have-key (c-tl ls) key = 0
  from A1 have S1: c-tl ls > 0 by (rule c-assoc-have-key-pos)
  have S2: c-tl ls ≤ ls by (rule c-tl-le)
  from S1 S2 have S3: ls ≠ 0 by auto
  from A1 S3 show ?thesis by (auto simp add: c-assoc-have-key-lm-1)
qed

lemma lm3: c-assoc-have-key (pr-gr x) key = 0 ⇒ c-assoc-have-key (pr-gr (Suc x)) key = 0
proof -
  assume A1: c-assoc-have-key (pr-gr x) key = 0
  have S1: pr-gr (Suc x) = pr-gr x ∨ c-tl (pr-gr (Suc x)) = pr-gr x by (rule lm1)
  from A1 have S2: pr-gr (Suc x) = pr-gr x ⇒ ?thesis by auto
  have S3: c-tl (pr-gr (Suc x)) = pr-gr x ⇒ ?thesis
  proof -
    assume c-tl (pr-gr (Suc x)) = pr-gr x (is c-tl ?ls = -)
    with A1 have c-assoc-have-key (c-tl ?ls) key = 0 by auto
    then show c-assoc-have-key ?ls key = 0 by (rule lm2)
  qed
  from S1 S2 S3 show ?thesis by auto
qed

lemma lm4: \[ \mathrm{c-assoc-have-key} (\mathrm{pr-gr} x) \mathrm{key} = 0; \ 0 \leq y \] ⇒ c-assoc-have-key (pr-gr (x+y)) key = 0
apply (induct-tac y)
apply (auto)
apply (simp add: lm3)
done
lemma lm5: \[ \{ \text{c-assoc-have-key (pr-gr x) key = 0; x \leq y} \} \implies \text{c-assoc-have-key (pr-gr y) key = 0} \]
proof -
  assume A1: c-assoc-have-key (pr-gr x) key = 0
  assume A2: x \leq y
  let \(?z = y-x\)
  from A2 have S1: 0 \leq \(?z\) by auto
  from A2 have S2: y = x + \(?z\) by auto
  from A1 S1 have S3: c-assoc-have-key (pr-gr (x+\(?z\))) key = 0 by (rule lm4)
  from S2 S3 show \(?\)thesis by auto
qed

lemma loc-upb-lm-1: n = 0 \implies (c-fst n) mod 7 = 0
apply(simp add: c-fst-at-0)
done

lemma loc-upb-lm-2: (c-fst n) mod 7 > 1 \implies c-snd n < n
proof -
  assume A1: c-fst n mod 7 > 1
  from A1 have S1: 1 < c-fst n by simp
  have S2: c-fst n \leq n by (rule c-fst-le-arg)
  from S1 S2 have S3: 1 < n by simp
  from S3 have S4: n > 1 by simp
  from S4 show \(?\)thesis by (rule c-snd-less-arg)
qed

lemma loc-upb-lm-2-0: (c-fst n) mod 7 = 4 \implies c-fst (c-snd n) < n
proof
  assume A1: c-fst n mod 7 = 4
  then have S0: c-fst n mod 7 > 1 by auto
  then have S1: c-snd n < n by (rule loc-upb-lm-2)
  have S2: c-fst (c-snd n) \leq c-snd n by (rule c-fst-le-arg)
  from S1 S2 show c-fst (c-snd n) < n by auto
qed

lemma loc-upb-lm-2-2: (c-fst n) mod 7 = 4 \implies c-snd (c-snd n) < n
proof
  assume A1: c-fst n mod 7 = 4
  then have S0: c-fst n mod 7 > 1 by auto
  then have S1: c-snd n < n by (rule loc-upb-lm-2)
  have S2: c-snd (c-snd n) \leq c-snd n by (rule c-snd-le-arg)
  from S1 S2 show c-snd (c-snd n) < n by auto
qed

lemma loc-upb-lm-2-3: (c-fst n) mod 7 = 5 \implies c-fst (c-snd n) < n
proof
  assume A1: c-fst n mod 7 = 5
  then have S0: c-fst n mod 7 > 1 by auto
  then have S1: c-snd n < n by (rule loc-upb-lm-2)
have $S_2$: $c\text{-}\text{fst} \ (c\text{-}\text{snd} \ n) \leq c\text{-}\text{snd} \ n$ by (rule $c\text{-}\text{fst-le-arg}$)
from $S_1 \ S_2$ show $c\text{-}\text{fst} \ (c\text{-}\text{snd} \ n) < n$ by auto
qed

lemma $\text{loc-upb-lm-2-4}$: $(c\text{-}\text{fst} \ n) \mod 7 = 5 \implies c\text{-}\text{snd} \ (c\text{-}\text{snd} \ n) < n$
proof
assume $A_1$: $c\text{-}\text{fst} \ n \mod 7 = 5$
then have $S_0$: $c\text{-}\text{fst} \ n \mod 7 > 1$ by auto
then have $S_1$: $c\text{-}\text{snd} \ n < n$ by (rule $\text{loc-upb-lm-2}$)
have $S_2$: $c\text{-}\text{snd} \ (c\text{-}\text{snd} \ n) \leq c\text{-}\text{snd} \ n$ by (rule $c\text{-}\text{snd-le-arg}$)
from $S_1 \ S_2$ show $c\text{-}\text{snd} \ (c\text{-}\text{snd} \ n) < n$ by auto
qed

lemma $\text{loc-upb-lm-2-5}$: $(c\text{-}\text{fst} \ n) \mod 7 = 6 \implies c\text{-}\text{fst} \ (c\text{-}\text{snd} \ n) < n$
proof
assume $A_1$: $c\text{-}\text{fst} \ n \mod 7 = 6$
then have $S_0$: $c\text{-}\text{fst} \ n \mod 7 > 1$ by auto
then have $S_1$: $c\text{-}\text{snd} \ n < n$ by (rule $\text{loc-upb-lm-2}$)
have $S_2$: $c\text{-}\text{snd} \ (c\text{-}\text{snd} \ n) \leq c\text{-}\text{snd} \ n$ by (rule $c\text{-}\text{snd-le-arg}$)
from $S_1 \ S_2$ show $c\text{-}\text{snd} \ (c\text{-}\text{snd} \ n) < n$ by auto
qed

lemma $\text{loc-upb-lm-2-6}$: $(c\text{-}\text{fst} \ n) \mod 7 = 6 \implies c\text{-}\text{snd} \ (c\text{-}\text{snd} \ n) < n$
proof
assume $A_1$: $c\text{-}\text{fst} \ n \mod 7 = 6$
then have $S_0$: $c\text{-}\text{fst} \ n \mod 7 > 1$ by auto
then have $S_1$: $c\text{-}\text{snd} \ n < n$ by (rule $\text{loc-upb-lm-2}$)
have $S_2$: $c\text{-}\text{snd} \ (c\text{-}\text{snd} \ n) \leq c\text{-}\text{snd} \ n$ by (rule $c\text{-}\text{snd-le-arg}$)
from $S_1 \ S_2$ show $c\text{-}\text{snd} \ (c\text{-}\text{snd} \ n) < n$ by auto
qed

lemma $\text{loc-upb-lm-2-7}$: $[y_2 = y_1 - 1::\text{nat}; \ 0 < y_1; \ x_1 = c\text{-}\text{snd} \ x; \ y_1 = c\text{-}\text{fst} \ x] \implies \text{c\text{-}\text{pair} \ y}_2 \ x_1 < x$
proof
assume $A_1$: $y_2 = y_1 - 1::\text{nat}$ and $A_2$: $0 < y_1$ and $A_3$: $x_1 = c\text{-}\text{snd} \ x$ and
$A_4$: $y_1 = c\text{-}\text{fst} \ x$
from $A_1 \ A_2$ have $S_1$: $y_2 < y_1$ by auto
from $S_1$ have $S_2$: $\text{c\text{-}\text{pair} \ y}_2 \ x_1 < \text{c\text{-}\text{pair} \ y}_1 \ x_1$ by (rule $\text{c\text{-}\text{pair-strict-mono1}$)
from $A_3 \ A_4$ have $S_3$: $\text{c\text{-}\text{pair} \ y}_1 \ x_1 = x$ by auto
from $S_2 \ S_3$ show $\text{c\text{-}\text{pair} \ y}_2 \ x_1 < x$ by auto
qed

function $\text{loc-upb} :: \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat}$ where
aa: $\text{loc-upb} \ n \ x = ($
let $n_1 = (c\text{-}\text{fst} \ n) \mod 7$ in
if $n_1 = 0$ then $(\text{c\text{-}\text{pair} \ (c\text{-}\text{pair} \ n \ x)} \ 0) + 1$ else
if $n_1 = 1$ then $(\text{c\text{-}\text{pair} \ (c\text{-}\text{pair} \ n \ x)} \ 0) + 1$ else
if $n_1 = 2$ then $(\text{c\text{-}\text{pair} \ (c\text{-}\text{pair} \ n \ x)} \ 0) + 1$ else
if $n_1 = 3$ then $(\text{c\text{-}\text{pair} \ (c\text{-}\text{pair} \ n \ x)} \ 0) + 1$ else
116
if \( n = 4 \) then 
let \( m = c-snd n \); \( m1 = c-fst m \); \( m2 = c-snd m \);
y = \( c\text{-assoc-value} \) \((pr-gr \ (loc-upb m2 x))\) \((c\text{-pair} \ m2 \ x)\) \in
(c\text{-pair} \ (c\text{-pair} \ n \ x) \ (loc-upb m2 x + loc-upb m1 y)) + 1
) else
if \( n = 5 \) then 
let \( m = c-snd n \); \( m1 = c-fst m \); \( m2 = c-snd m \) in
(c\text{-pair} \ (c\text{-pair} \ n \ x) \ (loc-upb m1 x + loc-upb m2 x)) + 1
) else
if \( n = 6 \) then 
let \( m = c-snd n \); \( m1 = c-fst m \); \( m2 = c-snd m \); \( y1 = c-fst x \); \( x1 = c-snd x \) in
if \( y1 = 0 \) then 
(c\text{-pair} \ (c\text{-pair} \ n \ x) \ (loc-upb m1 x1)) + 1
) else 
let \( y2 = y1 - (1::nat) \);
\( t1 = c\text{-assoc-value} \) \((pr-gr \ (loc-upb n (c\text{-pair} \ y2 x1)))\) \((c\text{-pair} \ n \ (c\text{-pair} \ y2 x1))\); \( t2 = c\text{-pair} \ (c\text{-pair} \ y2 t1) \ x1 \) \in
(c\text{-pair} \ (c\text{-pair} \ n \ (c\text{-pair} \ y2 x1) + loc-upb m2 t2)) + 1
)
) else 0
)

by auto

termination
apply (relation measure (\( \lambda \ m, m \). \langle \text{lex} \rangle \)) measure (\( \lambda \ n, n \))
apply (simp-all add: loc-upb-lm-2-0 loc-upb-lm-2-2 loc-upb-lm-2-3 loc-upb-lm-2-4
loc-upb-lm-2-5 loc-upb-lm-2-6 loc-upb-lm-2-7)
apply auto
done

definition
\( \text{lex-p} :: ((\text{nat} \times \text{nat}) \times \text{nat} \times \text{nat}) \text{ set} \) where
\( \text{lex-p} = (\langle \text{measure} \ (\lambda \ m, m) \ \langle \text{lex} \rangle \rangle \) measure (\( \lambda \ n, n \)))

lemma wf-lex-p: wf(\( \text{lex-p} \))
apply(simp add: lex-p-def)
apply(auto)
done

lemma lex-p-eq: \( ((n', x'), (n, x)) \in \text{lex-p} = (n' < n \lor n' = n \land x' < x) \)
apply(simp add: lex-p-def)
done

lemma loc-upb-lex-0: \( c\text{-fst} n \mod 7 = 0 \implies \text{c-assoc-have-key} \) \((pr-gr \ (loc-upb n \ x))\) \((c\text{-pair} \ n \ x)\) = 0
proof –
assume A1: \( c\text{-fst} n \mod 7 = 0 \)
let \( \textit{key} = \text{c-pair } n \ x \)
let \( \textit{s} = \text{c-pair } \textit{key} 0 \)
let \( \textit{ls} = \text{pr-gr } \textit{s} \)
from \( A1 \) have \( \text{loc-upb } n \ x = \textit{s} + 1 \) by simp
then have \( S1: \text{pr-gr } (\text{loc-upb } n \ x) = \text{g-step } (\text{pr-gr } \textit{s}) \ (\text{c-fst } \textit{s}) \) by (simp add: \\
\text{pr-gr-at-Suc})
from \( A1 \) have \( S2: \text{g-step } \textit{ls} \ \textit{key} = \text{c-cons } (\text{c-pair } \textit{key} 0) \ \textit{ls} \) by (simp add: \\
\text{g-step-def})
from \( S1 \ S2 \) have \( \text{pr-gr } (\text{loc-upb } n \ x) = \text{c-cons } (\text{c-pair } \textit{key} 0) \ \textit{ls} \) by auto
thus \( \textbf{?thesis} \) by (simp add: \text{c-assoc-lm-1})
qed

\textbf{lemma} \( \text{loc-upb-lex-1}: \text{c-fst } n \ \text{mod } 7 = 1 \implies \text{c-assoc-have-key } (\text{pr-gr } (\text{loc-upb } n \ x)) \ (\text{c-pair } n \ x) = 0 \)
\textbf{proof} –
\begin{itemize}
  \item assume \( A1: \text{c-fst } n \ \text{mod } 7 = 1 \)
  \item let \( \textit{key} = \text{c-pair } n \ x \)
  \item let \( \textit{s} = \text{c-pair } \textit{key} 0 \)
  \item let \( \textit{ls} = \text{pr-gr } \textit{s} \)
  \item from \( A1 \) have \( \text{loc-upb } n \ x = \textit{s} + 1 \) by simp
  \item then have \( S1: \text{pr-gr } (\text{loc-upb } n \ x) = \text{g-step } (\text{pr-gr } \textit{s}) \ (\text{c-fst } \textit{s}) \) by (simp add: \\
  \text{pr-gr-at-Suc})
  \item from \( A1 \) have \( S2: \text{g-step } \textit{ls} \ \textit{key} = \text{c-cons } (\text{c-pair } \textit{key} \ (\text{Suc } x)) \ \textit{ls} \) by (simp add: \\
  \text{g-step-def})
  \item from \( S1 \ S2 \) have \( \text{pr-gr } (\text{loc-upb } n \ x) = \text{c-cons } (\text{c-pair } \textit{key} \ (\text{Suc } x)) \ \textit{ls} \) by auto
  \item thus \( \textbf{?thesis} \) by (simp add: \text{c-assoc-lm-1})
\end{itemize}
qed

\textbf{lemma} \( \text{loc-upb-lex-2}: \text{c-fst } n \ \text{mod } 7 = 2 \implies \text{c-assoc-have-key } (\text{pr-gr } (\text{loc-upb } n \ x)) \ (\text{c-pair } n \ x) = 0 \)
\textbf{proof} –
\begin{itemize}
  \item assume \( A1: \text{c-fst } n \ \text{mod } 7 = 2 \)
  \item let \( \textit{key} = \text{c-pair } n \ x \)
  \item let \( \textit{s} = \text{c-pair } \textit{key} 0 \)
  \item let \( \textit{ls} = \text{pr-gr } \textit{s} \)
  \item from \( A1 \) have \( \text{loc-upb } n \ x = \textit{s} + 1 \) by simp
  \item then have \( S1: \text{pr-gr } (\text{loc-upb } n \ x) = \text{g-step } (\text{pr-gr } \textit{s}) \ (\text{c-fst } \textit{s}) \) by (simp add: \\
  \text{pr-gr-at-Suc})
  \item from \( A1 \) have \( S2: \text{g-step } \textit{ls} \ \textit{key} = \text{c-cons } (\text{c-pair } \textit{key} \ (\text{c-fst } x)) \ \textit{ls} \) by (simp add: \\
  \text{g-step-def})
  \item from \( S1 \ S2 \) have \( \text{pr-gr } (\text{loc-upb } n \ x) = \text{c-cons } (\text{c-pair } \textit{key} \ (\text{c-fst } x)) \ \textit{ls} \) by auto
  \item thus \( \textbf{?thesis} \) by (simp add: \text{c-assoc-lm-1})
\end{itemize}
qed

\textbf{lemma} \( \text{loc-upb-lex-3}: \text{c-fst } n \ \text{mod } 7 = 3 \implies \text{c-assoc-have-key } (\text{pr-gr } (\text{loc-upb } n \ x)) \ (\text{c-pair } n \ x) = 0 \)
\textbf{proof} –
\begin{itemize}
  \item assume \( A1: \text{c-fst } n \ \text{mod } 7 = 3 \)
\end{itemize}
let \( \text{?key} = \text{c-pair n} \ x \)
let \( \text{?s} = \text{c-pair ?key} \ 0 \)
let \( \text{?ls} = \text{pr-gr ?s} \)

from \( A1 \) have \( \text{loc-upb n x = ?s + 1} \) by simp
then have \( S1: \text{pr-gr (loc-upb n x) = } g\text{-step (pr-gr ?s) (c-fst ?s)} \) by (simp add: pr-gr-at-Suc)
from \( A1 \) have \( S2: g\text{-step ?ls \ ?key = } c\text{-cons (c-pair ?key (c-snd x)) ?ls} \) by (simp add: g-step-def)
from \( S1 \ S2 \) have \( \text{pr-gr (loc-upb n x) = c-cons (c-pair ?key (c-snd x)) ?ls} \) by auto
thus \( ?\text{thesis by (simp add: c-assoc-lm-1)} \)

qed

lemma \( \text{loc-upb-lex-4: } \bigwedge n' x'. ((n', x'), (n, x)) \in \text{lex-p } \Rightarrow \text{c-assoc-have-key (pr-gr (loc-upb n' x')) (c-pair n' x') = 0;} \)
c-fst n mod 7 = 4 \Rightarrow 
\text{c-assoc-have-key (pr-gr (loc-upb n x)) (c-pair n x) = 0 }

proof –

assume \( A1: \bigwedge n' x'. ((n', x'), (n, x)) \in \text{lex-p } \Rightarrow \text{c-assoc-have-key (pr-gr (loc-upb n' x')) (c-pair n' x') = 0} \)
assume \( A2: \text{c-fst n mod 7 = 4 } \)
let \( \text{?key} = \text{c-pair n} \ x \)
let \( ?m1 = \text{c-fst (c-snd n)} \)
let \( ?m2 = \text{c-snd (c-snd n)} \)

def \( D1: \text{upb1 } = = \text{ loc-upb ?m2 x} \)
from \( A2 \) have \( m2\text{-ll-n: } ?m2 < n \) by (simp add: loc-upb-lm-2-2)
then have \( M2: ((?m2, x), (n, x)) \in \text{lex-p } \) by (simp add: lex-p-eq)
with \( A1 \ D1 \) have \( S1: \text{c-assoc-have-key (pr-gr upb1) (c-pair ?m2 x) = 0} \) by auto
from \( M2 \) have \( M2': ((?m2, x), n, x) \in \text{measure (\lambda m. m) c-lex*> measure (\lambda n. n)} \) by (simp add: lex-p-def)

have \( T1: \text{c-is-sub-fun (pr-gr upb1) univ-for-pr by (rule pr-gr-1)} \)
from \( T1 \ S1 \) have \( T2: \text{c-assoc-value (pr-gr upb1) (c-pair ?m2 x) = univ-for-pr (c-pair ?m2 x) by (rule c-is-sub-fun-lm-1)} \)
def \( D-y: y = = \text{c-assoc-value (pr-gr upb1) (c-pair ?m2 x) } \)
from \( T2 \ D-y \) have \( T3: y = = \text{univ-for-pr (c-pair ?m2 x) by auto } \)

def \( D2: \text{upb2 } = = \text{ loc-upb ?m1 y} \)
from \( A2 \) have \( ?m1 < n \) by (simp add: loc-upb-lm-2-0)
then have \( M1: ((?m1, y), (n, x)) \in \text{lex-p } \) by (simp add: lex-p-eq)
with \( A1 \) have \( S2: \text{c-assoc-have-key (pr-gr (loc-upb ?m1 y)) (c-pair ?m1 y) = 0 } \)
by auto

from \( M1 \) have \( M1': ((?m1, y), n, x) \in \text{measure (\lambda m. m) c-lex*> measure (\lambda n. n)} \) by (simp add: lex-p-def)
from \( S1 \ D1 \) have \( S3: \text{c-assoc-have-key (pr-gr upb1) (c-pair ?m2 x) = 0} \) by auto
from \( S2 \ D2 \) have \( S4: \text{c-assoc-have-key (pr-gr upb2) (c-pair ?m1 y) = 0 } \) by auto

let \( ?s = \text{c-pair ?key (upb1 + upb2)} \)
let \( ?ls = \text{pr-gr ?s} \)
let ?sum-upb = upb1 + upb2
from A2 have ?m1 < n by (simp add: loc-upb-hm-2-0)
then have ((?m1, x), (n,x)) ∈ lex-p by (simp add: lex-p-eq)
then have M1": ((?m1, x), n, x) ∈ measure (λm. m) < lex' > measure (λn. n) by (simp add: lex-p-def)
from A2 M2' M1" have S11: loc-upb n x = (let y = c-assoc-value (pr-gr (loc-upb ?m2 x)) (c-pair ?m2 x)
     in (c-pair (c-pair n x)
         (loc-upb ?m2 x + loc-upb ?m1 y)) + 1)
apply(simp add: Let-def)
done
def upb-def: upb == loc-upb n x
from S11 D-y D1 D2 have loc-upb n x = ?s + 1 by (simp add: Let-def)
with upb-def have S11: upb = ?s + 1 by auto
have S7: ?sum-upb ≤ ?s by (rule arg2-le-c-pair)
have upb1-le-s: upb1 ≤ ?s
proof
  have S1: upb1 ≤ ?sum-upb by (rule Nat.le-add1)
  from S1 S7 show ?thesis by auto
qed
have upb2-le-s: upb2 ≤ ?s
proof
  have S1: upb2 ≤ ?sum-upb by (rule Nat.le-add2)
  from S1 S7 show ?thesis by auto
qed
have S18: pr-gr upb = g-comp ?ls ?key
proof
  from S11 have S1: pr-gr upb = g-step (pr-gr ?ls) (c-fst ?ls) by (simp add: pr-gr-at-Suc)
  from A2 have S2: g-step ?ls ?key = g-comp ?ls ?key by (simp add: g-step-def)
  from S1 S2 show ?thesis by auto
qed
from S3 upb1-le-s have S19: c-assoc-have-key ?ls (c-pair ?m2 x) = 0 by (rule lm5)
from S4 upb2-le-s have S20: c-assoc-have-key ?ls (c-pair ?m1 y) = 0 by (rule lm5)
have T-ls: c-is-sub-fun ?ls univ-for-pr by (rule pr-gr-1)
from T-ls S19 have T-ls2: c-assoc-value ?ls (c-pair ?m2 x) = univ-for-pr (c-pair ?m2 x) by (rule c-is-sub-fun-ln-1)
from T-y S19 S20 have S21: g-comp ?ls ?key = c-cons (c-pair ?key (c-assoc-value ?ls (c-pair ?m1 y))) ?ls
apply(unfold g-comp-def)
apply(simp del: loc-upb.simps add: Let-def)
done
from S18 S21 have pr-gr upb = c-cons (c-pair ?key (c-assoc-value ?ls (c-pair ?m1 y))) by (rule lm5)
lemma loc-upb-lex-5: \[ \bigwedge \ n' x' \cdot ((n', x'), (n, x)) \in \lex-p \Longrightarrow \c-assoc-have-key (pr-gr (loc-upb n' x')) (c-pair n' x') = 0; \]
\[
c\text{fst} n \mod 7 = 5 \Longrightarrow \c-assoc-have-key (pr-gr (loc-upb n x)) (c-pair n x) = 0 \]

proof –
assume \( A1: \bigwedge n' x', ((n', x'), (n, x)) \in \lex-p \Longrightarrow \c-assoc-have-key (pr-gr (loc-upb n' x')) (c-pair n' x') = 0 \)
assume \( A2: \c\text{fst} n \mod 7 = 5 \)
let \( \?m1 = \c\text{fst} (c\text{-snd} n) \)
let \( \?m2 = c\text{-snd} (c\text{-snd} n) \)
from \( A2 \) have \( \?m1 < n \) by \( \text{(simp add: loc-upb-lm-2-3)} \)
then have \( ((?m1, x), (n, x)) \in \lex-p \) by \( \text{(simp add: lex-p-eq)} \)
with \( A1 \) have \( S1: \c-assoc-have-key (pr-gr (loc-upb ?m1 x)) (c-pair ?m1 x) = 0 \)
by auto
from \( A2 \) have \( \?m2 < n \) by \( \text{(simp add: loc-upb-lm-2-4)} \)
then have \( ((?m2, x), (n, x)) \in \lex-p \) by \( \text{(simp add: lex-p-eq)} \)
with \( A1 \) have \( S2: \c-assoc-have-key (pr-gr (loc-upb ?m2 x)) (c-pair ?m2 x) = 0 \)
by auto
def \( D1: \) \text{upb}1 = \text{loc-upb} ?m1 x
def \( D2: \) \text{upb}2 = \text{loc-upb} ?m2 x
from \( D1 \) \( S1 \) have \( S3: \c-assoc-have-key (pr-gr \text{upb}1) (c-pair ?m1 x) = 0 \) by auto
from \( D2 \) \( S2 \) have \( S4: \c-assoc-have-key (pr-gr \text{upb}2) (c-pair ?m2 x) = 0 \) by auto
let \( \?s = (c-pair \?key \?sum-upb) \)
have \( S5: \text{upb}1 \leq \?s \) by \( \text{(rule Nat.le-add1)} \)
have \( S6: \text{upb}2 \leq \?s \) by \( \text{(rule Nat.le-add2)} \)
let \( \?s = (c-pair \?key \?sum-upb) \)
have \( S7: \?sum-upb \leq \?s \) by \( \text{(rule arg2-le-c-pair)} \)
from \( S5 \) \( S7 \) have \( S8: \text{upb}1 \leq \?s \) by auto
from \( S6 \) \( S7 \) have \( S9: \text{upb}2 \leq \?s \) by auto
let \( \?ls = \text{pr-gr} \?s \)
from \( A2 \) \( D1 \) \( D2 \) have \( S10: \text{loc-upb} n x = \?s + 1 \) by \( \text{(simp add: Let-def)} \)
def \( D3: \) \text{upb} = \text{loc-upb} n x
from \( D3 \) \( S10 \) have \( S11: \text{upb} = \?s + 1 \) by auto
from \( S11 \) have \( S12: \text{pr-gr} \text{upb} = \text{g-step} (pr-gr \?s) (c-fst \?s) \) by \( \text{(simp add: pr-gr-at-Suc)} \)
from \( S8 \) \( S10 \) \( D3 \) have \( S13: \text{upb}1 \leq \text{upb} \) by \( \text{(simp only:)} \)
from \( S9 \) \( S10 \) \( D3 \) have \( S14: \text{upb}2 \leq \text{upb} \) by \( \text{(simp only:)} \)
from \( S3 \) \( S13 \) have \( S15: \c-assoc-have-key (pr-gr \text{upb}) (c-pair ?m1 x) = 0 \) by \( \text{(rule lm5)} \)
from \( S4 \) \( S14 \) have \( S16: \c-assoc-have-key (pr-gr \text{upb}) (c-pair ?m2 x) = 0 \) by \( \text{(rule lm5)} \)
from \( A2 \) have \( S17: \text{g-step} \?ls \?key = \text{g-pair} \?ls \?key \) by \( \text{(simp add: g-step-def)} \)

121
from S12 S17 have S18: pr-gr upb = g-pair ?ls ?key by auto
from S3 S8 have S19: c-assoc-have-key ?ls (c-pair ?m1 x) = 0 by (rule lm5)
from S4 S9 have S20: c-assoc-have-key ?ls (c-pair ?m2 x) = 0 by (rule lm5)
let ?y1 = c-assoc-value ?ls (c-pair ?m1 x)
let ?y2 = c-assoc-value ?ls (c-pair ?m2 x)
let ?y = c-pair ?y1 ?y2
from S19 S20 have S21: g-pair ?ls ?key = c-cons (c-pair ?key ?y) ?ls by (unfold g-pair-def, simp add: Let-def)
from S18 S21 have S22: pr-gr upb = c-cons (c-pair ?key ?y) ?ls by auto
from D3 S22 have S23: pr-gr (loc-upb n x) = c-cons (c-pair ?key ?y) ?ls by auto
from S23 show ?thesis by (simp add: c-assoc-lm-1)
qed

lemma loc-upb-6-z: [ c-fst n mod 7 = 6; c-fst x = 0 ] ⇒ loc-upb n x = c-pair (loc-upb (c-fst (c-snd n))) (c-snd x) + 1 by (simp add: Let-def)

lemma loc-upb-6: [ c-fst n mod 7 = 6; c-fst x ≠ 0 ] ⇒ loc-upb n x = (let m = c-snd n; m1 = c-fst m; m2 = c-snd m; y1 = c-fst n (c-pair y2 x1)); t2 = c-pair (c-pair y2 t1) x1 in c-pair (c-pair n x) (loc-upb n (c-pair y2 x1) + (loc-upb m2 t2)) + 1) by (simp add: Let-def)

lemma loc-upb-lex-6: [ (n', x'), (n, x) ∈ lex-p ⇒ c-assoc-have-key (pr-gr (loc-upb n' x')) (c-pair n' x') = 0; c-fst n mod 7 = 6 ] ⇒ c-assoc-have-key (pr-gr (loc-upb n x)) (c-pair n x) = 0
proof –
assume A1: (n', x'), (n, x) ∈ lex-p ⇒ c-assoc-have-key (pr-gr (loc-upb n' x')) (c-pair n' x') = 0
assume A2: c-fst n mod 7 = 6
let ?key = c-pair n x
let ?m1 = c-fst (c-snd n)
let ?m2 = c-snd (c-snd n)
let ?y1 = c-fst x
let ?x1 = c-snd x
def upb-def: upb == loc-upb n x
show ?thesis
proof (cases)
assume A: ?y1 = 0
from A2 A have S1: loc-upb n x = c-pair ?key (loc-upb ?m1 (c-snd x)) + 1 by (rule loc-upb-6-z)
def upb1-def: upb1 == loc-upb ?m1 (c-snd x)
from upb1-def S1 have S2: loc-upb n x = c-pair ?key upb1 + 1 by auto
let ?s = c-pair ?key upb1
from S2 have S3: pr-gr (loc-upb n x) = pr-gr (Suc ?s) by simp
have pr-gr (Suc ?s) = g-step (pr-gr ?s) (c-fst ?s) by (rule pr-gr-at-Suc)
with S3 have S4: pr-gr (loc-upb n x) = g-step (pr-gr ?s) ?key by auto
let ?ls = pr-gr ?s
from A2 have g-step ?ls ?key = g-rec ?ls ?key by (simp add: g-step-def)
with S4 have S5: pr-gr (loc-upb n x) = g-rec ?ls ?key by auto
have S6: c-assoc-have-key ?ls (c-pair ?m1 ?x1) = 0
proof
  from A2 have ?m1 < n by (simp add: loc-upb-lm-2-5)
  then have ((?m1,?x1), n, x) ∈ lex-p by (simp add: lex-p-eq)
  with A1 upb1-def have c-assoc-have-key (pr-gr upb1) (c-pair ?m1 ?x1) = 0
    by auto
  also have upb1 ≤ ?s by (rule arg2-le-c-pair)
ultimately show ?thesis by (rule lm5)
qed

next
assume A: c-fst x ≠ 0 then have y1-pos: c-fst x > 0 by auto
let ?y2 = ?y1 - 1
from A2 A have loc-upb n x = (let m = c-snd n; m1 = c-fst m; m2 = c-snd m; y1 = c-fst
x; x1 = c-snd x;
y2 = y1 - 1;
t1 = c-assoc-value (pr-gr (loc-upb n (c-pair y2 x1))); (c-pair
n (c-pair y2 x1));
t2 = c-pair (c-pair y2 t1) x1 in
  c-pair (c-pair n x) (loc-upb n (c-pair y2 x1) + (loc-upb
m2 t2)) + 1) by (rule loc-upb-6)
then have S1: loc-upb n x = (let
  t1 = c-assoc-value (pr-gr (loc-upb n (c-pair ?y2 ?x1)));
  t2 = c-pair (c-pair ?y2 t1) ?x1 in
  c-pair (c-pair ?y2 t1) ?x1 in
  c-pair (c-pair n x) (loc-upb n (c-pair ?y2 ?x1) + (loc-upb
?m2 t2)) + 1) by (simp del: loc-upb.simps add: Let-def)
let ?t1 = univ-for-pr (c-pair n (c-pair ?y2 ?x1))
let ?t2 = c-pair (c-pair ?y2 ?t1) ?x1
have S1-1: c-assoc-have-key (pr-gr (loc-upb n (c-pair ?y2 ?x1))) (c-pair n (c-pair
?y2 ?x1)) = 0
proof
  from A have ?y2 < ?y1 by auto
  then have c-pair ?y2 ?x1 < c-pair ?y1 ?x1 by (rule c-pair-strict-mono1)
  then have ((n, c-pair ?y2 ?x1),n,x) ∈ lex-p by (simp add: lex-p-eq)
  with A1 show ?thesis by auto
qed
have \( S2: c\text{-assoc-value} (pr\text{-gr} (loc\text{-upb} n (c\text{-pair } y2 \ ?x1))) (c\text{-pair } n (c\text{-pair } y2 \ ?x1)) = \text{univ-for-pr} (c\text{-pair } n (c\text{-pair } y2 \ ?x1)) \)

proof 

have c-is-sub-fun (pr-gr (loc-upb n (c-pair ?y2 \ ?x1))) univ-for-pr by (rule pr-gr-1)

with \( S1-1 \) show thesis by (simp add: c-is-sub-fun-lm-1)

qed

from \( S1 \) \( S2 \) have \( S3: \) loc-upb \( n \) \( x = \) c-pair \( (c\text{-pair } n \ x) \) \( (c\text{-pair } y2 \ ?x1) + \text{loc-upb} \ ?m2 \ ?t2 \) + 1 by (simp del: loc-upb.simps add: Let-def)

let \( s = \text{c-pair} (c\text{-pair } n \ x) \) \( (c\text{-pair } y2 \ ?x1) + \text{loc-upb} \ ?m2 \ ?t2 \)

from \( S3 \) have \( S4: \) pr-gr \( (\text{loc-upb} \ n \ x) = \) pr-gr \( (\text{Suc } s) \) by (simp del: loc-upb.simps)

have pr-gr (Suc \( s \)) = g-step (pr-gr \( s \)) (c-fst \( s \)) by (rule pr-gr-at-Suc)

with \( S4 \) have \( S5: \) pr-gr \( (\text{loc-upb} \ n \ x) = \) g-step (pr-gr \( s \)) \( ?k \) by (simp del: loc-upb.simps)

let \( \ls = \text{pr-gr} \ s \)

from \( A2 \) have \( \ls \) \( \leq \ls \) \( \leq \ls \text{ by} \) (auto simp del: loc-upb.simps)

ultimately have \( S7-1: \) loc-upb \( n \) \( (c\text{-pair } y2 \ ?x1) \leq \ls \text{ by} \) (auto simp del: loc-upb.simps)

let \( \ls = \text{pr-gr} \ s \)

from \( A2 \) have \( ?m2 < n \) by (simp add: loc-upb-lnm-2-6)

then have \( ((?m2, ?t2), n, x) \in \text{lex-p} \) by (simp add: lex-p-eq)

with \( A1 \) have c-assoc-have-key (pr-gr (loc-upb (?m2 \ ?t2))) (c-pair ?m2 \ ?t2) = 0 by auto

also have loc-upb \( ?m2 \ ?t2 \leq ?s \)

ultimately have \( \) thesis by (auto simp del: loc-upb.simps)

qed
(c-pair ?m2 ?t2)) ?ls by (simp del: loc-upb.simps add: g-rec-def Let-def)
  with S6 show ?thesis by (simp add: c-assoc-lm-1)
qed

lemma wf-upb-step-0:
  [\(\forall n' x'. ((n',x'), (n,x)) \in \text{lex-p} \implies \text{c-assoc-have-key} (\text{pr-gr} (\text{loc-upb } n' x'))\)
  (c-pair n' x') = 0] \implies
  c-assoc-have-key (pr-gr (loc-upb n x)) (c-pair n x) = 0

proof –
  assume A1: \(\forall n' x'. ((n',x'), (n,x)) \in \text{lex-p} \implies \text{c-assoc-have-key} (\text{pr-gr} (\text{loc-upb } n' x'))\)
  (c-pair n' x') = 0
  let ?n1 = (c-fst n) mod 7
  have S1: ?n1 = 0 \implies ?thesis
    proof
      assume A: ?n1 = 0
      thus ?thesis by (rule loc-upb-lex-0)
    qed
  have S2: ?n1 = 1 \implies ?thesis
    proof
      assume A: ?n1 = 1
      thus ?thesis by (rule loc-upb-lex-1)
    qed
  have S3: ?n1 = 2 \implies ?thesis
    proof
      assume A: ?n1 = 2
      thus ?thesis by (rule loc-upb-lex-2)
    qed
  have S4: ?n1 = 3 \implies ?thesis
    proof
      assume A: ?n1 = 3
      thus ?thesis by (rule loc-upb-lex-3)
    qed
  have S5: ?n1 = 4 \implies ?thesis
    proof
      assume A: ?n1 = 4
      from A1 A show ?thesis by (rule loc-upb-lex-4)
    qed
  have S6: ?n1 = 5 \implies ?thesis
    proof
      assume A: ?n1 = 5
      from A1 A show ?thesis by (rule loc-upb-lex-5)
    qed
  have S7: ?n1 = 6 \implies ?thesis
    proof
      assume A: ?n1 = 6
      from A1 A show ?thesis by (rule loc-upb-lex-6)
    qed
  have S8: ?n1=0 \lor ?n1=1 \lor ?n1=2 \lor ?n1=3 \lor ?n1=4 \lor ?n1=5 \lor ?n1=6
by (rule mod7-lm)
  from S1 S2 S3 S4 S5 S6 S7 S8 show ?thesis by fast
qed

lemma wf-upb-step:
  assumes A1: \( \forall p2. (p2, p1) \in \text{lex-p} \implies \)
  \( \text{c-assoc-have-key} (\text{pr-gr} (\text{loc-upb} (\text{fst} p2) (\text{snd} p2))) (\text{c-pair} (\text{fst} p2) (\text{snd} p2)) \) = 0
  shows \( \text{c-assoc-have-key} (\text{pr-gr} (\text{loc-upb} (\text{fst} p1) (\text{snd} p1))) (\text{c-pair} (\text{fst} p1) (\text{snd} p1)) \) = 0
proof –
  let \(?n = \text{fst} p1\)
  let \(?x = \text{snd} p1\)
  from A1 have S1: \( \forall p2. ((?n, ?x), (p2, ?p)) \in \text{lex-p} \implies \)
  \( \text{c-assoc-have-key} (\text{pr-gr} (\text{loc-upb} (?n x')) (\text{c-pair} (?n x')) = 0) \implies \)
  \( \text{c-assoc-have-key} (\text{pr-gr} (\text{loc-upb} (\text{fst} p1) (\text{snd} p1))) (\text{c-pair} (\text{fst} p1) (\text{snd} p1)) \) = 0
  by auto
have S2: \( \forall n' x'. ((n', x'), (\text{fst} p1, \text{snd} p1)) \in \text{lex-p} \implies \)
  \( \text{c-assoc-have-key} (\text{pr-gr} (\text{loc-upb} n' x')) (\text{c-pair} n' x') = 0 \)
proof –
  fix \( n' x' \)
  assume A4-1: \( ((n', x'), (n, p1)) \in \text{lex-p} \)
  let \(?p2 = (n', x')\)
  from A4-1 have S4-1: \( (\text{fst} ?, \text{snd} p2) \) \( \in \text{lex-p} \) by auto
  from S4-1 have \( \text{c-assoc-have-key} (\text{pr-gr} (\text{loc-upb} (\text{fst} ?) (\text{snd} ?))) (\text{c-pair} (\text{fst} ?) (\text{snd} ?2)) \) = 0
  by (rule A1)
  then show \( \text{c-assoc-have-key} (\text{pr-gr} (\text{loc-upb} n' x')) (\text{c-pair} n' x') = 0 \) by auto
qed
from S4 S3 show ?thesis by auto
qed

theorem loc-upb-main: \( \text{c-assoc-have-key} (\text{pr-gr} (\text{loc-upb} n x)) (\text{c-pair} n x) = 0 \)
proof –
  have loc-upb-lm: \( \forall p. \text{c-assoc-have-key} (\text{pr-gr} (\text{loc-upb} (\text{fst} p) (\text{snd} p))) (\text{c-pair} (\text{fst} p) (\text{snd} p)) \) = 0
  proof –
    fix p show \( \text{c-assoc-have-key} (\text{pr-gr} (\text{loc-upb} (\text{fst} p) (\text{snd} p))) (\text{c-pair} (\text{fst} p) (\text{snd} p)) \) = 0
    proof –
      have S1: \( \text{wf \ lex-p} \) by (auto simp add: lex-p-def)
from S1 wf-upb-step show ?thesis by (rule wf-induct-rule)
qed
qed
let ?p = (n,x)
have c-assoc-have-key (pr-gr (loc-upb (fst ?p) (snd ?p))) (c-pair (fst ?p) (snd ?p)) = 0 by (rule loc-upb-lm)
thus ?thesis by simp
qed

theorem pr-gr-value: c-assoc-value (pr-gr (loc-upb n x)) (c-pair n x) = univ-for-pr (c-pair n x) by (simp del: loc-upb.simps add: loc-upb-main pr-gr-1 c-is-sub-fun-lm-1)

theorem g-comp-is-pr: g-comp ∈ PrimRec2
proof –
from c-assoc-have-key-is-pr c-assoc-value-is-pr c-cons-is-pr have (λ x y. g-comp x y) ∈ PrimRec2
unfolding g-comp-def Let-def by prec
thus ?thesis by auto
qed

theorem g-pair-is-pr: g-pair ∈ PrimRec2
proof –
from c-assoc-have-key-is-pr c-assoc-value-is-pr c-cons-is-pr have (λ x y. g-pair x y) ∈ PrimRec2
unfolding g-pair-def Let-def by prec
thus ?thesis by auto
qed

theorem g-rec-is-pr: g-rec ∈ PrimRec2
proof –
from c-assoc-have-key-is-pr c-assoc-value-is-pr c-cons-is-pr have (λ x y. g-rec x y) ∈ PrimRec2
unfolding g-rec-def Let-def by prec
thus ?thesis by auto
qed

theorem g-step-is-pr: g-step ∈ PrimRec2
proof –
from g-comp-is-pr g-pair-is-pr g-rec-is-pr mod-is-pr c-assoc-have-key-is-pr c-assoc-value-is-pr c-cons-is-pr have
(λ ls key. g-step ls key) ∈ PrimRec2 unfolding g-step-def Let-def by prec
thus ?thesis by auto
qed

theorem pr-gr-is-pr: pr-gr ∈ PrimRec1
proof –
have S1: (λ x. pr-gr x) = PrimRecOp1 0 (λ x y. g-step y (c-fst x)) (is - = ?f)
proof
7 Computably enumerable sets of natural numbers

theory RecEnSet
imports PRecList PRecFun2 PRecFinSet PRecUnGr
begin

7.1 Basic definitions

definition fn-to-set :: (nat ⇒ nat ⇒ nat) ⇒ nat set where
  fn-to-set f = { x. ∃ y. f x y = 0 }

definition ce-sets :: (nat set) set where
  ce-sets = { (fn-to-set p) | p. p ∈ PrimRec2 }

7.2 Basic properties of computably enumerable sets

lemma ce-set-lm-1: p ∈ PrimRec2 ⇒ fn-to-set p ∈ ce-sets by (auto simp add: ce-sets-def)

lemma ce-set-lm-2: [ p ∈ PrimRec2; ∀ x. (x ∈ A) = (∃ y. p x y = 0) ] ⇒ A ∈ ce-sets
  proof –
  assume p-is-pr: p ∈ PrimRec2
  assume ∀ x. (x ∈ A) = (∃ y. p x y = 0)
  then have A = fn-to-set p by (unfold fn-to-set-def , auto)
  with p-is-pr show A ∈ ce-sets by (simp add: ce-set-lm-1)
  qed

lemma ce-set-lm-3: A ∈ ce-sets ⇒ ∃ p ∈ PrimRec2. A = fn-to-set p
  proof –
  assume A ∈ ce-sets
  then have A ∈ { (fn-to-set p) | p. p ∈ PrimRec2 } by (simp add: ce-sets-def)
  thus ?thesis by auto
  qed
lemma ce-set-lm-4: $A \in \text{ce-sets} \implies \exists \ p \in \text{PrimRec2}. \ \forall x. \ (x \in A) = (\exists y. \ p \ x \ y = 0)$

proof –
  assume $A \in \text{ce-sets}$
  then have $\exists \ p \in \text{PrimRec2}. \ A = \text{fn-to-set} \ p$ by (rule ce-set-lm-3)
  then obtain $p$ where $p$-is-pr: $p \in \text{PrimRec2}$ and $L1: A = \text{fn-to-set} \ p$ ..
  from $p$-is-pr $L1$ show \text{thesis} by (unfold fn-to-set-def, auto)
qed

lemma ce-set-lm-5: $\ [ A \in \text{ce-sets}; \ p \in \text{PrimRec1} \ ] \implies \{ x . \ p \ x \in A \} \in \text{ce-sets}$

proof –
  assume $A1: A \in \text{ce-sets}$
  assume $A2: p \in \text{PrimRec1}$
  from $A1$ have $\exists \ pA \in \text{PrimRec2}. \ A = \text{fn-to-set} \ pA$ by (rule ce-set-lm-3)
  then obtain $pA$ where $pA$-is-pr: $pA \in \text{PrimRec2}$ and $S1: A = \text{fn-to-set} \ pA$ ..
  from $S1$ have $S2: A = \{ x . \ \exists y. \ pA \ x \ y = 0 \}$ by (simp add: fn-to-set-def)
  def $q$-def: $q \equiv \lambda x y. \ pA (p \ x) \ y$
  from $pA$-is-pr $A2$ have $q$-is-pr: $q \in \text{PrimRec2}$ unfolding $q$-def by prec
  have $\bigwedge x. \ (p \ x \in A) = (\exists y. \ q \ x \ y = 0)$
proof –
  fix $x$ show $(p \ x \in A) = (\exists y. \ q \ x \ y = 0)$
proof –
  assume $A: p \ x \in A$
  with $S2$ obtain $y$ where $L1: pA (p \ x) \ y = 0$ by auto
  then have $q \ x \ y = 0$ by (simp add: $q$-def)
  thus $\exists y. \ q \ x \ y = 0$ ..
next
  assume $A: \exists y. \ q \ x \ y = 0$
  then obtain $y$ where $L1: q \ x \ y = 0$ ..
  then have $pA (p \ x) \ y = 0$ by (simp add: $q$-def)
  with $S2$ show $p \ x \in A$ by auto
qed
qed

then have $\{ x . \ p \ x \in A \} = \{ x . \ \exists y. \ q \ x \ y = 0 \}$ by auto
then have $\{ x . \ p \ x \in A \} = \text{fn-to-set} \ q$ by (simp add: fn-to-set-def)
moreover from $q$-is-pr have \text{fn-to-set} $q \in$ \text{ce-sets} by (rule ce-set-lm-1)
ultimately show \text{thesis} by auto
qed

lemma ce-set-lm-6: $\ [ A \in \text{ce-sets}; \ A \neq \{\} \ ] \implies \exists q \in \text{PrimRec1}. \ A = \{ q \ x \ | \ x. \ x \in \text{UNIV} \}$

proof –
  assume $A1: A \in \text{ce-sets}$
  assume $A2: A \neq \{\}$
  from $A1$ have $\exists \ pA \in \text{PrimRec2}. \ A = \text{fn-to-set} \ pA$ by (rule ce-set-lm-3)
  then obtain $pA$ where $pA$-is-pr: $pA \in \text{PrimRec2}$ and $S1: A = \text{fn-to-set} \ pA$ ..
  from $S1$ have $S2: A = \{ x . \ \exists y. \ pA \ x \ y = 0 \}$ by (simp add: fn-to-set-def)
  from $A2$ obtain $a$ where $a$-in: $a \in A$ by auto

129
\textbf{def} q-def: \(q \equiv \lambda z. \text{if } pA \ (c\text{-fst } z) \ (c\text{-snd } z) = 0 \text{ then } c\text{-fst } z \text{ else } a\)

from pA-is-pr have q-is-pr: \(q \in \text{PrimRec1}\) unfolding q-def by \textit{prece}

have S3: \(\forall \ z. \ q \ z \in A\)

proof
  \begin{itemize}
  \item \textbf{fix} \(z\) show \(q \ z \in A\)
  \item \textbf{proof cases}
  \item \textbf{assume} \(A: \ pA \ (c\text{-fst } z) \ (c\text{-snd } z) = 0\)
  \item \textbf{with} \(S2\) have \(c\text{-fst } z \in A\) by \textit{auto}
  \item moreover from \(A\) q-def have \(q \ z = c\text{-fst } z\) by \textit{simp}
  \item ultimately show \(q \ z \in A\) by \textit{auto}
  \item next
  \item \textbf{assume} \(A: \ pA \ (c\text{-fst } z) \ (c\text{-snd } z) \neq 0\)
  \item \textbf{with} q-def have \(q \ z = a\) by \textit{simp}
  \item \textbf{with} a-in show \(q \ z \in A\) by \textit{auto}
  \end{itemize}
\end{proof}

\textbf{qed}

\textbf{qed}

then have \(S4: \{ \ q \ x \ | \ x \in \text{UNIV} \ \} \subseteq A\) by \textit{auto}

have \(S5: A \subseteq \{ \ q \ x \ | \ x \in \text{UNIV} \ \}\)

proof
  \begin{itemize}
  \item \textbf{fix} \(x\) assume \(A: \ x \in A\) show \(x \in \{ q \ x | x \in \text{UNIV} \}\)
  \item \textbf{proof}
  \item \textbf{from} \(A\) \(S2\) obtain \(y\) where \(L1: \ pA \ x y = 0\) by \textit{auto}
  \item \textbf{let} \(?z = c\text{-pair } x \ y\)
  \item \textbf{from} \(L1\) have \(q \ ?z = x\) by \textit{(simp add: q-def)}
  \item \textbf{then have} \(\exists \ u. \ q u = x\) by \textit{blast}
  \item \textbf{then show} \(\exists \ u. \ x = q u \land u \in \text{UNIV}\) by \textit{auto}
  \end{itemize}
\end{proof}

\textbf{qed}

from \(S4\) \(S5\) have \(S6: A = \{ \ q \ x \ | \ x \in \text{UNIV} \ \}\) by \textit{auto}

with q-is-pr show \(\exists \ q\) \textit{thesis} by \textit{blast}

\textbf{qed}

\textbf{lemma} ce-set-lm-7: \(\forall \ A \in \text{ce-sets}; \ p \in \text{PrimRec1}\) \(\Rightarrow \{ \ p \ x | x \in A \ \} \in \text{ce-sets}\)

\textbf{proof –}

\begin{itemize}
  \item \textbf{assume} \(A1: \ A \in \text{ce-sets}\)
  \item \textbf{assume} \(A2: \ p \in \text{PrimRec1}\)
  \item \textbf{let} \(\ ?B = \{ \ p \ x | x, x \in A \ \}\)
  \item fix \(y\) have \(S1: \ (y \in \ ?B) = (\exists \ x. \ x \in A \land (y = p \ x))\) by \textit{auto}
  \item from \(A1\) have \(\exists \ pA \in \text{PrimRec2}. \ A = \text{fn-to-set } pA\) by \textit{(rule ce-set-lm-3)}
  \item then obtain \(pA\) where \(pA\text{-is-pr}: \ pA \in \text{PrimRec2} \text{ and } S2: \ A = \text{fn-to-set } pA\) ..
  \item from \(S2\) have \(S3: A = \{ \ x. \ \exists y. \ pA \ x y = 0 \ \}\) by \textit{(simp add: fn-to-set-def)}
  \item def q-def: \(q \equiv \lambda \ y. \ t. \ \text{if } y = p \ (c\text{-snd } t) \text{ then } pA \ (c\text{-snd } t) \ (c\text{-fst } t) \text{ else } 1\)
  \item from pA-is-pr A2 have q-is-pr: \(q \in \text{PrimRec2}\) unfolding q-def by \textit{prece}
  \item have \(L1: \ \land \ y. \ (y \in \ ?B) = (\exists \ z. \ q y z = 0)\)
  \item \textbf{proof –} \textbf{fix} \(y\) show \(\ (y \in \ ?B) = (\exists \ z. \ q y z = 0)\)
  \item \textbf{proof –}
  \item \textbf{assume} \(\ A A1: \ y \in ?B\)
  \item then obtain \(x0\) where \(LL-2: x0 \in A\) and \(LL-3: y = p x0\) by \textit{auto}
  \item from \(S3\) have \(LL-4: (x0 \in A) = (\exists \ z. \ pA x0 z = 0)\) by \textit{auto}
  \end{itemize}
from LL-2 LL-4 obtain z0 where LL-5: pA z0 z0 = 0 by auto

def t-def: t ≡ c-pair z0 x0

from t-def q-def LL-3 LL-5 have q y t = 0 by simp
then show ∃ z. q y z = 0 by auto

next
assume A1: ∃ z. q y z = 0
then obtain z0 where LL-1: q y z0 = 0..

have LL2: y = p (c-snd z0)
proof (rule ccontr)
assume y ≠ p (c-snd z0)
with q-def LL-1 have q y z0 = 1 by auto
with LL-1 show False by auto
qed

from LL2 LL-1 q-def have LL3: pA (c-snd z0) (c-fst z0) = 0 by auto
with S3 have LL4: c-snd z0 ∈ A by auto
with LL2 show y ∈ {p x | x. x ∈ A} by auto
qed

then have L2: ?B = {y | y. ∃ z. q y z = 0} by auto
with fn-to-set-def have ?B = fn-to-set q by auto
with q-is-pr ce-set-lm-1 show ?thesis by auto
qed

theorem ce-empty: {} ∈ ce-sets
proof –
let ?f = (λ x a. (1::nat))
have S1: ?f ∈ PrimRec2 by (rule const-is-pr-2)
then have ∀ x a. ?f x a ≠ 0 by simp
then have {x. ∃ a. ?f x a = 0} = {} by auto
also have fn-to-set ?f = . . . by (simp add: fn-to-set-def)
with S1 show ?thesis by (auto simp add: ce-sets-def)
qed

theorem ce-univ: UNIV ∈ ce-sets
proof –
let ?f = (λ x a. (0::nat))
have S1: ?f ∈ PrimRec2 by (rule const-is-pr-2)
then have ∀ x a. ?f x a = 0 by simp
then have {x. ∃ a. ?f x a = 0} = UNIV by auto
also have fn-to-set ?f = . . . by (simp add: fn-to-set-def)
with S1 show ?thesis by (auto simp add: ce-sets-def)
qed

theorem ce-singleton: {a} ∈ ce-sets
proof –
let ?f = λ x y. (abs-of-diff x a) + y
have S1: ?f ∈ PrimRec2 using const-is-pr-2 [where ?n=a] by prec
then have ∀ x y. (?f x y = 0) = (x=a ∧ y=0) by (simp add: abs-of-diff-eq)
then have S2: {x. ∃ y. ?f x y = 0} = {a} by auto
have \( fn-to-set \ \{ x. \ \exists y. \ \exists f x y = 0 \} \) by (simp add: fn-to-set-def)
with \( S2 \) have \( fn-to-set \ \{ a \} \) by simp
with \( S1 \) show \( \text{thesis} \) by (auto simp add: ce-sets-def)
qed

**Theorem** ce-union: \[ A \in \text{ce-sets}; B \in \text{ce-sets} \] \implies A \cup B \in \text{ce-sets}

**Proof**

1. Assume \( A1: A \in \text{ce-sets} \)
2. Then obtain \( p-a \) where \( S2: p-a \in \text{PrimRec2 and} \ S3: A = fn-to-set p-a \)
   by (auto simp add: ce-sets-def)
3. Assume \( A2: B \in \text{ce-sets} \)
4. Then obtain \( p-b \) where \( S5: p-b \in \text{PrimRec2 and} \ S6: B = fn-to-set p-b \)
   by (auto simp add: ce-sets-def)
5. Let \( ?p = (\lambda x y. (p-a x y) \ast (p-b x y)) \)
6. From \( S2 \) have \( S5: \ ?p \in \text{PrimRec2 by prec} \)
7. Have \( S8: \ \forall x y. (\exists z. p-a x z = 0) \lor (\exists z. p-b x z = 0) \) by simp
8. Let \( ?C = \text{fn-to-set} \ ?p \)
9. Have \( S9: ?C = \{ x. \ \exists y. ?p x y = 0 \} \) by (simp add: fn-to-set-def)
10. From \( S3 \) have \( S10: A = \{ x. \ \exists y. p-a x y = 0 \} \) by (simp add: fn-to-set-def)
11. From \( S6 \) have \( S11: B = \{ x. \ \exists y. p-b x y = 0 \} \) by (simp add: fn-to-set-def)
12. From \( S10 \) \( S11 \) \( S9 \) \( S8 \) have \( S12: ?C = A \cup B \) by auto
13. From \( S7 \) have \( ?C \in \text{ce-sets} \) by (auto simp add: ce-sets-def)

With \( S12 \) show \( \text{thesis} \) by simp
qed

**Theorem** ce-intersect: \[ A \in \text{ce-sets}; B \in \text{ce-sets} \] \implies A \cap B \in \text{ce-sets}

**Proof**

1. Assume \( A1: A \in \text{ce-sets} \)
2. Then obtain \( p-a \) where \( S2: p-a \in \text{PrimRec2 and} \ S3: A = fn-to-set p-a \)
   by (auto simp add: ce-sets-def)
3. Assume \( A2: B \in \text{ce-sets} \)
4. Then obtain \( p-b \) where \( S5: p-b \in \text{PrimRec2 and} \ S6: B = fn-to-set p-b \)
   by (auto simp add: ce-sets-def)
5. Let \( ?p = (\lambda x y. (p-a x (c-fst y)) + (p-b x (c-snd y))) \)
6. From \( S2 \) \( S5 \) have \( S7: \ ?p \in \text{PrimRec2 by prec} \)
7. Have \( S8: \ \forall x. (\exists y. ?p x y = 0) = (\exists z. p-a x z = 0) \land (\exists z. p-b x z = 0) \) by auto
   - Fix \( x \) show \( (\exists y. ?p x y = 0) = (\exists z. p-a x z = 0) \land (\exists z. p-b x z = 0) \)
     - Have \( I: (\exists y. ?p x y = 0) \implies (\exists z. p-a x z = 0) \land (\exists z. p-b x z = 0) \)
       by blast
     - Have \( 2: (\exists z. p-a x z = 0) \land (\exists z. p-b x z = 0) \implies (\exists y. ?p x y = 0) \)
       by auto
     - Assume \( (\exists z. p-a x z = 0) \land (\exists z. p-b x z = 0) \)
     - Then obtain \( z1 \) \( z2 \) where \( s-23: p-a x z1 = 0 \) and \( s-24: p-b x z2 = 0 \) by auto
6. Let \( ?y1 = c-pair z1 z2 \)
7. From \( s-23 \) have \( s-25: p-a x (c-fst ?y1) = 0 \) by simp
8. From \( s-24 \) have \( s-26: p-b x (c-snd ?y1) = 0 \) by simp

132
from s-25 s-26 have s-27: p-a x (c-fst ?y1) + p-b x (c-snd ?y1) = 0 by simp then show ?thesis .. qed
from 1 2 have (∃ y. ?p x y = 0) = ((∃ z. p-a x z = 0) ∧ (∃ z. p-b x z = 0)) by (rule iffI) then show ?thesis by auto qed

let ?C = fn-to-set ?p have S9: ?C = { x. ∃ y. ?p x y = 0} by (simp add: fn-to-set-def)
from S3 have S10: A = {x. ∃ y. p-a x y = 0} by (simp add: fn-to-set-def)
from S6 have S11: B = {x. ∃ y. p-b x y = 0} by (simp add: fn-to-set-def)
from S10 S11 S9 S8 have S12: ?C = A ∩ B by auto
from S7 have ?C ∈ ce-sets by (auto simp add: ce-sets-def)
with S12 show ?thesis by simp qed

7.3 Enumeration of computably enumerable sets
definition
  nat-to-ce-set :: nat ⇒ (nat set) where
  nat-to-ce-set = (λ n. fn-to-set (pr-conv-1-to-2 (nat-to-pr n)))
lemma nat-to-ce-set-lm-1: nat-to-ce-set n = { x . ∃ y. (nat-to-pr n) (c-pair x y) = 0 }
  proof – have S1: nat-to-ce-set n = fn-to-set (pr-conv-1-to-2 (nat-to-pr n)) by (simp add: nat-to-ce-set-def)
  then have S2: nat-to-ce-set n = { x . ∃ y. (pr-conv-1-to-2 (nat-to-pr n)) x y = 0} by (simp add: fn-to-set-def)
  have S3: ∃ x y. (pr-conv-1-to-2 (nat-to-pr n)) x y = (nat-to-pr n) (c-pair x y)
  by (simp add: pr-conv-1-to-2-def)
  from S2 S3 show ?thesis by auto qed
lemma nat-to-ce-set-into-ce: nat-to-ce-set n ∈ ce-sets
  proof – have S1: nat-to-ce-set n = fn-to-set (pr-conv-1-to-2 (nat-to-pr n)) by (simp add: nat-to-ce-set-def)
  have (nat-to-pr n) ∈ PrimRec1 by (rule nat-to-pr-into-pr)
  then have S2: (pr-conv-1-to-2 (nat-to-pr n)) ∈ PrimRec2 by (rule pr-conv-1-to-2-lm)
  from S2 S1 show ?thesis by (simp add: ce-set-lm-1)
  qed
lemma nat-to-ce-set-srj: A ∈ ce-sets ⇒ ∃ n. A = nat-to-ce-set n
  proof – assume A: A ∈ ce-sets
  then have ∃ p ∈ PrimRec2. A = fn-to-set p by (rule ce-set-lm-3)
then obtain \( p \) where \( p \text{-is-pr} \): \( p \in \text{PrimRec2} \) and \( S1: A = \text{fn-to-set} p \)

**def** \( q \text{-def} \): \( q \equiv \text{pr-conv-2-to-1} p \)

from \( p \text{-is-pr} \) have \( q \text{-is-pr} \): \( q \in \text{PrimRec1} \) by (unfold \( q \text{-def} \), rule \( \text{pr-conv-2-to-1-lm} \))

from \( q \text{-def} \) have \( S2: \text{pr-conv-1-to-2} q = p \) by simp

let \( ?n = \text{index-of-pr} q \)

from \( q \text{-is-pr} \) have \( \text{nat-to-pr} ?n = q \) by (rule \( \text{index-of-pr-is-real} \))

with \( S2 S1 \) have \( A = \text{fn-to-set} (\text{pr-conv-1-to-2} (\text{nat-to-pr} ?n)) \) by auto

thus \( \text{?thesis} \) ..

qed

7.4 Characteristic functions

**definition**

\[ \text{chf} :: \text{nat set} \Rightarrow (\text{nat} \Rightarrow \text{nat}) \] — Characteristic function

**where**

\[ \text{chf} = (\lambda A \ x. \text{if } x \in A \text{ then } 0 \text{ else } 1) \]

**definition**

\[ \text{zero-set} :: (\text{nat} \Rightarrow \text{nat}) \Rightarrow \text{nat set} \]

**where**

\[ \text{zero-set} = (\lambda f. \{ x. f x = 0 \}) \]

**lemma** \( \text{chf-lm-1} \) [simp]: \( \text{zero-set} (\text{chf} A) = A \) by (unfold \( \text{chf-def} \), unfold \( \text{zero-set-def} \), simp)

**lemma** \( \text{chf-lm-2} \): \( (x \in A) = (\text{chf} A x = 0) \) by (unfold \( \text{chf-def} \), simp)

**lemma** \( \text{chf-lm-3} \): \( (x \not\in A) = (\text{chf} A x = 1) \) by (unfold \( \text{chf-def} \), simp)

**lemma** \( \text{chf-lm-4} \):

\[ \text{chf} A \in \text{PrimRec1} \Rightarrow A \in \text{ce-sets} \]

**proof** –

assume \( A: \text{chf} A \in \text{PrimRec1} \)

**def** \( p \text{-def} \): \( p \equiv \text{chf} A \)

from \( A p \text{-def} \) have \( p \text{-is-pr} \): \( p \in \text{PrimRec1} \) by auto

**def** \( q \text{-def} \): \( q \equiv \lambda x. (y: \text{nat}). \ p \ x \)

from \( p \text{-is-pr} \) have \( q \text{-is-pr} \): \( q \in \text{PrimRec2} \) unfolding \( q \text{-def} \) by prec

have \( S1: A = \{ x. \ x \in A \} \)

**proof** –

have \( \text{zero-set} p = A \) by (unfold \( p \text{-def} \), simp)

thus \( \text{?thesis} \) by (simp add: \( \text{zero-set-def} \))

qed

have \( S2: \text{fn-to-set} q = \{ x. \exists y. \ x y = 0 \} \) by (simp add: \( \text{fn-to-set-def} \))

have \( S3: \\forall x. (p x = 0) = (\exists y. \ x y = 0) \) by (unfold \( q \text{-def} \), auto)

then have \( S4: \{ x. \ p x = 0 \} = \{ x. \exists y. \ x y = 0 \} \) by auto

with \( S1 S2 \) have \( S5: \text{fn-to-set} q = A \) by auto

from \( q \text{-is-pr} \) have \( \text{fn-to-set} q \in \text{ce-sets} \) by (rule \( \text{ce-set-lm-1} \))

with \( S5 \) show \( \text{?thesis} \) by auto

qed

**lemma** \( \text{chf-lm-5} \): \( \text{finite} A \Rightarrow \text{chf} A \in \text{PrimRec1} \)

134
proof
  assume A: finite A
  def u-def: u ≡ set-to-nat A
  from A have S1: nat-to-set u = A by (unfold u-def, rule nat-to-set-srj)
  have chf A = (λ x. sgn2 (c-in x u))
  proof
    fix x show chf A x = sgn2 (c-in x u)
    proof cases
      assume A: x ∈ A
      then have S1-1: chf A x = 0 by (simp add: chf-lm-2)
      from A S1 have x ∈ nat-to-set u by auto
      then have c-in x u = 1 by (simp add: x-in-u-eq)
      with S1-1 show ?thesis by simp
    next
      assume A: x /∈ A
      then have S1-1: chf A x = 1 by (simp add: chf-def)
      from A S1 have x /∈ nat-to-set u by auto
      then have c-in x u = 0 by (simp add: x-in-u-eq c-in-def)
      with S1-1 show ?thesis by simp
    qed
  qed
  moreover from c-in-is-pr have (λ x. sgn2 (c-in x u)) ∈ PrimRec1 by prec
  ultimately show ?thesis by auto
  qed

theorem ce-finite: finite A ⇒ A ∈ ce-sets
proof
  assume A: finite A
  then have chf A ∈ PrimRec1 by (rule chf-lm-5)
  then show ?thesis by (rule chf-lm-4)
  qed

7.5 Computably enumerable relations

definition
  ce-set-to-rel :: nat set ⇒ (nat * nat) set where
  ce-set-to-rel = (λ A. { (c-fst x, c-snd x) | x. x ∈ A})

definition
  ce-rel-to-set :: (nat * nat) set ⇒ nat set where
  ce-rel-to-set = (λ R. { c-pair x y | x y. (x,y) ∈ R})

definition
  ce-rels :: ((nat * nat) set) set where
  ce-rels = { R | R. ce-rel-to-set R ∈ ce-sets }

lemma ce-rel-lm-1 [simp]: ce-set-to-rel (ce-rel-to-set r) = r
proof
  show ce-set-to-rel (ce-rel-to-set r) ⊆ r

135
proof fix \( z \)

assume \( A : z \in \text{ce-set-to-rel (ce-rel-to-set } r) \)
then obtain \( u \) where \( L1 : u \in (\text{ce-rel-to-set } r) \) and \( L2 : z = (\text{c-fst } u, \text{c-snd } u) \)
unfolding \( \text{ce-set-to-rel-def by auto} \)
from \( L1 \) obtain \( x y \) where \( L3 : (x, y) \in r \) and \( L4 : u = \text{c-pair } x y \)
unfolding \( \text{ce-rel-to-set-def by auto} \)
from \( L4 \) have \( L5 : \text{c-fst } u = x \) by simp
from \( L4 \) have \( L6 : \text{c-snd } u = y \) by simp
from \( L5 L6 \) have \( z = (x, y) \) by simp
with \( L3 \) show \( z \in r \) by auto
qed

next

show \( r \subseteq \text{ce-set-to-rel (ce-rel-to-set } r) \)
proof fix \( z \) show \( z \in r \implies z \in \text{ce-set-to-rel (ce-rel-to-set } r) \)
proof
assume \( A : z \in r \)
def \( x\text{-def} : x \equiv \text{fst } z \)
def \( y\text{-def} : y \equiv \text{snd } z \)
from \( \text{x-def y-def have } L1 : z = (x, y) \) by simp
def \( u\text{-def} : u \equiv \text{c-pair } x y \)
from \( A \) \( u\text{-def have } L2 : u \in \text{ce-rel-to-set } r \) by (unfold \( \text{ce-rel-to-set-def, auto} \))
from \( L1 \) \( u\text{-def have } L3 : z = (\text{c-fst } u, \text{c-snd } u) \) by simp
from \( L2 L3 \) show \( z \in \text{ce-set-to-rel (ce-rel-to-set } r) \) by (unfold \( \text{ce-set-to-rel-def, auto} \))
qed
qed

lemma \( \text{ce-rel-lm-2 } [\text{simp}]: \text{ce-rel-to-set } (\text{ce-set-to-rel } A) = A \)
proof
show \( \text{ce-rel-to-set } (\text{ce-set-to-rel } A) \subseteq A \)
proof fix \( z \) show \( z \in \text{ce-rel-to-set } (\text{ce-set-to-rel } A) \implies z \in A \)
proof
assume \( A : z \in \text{ce-rel-to-set } (\text{ce-set-to-rel } A) \)
then obtain \( x y \) where \( L1 : z = \text{c-pair } x y \) and \( L2 : (x, y) \in \text{ce-set-to-rel } A \)
unfolding \( \text{ce-rel-to-set-def by auto} \)
from \( L2 \) obtain \( u \) where \( L3 : (x, y) = (\text{c-fst } u, \text{c-snd } u) \) and \( L4 : u \in A \)
unfolding \( \text{ce-set-to-rel-def by auto} \)
from \( L3 L1 \) have \( L5 : z = u \) by simp
with \( L4 \) show \( z \in A \) by auto
qed
qed

next

show \( A \subseteq \text{ce-rel-to-set } (\text{ce-set-to-rel } A) \)
proof fix \( z \) show \( z \in A \implies z \in \text{ce-rel-to-set } (\text{ce-set-to-rel } A) \)
proof
assume \( A : z \in A \)
then have \( L1 : (\text{c-fst } z, \text{c-snd } z) \in \text{ce-set-to-rel } A \) by (unfold \( \text{ce-set-to-rel-def,} \)
\(auto\)

\begin{verbatim}
def x-def: x \equiv c-fst z
def y-def: y \equiv c-snd z
from L1 x-def y-def have L2: (x,y) \in ce-set-to-rel A by simp
then have L3: c-pair x y \in ce-rel-to-set (ce-set-to-rel A) by (unfold ce-rel-to-set-def, auto)
with x-def y-def show z \in ce-rel-to-set (ce-set-to-rel A) by simp
qed
qed

lemma ce-rels-def1: ce-rels = \{ ce-set-to-rel A \mid A. A \in ce-sets \}
proof
show ce-rels \subseteq \{ ce-set-to-rel A \mid A. A \in ce-sets \}
proof fix r show r \in ce-rels \implies r \in \{ ce-set-to-rel A \mid A. A \in ce-sets \}
proof -
  assume A: r \in ce-rels
  then have L1: ce-rel-to-set r \in ce-sets by (unfold ce-rels-def, auto)
def A-def: A \equiv ce-rel-to-set r
from A-def L1 have L2: A \in ce-sets by auto
from A-def have L3: ce-set-to-rel A = r by simp
with L2 show r \in \{ ce-set-to-rel A \mid A. A \in ce-sets \} by auto
qed
qed
next
show \{ ce-set-to-rel A \mid A. A \in ce-sets \} \subseteq ce-rels
proof fix r show r \in \{ ce-set-to-rel A \mid A. A \in ce-sets \} \implies r \in ce-rels
proof -
  assume A: r \in \{ ce-set-to-rel A \mid A. A \in ce-sets \}
  then obtain A where L1: r = ce-set-to-rel A and L2: A \in ce-sets by auto
from L1 have ce-rel-to-set r = A by simp
with L2 show r \in ce-rels unfolding ce-rels-def by auto
qed
qed

lemma ce-rel-to-set-inj: inj ce-rel-to-set
proof (rule inj-on-inverseI)
  fix x assume A: (x::(nat\times nat) set) \in UNIV show ce-set-to-rel (ce-rel-to-set x) = x by (rule ce-rel-lm-1)
qed

lemma ce-rel-to-set-srj: surj ce-rel-to-set
proof (rule surjI [where \?f=ce-set-to-rel])
  fix x show ce-rel-to-set (ce-set-to-rel x) = x by (rule ce-rel-lm-2)
qed

lemma ce-rel-to-set-bij: bij ce-rel-to-set
proof (rule bijI)
\end{verbatim}
show inj ce-rel-to-set by (rule ce-rel-to-set-inj)
next
show surj ce-rel-to-set by (rule ce-rel-to-set-srj)
qed

lemma ce-set-to-rel-inj: inj ce-set-to-rel
proof (rule inj-on-inverseI)
  fix x assume A: (x::nat set) ∈ UNIV show ce-rel-to-set (ce-set-to-rel x) = x
  by (rule ce-rel-lm-2)
qed

lemma ce-set-to-rel-srj: surj ce-set-to-rel
proof (rule surjI [where ?f = ce-rel-to-set])
  fix x show ce-set-to-rel (ce-rel-to-set x) = x by (rule ce-rel-lm-1)
qed

lemma ce-set-to-rel-bij: bij ce-set-to-rel
proof (rule bijI)
  show inj ce-set-to-rel by (rule ce-set-to-rel-inj)
  next
  show surj ce-set-to-rel by (rule ce-set-to-rel-srj)
qed

lemma ce-rel-lm-3: A ∈ ce-sets ⇒ ce-set-to-rel A ∈ ce-rels
proof –
  assume A: A ∈ ce-sets
  from A ce-rels-def1 show ?thesis by auto
qed

lemma ce-rel-lm-4: ce-set-to-rel A ∈ ce-rels ⇒ A ∈ ce-sets
proof –
  assume A: ce-set-to-rel A ∈ ce-rels
  from A show ?thesis by (unfold ce-rels-def, auto)
qed

lemma ce-rel-lm-5: (A ∈ ce-sets) = (ce-set-to-rel A ∈ ce-rels)
proof
  assume A ∈ ce-sets then show ce-set-to-rel A ∈ ce-rels by (rule ce-rel-lm-3)
  next
  assume ce-set-to-rel A ∈ ce-rels then show A ∈ ce-sets by (rule ce-rel-lm-4)
qed

lemma ce-rel-lm-6: r ∈ ce-rels ⇒ ce-rel-to-set r ∈ ce-sets
proof –
  assume A: r ∈ ce-rels
  then show ?thesis by (unfold ce-rels-def, auto)
qed

lemma ce-rel-lm-7: ce-rel-to-set r ∈ ce-sets ⇒ r ∈ ce-rels
proof
assume \( r \in \text{ce-sets} \)
then show \(?\text{thesis}\) by (unfold \text{ce-rels-def}, auto)
qed

lemma \text{ce-rel-lm-8}: (r \in \text{ce-rels}) = (ce-rel-to-set r \in \text{ce-sets}) by (unfold \text{ce-rels-def},
auto)

lemma \text{ce-rel-lm-9}: (x,y) \in r \implies c\text{-pair x y} \in ce-rel-to-set r by (unfold \text{ce-rel-to-set-def},
auto)

lemma \text{ce-rel-lm-10}: x \in A \implies (c\text{-fst x}, c\text{-snd x}) \in \text{ce-set-to-rel A} by (unfold \text{ce-set-to-rel-def}, auto)

lemma \text{ce-rel-lm-11}: c\text{-pair x y} \in ce-rel-to-set r \implies (x,y) \in r by simp
qed

lemma \text{ce-rel-lm-12}: (c\text{-pair x y} \in ce-rel-to-set r) = (\text{(x,y)} \in r)
proof
assume c\text{-pair x y} \in ce-rel-to-set r then show (x, y) \in r by (rule ce-rel-lm-11)
next
assume (x, y) \in r then show c\text{-pair x y} \in ce-rel-to-set r by (rule ce-rel-lm-9)
qed

lemma \text{ce-rel-lm-13}: (x,y) \in ce-set-to-rel A \implies c\text{-pair x y} \in A
proof
assume (x,y) \in ce-set-to-rel A
then have c\text{-pair x y} \in ce-set-to-rel (ce-set-to-rel A) by (rule ce-rel-lm-9)
then show ?thesis by simp
qed

lemma \text{ce-rel-lm-14}: c\text{-pair x y} \in A \implies (x,y) \in ce-set-to-rel A
proof
assume c\text{-pair x y} \in A
then have c\text{-pair x y} \in ce-set-to-rel (ce-set-to-rel A) by simp
then show ?thesis by (rule ce-rel-lm-11)
qed

lemma \text{ce-rel-lm-15}: ((x,y) \in ce-set-to-rel A) = (c\text{-pair x y} \in A)
proof
assume (x, y) \in ce-set-to-rel A then show c\text{-pair x y} \in A by (rule ce-rel-lm-13)
next
assume c\text{-pair x y} \in A then show (x, y) \in ce-set-to-rel A by (rule ce-rel-lm-14)
qed
lemma ce-rel-lm-16: \( x \in \text{ce-rel-to-set } r \implies (\text{c-fst } x, \text{c-snd } x) \in r \)
proof 
  assume \( x \in \text{ce-rel-to-set } r \)
  then have \((\text{c-fst } x, \text{c-snd } x) \in \text{ce-set-to-rel } (\text{ce-rel-to-set } r)\) by (rule ce-rel-lm-10)
  then show ?thesis by simp
qed

lemma ce-rel-lm-17: \((\text{c-fst } x, \text{c-snd } x) \in \text{ce-set-to-rel } A \implies x \in A\)
proof 
  assume \((\text{c-fst } x, \text{c-snd } x) \in \text{ce-set-to-rel } A\)
  then have \(\text{c-pair } (\text{c-fst } x) (\text{c-snd } x) \in A\) by (rule ce-rel-lm-13)
  then show ?thesis by simp
qed

lemma ce-rel-lm-18: \(((\text{c-fst } x, \text{c-snd } x) \in \text{ce-set-to-rel } A) \iff x \in A\)
proof 
  assume \((\text{c-fst } x, \text{c-snd } x) \in \text{ce-set-to-rel } A\) then show \(x \in A\) by (rule ce-rel-lm-17)
  next 
  assume \(x \in A\) then show \((\text{c-fst } x, \text{c-snd } x) \in \text{ce-set-to-rel } A\) by (rule ce-rel-lm-10)
qed

lemma ce-rel-lm-19: \((\text{c-fst } x, \text{c-snd } x) \in r \implies x \in \text{ce-rel-to-set } r\)
proof 
  assume \((\text{c-fst } x, \text{c-snd } x) \in r\)
  then have \((\text{c-fst } x, \text{c-snd } x) \in \text{ce-set-to-rel } (\text{ce-rel-to-set } r)\) by simp
  then show ?thesis by (rule ce-rel-lm-17)
qed

lemma ce-rel-lm-20: \(((\text{c-fst } x, \text{c-snd } x) \in r) \iff x \in \text{ce-rel-to-set } r\)
proof 
  assume \((\text{c-fst } x, \text{c-snd } x) \in r\) then show \(x \in \text{ce-rel-to-set } r\) by (rule ce-rel-lm-19)
  next 
  assume \(x \in \text{ce-rel-to-set } r\) then show \((\text{c-fst } x, \text{c-snd } x) \in r\) by (rule ce-rel-lm-16)
qed

lemma ce-rel-lm-21: \(r \in \text{ce-rels } \implies \exists p \in \text{PrimRec}3. \forall x y. ((x,y) \in r) = (\exists u. p \ x \ y \ u = 0)\)
proof 
  assume \(r-ce\): \(r \in \text{ce-rels}\)
  def A-def: \(A \equiv \text{ce-rel-to-set } r\)
  from \(r-ce\) have A-cc: \(A \in \text{ce-sets}\) by (unfold A-def, rule ce-rel-lm-6)
  then have \(\exists p \in \text{PrimRec}2. A = \text{fn-to-set } p\) by (rule ce-set-lm-3)
  then obtain q where q-is-pr: \(q \in \text{PrimRec}2\) and A-def1: \(A = \text{fn-to-set } q\) ..
  from A-def1 have A-def2: \(A = \{ x, \exists y. q \ x \ y = 0\}\) by (unfold fn-to-set-def)
  def p-def: \(p \equiv \lambda x y u. q \ (\text{c-pair } x \ y) \ u\)
  from q-is-pr have p-is-pr: \(p \in \text{PrimRec}3\) unfolding p-def by prec
  have \(\bigwedge x y. ((x,y) \in r) = (\exists u. p \ x \ y \ u = 0)\)
  proof 
  fix x y show \((x,y) \in r) = (\exists u. p \ x \ y \ u = 0)\)
  qed
proof

assume A: (x, y) ∈ r

def z-def: z ≡ c-pair x y

with A-def A have z-in-A: z ∈ A by (unfold ce-rel-to-set-def, auto)

with A-def2 have z ∈ { x. ∃ y. q x y = 0 } by auto

then obtain u where q z u = 0 by (simp add: z-def p-def)

then show ∃ u. p x y u = 0 by auto

next

assume A: ∃ u. p x y u = 0

def z-def: z ≡ c-pair x y

from A obtain u where p x y u = 0 by auto

then have q-z: q z u = 0 by (simp add: z-def p-def)

with A-def2 have z-in-A: z ∈ A by auto

then have c-pair x y ∈ A by (unfold z-def)

then have c-pair x y ∈ ce-rel-to-set r by (unfold A-def)

then show (x, y) ∈ r by (rule ce-rel-lm-11)

qed

qed

with p-is-pr show ?thesis by auto

qed

lemma ce-rel-lm-22: r ∈ ce-rels ⟹ ∃ p ∈ PrimRec3. r = { (x, y). ∃ u. p x y u = 0 }

proof

assume r-ce: r ∈ ce-rels

then have ∃ p ∈ PrimRec3. ∀ x y. ((x, y) ∈ r) = (∃ u. p x y u = 0) by (rule ce-rel-lm-21)

then obtain p where p-is-pr: p ∈ PrimRec3 and L1: ∀ x y. ((x, y) ∈ r) = (∃ u. p x y u = 0) by auto

from p-is-pr L1 show ?thesis by blast

qed

lemma ce-rel-lm-23: [ p ∈ PrimRec3; ∀ x y. ((x, y) ∈ r) = (∃ u. p x y u = 0) ] ⟹ r ∈ ce-rels

proof

assume p-is-pr: p ∈ PrimRec3

assume A: ∀ x y. ((x, y) ∈ r) = (∃ u. p x y u = 0)

def q-def: q ≡ λ z u. p (c-fst z) (c-snd z) u

from p-is-pr have q-is-pr: q ∈ PrimRec2 unfolding q-def by prec

def A-def: A ≡ { x. ∃ y. q x y = 0 }

then have A-def1: A = fn-to-set q by (unfold fn-to-set-def, auto)

from q-is-pr A-def1 have A-ce: A ∈ ce-sets by (simp add: ce-set-lm-1)

have main: A = ce-rel-to-set r

proof

show A ⊆ ce-rel-to-set r

proof fix z assume z-in-A: z ∈ A show z ∈ ce-rel-to-set r

proof

def x-def: x ≡ c-fst z

qed
def y-def: y \equiv c-snd z

from z-in-A A-def obtain u where L2: q z u = 0 by auto
with x-def y-def q-def have L3: p x y u = 0 by simp
then have \exists u. p x y u = 0 by auto
with A have (x,y) \in r by auto
then have c-pair x y \in ce-rel-to-set r by (rule ce-rel-lm-9)
with x-def y-def show \?thesis by simp
qed
qed
next
show ce-rel-to-set r \subseteq A
proof fix z assume z-in-r: z \in ce-rel-to-set r show z \in A
proof –
  def x-def: x \equiv c-fst z
  def y-def: y \equiv c-snd z
  from z-in-r have (c-fst z, c-snd z) \in r by (rule ce-rel-lm-16)
  with x-def y-def have (x,y) \in r by simp
  with A obtain u where L1: p x y u = 0 by auto
  with x-def y-def q-def have q z u = 0 by simp
  with A-def show z \in A by auto
qed
qed
qed
with A-ce have ce-rel-to-set r \in ce-sets by auto
then show r \in ce-rels by (rule ce-rel-lm-7)
qed

lemma ce-rel-lm-24: \[ r \in ce-rels; s \in ce-rels \] \implies s O r \in ce-rels
proof –
  assume r-ce: r \in ce-rels
  assume s-ce: s \in ce-rels
  from r-ce have \exists p \in PrimRec3. \forall x y. ((x,y) \in r) = (\exists u. p x y u = 0) by (rule ce-rel-lm-21)
  then obtain p-r where p-r-is-pr: p-r \in PrimRec3 and R1: \forall x y. ((x,y) \in r) = (\exists u. p-r x y u = 0) by auto
  from s-ce have \exists p \in PrimRec3. \forall x y. ((x,y) \in s) = (\exists u. p x y u = 0) by auto
  then obtain p-s where p-s-is-pr: p-s \in PrimRec3 and S1: \forall x y. ((x,y) \in s) = (\exists u. p-s x y u = 0) by auto
  def p-def: p \equiv \lambda x z u. (p-s x (c-fst u) (c-fst (c-snd u))) + (p-r (c-fst u) z (c-snd (c-snd u)))
  from p-r-is-pr p-s-is-pr have p-is-pr: p \in PrimRec3 unfolding p-def by prec
  def sr-def: sr \equiv s O r
  have main: \forall x z. ((x,z) \in sr) = (\exists u. p x z u = 0)
proof (rule allI, rule allI) fix x z show ((x, z) \in sr) = (\exists u. p x z u = 0)
proof assume A: (x, z) \in sr show \exists u. p x z u = 0
proof –
  from A sr-def obtain y where L1: (x,y) \in s and L2: (y,z) \in r by auto
  from L1 S1 obtain u-s where L3: p-s x y u-s = 0 by auto

142
from L2 R1 obtain \( u \rightarrow r \) where \( \text{L4}: p \rightarrow r \) \( y z \) \( u \rightarrow r = 0 \) \text{by auto}\\
def \text{u-def}: u \equiv \text{c-pair } y \ (\text{c-pair } u-s \ u-r)\\
from L3 \text{L4} have \( p x z u = 0 \) \textbf{by} (\text{unfold } p\text{-def, unfold } u\text{-def, simp)}\\then show \ \text{?thesis} \ \textbf{by} \ \text{auto}\\qed\\nnext\\assume A: \exists u. \ p x z u = 0 \ \text{show} \ (x, z) \in sr\\proof –\\from A \text{ obtain } u \ \text{where} \ \text{L1}: \ p x z u = 0 \ \text{by auto}\\then have \ \text{L2}: (p-s x (c-fst u) (c-fst (c-snd u))) + (p-r (c-fst u) z (c-snd (c-snd u))) = 0 \ \text{by} (\text{unfold } p\text{-def)}\\from L2 \text{ have } \text{L3}: p-s x (c-fst u) (c-fst (c-snd u)) = 0 \ \text{by auto}\\from L2 \text{ have } \text{L4}: p-r (c-fst u) z (c-snd (c-snd u)) = 0 \ \text{by auto}\\from L3 S1 have \ \text{L5}: (x,(c-fst u)) \in s \ \text{by auto}\\from L4 R1 have \ \text{L6}: ((c-fst u),z) \in r \ \text{by auto}\\from L5 L6 have \ (x,z) \in s \ O \ r \ \text{by auto}\\with \sr\text{-def} \ \text{show} \ \text{?thesis} \ \text{by} \ \text{auto}\\qed\\qed\\qed\\from p\text{-is-pr} \ \text{main} \ \text{have} \ sr \in \text{ce-rels} \ \textbf{by} (\text{rule } ce\text{-rel-lm-23)}\\then show \ \text{?thesis} \ \text{by} (\text{unfold } sr\text{-def)}\\qed\\lemma \ \text{ce-rel-lm-25}: r \in \text{ce-rels} \ \implies \ r^{-1} \in \text{ce-rels}\\proof –\\\assume r-ce: r \in \text{ce-rels}\\have r^{-1} \ = \ \{(y,x). \ (x,y) \in r\} \ \textbf{by auto}\\then have \ \text{L1}: \forall x y. \ ((x,y) \in r) = (((y,x) \in r^{-1}) \ \textbf{by auto}\\from r-ce \ \text{have} \ \exists p \in \text{PrimRec3}. \ \forall x y. \ ((x,y) \in r) = (\exists u. \ p x y u = 0)\ \textbf{by (rule } ce\text{-rel-lm-21)}\\then obtain p \ \text{where} \ \text{p\text{-is-pr}: } p \in \text{PrimRec3} \ \text{and } R1: \forall x y. \ ((x,y) \in r) = (\exists u. \ p x y u = 0) \ \textbf{by auto}\\def \text{q-def}: q \equiv \lambda x \ y \ u. \ p \ x \ y \ u\\from p\text{-is-pr} \ \text{have } q\text{-is-pr}: q \in \text{PrimRec3 unfolding } q\text{-def } \\textbf{by prec}\\from L1 R1 have \ \text{L2}: \forall x y. \ ((x,y) \in r^{-1}) = (\exists u. \ p x y u = 0) \ \textbf{by auto}\\\with q\text{-def} \ \text{have} \ \text{L3}: \forall x y. \ ((x,y) \in r^{-1}) = (\exists u. \ q x y u = 0) \ \textbf{by auto}\\\with q\text{-is-pr} \ \text{show} \ \text{?thesis } \ \textbf{by (rule } ce\text{-rel-lm-23)}\\qed\\lemma \ \text{ce-rel-lm-26}: r \in \text{ce-rels} \ \implies \ \text{Domain } r \in \text{ce-sets}\\proof –\\\assume r-ce: r \in \text{ce-rels}\\have L1: \forall x. \ (x \in \text{Domain } r) = (\exists y. \ (x,y) \in r) \ \textbf{by auto}\\def A\text{-def}: A \equiv \text{ce-rel-to-set } r\\from r-ce \ \text{have} \ \text{ce-rel-to-set } r \ \in \text{ce-sets } \ \textbf{by (rule } ce\text{-rel-lm-6)}\\\then have A-ce: A \in \text{ce-sets } \ \textbf{by (unfold } A\text{-def)}\\\have \forall x y. \ ((x,y) \in r) = (c\text{-pair } x y \in \text{ce-rel-to-set } r) \ \textbf{by (simp add: } ce\text{-rel-lm-12)}\\\then have L2: \forall x y. \ ((x,y) \in r) = (c\text{-pair } x y \in A) \ \textbf{by (unfold } A\text{-def)}
from A-ce c-fst-is-pr have L3: \{ c-fst z \mid z. z \in A \} \in ce-sets by (rule ce-set-lm-7)

have L4: \forall x. (x \in \{ c-fst z \mid z. z \in A \}) = (\exists y. c-pair x y \in A)

proof
fix x show \( x \in \{ c-fst z \mid z. z \in A \} \) = (\exists y. c-pair x y \in A)
proof

assume A: \( x \in \{ c-fst z \mid z. z \in A \} \)
then obtain z where z-in-A: z \in A and x-z: x = c-fst z by auto
from x-z have z = c-pair x (c-snd z) by simp

then have c-pair x (c-snd z) \in A by auto
then show \( \exists y. c-pair x y \in A \) by auto

next

assume A: \( \exists y. c-pair x y \in A \)
then obtain y where y-1: c-pair x y \in A by auto
def z-def: \( z \equiv c-pair x y \)
from y-1 have z-in-A: z \in A by (unfold z-def)
from z-def have x-z: x = c-fst z by (unfold z-def, simp)
from z-in-A x-z show \( x \in \{ c-fst z \mid z. z \in A \} \) by auto

qed

qed

from L1 L2 have L5: \( \forall x. (x \in Domain r) = (\exists y. c-pair x y \in A) \) by auto
from L4 L5 have L6: \( \forall x. (x \in Domain r) = (x \in \{ c-fst z \mid z. z \in A \} \) by auto
then have Domain r = \{ c-fst z \mid z. z \in A \} by auto
with L3 show Domain r \in ce-sets by auto

qed

lemma ce-rel-lm-27: \( r \in ce-rels \implies Range r \in ce-sets \)

proof

assume r-ce: \( r \in ce-rels \)
then have r^-1 \in ce-rels by (rule ce-rel-lm-25)
then have Domain (r^-1) \in ce-sets by (rule ce-rel-lm-26)
then show \( \exists \text{thesis by (unfold Domain-converse [symmetric])} \)

qed

lemma ce-rel-lm-28: \( r \in ce-rels \implies Field r \in ce-sets \)

proof

assume r-ce: \( r \in ce-rels \)
from r-ce have L1: Domain r \in ce-sets by (rule ce-rel-lm-26)
from r-ce have L2: Range r \in ce-sets by (rule ce-rel-lm-27)
from L1 L2 have L3: Domain r \cup Range r \in ce-sets by (rule ce-union)
then show \( \exists \text{thesis by (unfold Field-def)} \)

qed

lemma ce-rel-lm-29: \( [ A \in ce-sets; B \in ce-sets ] \implies A \times B \in ce-rels \)

proof

assume A-ce: \( A \in ce-sets \)
assume B-ce: \( B \in ce-sets \)
def r-a-def: \r-a \equiv \{ (x,0::nat) \mid x. x \in A \}
def r-b-def: \r-b \equiv \{ ((0::nat),z) \mid z. z \in B \}

144
have \( L1: r-a O r-b = A \times B \) by (unfold \( r-a-def \), unfold \( r-b-def \), auto)

have \( r-a-ce: r-a \in \text{ce-rels} \)

proof –

have loc1: \( \text{ce-rel-to-set} \ r-a = \{ \text{c-pair} x 0 \mid x. x \in A \} \) by (unfold \( r-a-def \),
unfold \( \text{ce-rel-to-set-def} \), auto)

\[
\begin{align*}
def p-def: p &\equiv \lambda x. \text{c-pair} x 0 \\
have p-is-pr: p &\in \text{PrimRec1} \text{ unfolding} p-def \text{ by prec} \\
\text{from} A-ce \ p-is-pr \ have \( \{ \text{c-pair} x 0 \mid x. x \in A \} \in \text{ce-sets} \) by (simp add: p-def
ce-set-lm-7)
\end{align*}
\]

with loc1 have \( \text{ce-rel-to-set} \ r-a \in \text{ce-sets} \) by auto

then show \( \text{?thesis} \) by (rule ce-rel-lm-7)

qed

have \( r-b-ce: r-b \in \text{ce-rels} \)

proof –

have loc1: \( \text{ce-rel-to-set} \ r-b = \{ \text{c-pair} 0 z \mid z. z \in B \} \) by (unfold \( r-b-def \), unfold
\( \text{ce-rel-to-set-def} \), auto)

\[
\begin{align*}
def p-def: p &\equiv \lambda z. \text{c-pair} 0 z \\
have p-is-pr: p &\in \text{PrimRec1} \text{ unfolding} p-def \text{ by prec} \\
\text{from} B-ce \ p-is-pr \ have \( \{ \text{c-pair} 0 z \mid z. z \in B \} \in \text{ce-sets} \) by (simp add: p-def
ce-set-lm-7)
\end{align*}
\]

with loc1 have \( \text{ce-rel-to-set} \ r-b \in \text{ce-sets} \) by auto

then show \( \text{?thesis} \) by (rule ce-rel-lm-7)

qed

from \( r-b-ce \ r-a-ce \) have \( r-a O r-b \in \text{ce-rels} \) by (rule ce-rel-lm-24)

with \( L1 \) show \( \text{?thesis} \) by auto

qed

lemma ce-rel-lm-30: \( \emptyset \in \text{ce-rels} \)

proof –

have \( \text{ce-rel-to-set} \ \emptyset = \emptyset \) by (unfold \( \text{ce-rel-to-set-def} \), auto)

with \( \text{ce-empty} \) have \( \text{ce-rel-to-set} \ \emptyset \in \text{ce-sets} \) by auto

then show \( \text{?thesis} \) by (rule ce-rel-lm-7)

qed

lemma ce-rel-lm-31: \( \text{UNIV} \in \text{ce-rels} \)

proof –

from \( \text{ce-univ} \ \text{ce-univ} \) have \( \text{UNIV} \times \text{UNIV} \in \text{ce-rels} \) by (rule ce-rel-lm-29)

then show \( \text{?thesis} \) by auto

qed

lemma ce-rel-lm-32: \( \text{ce-rel-to-set} \ (r \cup s) = (\text{ce-rel-to-set} \ r) \cup (\text{ce-rel-to-set} \ s) \)

by (unfold \( \text{ce-rel-to-set-def} \), auto)

lemma ce-rel-lm-33: \( [ \ r \in \text{ce-rels}; s \in \text{ce-rels} ] \Rightarrow r \cup s \in \text{ce-rels} \)

proof –

assume \( r \in \text{ce-rels} \)

then have \( r-ce: \text{ce-rel-to-set} \ r \in \text{ce-sets} \) by (rule ce-rel-lm-6)

assume \( s \in \text{ce-rels} \)

then have \( s-ce: \text{ce-rel-to-set} \ s \in \text{ce-sets} \) by (rule ce-rel-lm-6)

145
have \( (r \cup s) = (\text{ce-rel-to-set } r) \cup (\text{ce-rel-to-set } s) \) by (unfold ce-rel-to-set-def, auto)

moreover from \( r \cup s \) have \( (\text{ce-rel-to-set } r) \cup (\text{ce-rel-to-set } s) \in \text{ce-sets} \) by (rule ce-union)

ultimately have \( (\text{ce-rel-to-set } (r \cup s)) \in \text{ce-sets} \) by auto then show \( \text{thesis} \) by (rule ce-rel-lm-7)

qed

lemma **ce-rel-lm-34**: \( \text{ce-rel-to-set } (r \cap s) = (\text{ce-rel-to-set } r) \cap (\text{ce-rel-to-set } s) \)

proof

show \( (\text{ce-rel-to-set } (r \cap s)) \subseteq (\text{ce-rel-to-set } r) \cap (\text{ce-rel-to-set } s) \) by (unfold ce-rel-to-set-def, auto)

next

show \( (\text{ce-rel-to-set } r) \cap (\text{ce-rel-to-set } s) \subseteq (\text{ce-rel-to-set } (r \cap s)) \)

proof

fix \( x \) assume \( A: x \in (\text{ce-rel-to-set } r) \cap (\text{ce-rel-to-set } s) \) by auto

from \( A \) have \( L1: x \in (\text{ce-rel-to-set } r) \) by auto

from \( A \) have \( L2: x \in (\text{ce-rel-to-set } s) \) by auto

from \( L1 \) obtain \( u v \) where \( L3: (u,v) \in r \) and \( L4: x = \text{c-pair } u v \) unfolding ce-rel-to-set-def by auto

from \( L2 \) obtain \( u1 v1 \) where \( L5: (u1,v1) \in s \) and \( L6: x = \text{c-pair } u1 v1 \) unfolding ce-rel-to-set-def by auto

from \( L4, L6 \) have \( L7: \text{c-pair } u1 v1 = \text{c-pair } u v \) by auto

ultimately have \( (u,v) = (u1,v1) \) by auto

with \( L3, L5 \) have \( (u,v) \in (r \cap s) \) by auto

with \( L4 \) show \( x \in (\text{ce-rel-to-set } (r \cap s)) \) by (unfold ce-rel-to-set-def, auto)

qed

qed

lemma **ce-rel-lm-35**: \( r \in \text{ce-rels} ; \ s \in \text{ce-rels} \) \( \Rightarrow r \cap s \in \text{ce-rels} \)

proof

assume \( r \in \text{ce-rels} \)

then have \( r \cup s \) have \( (\text{ce-rel-to-set } r) \cap (\text{ce-rel-to-set } s) \in \text{ce-sets} \) by (rule ce-rel-lm-6)

assume \( s \in \text{ce-rels} \)

then have \( s \cup r \) have \( (\text{ce-rel-to-set } s) \cap (\text{ce-rel-to-set } r) \in \text{ce-sets} \) by (rule ce-rel-lm-6)

have \( (\text{ce-rel-to-set } (r \cap s)) \in \text{ce-sets} \) by (rule ce-rel-lm-34)

moreover from \( r \cup s \) have \( (\text{ce-rel-to-set } r) \cap (\text{ce-rel-to-set } s) \in \text{ce-sets} \) by (rule ce-intersect)

ultimately have \( (\text{ce-rel-to-set } (r \cap s)) \in \text{ce-sets} \) by auto then show \( \text{thesis} \) by (rule ce-rel-lm-7)

qed

lemma **ce-rel-lm-36**: \( \text{ce-set-to-rel } (A \cup B) = (\text{ce-set-to-rel } A) \cup (\text{ce-set-to-rel } B) \)

by (unfold ce-set-to-rel-def, auto)

lemma **ce-rel-lm-37**: \( \text{ce-set-to-rel } (A \cap B) = (\text{ce-set-to-rel } A) \cap (\text{ce-set-to-rel } B) \)

proof

\( \text{def } f \text{-def: } f \equiv \lambda x. (c-fst x, c-snd x) \)
have \( f \)-inj: \( \text{inj} \ f \)

proof (unfold \( f \)-def, rule inj-on-inverseI [where \( g = \lambda (u,v). \ c\text{-pair \( u \) \( v \))}]

fix \( x \) assume \((x::\text{nat}) \in \text{UNIV} \) show \( \text{case-prod} \ c\text{-pair} \ (c\text{-fst} \( x \), c\text{-snd} \( x \)) = x \)
by simp

qed

from \( f \)-inj have \( f ' \ (A \cap B) = f ' A \cap f ' B \) by (rule image-Int)
then show \( \forall \text{thesis} \) by (unfold \( f \)-def, unfold ce-set-to-rel-def, auto)

qed

lemma ce-rel-lm-38: \( \forall r \in \text{ce-rels}; \ A \in \text{ce-sets} \) \( \Rightarrow \ r'^*A \in \text{ce-sets} \)

proof −
assume \( r\)-ce: \( r \in \text{ce-rels} \)
assume \( A\)-ce: \( A \in \text{ce-sets} \)
have \( L1: r'^*A = \text{Range} \ (r \cap A \times \text{UNIV}) \) by blast
have \( L2: \text{Range} \ (r \cap A \times \text{UNIV}) \in \text{ce-sets} \)

proof (rule ce-rel-lm-27)
show \( r \cap A \times \text{UNIV} \in \text{ce-rels} \)
proof (rule ce-rel-lm-35)
show \( r \in \text{ce-rels} \) by (rule \( r\)-ce)
next
show \( A \times \text{UNIV} \in \text{ce-rels} \)
proof (rule ce-rel-lm-29)
show \( A \in \text{ce-sets} \) by (rule \( A\)-ce)
next
show \( \text{UNIV} \in \text{ce-sets} \) by (rule ce-univ)
qed
qed

from \( L1 \) \( L2 \) show \( \forall \text{thesis} \) by auto

qed

7.6 Total computable functions

definition
\( \text{graph} :: (\text{nat} \Rightarrow \text{nat}) \Rightarrow (\text{nat} \times \text{nat}) \text{ set} \) where
\( \text{graph} = (\lambda f. \{ (x, f \ x) | x. \ x \in \text{UNIV} \}) \)

lemma graph-lm-1: \((x,y) \in \text{graph} \ f \) \( \Rightarrow \ y = f \ x \) by (unfold graph-def, auto)

lemma graph-lm-2: \( y = f \ x \) \( \Rightarrow \ (x,y) \in \text{graph} \ f \) by (unfold graph-def, auto)

lemma graph-lm-3: \((x,y) \in \text{graph} \ f \) = \( (y = f \ x) \) by (unfold graph-def, auto)

lemma graph-lm-4: \( \text{graph} \ (f \circ g) = (\text{graph} \ g) \circ (\text{graph} \ f) \) by (unfold graph-def, auto)

definition
\( \text{c-graph} :: (\text{nat} \Rightarrow \text{nat}) \Rightarrow \text{nat set} \) where
\( \text{c-graph} = (\lambda f. \{ \text{c-pair} \ (f \ x) | x. \ x \in \text{UNIV} \}) \)

147
lemma c-graph-lm-1: c-pair x y ∈ c-graph f ⇒ y = f x
proof −
  assume A: c-pair x y ∈ c-graph f
  have S1: c-graph f = {c-pair x (f x) | x. x ∈ UNIV} by (simp add: c-graph-def)
  from A S1 obtain z where S2: c-pair x y = c-pair z (f z) by auto
  then have x = z by (rule c-pair-inj1)
  moreover from S2 have y = f z by (rule c-pair-inj2)
  ultimately show ?thesis by auto
qed

lemma c-graph-lm-2: y = f x =⇒ c-pair x y ∈ c-graph f
by (unfold c-graph-def, auto)

lemma c-graph-lm-3: (c-pair x y ∈ c-graph f) = (y = f x)
proof
  assume c-pair x y ∈ c-graph f then show y = f x by (rule c-graph-lm-1)
next
  assume y = f x then show c-pair x y ∈ c-graph f by (rule c-graph-lm-2)
qed

lemma c-graph-lm-4: c-graph f = ce-rel-to-set (graph f)
by (unfold c-graph-def ce-rel-to-set-def graph-def, auto)

lemma c-graph-lm-5: graph f = ce-set-to-rel (c-graph f)
by (simp add: c-graph-lm-4)

definition total-recursive :: (nat ⇒ nat) ⇒ bool where
  total-recursive = (λ f. graph f ∈ ce-rels)

lemma total-recursive-def1: total-recursive = (λ f. c-graph f ∈ ce-sets)
proof (rule ext) fix f show total-recursive f = (c-graph f ∈ ce-sets)
proof
  assume A: total-recursive f
  then have graph f ∈ ce-rels by (unfold total-recursive-def)
  then have ce-rel-to-set (graph f) ∈ ce-sets by (rule ce-rel-lm-6)
  then show c-graph f ∈ ce-sets by (simp add: c-graph-lm-4)
next
  assume c-graph f ∈ ce-sets
  then have ce-rel-to-set (graph f) ∈ ce-sets by (simp add: c-graph-lm-4)
  then have graph f ∈ ce-rels by (rule ce-rel-lm-7)
  then show total-recursive f by (unfold total-recursive-def)
qed

theorem pr-is-total-rec: f ∈ PrimRec1 =⇒ total-recursive f
proof −
  assume A: f ∈ PrimRec1
  def p-def: p ≡ λ x. c-pair x (f x)
from A have p-is-pr: p ∈ PrimRec1 unfolding p-def by prec
let ?U = { p x | x. x ∈ UNIV }
from ce-univ p-is-pr have U-ce: ?U ∈ ce-sets by (rule ce-set-lm-7)
have U-1: ?U = { c-pair x (f x) | x. x ∈ UNIV } by (simp add: p-def)
with c-graph-def have c-graph-f-is-ce: c-graph f ∈ ce-sets by (unfold c-graph-def, auto)
  then show ?thesis by (unfold total-recursive-def1, auto)
qed

theorem comp-tot-rec: [ total-recursive f; total-recursive g ] ⇒ total-recursive (f o g)
proof −
  assume total-recursive f
  then have f-ce: graph f ∈ ce-rels by (unfold total-recursive-def)
  assume total-recursive g
  then have g-ce: graph g ∈ ce-rels by (unfold total-recursive-def)
  from f-ce g-ce have graph g O graph f ∈ ce-rels by (rule ce-rel-lm-24)
  then have graph (f o g) ∈ ce-rels by (simp add: graph-lm-4)
  then show ?thesis by (unfold total-recursive-def)
qed

lemma univ-for-pr-tot-rec-lm: c-graph univ-for-pr ∈ ce-sets
proof −
def A-def: A ≡ c-graph univ-for-pr
from A-def have S1: A = { c-pair x (univ-for-pr x) | x. x ∈ UNIV } by (simp add: c-graph-def)
  from S1 have S2: A = { z. ∃ x. z = c-pair x (univ-for-pr x) } by auto
  have S3: ∃ z. (3 x. (z = c-pair x (univ-for-pr x))) = (univ-for-pr (c-fst z) = c-snd z)
  proof −
    fix z show (3 x. (z = c-pair x (univ-for-pr x))) = (univ-for-pr (c-fst z) = c-snd z)...
    proof
      assume A: 3 x. z = c-pair x (univ-for-pr x)
      then obtain x where S3-1: z = c-pair x (univ-for-pr x) ..
      then show univ-for-pr (c-fst z) = c-snd z by simp
    next
      assume A: univ-for-pr (c-fst z) = c-snd z
      from A have z = c-pair (c-fst z) (univ-for-pr (c-fst z)) by simp
      thus ∃ x. z = c-pair x (univ-for-pr x) ..
    qed
qed
with S2 have S4: A = { z. univ-for-pr (c-fst z) = c-snd z } by auto
def p-def: p ≡ λ x y. if c-assoc-have-key (pr-gr y) (c-fst x) = 0 then
  (if c-assoc-value (pr-gr y) (c-fst x) = c-snd x then (0::nat) else
  1) else 1
  from c-assoc-have-key-is-pr c-assoc-value-is-pr pr-gr-is-pr have p-is-pr: p ∈ PrimRec2
unfolding $p$-def by prec

have $S5$: $\forall z. (\text{univ-for-pr} (c\text{-fst } z) = c\text{-snd } z) = (\exists y. p\ z\ y = 0)$

proof

fix $z$ show $(\text{univ-for-pr} (c\text{-fst } z) = c\text{-snd } z) = (\exists y. p\ z\ y = 0)$

proof

assume $A$: $\text{univ-for-pr} (c\text{-fst } z) = c\text{-snd } z$

let $?n = c\text{-fst } (c\text{-fst } z)$

let $?x = c\text{-snd } (c\text{-fst } z)$

let $?y = \text{loc-upb } ?n ?x$

have $S5\text{-}1$: $c\text{-assoc-have-key} (\text{pr-gr } ?y) (c\text{-pair } ?n ?x) = (\exists y. p\ z\ y = 0)$

proof (rule loc-upb-main)

have $S5\text{-}2$: $c\text{-assoc-value} (\text{pr-gr } ?y) (c\text{-pair } ?n ?x) = \text{univ-for-pr} (c\text{-pair } ?n ?x)$

proof (rule pr-gr-value)

from $S5\text{-}1$ have $S5\text{-}3$: $c\text{-assoc-have-key} (\text{pr-gr } y) (c\text{-fst } z) = 0$

proof (rule ccontr)

assume $A\text{-}1$: $c\text{-assoc-have-key} (\text{pr-gr } y) (c\text{-fst } z) \neq 0$

then have $p\ z\ y = 1$ by (simp add: $p$-def)

with $S5\text{-}1$ show False by auto

qed

from $S5\text{-}3$ $S5\text{-}2$ have $S5\text{-}4$: $c\text{-assoc-value} (\text{pr-gr } y) (c\text{-fst } z) = c\text{-snd } z$

proof (rule c-is-sub-fun-lm-1)

have $S5\text{-}5$: $c\text{-is-sub-fun} (\text{pr-gr } y) \text{univ-for-pr}$

proof (rule pr-gr-1)

have $S5\text{-}6$: $c\text{-assoc-value} (\text{pr-gr } y) (c\text{-fst } z) = \text{univ-for-pr} (c\text{-pair } ?n ?x)$

proof (rule ce-set-lm-1)

ultimately have $A \in \text{ce-sets}$ by auto

with $A\text{-def}$ show $\text{thesis}$ by auto

qed

theorem $\text{univ-for-pr-tot-rec}$: total-recursive $\text{univ-for-pr}$

proof

have $c\text{-graph} \text{univ-for-pr} \in \text{ce-sets}$ by (rule univ-for-pr-tot-rec-lm)
then show ?thesis by (unfold total-recursive-def1, auto)
qed

7.7 Computable sets, Post’s theorem

definition
computable :: nat set ⇒ bool where
computable = (λ A. A ∈ ce-sets ∧ −A ∈ ce-sets)

lemma computable-complement-1: computable A ⇒ computable (− A)
proof –
  assume computable A
then show ?thesis by (unfold computable-def, auto)
qed

lemma computable-complement-2: computable (− A) ⇒ computable A
proof –
  assume computable (− A)
then show ?thesis by (unfold computable-def, auto)
qed

lemma computable-complement-3: (computable A) = (computable (− A)) by (unfold computable-def, auto)

theorem comp-impl-tot-rec: computable A ⇒ total-recursive (chf A)
proof –
  assume A: computable A
from A have A1: A ∈ ce-sets by (unfold computable-def, simp)
from A have A2: −A ∈ ce-sets by (unfold computable-def, simp)
def p-def: p ≡ λ x. c-pair x 0
def q-def: q ≡ λ x. c-pair x 1
from p-def have p-is-pr: p ∈ PrimRec1 unfolding p-def by prec
from q-def have q-is-pr: q ∈ PrimRec1 unfolding q-def by prec
def U0-def: U0 ≡ { p x | x. x ∈ A}
def U1-def: U1 ≡ { q x | x. x ∈ − A}
from A1 p-is-pr have U0-ce: U0 ∈ ce-sets by(unfold U0-def, rule ce-set-lm-7)
from A2 q-is-pr have U1-ce: U1 ∈ ce-sets by(unfold U1-def, rule ce-set-lm-7)
def U-def: U ≡ U0 ∪ U1
from U0-ce U1-ce have U-ce: U ∈ ce-sets by (unfold U-def, rule ce-union)
def V-def: V ≡ c-graph (chf A)
  have V-1: V = { c-pair x (chf A x) | x. x ∈ UNIV} by (simp add: V-def c-graph-def)
    from U0-def p-def have U0-1: U0 = { c-pair x y | x y. x ∈ A ∧ y=0} by auto
    from U1-def q-def have U1-1: U1 = { c-pair x y | x y. x ̸∈ A ∧ y=1} by auto
    from U0-1 U1-1 U-def have U-1: U = { c-pair x y | x y. (x ∈ A ∧ y=0) ∨ (x ̸∈ A ∧ y=1)} by auto
    from V-1 have V-2: V = { c-pair x y | x y. y = chf A x} by auto
  have L1: A ∧ x y. ((x ∈ A ∧ y=0) ∨ (x ̸∈ A ∧ y=1)) = (y = chf A x)
proof – fix x y show ((x ∈ A ∧ y=0) ∨ (x ̸∈ A ∧ y=1)) = (y = chf A x)
by (unfold chf-def, auto) 
qed
from V-2 U-1 L1 have U=V by simp
with U-ce have V-ce: V ∈ ce-sets by auto
with V-def have c-graph (chf A) ∈ ce-sets by auto
then show ?thesis by (unfold total-recursive-def1)
qed

theorem tot-rec-impl-comp: total-recursive (chf A) ⟹ computable A
proof –
  assume A: total-recursive (chf A)
  then have A1: c-graph (chf A) ∈ ce-sets by (unfold total-recursive-def1)
  let ?U = c-graph (chf A)
  have L1: ?U = { c-pair x (chf A x) | x, x ∈ UNIV} by (simp add: c-graph-def)
  have L2: \( x y. ((x ∈ A ∧ y=0) ∨ (x ∉ A ∧ y=1)) = (y = chf A x) \) 
  proof – fix x y show \( ((x ∈ A ∧ y=0) ∨ (x ∉ A ∧ y=1)) = (y = chf A x) \) 
  by (unfold chf-def, auto)
  qed
  from L1 L2 have L3: ?U = { c-pair x y | x y. (x ∈ A ∧ y=0) ∨ (x ∉ A ∧ y=1) } by auto
  def p-def: p ≡ λ x. c-pair x 0
  def q-def: q ≡ λ x. c-pair x 1
  have p-is-pr: p ∈ PrimRec1 unfolding p-def by prec
  have q-is-pr: q ∈ PrimRec1 unfolding q-def by prec
  def V-def: V ≡ { c-pair x y | x y. (x ∈ A ∧ y=0) ∨ (x ∉ A ∧ y=1) }
  from V-def L3 A1 have V-ce: V ∈ ce-sets by auto
  from V-def have L4: ∀ z. (z ∈ V) = (∃ x y. z = c-pair x y ∧ ((x ∈ A ∧ y=0) ∨ (x ∉ A ∧ y=1))) by blast
  have L5: \( \bigwedge x. (p x ∈ V) = (x ∈ A) \)
  proof – fix x show (p x ∈ V) = (x ∈ A)
    proof
      assume A: p x ∈ V
      then have c-pair x 0 ∈ V by (unfold p-def)
      with V-def obtain x1 y1 where L5-2: c-pair x 0 = c-pair x1 y1
      and L5-3: ((x1 ∈ A ∧ y1=0) ∨ (x1 ∉ A ∧ y1=1)) by auto
      from L5-2 have X-eq-X1: x=x1 by (rule c-pair-inj1)
      from L5-2 have Y1-eq-0: 0=y1 by (rule c-pair-inj2)
      from L5-3 X-eq-X1 Y1-eq-0 show x ∈ A by auto
    next
      assume A: x ∈ A
      let ?z = c-pair x 0
      from A have L5-1: ∃ x1 y1. c-pair x 0 = c-pair x1 y1 ∧ ((x1 ∈ A ∧ y1=0) ∨ (x1 ∉ A ∧ y1=1)) by auto
      with V-def have c-pair x 0 ∈ V by auto
      with p-def show p x ∈ V by simp
      qed
    qed
  then have A-eq: A = { x. p x ∈ V } by auto
  from V-ce p-is-pr have { x. p x ∈ V } ∈ ce-sets by (rule ce-set-lm-5)
with A-eq have A-ce: A ∈ ce-sets by simp
have CA-eq: − A = \{ x. q x ∈ V \}
proof −
have \( \forall x. (q x ∈ V) = (x \notin A) \)
proof − fix x show (q x ∈ V) = (x \notin A)
proof
assume A: q x ∈ V
then have c-pair x 1 ∈ V by (unfold q-def)
with V-def obtain x1 y1 where L5-2: c-pair x 1 = c-pair x1 y1
and L5-1: ((x1 ∈ A ∧ y1=0) ∨ (x1 \notin A ∧ y1=1)) by auto
from L5-2 have X-eq-X1: x=x1 by (rule c-pair-inj1)
from L5-2 have Y1-eq-1: 1=y1 by (rule c-pair-inj2)
from L5-3 X-eq-X1 Y1-eq-1 show x \notin A by auto
next
assume A: x \notin A
from A have L5-1: \( \exists x1 y1. c-pair x 1 = c-pair x1 y1 \land ((x1 ∈ A ∧ y1=0) ∨ (x1 \notin A ∧ y1=1)) \) by auto
with V-def have c-pair x 1 ∈ V by auto
with q-def show q x ∈ V by simp
qed
qed
then show ?thesis by auto
qed
from V-ce q-is-pr have \{ x. q x ∈ V \} ∈ ce-sets by (rule ce-set-lm-5)
with CA-eq have CA-ce: − A ∈ ce-sets by simp
from A-ce CA-ce show ?thesis by (simp add: computable-def)
qed

theorem post-th-0: (computable A) = (total-recursive (chf A))
proof
assume computable A then show total-recursive (chf A) by (rule comp-impl-tot-rec)
next
assume total-recursive (chf A) then show computable A by (rule tot-rec-impl-comp)
qed

7.8 Universal computably enumerable set
definition
univ-ce :: nat set where
univ-ce = \{ c-pair n x | n x. x ∈ nat-to-ce-set n \}
lemma univ-for-pr-lm: univ-for-pr (c-pair n x) = (nat-to-pr n) x by (simp add: univ-for-pr-def pr-conv-2-to-1-def)

theorem univ-is-ce: univ-ce ∈ ce-sets
proof −
def A-def: A ≡ c-graph univ-for-pr
then have A ∈ ce-sets by (simp add: univ-for-pr-tot-rec-lm)
then have \( \exists pA ∈ PrimRec2. A = fn-to-set pA \) by (rule ce-set-lm-3)
then obtain \( pA \) where \( pA \text{-is-pr} : pA \in \text{PrimRec2} \) and \( S1 : A = \text{fn-to-set } pA \) by auto
from \( S1 \) have \( S2 : A = \{ x . \exists y. pA \, x \, y = 0 \} \) by (simp add: \text{fn-to-set-def})
def \( p\text{-def} : p \equiv \lambda z \, y. pA \, (\text{c-pair } (\text{c-fst } z) \, (\text{c-pair } (\text{c-snd } z) \, (\text{c-fst } y))) \, 0 \)
(c-snd y)
from \( pA \text{-is-pr} \) have \( p\text{-is-pr} : p \in \text{PrimRec2} \) unfolding \( p\text{-def} \) by prec
have \( \land z. (\exists n \, x. z = \text{c-pair } n \, x \land x \in \text{nat-to-ce-set } n) = (\text{c-snd } z \in \text{nat-to-ce-set } (\text{c-fst } z)) \)
proof –
  fix \( z \) show \((\exists n \, x. z = \text{c-pair } n \, x \land x \in \text{nat-to-ce-set } n) = (\text{c-snd } z \in \text{nat-to-ce-set } (\text{c-fst } z)) \)
proof
  assume \( A \): \( \exists n \, x. z = \text{c-pair } n \, x \land x \in \text{nat-to-ce-set } n \)
  then obtain \( n \, x \) where \( L1 : z = \text{c-pair } n \, x \land x \in \text{nat-to-ce-set } n \) by auto
  from \( L1 \) have \( L2 : z = \text{c-pair } n \, x \) by auto
  from \( L1 \) have \( L3 : x \in \text{nat-to-ce-set } n \) by auto
  from \( L1 \) have \( L4 : \text{c-fst } z = n \) by simp
  from \( L1 \) have \( L5 : \text{c-snd } z = x \) by simp
  from \( L3 \) \( \boldsymbol{L4} \) \( \boldsymbol{L5} \) show \( \text{c-snd } z \in \text{nat-to-ce-set } (\text{c-fst } z) \) by auto
next
  assume \( A \): \( \text{c-snd } z \in \text{nat-to-ce-set } (\text{c-fst } z) \)
  let \( ?n = \text{c-fst } z \)
  let \( ?x = \text{c-snd } z \)
  have \( L1 : z = \text{c-pair } ?n \, ?x \) by simp
  from \( L1 \) \( A \) have \( z = \text{c-pair } ?n \, ?x \land ?x \in \text{nat-to-ce-set } ?n \) by auto
  thus \( \exists n \, x. z = \text{c-pair } n \, x \land x \in \text{nat-to-ce-set } n \) by blast
qed
qed
then have \( \{ \text{c-pair } n \, x \mid n \, x \in \text{nat-to-ce-set } n \} = \{ z. \text{c-snd } z \in \text{nat-to-ce-set } (\text{c-fst } z) \} \) by auto
then have \( S3 : \text{univ-ce } = \{ z. \text{c-snd } z \in \text{nat-to-ce-set } (\text{c-fst } z) \} \) by (simp add: univ-ce-def)
have \( S4 : \land z. (\text{c-snd } z \in \text{nat-to-ce-set } (\text{c-fst } z)) = (\exists y. y \, p \, z \, y = 0) \)
proof –
  fix \( z \) show \((\text{c-snd } z \in \text{nat-to-ce-set } (\text{c-fst } z)) = (\exists y. y \, p \, z \, y = 0) \)
proof
  assume \( A \): \( \text{c-snd } z \in \text{nat-to-ce-set } (\text{c-fst } z) \)
  have \( \text{nat-to-ce-set } (\text{c-fst } z) = \{ x. \exists y. (\text{nat-to-pr } (\text{c-fst } z)) \, (\text{c-pair } x \, y) = 0 \} \) by (simp add: nat-to-ce-set-lm-1)
  with \( A \) obtain \( u \) where \( S4\text{-1} : (\text{nat-to-pr } (\text{c-fst } z)) \, (\text{c-pair } (\text{c-snd } z) \, u) = 0 \) by auto
  then have \( S4\text{-2} : \text{univ-for-pr } (\text{c-pair } (\text{c-fst } z) \, (\text{c-pair } (\text{c-snd } z) \, u)) = 0 \) by (simp add: univ-for-pr-lm)
  from \( A\text{-def} \) have \( S4\text{-3} : A = \{ \text{c-pair } x \mid \text{univ-for-pr } x \mid x. x \in \text{UNIV} \} \) by (simp add: c-graph-def)
  then have \( S4\text{-4} : \land x. \text{c-pair } x \, (\text{univ-for-pr } x) \in A \) by auto
  then have \( \text{c-pair } (\text{c-pair } (\text{c-fst } z) \, (\text{c-pair } (\text{c-snd } z) \, u)) \, (\text{univ-for-pr } (\text{c-pair } (\text{c-fst } z) \, (\text{c-pair } (\text{c-snd } z) \, u))) \in A \) by auto
  with \( S4\text{-2} \) have \( S4\text{-5} : \text{c-pair } (\text{c-pair } (\text{c-fst } z) \, (\text{c-pair } (\text{c-snd } z) \, u)) \) \( 0 \in A \) by auto

154
auto

with S2 obtain v where S4-6: pA (c-pair (c-fst z) (c-pair (c-snd z) u)) 0) v = 0
  by auto
def y-def: y ≡ c-pair u v
from y-def have S4-7: u = c-fst y by simp
from y-def have S4-8: v = c-snd y by simp
from S4-6 S4-7 S4-8 p-def have p z y = 0 by simp
thus ∃ y. p z y = 0 ..

next
  assume A: ∃ y. p z y = 0
  then obtain y where S4-1: p z y = 0 ..
  from S4-1 p-def have S4-2: pA (c-pair (c-fst z) (c-pair (c-snd z) (c-fst y))) 0) (c-snd y) = 0 by simp
  with S2 have S4-3: c-pair (c-pair (c-fst z) (c-pair (c-snd z) (c-fst y))) 0 ∈ A by auto
  with A-def have c-pair (c-pair (c-fst z) (c-pair (c-snd z) (c-fst y))) 0 ∈ c-graph uni-for-pr by simp
  then have S4-4: 0 = uni-for-pr (c-pair (c-fst z) (c-pair (c-snd z) (c-fst y))) by (rule c-graph-lm-1)
  then have S4-5: uni-for-pr (c-pair (c-fst z) (c-pair (c-snd z) (c-fst y))) = 0 by auto
  then have S4-6: (nat-to-pr (c-fst z)) (c-pair (c-snd z) (c-fst y)) = 0 by (simp add: uni-for-pr-lm)
  then have S4-7: ∃ y. (nat-to-pr (c-fst z)) (c-pair (c-snd z) y) = 0 ..
  have S4-8: nat-to-ce-set (c-fst z) = { x . ∃ y. (nat-to-pr (c-fst z)) (c-pair x y) = 0 } by (simp add: nat-to-ce-set-lm-1)
  from S4-7 have S4-9: c-snd z ∈ { x . ∃ y. (nat-to-pr (c-fst z)) (c-pair x y) = 0 } by auto
  with S4-8 show c-snd z ∈ nat-to-ce-set (c-fst z) by auto
dq

dq
with S2 have uniiv-ce = { z . ∃ y. p z y = 0 } by auto
then have uniiv-ce = fn-to-set p by (simp add: fn-to-set-def)
moreover from p-is-pr have fn-to-set p ∈ ce-sets by (rule ce-set-lm-1)
ultimately show uniiv-ce ∈ ce-sets by auto

dq

lemma uniiv-ce-lm-1: (c-pair n x ∈ uniiv-ce) = (x ∈ nat-to-ce-set n)
proof
  from uniiv-ce-def have S1: uniiv-ce = { z . ∃ n x. z = c-pair n x ∧ x ∈ nat-to-ce-set n} by auto
  have S2: (∃ n1 x1. c-pair n x = c-pair n1 x1 ∧ x1 ∈ nat-to-ce-set n1) = (x ∈ nat-to-ce-set n)
  proof
    assume ∃ n1 x1. c-pair n x = c-pair n1 x1 ∧ x1 ∈ nat-to-ce-set n1
    then obtain n1 x1 where L1: c-pair n x = c-pair n1 x1 and L2: x1 ∈ nat-to-ce-set n1 by auto
    from L1 have L3: n = n1 by (rule c-pair-inj1)
  qed

  dq

155
from L1 have L4: \( x = x_1 \) by (rule c-pair-inj2)
from L2 L3 L4 show \( x \in \text{nat-to-ce-set} \, n \) by auto

next
assume A: \( x \in \text{nat-to-ce-set} \, n \)
then have c-pair \( n \, x = \text{c-pair} \, n \, x \land x \in \text{nat-to-ce-set} \, n \) by auto
thus \( \exists \, n1 \, x1. \, \text{c-pair} \, n \, x = \text{c-pair} \, n1 \, x1 \land x1 \in \text{nat-to-ce-set} \, n1 \) by blast
qed

with S1 show ?thesis by auto
qed

theorem univ-ce-is-not-comp1: \( \neg \text{univ-ce} \notin \text{ce-sets} \)
proof (rule ccontr)
assume \( \neg \neg \, \text{univ-ce} \notin \text{ce-sets} \)
then have A: \( \neg \, \text{univ-ce} \in \text{ce-sets} \) by auto
def p-def: \( p \equiv \lambda \, x. \, \text{c-pair} \, x \, x \)
have p-is-pr: \( p \in \text{PrimRec1} \) unfolding p-def by prec
def A-def: \( A \equiv \{ \, x. \, p \, x \notin \text{univ-ce} \, \} \)
from A p-is-pr have \( \{ \, x. \, p \, x \notin \text{univ-ce} \, \} \in \text{ce-sets} \) by (rule ce-set-lm-5)
with A-def have S1: \( A \in \text{ce-sets} \) by auto
then have \( \exists \, n. \, A = \text{nat-to-ce-set} \, n \) by (rule nat-to-ce-set-srj)
then obtain n where S2: \( A = \text{nat-to-ce-set} \, n \) ..
from A-def have \( (n \in A) = (p \, n \notin \text{univ-ce}) \) by auto
with p-def have \( (n \in A) = (\text{c-pair} \, n \, n \notin \text{univ-ce}) \) by auto
with univ-ce-def univ-ce-lm-1 have \( (n \in A) = (n \notin \text{nat-to-ce-set} \, n) \) by auto
with S2 have \( (n \in A) = (n \notin A) \) by auto
thus False by auto
qed

theorem univ-ce-is-not-comp2: \( \neg \text{total-recursive} \, (\text{chf univ-ce}) \)
proof
assume total-recursive (chf univ-ce)
then have computable univ-ce by (rule tot-rec-impl-comp)
then have \( \neg \, \text{univ-ce} \in \text{ce-sets} \) by (unfold computable-def, auto)
with univ-ce-is-not-comp1 show False by auto
qed

theorem univ-ce-is-not-comp3: \( \neg \, \text{computable univ-ce} \)
proof (rule ccontr)
assume \( \neg \neg \, \text{computable univ-ce} \)
then have computable univ-ce by auto
then have \( \neg \text{total-recursive} \, (\text{chf univ-ce}) \) by (rule comp-impl-tot-rec)
with univ-ce-is-not-comp2 show False by auto
qed

7.9 s-1-1 theorem, one-one and many-one reducibilities

definition index-of-r-to-l :: nat where
index-of-r-to-l =
pair-by-index
(pair-by-index index-of-c-fst (comp-by-index index-of-c-fst index-of-c-snd))
(comp-by-index index-of-c-snd index-of-c-snd)

lemma index-of-r-to-l-lm: nat-to-pr index-of-r-to-l (c-pair x (c-pair y z)) = c-pair (c-pair x y) z
  apply (unfold index-of-r-to-l-def)
  apply (simp add: pair-by-index-main)
  apply (unfold c-f-pair-def)
  apply (simp add: index-of-c-fst-main)
  apply (simp add: comp-by-index-main)
  apply (simp add: index-of-c-fst-main)
  apply (simp add: index-of-c-snd-main)
done

definition s-ce :: nat ⇒ nat ⇒ nat where
s-ce == (λ e x. s1-1 (comp-by-index e index-of-r-to-l) x)

lemma s-ce-is-pr: s-ce ∈ PrimRec2
  unfolding s-ce-def using comp-by-index-is-pr s1-1-is-pr by prec

lemma s-ce-inj: s-ce e1 x1 = s-ce e2 x2 ⟹ e1 = e2 ∧ x1 = x2
proof
  let ?n1 = index-of-r-to-l
  assume s-ce e1 x1 = s-ce e2 x2
  then have s1-1 (comp-by-index e1 ?n1) x1 = s1-1 (comp-by-index e2 ?n1) x2
    by (unfold s-ce-def)
  then have L1: comp-by-index e1 ?n1 = comp-by-index e2 ?n1 ∧ x1 = x2
    by (rule s1-1-inj)
  from L1 have comp-by-index e1 ?n1 = comp-by-index e2 ?n1 ..
    then have e1 = e2 by (rule comp-by-index-inj1)
  moreover from L1 have x1 = x2 by auto
  ultimately show ?thesis by auto
qed

lemma s-ce-inj1: s-ce e1 x = s-ce e2 x ⟹ e1 = e2
proof
  assume s-ce e1 x = s-ce e2 x
  then have e1 = e2 ∧ x = x
    by (rule s-ce-inj)
  then show e1 = e2 by auto
qed

lemma s-ce-inj2: s-ce e x1 = s-ce e x2 ⟹ x1 = x2
proof
  assume s-ce e x1 = s-ce e x2
  then have e = e ∧ x1 = x2
    by (rule s-ce-inj)
  then show x1 = x2 by auto
qed
Theorem s1-1-th1: \( \forall n x y. \ (\text{nat-to-pr} \ n \ (\text{c-pair} \ x \ y)) = (\text{nat-to-pr} \ (s1-1 \ n \ x)) \ y \)

Proof (rule allI, rule allI, rule allI)

Fix \( n x y \) show \( \text{nat-to-pr} \ n \ (\text{c-pair} \ x \ y) = \text{nat-to-pr} \ (s1-1 \ n \ x) \ y \)

Proof

Have \( \lambda y. \ (\text{nat-to-pr} \ n \ (\text{c-pair} \ x \ y)) = \text{nat-to-pr} \ (s1-1 \ n \ x) \ y \) by (rule s1-1-th)

Then show \( \text{thesis} \) by (simp add: fun-eq-iff)

Qed

Qed

Lemma s-lm: \( (\text{nat-to-pr} \ (s-ce \ e \ x)) \ (\text{c-pair} \ y \ z) = (\text{nat-to-pr} \ e \ (\text{c-pair} \ x \ y) \ z) \)

Proof —

Let \( ?n1 = \text{index-of-r-to-l} \)

Have \( (\text{nat-to-pr} \ (s-ce \ e \ x)) \ (\text{c-pair} \ y \ z) = \text{nat-to-pr} \ (s1-1 \ (\text{comp-by-index} \ e \ ?n1)) \ x) \ (\text{c-pair} \ y \ z) \) by (unfold s-ce-def, simp)

Also have \( \ldots = (\text{nat-to-pr} \ (\text{comp-by-index} \ e \ ?n1)) \ (\text{c-pair} \ x \ (\text{c-pair} \ y \ z)) \) by (simp add: s1-1-th1)

Also have \( \ldots = (\text{nat-to-pr} \ e \ ((\text{nat-to-pr} \ ?n1) \ (\text{c-pair} \ x \ (\text{c-pair} \ y \ z)))) \) by (simp add: comp-by-index-main)

Finally show \( \text{thesis} \) by (simp add: index-of-r-to-l-lm)

Qed

Theorem s-ce-1-1-th: \( (\text{c-pair} \ x \ y \in \text{nat-to-ce-set} \ e) = (y \in \text{nat-to-ce-set} \ (s-ce \ e \ x)) \)

Proof

Assume \( A: \text{c-pair} \ x \ y \in \text{nat-to-ce-set} \ e \)

Then obtain \( z \) where \( L1: (\text{nat-to-pr} \ e) \ (\text{c-pair} \ (\text{c-pair} \ x \ y) \ z) = 0 \)

By (auto simp add: nat-to-ce-set-lm-1)

Have \( (\text{nat-to-pr} \ (s-ce \ e \ x)) \ (\text{c-pair} \ y \ z) = 0 \) by (simp add: s-lm L1)

With \( \text{nat-to-ce-set-lm-1} \) show \( y \in \text{nat-to-ce-set} \ (s-ce \ e \ x) \) by auto

Next

Assume \( A: y \in \text{nat-to-ce-set} \ (s-ce \ e \ x) \)

Then obtain \( z \) where \( L1: (\text{nat-to-pr} \ (s-ce \ e \ x)) \ (\text{c-pair} \ y \ z) = 0 \)

By (auto simp add: nat-to-ce-set-lm-1)

Then have \( (\text{nat-to-pr} \ e) \ (\text{c-pair} \ (\text{c-pair} \ x \ y) \ z) = 0 \) by (simp add: s-lm)

With \( \text{nat-to-ce-set-lm-1} \) show \( \text{c-pair} \ x \ y \in \text{nat-to-ce-set} \ e \) by auto

Qed

Definition one-reducible-to-via :: \( (nat \ set) \Rightarrow \ (nat \ set) \Rightarrow \ (nat \Rightarrow \ (nat \Rightarrow \ bool)) \) where

one-reducible-to-via = \( \lambda A B f. \ \text{total-recursive} \ f \land \ \text{inj} \ f \land (\forall x. \ (x \in A) = (f x \in B)) \)

Definition one-reducible-to :: \( (nat \ set) \Rightarrow \ (nat \ set) \Rightarrow bool \) where

one-reducible-to = \( \lambda A B. \ \exists f. \ \text{one-reducible-to-via} \ A \ B \ f \)
many-reducible-to-via :: (nat set) ⇒ (nat set) ⇒ (nat ⇒ nat) ⇒ bool where
many-reducible-to-via = (λ A B f. total-recursive f ∧ (∀ x. (x ∈ A) = (f x ∈ B)))

definition
many-reducible-to :: (nat set) ⇒ (nat set) ⇒ bool where
many-reducible-to = (λ A B. ∃ f. many-reducible-to-via A B f)

lemma one-reducible-to-via-trans: [ one-reducible-to-via A B f; one-reducible-to-via B C g ] ⇒ one-reducible-to-via A C (g o f)
proof –
  assume A1: one-reducible-to-via A B f
  assume A2: one-reducible-to-via B C g
  from A1 have f-tr: total-recursive f by (unfold one-reducible-to-via-def, auto)
  from A1 have f-inj: inj f by (unfold one-reducible-to-via-def, auto)
  from A1 have L1: ∀ x. (x ∈ A) = (f x ∈ B) by (unfold one-reducible-to-via-def, auto)
  from A2 have g-tr: total-recursive g by (unfold one-reducible-to-via-def, auto)
  from A2 have g-inj: inj g by (unfold one-reducible-to-via-def, auto)
  from A2 have L2: ∀ x. (x ∈ B) = (g x ∈ C) by (unfold one-reducible-to-via-def, auto)
  from g-tr f-tr have fg-tr: total-recursive (g o f) by (rule comp-tot-rec)
  from g-inj f-inj have fg-inj: inj (g o f) by (rule inj-comp)
  from L1 L2 have L3: (∀ x. (x ∈ A) = ((g o f) x ∈ C)) by auto
  with fg-tr fg-inj show ?thesis by (unfold one-reducible-to-via-def, auto)
qed

lemma one-reducible-to-trans: [ one-reducible-to A B; one-reducible-to B C ] ⇒ one-reducible-to A C
proof –
  assume one-reducible-to A B
  then obtain f where A1: one-reducible-to-via A B f unfolding one-reducible-to-def by auto
  assume one-reducible-to B C
  then obtain g where A2: one-reducible-to-via B C g unfolding one-reducible-to-def by auto
  from A1 A2 have one-reducible-to-via A C (g o f) by (rule one-reducible-to-via-trans)
  then show ?thesis unfolding one-reducible-to-def by auto
qed

lemma one-reducible-to-via-refl: one-reducible-to-via A A (λ x. x)
proof –
  have is-pr: (λ x. x) ∈ PrimRec1 by (rule pr-id1-1)
  then have is-tr: total-recursive (λ x. x) by (rule pr-is-total-rec)
  have is-inj: inj (λ x. x) by simp
  have L1: ∀ x. (x ∈ A) = (((λ x. x) x) ∈ A) by simp
  with is-tr is-inj show ?thesis by (unfold one-reducible-to-via-def, auto)
qed
lemma one-reducible-to-refl : one-reducible-to A A
proof –
  have one-reducible-to-via A A (λ x. x) by (rule one-reducible-to-via-refl)
  then show ?thesis by (unfold one-reducible-to-def, auto)
qed

lemma many-reducible-to-via-trans : [ [ many-reducible-to A B f ; many-reducible-to B C g ] ] ⇒ many-reducible-to A C (g o f)
proof –
  assume A1 : many-reducible-to-via A B f
  assume A2 : many-reducible-to-via B C g
  from A1 have f-tr : total-recursive f by (unfold many-reducible-to-via-def, auto)
  from A1 have L1 : ∀ x. (x ∈ A) = (f x ∈ B) by (unfold many-reducible-to-via-def, auto)
  from A2 have g-tr : total-recursive g by (unfold many-reducible-to-via-def, auto)
  from A2 have L2 : ∀ x. (x ∈ B) = (g x ∈ C) by (unfold many-reducible-to-via-def, auto)
  from g-tr f-tr have fg-tr : total-recursive (g o f) by (rule comp-tot-rec)
  from L1 L2 have L3 : ( ∀ x. (x ∈ A) = ((g o f) x ∈ C)) by auto
  with fg-tr show ?thesis by (unfold many-reducible-to-via-def, auto)
qed

lemma many-reducible-to-trans : [ [ many-reducible-to A B ; many-reducible-to B C ] ] ⇒ many-reducible-to A C
proof –
  assume many-reducible-to A B
  then obtain f where A1 : many-reducible-to-via A B f
    unfolding many-reducible-to-def by auto
  assume many-reducible-to B C
  then obtain g where A2 : many-reducible-to-via B C g
    unfolding many-reducible-to-def by auto
  from A1 A2 have many-reducible-to-via A C (g o f) by (rule many-reducible-to-via-trans)
  then show ?thesis unfolding many-reducible-to-def by auto
qed

lemma one-reducibility-via-is-many : one-reducible-to-via A B f ⇒ many-reducible-to-via A B f
proof –
  assume A : one-reducible-to-via A B f
  from A have f-tr : total-recursive f by (unfold one-reducible-to-via-def, auto)
  from A have ∀ x. (x ∈ A) = (f x ∈ B) by (unfold one-reducible-to-via-def, auto)
  with f-tr show ?thesis by (unfold many-reducible-to-via-def, auto)
qed

lemma one-reducibility-is-many : one-reducible-to A B ⇒ many-reducible-to A B
proof –
  assume one-reducible-to A B
  then obtain f where A : one-reducible-to-via A B f
unfolding one-reducible-to-def by auto
then have many-reducible-to-via A B f by (rule one-reducibility-via-many)
then show thesis unfolding many-reducible-to-def by auto
qed

lemma many-reducible-to-via-refl: many-reducible-to A A (λ x. x)
proof –
  have one-reducible-to-via A A (λ x. x) by (rule one-reducibility-via-refl)
  then show thesis by (rule one-reducibility-is-many)
qed

lemma many-reducible-to-refl: many-reducible-to A A
proof –
  have one-reducible-to A A by (rule one-reducible-to-refl)
  then show thesis by (rule one-reducibility-is-many)
qed

definition m-red-to-comp: [ many-reducible-to A B; computable B ] ⇒ computable A
proof –
  assume many-reducible-to A B
  then obtain f where A1: many-reducible-to-via A B f
  unfolding many-reducible-to-def by auto
  from A1 have f-tr: total-recursive f by (unfold many-reducible-to-def, auto)
  from A1 have L1: ∀ x. (x ∈ A) = (f x ∈ B) by (unfold many-reducible-to-def, auto)
  assume computable B
  then have L2: total-recursive (chf B) by (rule comp-impl-tot-rec)
  have L3: chf A = (chf B) o f
  proof fix x
    have chf A x = (chf B) (f x)
    proof cases
      assume A: x ∈ A
      then have L3-1: chf A x = 0 by (simp add: chf-lm-2)
      from A L1 have f x ∈ B by auto
      then have L3-2: (chf B) (f x) = 0 by (simp add: chf-lm-2)
      from L3-1 L3-2 show chf A x = (chf B) (f x) by auto
    next
      assume A: x /∈ A
      then have L3-1: chf A x = 1 by (simp add: chf-lm-3)
      from A L1 have f x /∈ B by auto
      then have L3-2: (chf B) (f x) = 1 by (simp add: chf-lm-3)
      from L3-1 L3-2 show chf A x = (chf B) (f x) by auto
    qed
  qed
  from L2 f-tr have total-recursive (chf B o f) by (rule comp-tot-rec)
  with L3 have total-recursive (chf A) by auto
  then show thesis by (rule tot-rec-impl-comp)
qed

lemma many-reducible-lm-1: many-reducible-to univ-ce A \implies \neg \text{computable } A

proof (rule ccontr)
  assume A1: many-reducible-to univ-ce A
  assume \neg \neg \text{computable } A
  then have A2: \text{computable } A \by \text{auto}
  from A1 A2 have \text{computable } univ-ce \by (rule m-red-to-comp)
  with univ-ce-is-not-comp3 show \text{False} \by \text{auto}
qed

lemma one-reducible-lm-1: one-reducible-to univ-ce A \implies \neg \text{computable } A

proof
  assume one-reducible-to univ-ce A
  then have many-reducible-to univ-ce A \by (rule one-reducibility-is-many)
  then show \text{thesis} \by (rule many-reducible-lm-1)
qed

lemma one-reducible-lm-2: one-reducible-to-via (nat-to-ce-set n) univ-ce (\lambda x. c-pair n x)

proof
  def f-def: f \equiv \lambda x. c-pair n x
  have f-is-pr: f \in \text{PrimRec1} \by \text{unfolding } f-def \by \text{prec}
  then have f-tr: \text{total-recursive } f \by (rule pr-is-total-rec)
  have f-inj: inj f
    proof (rule injI)
      fix x y assume A: f x = f y
      then have c-pair n x = c-pair n y \by (unfold f-def)
      then show x = y \by (rule c-pair-inj2)
    qed
  have \forall x. (x \in (nat-to-ce-set n)) = (f x \in \text{univ-ce})
  proof fix x show (x \in nat-to-ce-set n) = (f x \in \text{univ-ce}) \by (unfold f-def, simp add: univ-ce-lm-1)
    qed
  with f-tr f-inj show \text{thesis} \by (unfold f-def, unfold one-reducible-to-via-def, auto)
qed

lemma one-reducible-lm-3: one-reducible-to (nat-to-ce-set n) univ-ce

proof
  have one-reducible-to-via (nat-to-ce-set n) univ-ce (\lambda x. c-pair n x) \by (rule one-reducible-lm-2)
  then show \text{thesis} \by (unfold one-reducible-to-def, auto)
qed

lemma one-reducible-lm-4: A \in \text{ce-sets} \implies one-reducible-to A univ-ce

proof
  assume A \in \text{ce-sets}
  then have \exists n. A = nat-to-ce-set n \by (rule nat-to-ce-set-srj)

162
then obtain \( n \) where \( A = \text{nat-to-ce-set} \ n \) by auto with one-reducible-lm-3 show \( ?\text{thesis} \) by auto qed

7.10 One-complete sets

definition one-complete :: \( \text{nat set} \Rightarrow \text{bool} \) where
one-complete = (\( \lambda \ A. \ A \in \text{ce-sets} \land (\forall \ B. \ B \in \text{ce-sets} \longrightarrow \text{one-reducible-to} \ B \ A) \))

theorem univ-is-complete: one-complete univ-ce
proof (unfold one-complete-def)
show univ-ce \( \in \text{ce-sets} \land (\forall \ B. \ B \in \text{ce-sets} \longrightarrow \text{one-reducible-to} \ B \ \text{univ-ce}) \)
proof
show univ-ce \( \in \text{ce-sets} \) by (rule univ-is-ce)
next
show \( \forall \ B. \ B \in \text{ce-sets} \longrightarrow \text{one-reducible-to} \ B \ \text{univ-ce} \)
proof (rule allI, rule impI)
fix \( B \) assume \( B \in \text{ce-sets} \) then show \( \text{one-reducible-to} \ B \ \text{univ-ce} \) by (rule one-reducible-lm-4)
qed
qed

7.11 Index sets, Rice’s theorem

definition index-set :: \( \text{nat set} \Rightarrow \text{bool} \) where
index-set = (\( \lambda \ A. \ \forall \ n \ m. \ n \in A \land (\text{nat-to-ce-set} \ n = \text{nat-to-ce-set} \ m) \longrightarrow m \in A \))

lemma index-set-lm-1: \[ \text{index-set} \ A; \ n \in A; \ \text{nat-to-ce-set} \ n = \text{nat-to-ce-set} \ m \]
\( \Rightarrow m \in A \)
proof –
assume A1: index-set A
assume A2: \( n \in A \)
assume A3: \( \text{nat-to-ce-set} \ n = \text{nat-to-ce-set} \ m \)
from A2 A3 have L1: \( n \in A \land (\text{nat-to-ce-set} \ n = \text{nat-to-ce-set} \ m) \) by auto
from A1 have L2: \( \forall \ n. \ n \in A \land (\text{nat-to-ce-set} \ n = \text{nat-to-ce-set} \ m) \longrightarrow m \in A \)
by (unfold index-set-def)
from L1 L2 show \( ?\text{thesis} \) by auto
qed

lemma index-set-lm-2: index-set A \( \Rightarrow \text{index-set} \ (-A) \)
proof –
assume A: index-set A
show index-set (-A)
proof (unfold index-set-def)
show \( \forall \ n. \ n \in - A \land \text{nat-to-ce-set} \ n = \text{nat-to-ce-set} \ m \longrightarrow m \in - A \)
proof (rule allI, rule allI, rule impl)
  fix n m assume A1: n ∈ - A ∧ nat-to-ce-set n = nat-to-ce-set m
  from A1 have A2: n ∈ - A by auto
  from A1 have A3: nat-to-ce-set m = nat-to-ce-set n by auto
  show m ∈ - A
    proof
      assume m ∈ A
      from A this A3 have n ∈ A by (rule index-set-lm-1)
      with A2 show False by auto
    qed
  qed
qed

lemma Rice-lm-1: [\[ index-set A; A \neq \{\}; A \neq UNIV; \exists n \in A. nat-to-ce-set n = \{\} \] \implies one-reducible-to univ-ce (- A)
proof
  assume A1: index-set A
  assume A2: A \neq \{\}
  assume A3: A \neq UNIV
  assume \exists n \in A. nat-to-ce-set n = \{\}
  then obtain e-0 where e-0-in-A
    e-0-empty: nat-to-ce-set e-0 = \{\} by auto
  from e-0-in-A A3 obtain e-1 where e-1-not-in-A
    have nat-to-ce-set e-0 \neq nat-to-ce-set e-1
      proof
        assume nat-to-ce-set e-0 = nat-to-ce-set e-1
        with A1 e-0-in-A have e-1 \in A by (rule index-set-lm-1)
        with e-1-not-in-A show False by auto
      qed
    with e-0-empty have e1-not-empty: nat-to-ce-set e-0 = \{\}
      proof
        def we-1-def: we-1 \equiv nat-to-ce-set e-1
        from e1-not-empty have we-1-not-empty: we-1 \neq \{\} by (unfold we-1-def)
        def r-def: r \equiv univ-ce \times we-1
        have loc-lm-1: \forall x. x \in univ-ce \implies \forall y. (y \in we-1) = ((x,y) \in r) by (unfold r-def, auto)
        have loc-lm-2: \forall x. x \notin univ-ce \implies \forall y. (y \in \{\}) = ((x,y) \in r) by (unfold r-def, auto)
        have r-ce: r \in ce-rels
          proof (unfold r-def, rule ce-rel-lm-29)
            show univ-ce \in ce-sets by (rule univ-is-ce)
          next
            show we-1 \in ce-sets by (unfold we-1-def, rule nat-to-ce-set-into-ce)
          qed
        def we-n-def: we-n \equiv ce-rel-to-set r
        from r-ce have we-n-ce: we-n \in ce-sets by (unfold we-n-def, rule ce-rel-lm-6)
        then have \exists n. we-n = nat-to-ce-set n by (rule nat-to-ce-set-srj)
        then obtain n where we-n-defl: we-n = nat-to-ce-set n by auto
\[\text{def } f\text{-def}: f \equiv \lambda x. s-ce n x\]
\[\text{from } s-ce-is-pr \text{ have } f\text{-is-pr: } f \in \text{PrimRec1 unfolding } f\text{-def by } \text{prec}\]
\[\text{then have } f-tr: \text{ total-recursive } f \text{ by } (\text{rule pr-is-total-rec})\]
\[\text{have } f\text{-inj: } \text{inj } f\]
\[\text{proof (rule injI)}\]
\[\text{fix } x y\]
\[\text{assume } f x = f y\]
\[\text{then have } s-ce n x = s-ce n y \text{ by (unfold } f\text{-def)}\]
\[\text{then show } x = y \text{ by (rule } s-ce-inj2)\]
\[\text{qed}\]
\[\text{have } \text{loc-lm-3: } \forall x y. (c\text{-pair } x y \in \text{we-n}) = (y \in \text{nat-to-ce-set } f x)\]
\[\text{proof (rule allI, rule allI)}\]
\[\text{fix } x y \text{ show } (c\text{-pair } x y \in \text{we-n}) = (y \in \text{nat-to-ce-set } f x) \text{ by (unfold } f\text{-def, unfold we-n-def1, simp add: } s-ce-I-1-th)\]
\[\text{qed}\]
\[\text{from } A1 \text{ have } \text{loc-lm-4: } \text{index-set } (\neg A) \text{ by (rule index-set-lm-2)}\]
\[\text{have } \text{loc-lm-5: } \forall x. (x \in \text{univ-ce}) = (f x \in \neg A)\]
\[\text{proof fix } x \text{ show } (x \in \text{univ-ce}) = (f x \in \neg A)\]
\[\text{proof}\]
\[\text{assume } A: x \in \text{univ-ce}\]
\[\text{then have } S1: \forall y. (y \in \text{we-1}) = ((x, y) \in r) \text{ by (rule loc-lm-1)}\]
\[\text{from } \text{ce-rel-lm-12 have } \forall y. (c\text{-pair } x y \in \text{ce-rel-to-set } r) = ((x, y) \in r) \text{ by auto}\]
\[\text{then have } \forall y. ((x, y) \in r) = (c\text{-pair } x y \in \text{we-n}) \text{ by (unfold we-n-def, auto)}\]
\[\text{with } S1 \text{ have } \forall y. (y \in \text{we-1}) = (c\text{-pair } x y \in \text{we-n}) \text{ by auto}\]
\[\text{with } \text{loc-lm-3 have } \forall y. (y \in \text{we-1}) = (y \in \text{nat-to-ce-set } f x) \text{ by auto}\]
\[\text{then have } S2: \text{we-1} = \text{nat-to-ce-set } f x \text{ by auto}\]
\[\text{then have } \text{nat-to-ce-set e-1} = \text{nat-to-ce-set } f x \text{ by (unfold we-1-def)}\]
\[\text{with } \text{loc-lm-4 e-1-nat-in-A show } f x \in \neg A \text{ by (rule index-set-lm-1)}\]
\[\text{next}\]
\[\text{show } f x \in \neg A \implies x \in \text{univ-ce}\]
\[\text{proof (rule ccontr)}\]
\[\text{assume } \text{fx-in-A: } f x \in \neg A\]
\[\text{assume } x\text{-not-in-univ: } x \notin \text{univ-ce}\]
\[\text{then have } S1: \forall y. (y \in \{\}) = ((x, y) \in r) \text{ by (rule loc-lm-2)}\]
\[\text{from } \text{ce-rel-lm-12 have } \forall y. (c\text{-pair } x y \in \text{ce-rel-to-set } r) = ((x, y) \in r) \text{ by auto}\]
\[\text{then have } \forall y. ((x, y) \in r) = (c\text{-pair } x y \in \text{we-n}) \text{ by (unfold } \text{we-n-def, auto)}\]
\[\text{auto}\]
\[\text{then have } S1 \text{ have } \forall y. (y \in \{\}) = (c\text{-pair } x y \in \text{we-n}) \text{ by auto}\]
\[\text{with } \text{loc-lm-3 have } \forall y. (y \in \{\}) = (y \in \text{nat-to-ce-set } f x) \text{ by auto}\]
\[\text{then have } S2: \{\} = \text{nat-to-ce-set } f x \text{ by auto}\]
\[\text{then have } \text{nat-to-ce-set e-0} = \text{nat-to-ce-set } f x \text{ by (unfold e-0-empty)}\]
\[\text{with } A1 \text{ e-0-in-A have } f x \in A \text{ by (rule index-set-lm-1)}\]
\[\text{with } \text{fx-in-A show False by auto}\]
\[\text{qed}\]
\[\text{qed}\]
\[\text{with } f-tr f\text{-inj have one-reducible-to-via univ-ce } (\neg A) f \text{ by (unfold one-reducible-to-via-def),}\]
then show \( ? \thesis \) by \((\text{unfold one-reducible-to-def, auto})\)

**lemma Rice-lm-2:** \([\text{index-set } A; A \neq \{\}; A \neq \text{UNIV}; n \in A; \text{nat-to-ce-set } n = \{\}] \implies \text{one-reducible-to univ-ce } (-A)\)

**proof** –
- assume \( A1: \text{index-set } A \)
- assume \( A2: A \neq \{\} \)
- assume \( A3: A \neq \text{UNIV} \)
- assume \( A4: n \in A \)
- assume \( A5: \text{nat-to-ce-set } n = \{\} \)
- from \( A4 \ A5 \) have \( S1: \exists n \in A. \text{nat-to-ce-set } n = \{\} \) by auto
- from \( A1 \ A2 \ A3 \ S1 \) show \( ? \thesis \) by \((\text{rule Rice-lm-1})\)

**qed**

**theorem Rice-1:** \([\text{index-set } A; A \neq \{\}; A \neq \text{UNIV}] \implies \text{one-reducible-to univ-ce } A \lor \text{one-reducible-to univ-ce } (-A)\)

**proof** –
- assume \( A1: \text{index-set } A \)
- assume \( A2: A \neq \{\} \)
- assume \( A3: A \neq \text{UNIV} \)
- from \( \text{ce-empty} \) have \( \exists n. \{\} = \text{nat-to-ce-set } n \) by \((\text{rule nat-to-ce-set-srj})\)
- then obtain \( n \) where \( n\text{-empty: nat-to-ce-set } n = \{\} \) by auto
- show \( ? \thesis \)
  **proof** cases
  - assume \( n \notin A \) then have \( A: n \in -A \) by auto
  - from \( A1 \ A2 \ A3 \ A \ n\text{-empty} \) have \( \text{one-reducible-to univ-ce } (-A) \) by \((\text{rule Rice-lm-2})\)
  - then show \( ? \thesis \) by auto
  - next
  - assume \( n \notin A \) then have \( A: n \in -A \) by auto
  - from \( A1 \) have \( S1: \text{index-set } (-A) \) by \((\text{rule index-set-lm-2})\)
  - from \( A3 \) have \( S2: -A \neq \{\} \) by auto
  - from \( A2 \) have \( S3: -A \neq \text{UNIV} \) by auto
  - from \( S1 \ S2 \ S3 \ A \ n\text{-empty} \) have \( \text{one-reducible-to univ-ce } (-(-A)) \) by \((\text{rule Rice-lm-2})\)
  - then have \( \text{one-reducible-to univ-ce } A \) by simp
  - then show \( ? \thesis \) by auto

**qed**

**theorem Rice-2:** \([\text{index-set } A; A \neq \{\}; A \neq \text{UNIV}] \implies \neg \text{computable } A\)

**proof** –
- assume \( A1: \text{index-set } A \)
- assume \( A2: A \neq \{\} \)
- assume \( A3: A \neq \text{UNIV} \)
- from \( A1 \ A2 \ A3 \) have \( \text{one-reducible-to univ-ce } A \lor \text{one-reducible-to univ-ce } (- A) \) by \((\text{rule Rice-1})\)

166
then have \( S1: \neg \text{one-reducible-to univ-ce} A \rightarrow \text{one-reducible-to univ-ce} (-A) \)
by auto

show \(?thesis

proof cases
  assume \text{one-reducible-to univ-ce} A
  then show \( \neg \text{computable} A \) by (rule one-reducible-lm-1)
next
  assume \( \neg \text{one-reducible-to univ-ce} A \)
  with \( S1 \) have \( \text{one-reducible-to univ-ce} (-A) \) by auto
  then have \( \neg \text{computable} (-A) \) by (rule one-reducible-lm-1)
  with computable-complement-3 show \( \neg \text{computable} A \) by auto
qed

qed

theorem Rice-3: \( \llbracket C \subseteq \text{ce-sets}; \text{computable} \{n. \text{nat-to-ce-set} n \in C\} \rrbracket = \llbracket C = \{} \lor C = \text{ce-sets} \rrbracket \)
proof (rule ccontr)
  assume \( A1: C \subseteq \text{ce-sets} \)
  assume \( A2: \text{computable} \{n. \text{nat-to-ce-set} n \in C\} \)
  assume \( A3: \neg (C = \{} \lor C = \text{ce-sets} \)
  from \( A3 \) have \( A4: C \neq \{} \) by auto
  from \( A3 \) have \( A5: C \neq \text{ce-sets} \) by auto
  def \( A\text{-def}: A \equiv \{n. \text{nat-to-ce-set} n \in C\} \)
  have \( S1: \text{index-set} A \)
proof (unfold index-set-def)
  show \( \forall n m. n \in A \land \text{nat-to-ce-set} n = \text{nat-to-ce-set} m \rightarrow m \in A \)
proof (rule allI, rule allI, rule impI)
  fix \( n m \) assume \( A1-1: n \in A \land \text{nat-to-ce-set} n = \text{nat-to-ce-set} m \)
  from \( A1-1 \) have \( n \in A \) by auto
  then have \( S1-1: \text{nat-to-ce-set} n \in C \) by (rule nat-to-ce-set-srj)
  then obtain \( n \) where \( \text{nat-to-ce-set} n = \text{nat-to-ce-set} m \) ...
  with \( S1-1 \) have \( \text{nat-to-ce-set} n \in C \) by auto
  then show \( m \in A \) by (unfold \( A\text{-def} \), auto)
qed

qed

have \( S2: A \neq \{} \)
proof
  from \( A4 \) obtain \( B \) where \( S2-1: B \in C \) by auto
  with \( A1 \) have \( B \in \text{ce-sets} \) by auto
  then have \( \exists n. B = \text{nat-to-ce-set} n \) by (rule nat-to-ce-set-srj)
  then obtain \( n \) where \( B = \text{nat-to-ce-set} n \) ..
  with \( S2-1 \) have \( \text{nat-to-ce-set} n \in C \) by auto
  then show \(?thesis \) by (unfold \( A\text{-def} \), auto)
qed

have \( S3: A \neq \text{UNIV} \)
proof
  from \( A1 A5 \) obtain \( B \) where \( S2-1: B \notin C \) and \( S2-2: B \in \text{ce-sets} \) by auto
  from \( S2-2 \) have \( \exists n. B = \text{nat-to-ce-set} n \) by (rule nat-to-ce-set-srj)
  then obtain \( n \) where \( B = \text{nat-to-ce-set} n \) ..
with S2-1 have nat-to-ce-set n \notin C by auto
then show \( ? \)thesis by (unfold A-def, auto)
qed
from S1 S2 S3 have \( \neg \)computable A by (rule Rice-2)
with A2 show False unfolding A-def by auto
qed
end

References
