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

\section{Pairing function}

definition
\texttt{sf :: nat \Rightarrow nat where}
\texttt{sf-def: sf x = x \ast (x+1) div 2}

definition
\texttt{c-pair :: nat \Rightarrow nat \Rightarrow nat where}
\texttt{c-pair x y = sf (x+y) + x}

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

lemma \texttt{sf-at-1: sf 1 = 1 by (simp add: sf-def)}

lemma \texttt{sf-at-Suc: sf (x+1) = sf x + x + 1}

proof –
\begin{itemize}
\item have \texttt{S1: sf(x+1) = ((x+1)*x+2) div 2 by (simp add: sf-def)}
\item have \texttt{S2: (x+1)*(x+2) = x*(x+1) + 2*(x+1) by (auto)}
\item have \texttt{S2-1: \forall x y. x=y \Longrightarrow x div 2 = y div 2 by auto}
\item from \texttt{S2} have \texttt{S3: (x+1)*(x+2) div 2 = (x*(x+1) + 2*(x+1)) div 2 by (rule S2-1)}
\item have \texttt{S4: (0::nat) < 2 by (auto)}
\item from \texttt{S4} have \texttt{S5: (x*(x+1) + 2*(x+1)) div 2 = (x+1) + x*(x+1) div 2 by simp}
\end{itemize}
from S1 S3 S5 show ?thesis by (simp add: sf-def)
qed

lemma arg-le-sf: \( x \leq sf\ x \)
proof –
  have \( x + x \leq x\cdot(x + 1) \) by simp
  hence \( x + x \) \# \text{div} 2 \leq x\cdot(x + 1) \# \text{div} 2 \) by (rule \#\, \text{div-le-mono})
  hence \( x \leq x\cdot(x + 1) \# \text{div} 2 \) by simp
  thus ?thesis by (simp add: sf-def)
qed

lemma sf-mono: \( x \leq y \Rightarrow sf\ x \leq sf\ y \)
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 \( x\cdot(x + 1) \# \text{div} 2 \leq y\cdot(y + 1) \# \text{div} 2 \) by (rule \#\#\, \text{div-le-mono})
  thus ?thesis by (simp add: sf-def)
qed

lemma sf-strict-mono: \( x < y \Rightarrow sf\ x < sf\ y \)
proof –
  assume A1: \( x < y \)
  from A1 have S1: \( x = ?y + 1 \) by simp
  from A1 have \( ?y > 0 \) by simp
  then have \( sf\ (\#\, y + 1) > 0 \) by (rule \#\, \text{sf-posI})
  have \( sf\ (\#\, y + 1) = sf\ (\#\, y) + \#\, y + 1 \) by (rule \#\, \text{sf-at-Suc})
  with S1 have \( sf\ (\#\, x) = sf\ (\#\, y) + x \) by simp
  with S2 show ?thesis by simp
qed

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

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

lemma sf-eq-arg: \( sf\ x = x \Rightarrow x \leq 1 \)
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 \(?thesis\) 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 \(?thesis\) by (auto)
qed

lemma sf-less-sfD: \( sf\ x < sf\ y \implies x < y \)
proof
  assume \( A1: sf\ x < sf\ y \)
  have \( S1: x \leq sf\ x \implies sf\ x \leq sf\ y \) by (rule sf-mono)
  have \( S2: x \leq y \lor y < x \) by (auto)
  from \( A1\ S1\ S2 \) show \(?thesis\) by (auto)
qed

lemma sf-inj: \( sf\ x = sf\ y \implies x = y \)
proof
  assume \( A1: sf\ x = sf\ y \)
  have \( S1: x \leq y \implies x \leq y \) by (rule sf-le-sfD)
  have \( S2: y \leq x \implies y \leq x \) by (rule sf-le-sfD)
  from \( A1\ S1\ S2 \) have \( S3: x \leq y \land y \leq x \) by (auto)
  from \( S3 S1 S2 \) have \( S4: x \leq y \land y \leq x \) by (auto)
  from \( S4 \) show \(?thesis\) 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 sf-at-Suc)
  from \( S2 S3 \) have \( S4: sf(x+y) + (x+y) + 1 \leq sf(z) \) by (auto)
  from \( S4 \) show \(?thesis\) 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 \( S1: x+y < sf(z) \) by (auto)
  from \( S1 \) sf-aux1 have \( S2: x+y < x \) by (auto)

from $S_2$ show $\text{thesis}$ by (auto)
qed

lemma $\text{sf-aux3}$: $sf(z) + m < sf(z+1)$ $\implies m \leq z$
proof
assume $A1$: $sf(z) + m < sf(z+1)$
have $S1$: $sf(z) + z + 1$ 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 $\text{thesis}$ by (auto)
qed

lemma $\text{sf-aux4}$: $(s::nat) < t$ $\implies$ $(sf s) + s < sf t$
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 $\text{thesis}$ 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 $\text{thesis}$ by (simp add: $sf$-$def$)
qed

Basic properties of $c$-$pair$ function

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

lemma $\text{arg1-le-c-pair}$: $x \leq c$-$pair$ $x$ $y$
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 $\text{thesis}$ by (simp)
qed

lemma $\text{arg2-le-c-pair}$: $y \leq c$-$pair$ $x$ $y$
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 $\text{thesis}$ by (simp)
lemma c-pair-sum-mono: \( (x1::nat) + y1 < x2 + y2 \implies c\text{-}pair\ x1\ y1 < c\text{-}pair\ x2\ y2 \)
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: \( c\text{-}pair\ x1\ y1 = c\text{-}pair\ x2\ y2 = \implies x1 = x2 + y2 \)
proof
  assume \( A1: c\text{-}pair\ x1\ y1 = c\text{-}pair\ x2\ y2 \)
  have \( S1: (x1::nat) + y1 < x2 + y2 \implies c\text{-}pair\ x1\ y1 \neq c\text{-}pair\ x2\ y2 \) by (rule less-not-refl3, rule c-pair-sum-mono, auto)
  have \( S2: (x2::nat) + y2 < x1 + y1 \implies c\text{-}pair\ x1\ y1 \neq c\text{-}pair\ x2\ y2 \) by (rule less-not-refl2, rule c-pair-sum-mono, auto)
  from \( S1\ S2 \) have \( (x1::nat) + y1 \neq x2 + y2 \implies c\text{-}pair\ x1\ y1 \neq c\text{-}pair\ x2\ y2 \) by (arith)
  with \( A1 \) show \(?thesis\) by (auto)
qed

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

lemma c-pair-mono1: \( x1\leq x2 \implies c\text{-}pair\ x1\ y < c\text{-}pair\ x2\ y \)
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: \( x1 < x2 \implies c\text{-}pair\ x1\ y < c\text{-}pair\ x2\ y \)
proof
  assume \( A1: x1 \leq x2 \)

show \( ?\text{thesis} \)
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 \( ?\text{thesis} \) by simp
next
assume \( \neg x_1 < x_2 \)
with \( A_1 \) have \( x_1 = x_2 \) by simp
then show \( ?\text{thesis} \) by simp
qed
qed

lemma \( \text{c-pair-strict-mono2} \): \( y_1 < y_2 \Longrightarrow \text{c-pair} \ x \ y_1 < \text{c-pair} \ x \ y_2 \)
proof –
assume \( A_1 \): \( y_1 < y_2 \)
from \( A_1 \) have \( S_1 \): \( x + y_1 < x + y_2 \) by simp
then show \( ?\text{thesis} \) by (rule \( \text{c-pair-sum-mono} \))
qed

lemma \( \text{c-pair-mono2} \): \( y_1 \leq y_2 \Longrightarrow \text{c-pair} \ x \ y_1 \leq \text{c-pair} \ x \ y_2 \)
proof –
assume \( A_1 \): \( y_1 \leq y_2 \)
show \( ?\text{thesis} \)
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 \( ?\text{thesis} \) by simp
next
assume \( \neg y_1 < y_2 \)
with \( A_1 \) have \( y_1 = y_2 \) by simp
then show \( ?\text{thesis} \) by simp
qed
qed

1.2 Inverse mapping

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

definition
\( \text{c-sum} :: \text{nat} \Rightarrow \text{nat} \ where \)
\( \text{c-sum} \ u = (\text{LEAST} \ z. \ u < \text{sf} \ (z+1)) \)
definition
\( \text{c-fst} :: \text{nat} \Rightarrow \text{nat} \ where \)
\( \text{c-fst} \ u = u - \text{sf} \ (\text{c-sum} \ u) \)
definition
\( \text{c-snd} :: \text{nat} \Rightarrow \text{nat} \ where \)
\( \text{c-snd} \ u = \text{c-sum} \ u - \text{c-fst} \ u \)
lemma arg-less-sf-at-Suc-of-c-sum: \( u < sf((c\text{-}sum\ u) + 1) \)
proof –
  have \( u+1 \leq sf(u+1) \) \( \text{by (rule arg-le-sf)} \)
  hence \( u < sf(u+1) \) \( \text{by simp} \)
  thus \( ?\text{thesis} \) \( \text{by (unfold c\text{-}sum-def, rule LeastI)} \)
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 \( ?\text{thesis} \) proof (cases \( x \))
    assume \( x=0 \)
    with \( A1 \) show \( ?\text{thesis} \) by (simp add: sf-def)
  next
    fix \( y \)
    assume \( A2: x = Suc\ y \)
    show \( ?\text{thesis} \) proof –
      from \( A1 \ A2 \) have \( u < sf(y+1) \) \( \text{by simp} \)
      hence \( (Least(\%z. u < sf(z+1))) \leq y \) \( \text{by (rule Least-le)} \)
      hence \( c\text{-}sum\ u \leq y \) \( \text{by (fold c\text{-}sum-def)} \)
      with \( A2 \) show \( ?\text{thesis} \) \( \text{by simp} \)
  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 \( S1: u < sf(?z+1) \) \( \text{by (auto)} \)
  have \( S2: \neg c\text{-}sum\ u < c\text{-}sum\ u \) \( \text{by (auto)} \)
  from arg-less-sf-imp-c-sum-less-arg \( S2 \) have \( S3: \neg u < sf(c\text{-}sum\ u) \) \( \text{by (auto)} \)
  from \( S3 \) show \( ?\text{thesis} \) \( \text{by (auto)} \)
qed

lemma c-sum-le-arg: \( c\text{-}sum\ u \leq u \)
proof –
  have \( c\text{-}sum\ u \leq sf(c\text{-}sum\ u) \) \( \text{by (rule arg-le-sf)} \)
  moreover have \( sf(c\text{-}sum\ u) \leq u \) \( \text{by (rule sf-c-sum-le-arg)} \)
  ultimately show \( ?\text{thesis} \) \( \text{by simp} \)
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 \( S1: ?u < sf(?z+1) \) \( \text{by (rule arg-less-sf-at-Suc-of-c-sum)} \)
  have \( S2: sf(?z) \leq ?u \) \( \text{by (rule sf-c-sum-le-arg)} \)
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 $S4$ have $S8$: $?z \leq x+y$ by (rule sf-aux2)
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

9
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 –
  have 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 –
  have 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: \( c\text{-sum} \ x = x \)
have S1: \( sf(c\text{-sum} \ x) + c\text{-fst} \ x = x \) by (rule sf-c-sum-plus-c-fst)
from A1 S1 have S2: \( sf \ x + c\text{-fst} \ x = x \) by simp
have S3: \( x \leq sf \ x \) by (rule arg-le-sf)
from S2 S3 have \( sf(x) = x \) by simp
thus \( \text{thesis} \) by (rule sf-eq-arg)
qed

lemma c-sum-eq-arg-2: \( c\text{-sum} \ x = x \Rightarrow c\text{-fst} \ x = 0 \)
proof –
  assume A1: \( c\text{-sum} \ x = x \)
  have S1: \( sf(c\text{-sum} \ x) + c\text{-fst} \ x = x \) by (rule sf-c-sum-plus-c-fst)
  from A1 S1 have S2: \( sf \ x + c\text{-fst} \ x = x \) by simp
  have S3: \( x \leq sf \ x \) by (rule arg-le-sf)
  from S2 S3 show \( \text{thesis} \) by simp
qed

lemma c-fst-eq-arg: \( c\text{-fst} \ x = x \Rightarrow x = 0 \)
proof –
  assume A1: \( c\text{-fst} \ x = x \)
  have S1: \( c\text{-fst} \ x \leq c\text{-sum} \ x \) by (rule c-fst-le-c-sum)
  have S2: \( c\text{-sum} \ x \leq x \) by (rule c-sum-le-arg)
  from A1 S1 S2 have \( c\text{-sum} \ x = x \) by simp
  then have \( c\text{-fst} \ x = 0 \) by (rule c-sum-eq-arg-2)
  with A1 show \( \text{thesis} \) by simp
qed

lemma c-fst-less-arg: \( x > 0 \Rightarrow c\text{-fst} \ x < x \)
proof –
  assume A1: \( x > 0 \)
  show \( \text{thesis} \)
  proof cases
    assume c-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-fst x \( \leq x \) by (rule c-fst-le-arg)
    with S1 have \( c\text{-fst} \ x = x \) by simp
    then have x = 0 by (rule c-fst-eq-arg)
    with A1 show \( \text{thesis} \) by simp
  qed
qed

lemma c-snd-eq-arg: \( c\text{-snd} \ x = x \Rightarrow x \leq 1 \)
proof –
  assume A1: \( c\text{-snd} \ x = x \)
  have S1: \( c\text{-snd} \ x \leq c\text{-sum} \ x \) by (rule c-snd-le-c-sum)
  have S2: \( c\text{-sum} \ x \leq x \) by (rule c-sum-le-arg)
from A1 S1 S2 have c-sum \( x = x \) by simp then show \( \text{thesis} \) 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 \( \text{thesis} \)
  proof cases
    assume c-snd x < x
    then show \( \text{thesis} \).
  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 \( \text{thesis} \) 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
| 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 0 x = g x
| PrimRecOp-last g h x (Suc y)= h x (PrimRecOp-last g h 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. f (g x y z (h 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)) ∈ 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 \( \text{pr-rec} = \text{PrimRec1}-\text{PrimRec2}-\text{PrimRec3} \)

ML-file \texttt{Utils.ML}

named-theorems \( \text{prec} \)

method-setup \( \text{prec0} = \langle \langle \\text{Attrib.thms} >> (\text{fn ths} => \text{fn ctxt} => \text{Method.METHOD} (\text{fn facts} => \text{HEADGOAL (prec0-tac ctxt (facts @ \text{Named-Theorems.get ctxt @\{\text{named-theorems prec\}})}\})) \rangle \rangle \)

apply primitive recursive functions

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

lemma \( \text{pr-swap} \): \footnotesize \( f \in \text{PrimRec2} \implies (\lambda x y. f y x) \in \text{PrimRec2} \) by \text{prec0}

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

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

proof (rule \( \text{pr-swap} \))
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 (rule \( \text{pr-rec} \))
    show \( (\lambda x. x) \in \text{PrimRec1} \) by (rule \( \text{pr-id1-1} \))
  next
    show \( (\lambda x y z. \text{Suc} \ y) \in \text{PrimRec3} \) by \text{prec0}
  qed
  have \( (\lambda x y. y+x) = \text{PrimRecOp} \ (\lambda x. x) \ (\lambda x y z. \text{Suc} \ y) \) (\text{is - = ?f})
  proof −
    have \( \lambda x y . \ (\?f \ y x = y + x) \) by (\text{induct-tac y, auto})
  thus \( \?\text{thesis} \) by \( \text{simp add: ext} \)
  qed
  with \( S1 \) show \( \?\text{thesis} \) by \text{simp}
  qed
  qed

qed
lemma \textit{op-mult-is-pr} \textit{[prec]}: \((\lambda x \ y \ x+y) \in \text{PrimRec2}\)

proof (rule pr-swap)
show \((\lambda x \ y \ y+x) \in \text{PrimRec2}\)
proof
have \(S1: \text{PrimRecOp} (\lambda x \ \theta) (\lambda x \ y \ z \ y+z) \in \text{PrimRec2}\)
proof (rule pr-rec)
  show \((\lambda x \ \theta) \in \text{PrimRec1}\) by (rule pr-zero)
next
  show \((\lambda x \ y \ z \ y+z) \in \text{PrimRec3}\) by prec0
qed
have \((\lambda x \ y \ y*x) = \text{PrimRecOp} (\lambda x \ \theta) (\lambda x \ y \ z \ y+z) \ (\text{is - = ?f})\)
proof −
  have \(\forall x y. \ (\text{iff} y \ x = y \ast x) \) by (induct-tac y, auto)
  thus \(?\text{thesis by simp add: ext}\)
qed
with \(S1\) show ?thesis by simp
qed

lemma \textit{const-is-pr}: \((\lambda x \ (n::\text{nat})) \in \text{PrimRec1}\)

proof (induct n)
show \((\lambda x \ \theta) \in \text{PrimRec1}\) by (rule pr-zero)
next
fix n assume \((\lambda x \ n) \in \text{PrimRec1}\)
then show \((\lambda x \ \text{Suc} \ n) \in \text{PrimRec1}\) by prec0
qed

lemma \textit{const-is-pr-2}: \((\lambda x \ y \ (n::\text{nat})) \in \text{PrimRec2}\)

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

lemma \textit{const-is-pr-3}: \((\lambda x \ y \ z \ (n::\text{nat})) \in \text{PrimRec3}\)

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

theorem \textit{pr-rec-last}: \([g \in \text{PrimRec1}; \ h \in \text{PrimRec3}] \implies \text{PrimRecOp-last} \ g \ h \in \text{PrimRec2}\)

proof −
  assume \(A1: g \in \text{PrimRec1}\)
  assume \(A2: h \in \text{PrimRec3}\)
  let \(\text{h1} = \lambda x \ y \ z. \ h \ z \ y \ x\)
  from \(A2\ \text{pr-id3-3} \ \text{pr-id3-2} \ \text{pr-id3-1}\) have \(\text{h1-is-pr}: \ \text{h1} \in \text{PrimRec3}\) by (rule \textit{pr-comp3-3})
let \( f_1 = \text{PrimRecOp} \ g \ h_1 \)

from \( A1 \): \( h_1 \text{-is-pr} \) have \( f_1 \text{-is-pr} \): \( f_1 \in \text{PrimRec2} \) by (rule pr-rec)

let \( f = \lambda x \ y.\ \ ?f_1\ y\ x \)

from \( f_1 \text{-is-pr} \) have \( f \text{-is-pr} \): \( f \in \text{PrimRec2} \) by (rule pr-swap)

have \( \forall x \ y.\ \ ?f\ x\ y = \text{PrimRecOp}\_\text{last} \ g\ h\ x\ y \) by (induct-tac y, simp-all)

then have \( ?f = \text{PrimRecOp}\_\text{last}\ g\ h \) by (simp add: ext)

with \( f \text{-is-pr} \) show \( ?\text{thesis} \) by simp

qed

**Theorem 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 \( g \text{-is-pr} \): \( g \in \text{PrimRec1} \) by (rule const-is-pr)
- let \( h_1 = (\lambda x\ y\ z.\ h\ x\ y) \)
- from \( A1 \) have \( h_1 \text{-is-pr} \): \( h_1 \in \text{PrimRec3} \) by pre0
- let \( f_1 = \text{PrimRecOp} \ g \ h_1 \)
- from \( g \text{-is-pr} \ h_1 \text{-is-pr} \) have \( f_1 \text{-is-pr} \): \( f_1 \in \text{PrimRec2} \) by (rule pr-rec)
- let \( f = (\lambda x.\ ?f_1\ x\ 0) \)
- from \( f_1 \text{-is-pr} \) pr-id1-1 pr-zero have \( f \text{-is-pr} \): \( f \in \text{PrimRec1} \) by (rule pr-comp2-1)
- have \( \forall y.\ \ ?f\ y = \text{PrimRecOp1}\ a\ h\ y \) by (induct-tac y, auto)
- then have \( f = \text{PrimRecOp1}\ a\ h \) by (simp add: ext)
- with \( f \text{-is-pr} \) show \( ?\text{thesis} \) by (auto)

qed

**Theorem 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 \( h \text{-is-pr} \): \( h \in \text{PrimRec2} \)
- assume \( f\text{-at-0} : f\ 0 = a \)
- assume \( f\text{-at-Suc} : \forall y.\ f\ (\text{Suc}\ y) = h\ y\ (f\ y) \)
- from \( f\text{-at-0} \) 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 \( h \text{-is-pr} \) show \( ?\text{thesis} \) by (simp add: pr-rec1)

qed

**Lemma pred-is-pr:** \( (\lambda x.\ x - (1::\text{nat} )) \in \text{PrimRec1} \)

**Proof:**

- have \( S1 : \text{PrimRecOp1}\ \emptyset \ (\lambda x\ y.\ x ) \in \text{PrimRec1} \) by (rule pr-rec1)
- show \( (\lambda x\ y.\ x ) \in \text{PrimRec2} \) by (rule pr-id2-1)

qed

have \( (\lambda x.\ x - (1::\text{nat} )) = \text{PrimRecOp1}\ \emptyset\ (\lambda x\ y.\ x ) \) (is = ?f)

**Proof:**

- have \( \forall x.\ ( ?f\ x = x - (1::\text{nat} )) \) by (induct-tac x, auto)
- thus \( ?\text{thesis} \) by (simp add: ext)

qed

with \( S1 \) show \( ?\text{thesis} \) by simp
qed

lemma op-sub-is-pr [prec]: \((\lambda x y. x-y) \in \text{PrimRec2}\)
proof (rule pr-swap)
  show \((\lambda x y. y - x) \in \text{PrimRec2}\)
  proof
    have S1: \text{PrimRecOp} (\lambda x. x) (\lambda x y z. y-(1::nat)) \in \text{PrimRec2}
    proof (rule pr-rec)
    show (\lambda x y z. y - x) \in \text{PrimRec2}
    proof
      have \((\lambda x y. y - x) = \text{PrimRecOp} (\lambda x. x) (\lambda x y z. y-(1::nat)) \text{ is - ?f})
      proof
        have \(\lambda x y. (\text{if } y = x - y) \text{ by (induct-tac } y, \text{ auto)}
        thus ?thesis by (simp add: ext)
      qed
      with S1 show ?thesis by simp
    qed
  qed
  from pred-is-pr pr-id3-2 show \((\lambda x y z. y - (1::nat)) \in \text{PrimRec3 by (rule pr-comp1-3)}\)
  qed
  have \((\lambda x y. x - y) \in \text{PrimRecOp} (\lambda x. x) (\lambda x y z. y-(1::nat)) \text{ is - ?f})
  proof
    from S1 show ?thesis by simp
  qed

definition sgn1 :: nat \Rightarrow nat where
sgn1 x = (case x of 0 \Rightarrow 0 | Suc y \Rightarrow 1)

definition sgn2 :: nat \Rightarrow nat where
sgn2 x \equiv (case x of 0 \Rightarrow 1 | Suc y \Rightarrow 0)

definition abs-of-diff :: nat \Rightarrow nat \Rightarrow nat where
abs-of-diff = \((\lambda x y. (x - y) + (y - x))\)

lemma [simp]: sgn1 0 = 0 by (simp add: sgn1-def)
lemma [simp]: sgn1 (Suc y) = 1 by (simp add: sgn1-def)
lemma [simp]: sgn2 0 = 1 by (simp add: sgn2-def)
lemma [simp]: sgn2 (Suc y) = 0 by (simp add: sgn2-def)
lemma [simp]: \(x \neq 0 \Rightarrow sgn1 x = 1\) by (simp add: sgn1-def, cases x, auto)
lemma [simp]: \(x \neq 0 \Rightarrow sgn2 x = 0\) by (simp add: sgn2-def, cases x, auto)

lemma sgn1-nz-impl-arg-pos: sgn1 x \neq 0 \Rightarrow x > 0 by (cases x) auto
lemma sgn1-zero-impl-arg-zero: sgn1 x = 0 \Rightarrow x = 0 by (cases x) auto
lemma sgn2-nz-impl-arg-zero: sgn2 x \neq 0 \Rightarrow x = 0 by (cases x) auto
lemma sgn2-zero-impl-arg-pos: sgn2 x = 0 \implies x > 0 by (cases x) auto

lemma sgn1-nz-eq-arg-pos: (sgn1 x \neq 0) = (x > 0) by (cases x) auto
lemma sgn1-zero-eq-arg-zero: (sgn1 x = 0) = (x = 0) by (cases x) auto
lemma sgn2-nz-eq-arg-pos: (sgn2 x \neq 0) = (x = 0) by (cases x) auto
lemma sgn2-zero-eq-arg-zero: (sgn2 x = 0) = (x > 0) by (cases x) auto

lemma sgn1-pos-eq-one: sgn1 x > 0 = \Rightarrow sgn1 x = 1 by (cases x) auto
lemma sgn2-pos-eq-one: sgn2 x > 0 = \Rightarrow sgn2 x = 1 by (cases x) auto

lemma sgn2-eq-1-sub-arg:
\begin{verbatim}
proof (rule ext)
  fix x show sgn2 x = 1 - x by (cases x) auto
qed
\end{verbatim}

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

lemma sgn2-is-pr [prec]: sgn2 \in PrimRec1
\begin{verbatim}
proof -
  have (\lambda x. 1 - x) \in PrimRec1 by prec0
  thus ?thesis by (simp add: sgn2-eq-1-sub-arg)
qed
\end{verbatim}

lemma sgn1-is-pr [prec]: sgn1 \in PrimRec1
\begin{verbatim}
proof -
  from sgn2-is-pr have (\lambda x. 1 - (sgn2 x)) \in PrimRec1 by prec0
  thus ?thesis by (simp add: sgn1-eq-1-sub-sgn2)
qed
\end{verbatim}

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

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

lemma sf-is-pr [prec]: sf \in PrimRec1
\begin{verbatim}
proof -
  have S1: PrimRecOp1 0 (\lambda x y. y + x + 1) \in PrimRec1
    proof (rule pr-rec1)
      show (\lambda x y. y + x + 1) \in PrimRec2 by prec0
    qed
  have (\lambda x. sf x) = PrimRecOp1 0 (\lambda x y. y + x + 1) (is - = ?f)
\end{verbatim}
proof
have \( \forall x. \ (\exists f \sf x) \)
proof (induct-tac x)
  show \( \forall f = 0 \ sf 0 \) by (simp add: sf-at-0)
next
  fix x assume \( \exists f \sf x \)
  with sf-at-Suc show \( \exists f \sf x \) by auto
qed
thus \(?thesis \) by (simp add: ext)
qed

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

lemma if-is-pr\[p1 \in PrimRec1; q1 \in PrimRec1; q2 \in PrimRec1\] = \( \lambda x. \)
proof
  have if-as-pr: \( \lambda x. \iff p1 x = 0 \ then \ q1 x \ else \ q2 x \) \( \in PrimRec1 \)
  proof (rule ext)
    fix x show \( \iff p1 x = 0 \ then \ q1 x \ else \ q2 x \) \( = (sgn2 \ p1 x) \) * \( q1 x \) + \( (sgn1 \ p1 x) \) * \( q2 x \) \( is \ ?left = ?right \)
    proof cases
      assume A1: \( p1 x = 0 \)
      then have S1: \( ?left = q1 x \) by simp
      from A1 have S2: \( ?right = q1 x \) by simp
      from S1 S2 show \(?thesis \) by simp
    next
      assume A2: \( p1 x \neq 0 \)
      then have S3: \( p1 x > 0 \) by simp
      then show \(?thesis \) by simp
    qed
  qed

assume p1 \in PrimRec1 and q1 \in PrimRec1 and q2 \in PrimRec1
then have \( \lambda x. \ (sgn2 \ p1 x) \) * \( q1 x \) + \( (sgn1 \ p1 x) \) * \( q2 x \) \( \in PrimRec1 \) by prec0
  with if-as-pr show \(?thesis \) by simp
qed

lemma if-eq-is-pr [prec]\( p1 \in PrimRec1; p2 \in PrimRec1; q1 \in PrimRec1; q2 \in PrimRec1 \) = \( \lambda x. \)
proof
  have S1: \( \lambda x. \iff p1 x = p2 x \ then \ q1 x \ else \ q2 x \) \( = (\abs-of-diff \ p1 \ p2) \)
x) (p2 x) = 0) then (q1 x) else (q2 x)) (is \( ?L = ?R \)) by (simp add: abs-of-diff-eq)
assume A1: p1 \in PrimRec1 and A2: p2 \in PrimRec1
with abs-of-diff-is-pr have S2: \( \lambda x. \, abs-of-diff \ (p1 \ x) \ (p2 \ x) \) \in PrimRec1 by prec0
assume q1 \in PrimRec1 and q2 \in PrimRec1
with S2 have \( ?R \in PrimRec1 \) by (rule if-is-pr)
with S1 show \( \?thesis \ by simp \)
qed

lemma if-is-pr2 \[ prec \]: \[ p \in PrimRec2; q1 \in PrimRec2; q2 \in PrimRec2 \] \implies \( \lambda x y. \, if (p x y = 0) then (q1 x y) else (q2 x y) \) \in PrimRec2
proof –
  have if-as-pr: \( \lambda x y. \, if (p x y = 0) then (q1 x y) else (q2 x y) \) = \( \lambda x y. \, (sgn2 (p x y)) * (q1 x y) \) + (sgn1 (p x y)) * (q2 x y)
  proof (rule ext, rule ext)
    fix x fix y show (if (p x y = 0) then (q1 x y) else (q2 x y)) = (sgn2 (p x y)) * (q1 x y) + (sgn1 (p x y)) * (q2 x y) (is \( ?L = ?R \))
    proof cases
      assume A1: p x y = 0
      then have S1: ?left = q1 x y by simp
      from A1 have S2: ?right = q1 x y by simp
      from S1 S2 show \( \?thesis \ by simp \)
    next
      assume A2: p x y \neq 0
      then have S3: p x y > 0 by simp
      then show \( \?thesis \ by simp \)
    qed
qed

assume p \in PrimRec2 and q1 \in PrimRec2 and q2 \in PrimRec2
then have \( \lambda x y. \, (sgn2 (p x y)) * (q1 x y) \) + (sgn1 (p x y)) * (q2 x y) \in PrimRec2 by prec0
with if-as-pr show \( \?thesis \ by simp \)
qed

lemma if-eq-is-pr2 \[ prec \]: \[ p1 \in PrimRec2; p2 \in PrimRec2; q1 \in PrimRec2; q2 \in PrimRec2 \] \implies \( \lambda x y. \, if (p1 x y = p2 x y) then (q1 x y) else (q2 x y) \) \in PrimRec2
proof –
  have S1: \( \lambda x y. \, if (p1 x y = p2 x y) then (q1 x y) else (q2 x y) \) = \( \lambda x y. \, if (abs-of-diff (p1 x y)) (p2 x y) = 0 \) then (q1 x y) else (q2 x y) (is \( ?L = ?R \)) by (simp add: abs-of-diff-eq)
  assume A1: p1 \in PrimRec2 and A2: p2 \in PrimRec2
  with abs-of-diff-is-pr have S2: \( \lambda x y. \, abs-of-diff \ (p1 x y) \ (p2 x y) \) \in PrimRec2 by prec0
  assume q1 \in PrimRec2 and q2 \in PrimRec2
  with S2 have \( ?R \in PrimRec2 \) by (rule if-is-pr2)
  with S1 show \( \?thesis \ by simp \)
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 (\text{if} \ (p \ x y z = 0) \text{then} \ (q1 \ x y z) \text{else} \ (q2 \ 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) \ (\text{is} \ ?\text{left} = \ ?\text{right})

    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 \text{thesis} by simp

    next

      assume A2: p x y z \neq 0
      then have S3: p x y z > 0 by simp

      then show \text{thesis} by simp

    qed

  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 \text{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 if-as-pr: (\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)) = (\lambda x y z. (\text{abs-of-diff} (p1 \ x y z) (p2 \ 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)

    assume A1: p1 \in \text{PrimRec3} and A2: p2 \in \text{PrimRec3}

    with \text{abs-of-diff-is-pr} have S2: (\lambda x y z. \text{abs-of-diff} (p1 \ x y z) (p2 \ 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 \text{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 = ⟨⟨
  Attr.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 \( b\text{-least-le-arg} \): \( b\text{-least} \, f \, x \leq x \)
proof
  have \( b\text{-least} \, f \, x = x \lor (b\text{-least} \, f \, x < x \land (f \, x \, (b\text{-least} \, f \, x)) \neq 0) \) by (rule b-least-aux1)
  from this show \( ?\text{thesis} \) by (arith)
qed

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

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

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

lemma \( b\text{-least-less-impl-eq} \): \( b\text{-least} \, f \, x < x \implies (b\text{-least} \, f \, x) = (\text{Least} \, (\%y. \, (f \, x \, y) \neq 0))) \)
proof
  assume \( A1: \, b\text{-least} \, f \, x < x \) (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 \( S2: \, ?B < ?b \) 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
\[\begin{align*}
\text{qed}\\
\text{lemma } \text{less-b-least-impl-zero2}: \forall y < x; b\text{-least } f x = x \implies f x y = 0 \text{ by (simp add: less-b-least-impl-zero)}\\
\text{lemma } \text{nz-impl-b-least-less}: \forall y < x; (f x y) \neq 0 \implies (b\text{-least } f x) < x\\
\text{proof} - \\
\quad \text{assume } A1: y < x \\
\quad \text{assume } f x y \neq 0 \\
\quad \text{then have } b\text{-least } f x \leq y \text{ by (rule nz-impl-b-least-le)} \\
\quad \text{with } A1 \text{ show } \text{thesis by simp} \\
\text{qed}\\
\text{lemma } \text{b-least-aux2}: \forall y < x; (f x y) \neq 0 \implies (b\text{-least } f x) = \text{Least}(\%y. (f x y) \neq 0)\\
\text{proof} - \\
\quad \text{assume } A1: y < x \text{ and } A2: f x y \neq 0 \\
\quad \text{from } A1 \ A2 \text{ have } S1: b\text{-least } f x < x \text{ by (rule nz-impl-b-least-less)} \\
\quad \text{thus } \text{thesis by (rule b-least-less-impl-eq)} \\
\text{qed}\\
\text{lemma } \text{b-least2-aux1}: 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)\\
\text{proof} - \\
\quad \text{let } ?P = \%z. z = y \lor (z < y \land (f x z) \neq 0) \\
\quad \text{have } ?P y \text{ by simp} \\
\quad \text{then have } ?P \text{ (Least } ?P) \text{ by (rule LeastI)} \\
\quad \text{thus } \text{thesis by (simp add: b-least2-def)} \\
\text{qed}\\
\text{lemma } \text{b-least2-le-arg}: b\text{-least2 } f x y \leq y\\
\text{proof} - \\
\quad \text{let } ?B = b\text{-least2 } f x y \\
\quad \text{have } ?B = y \lor (?B < y \land (f x ?B) \neq 0) \text{ by (rule b-least2-aux1)} \\
\quad \text{from this show } \text{thesis by (arith)} \\
\text{qed}\\
\text{lemma } \text{less-b-least2-impl-zero}: z < b\text{-least2 } f x y \implies f x z = 0\\
\text{proof} - \\
\quad \text{assume } A1: z < b\text{-least2 } f x y \text{ (is } < ?b) \\
\quad \text{have } b\text{-least2 } f x y \leq y \text{ by (rule b-least2-le-arg)} \\
\quad \text{with } A1 \text{ have } S1: z < y \text{ by simp} \\
\quad \text{with } A1 \text{ have } z < \text{Least}(\%z. z = y \lor (z < y \land (f x z) \neq 0)) \text{ by (simp add: b-least2-def)} \\
\quad \text{then have } \neg (z = y \lor (z < y \land (f x z) \neq 0)) \text{ by (rule not-less-Least)} \\
\quad \text{with } S1 \text{ show } \text{thesis by simp} \\
\text{qed}\\
\text{lemma } \text{nz-impl-b-least2-le}: (f x z) \neq 0 \implies (b\text{-least2 } f x y) \leq z\\
\end{align*}\]
proof –
  assume \( A1 : f \ x \ z \neq 0 \)
  have \( S1 : z < b - \text{least2} f \ x \ y \implies f \ x \ z = 0 \)
    by (rule \( \text{less-b-least2-impl-zero} \))
  from \( A1 \) \( S1 \) show \( ?\text{thesis} \) 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 \( ?\text{thesis} \) by simp
qed

lemma \( b - \text{least2-less-impl-eq} \): \( b - \text{least2} f \ x \ y < y \implies (b - \text{least2} f \ x \ y) = (\text{Least} (\forall z. (f \ x \ z) \neq 0)) \)
proof –
  assume \( A1 : b - \text{least2} f \ x \ y < y \) (is \( ?b < - \))
  let \( ?B = (\text{Least} (\forall z. (f \ x \ z) \neq 0)) \)
  from \( A1 \) have \( S1 : f \ x \ ?b \neq 0 \) by (rule \( b - \text{least2-less-impl-nz} \))
  from \( S1 \) have \( S2 : ?b \leq ?b \) by (rule \( \text{Least-le} \))
  from \( S1 \) have \( S3 : f \ x \ ?B \neq 0 \) by (rule \( \text{LeastI} \))
  from \( S3 \) have \( S4 : ?b \leq ?B \) by (rule \( \text{nz-impl-b-least2-le} \))
  from \( S2 \) \( S4 \) show \( ?\text{thesis} \) by simp
qed

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

lemma \( \text{nz-b-least2-impl-less} \): \([ z < y ; (f \ x \ z) \neq 0 ] \implies (b - \text{least2} f \ x \ y) < y \)
proof (rule \( \text{ccontr} \))
  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 \( \text{less-b-least2-impl-zero2} \))
  with \( A2 \) show \( \text{False} \) by simp
qed

lemma \( \text{b-least2-less-impl-eq2} \): \([ z < y ; (f \ x \ z) \neq 0 ] \implies (b - \text{least2} f \ x \ y) = (\text{Least} (\forall z. (f \ x \ z) \neq 0)) \)
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 \( \text{nz-b-least2-impl-less} \))

thus \( \text{thesis} \) by (rule b-least2-less-impl-eq)

qed

lemma b-least2-aux2: \( \text{b-least2 } f \ x \ y < y \implies \text{b-least2 } f \ x \ (\text{Suc } y) = \text{b-least2 } f \ x \ y \)

proof –
let \( ?B = \text{b-least2 } f \ x \ y \)
assume \( A1: ?B < y \)
from \( A1 \) have \( S1: f \ x \ ?B \neq 0 \) by (rule b-least2-less-impl-nz)
from \( S1 \) have \( S2: \text{b-least2 } f \ x \ (\text{Suc } y) \leq ?B \) by (simp add: nz-impl-b-least2-le)
from \( A1 \ S2 \) have \( S3: \text{b-least2 } f \ x \ (\text{Suc } y) < ?B \) by (simp)
(from \( S3 \) have \( S4: f \ x \ (\text{b-least2 } f \ x \ (\text{Suc } y)) \neq 0 \) by (rule b-least2-less-impl-nz)
from \( S4 \) have \( S5: ?B \leq \text{b-least2 } f \ x \ (\text{Suc } y) \) by (rule nz-impl-b-least2-le)
from \( S2 \ S5 \) show \( \text{thesis} \) by simp

qed

lemma b-least2-aux3: \( \left[ \text{b-least2 } f \ x \ y = y; f \ x \ y \neq 0 \right] \implies \text{b-least2 } f \ x \ (\text{Suc } y) = y \)

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

qed

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

qed

lemma b-least2-mono: \( y1 \leq y2 \implies \text{b-least2 } f \ x \ y1 \leq \text{b-least2 } f \ x \ y2 \)

proof (rule ccontr)
assume \( A1: y1 \leq y2 \)
let \( ?b1 = \text{b-least2 } f \ x \ y1 \) and \( ?b2 = \text{b-least2 } f \ x \ y2 \)
assume \( \neg ?b1 \leq ?b2 \) 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: \( \left[ \text{b-least2 } f \ x \ y = y; f \ x \ y = 0 \right] \implies \text{b-least2 } f \ x \ (\text{Suc } y) = \text{Suc } y \)

proof –
assume \( A1: \text{b-least2 } f \ x \ y = y \)
assume $A_2$: $f \times y = 0$

have $S_1$: $\text{b-least2 } f \times (\text{Suc } y) \leq \text{Suc } y$ by (rule $\text{b-least2-le-arg}$)

have $S_2$: $y \leq \text{b-least2 } f \times (\text{Suc } y)$

proof -

have $y \leq \text{Suc } y$ by simp

then have $\text{b-least2 } f \times y \leq \text{b-least2 } f \times (\text{Suc } y)$ by (rule $\text{b-least2-mono}$)

with $A_1$ show $\text{thesis}$ by simp

qed

from $S_1$ $S_2$ have $\text{b-least2 } f \times (\text{Suc } y) = y$ $\lor$ $\text{b-least2 } f \times (\text{Suc } y) = \text{Suc } y$ by arith

moreover

{} assume $A_3$: $\text{b-least2 } f \times (\text{Suc } y) = y$

have $f \times y \neq 0$

proof -

have $y < \text{Suc } y$ by simp

with $A_3$ have $\text{b-least2 } f \times (\text{Suc } y) < \text{Suc } y$ by simp

from this have $f \times (\text{b-least2 } f \times (\text{Suc } y)) \neq 0$ by (simp add: $\text{b-least2-less-impl-nz}$)

with $A_3$ show $f \times y \neq 0$ by simp

qed

with $A_2$ have $\text{thesis}$ by simp

moreover

{} assume $\text{b-least2 } f \times (\text{Suc } y) = \text{Suc } y$

then have $\text{thesis}$ by simp

ultimately show $\text{thesis}$ by blast

qed

lemma $\text{b-least2-at-zero}$: $\text{b-least2 } f \times 0 = 0$

proof -

have $S_1$: $\text{b-least2 } f \times 0 \leq 0$ by (rule $\text{b-least2-le-arg}$)

from $S_1$ show $\text{thesis}$ by auto

qed

theorem $\text{pr-b-least2}$: $f \in \text{PrimRec2} \implies \text{b-least2 } f \in \text{PrimRec2}$

proof -

define $\text{loc-Op1}$ where $\text{loc-Op1} = (\lambda (f::nat \Rightarrow nat \Rightarrow nat) \times y. (\text{sgn1} (z - y)) \times y + (\text{sgn2} (z - y)) \times ((\text{sgn1} (f \times z)) \times z + (\text{sgn2} (f \times z)) \times (\text{Suc } z)))$

define $\text{loc-Op2}$ where $\text{loc-Op2} = (\lambda f. \text{PrimRecOp-last} (\lambda x. 0) (\text{loc-Op1 } f))$

have $\text{loc-op2-lm-1}$: $\forall f \times y. \text{loc-Op2 } f \times y < y \implies \text{loc-Op2 } f \times (\text{Suc } y) = \text{loc-Op2 } f \times y$

proof -

fix $f \times y$

let $?b = \text{loc-Op2 } f \times y$

have $S_1$: $\text{loc-Op2 } f \times (\text{Suc } y) = (\text{loc-Op1 } f) \times y$ by (simp add: $\text{loc-Op2-def}$)

assume $?b < y$

then have $y - $?b $> 0$ by simp
then have \( \text{loc-Op1} \ f \ x \ ?b \ y = ?b \) by (simp add: loc-Op1-def)

with \( S1 \) show \( \text{loc-Op2} \ f \ x \ y < y \implies \text{loc-Op2} \ f \ x \ (\text{Suc} \ y) = \text{loc-Op2} \ f \ x \ y \) by simp

qed

have \( \text{op-2-lm-2} \): \( \forall f \ x \ y. [\neg (\text{loc-Op2} \ f \ x \ y < y); f x y \neq 0] \implies \text{loc-Op2} \ f \ x \ (\text{Suc} \ y) = y \)

proof –
  fix \( f \ x \ y \)
  let \( ?b = \text{loc-Op2} \ f \ x \ y \) and \( ?h = \text{loc-Op1} \ f \)
  have \( S1: \text{loc-Op2} \ f \ x \ (\text{Suc} \ y) = ?h \ x \ ?b \ y \) by (simp add: loc-Op2-def)
  assume \( \neg (?b < y) \)
  then have \( S2: y - ?b = 0 \) by simp
  assume \( f x y \neq 0 \)
  with \( S2 \) have \( ?h \ x \ ?b \ y = y \) by (simp add: loc-Op1-def)
  with \( S1 \) show \( \text{loc-Op2} \ f \ x \ (\text{Suc} \ y) = y \) by simp

qed

have \( \text{op-2-lm-3} \): \( \forall f \ x \ y. [\neg (\text{loc-Op2} \ f \ x \ y < y); f x y = 0] \implies \text{loc-Op2} \ f \ x \ (\text{Suc} \ y) = \text{Suc} \ y \)

proof –
  fix \( f \ x \ y \)
  let \( ?b = \text{loc-Op2} \ f \ x \ y \) and \( ?h = \text{loc-Op1} \ f \)
  have \( S1: \text{loc-Op2} \ f \ x \ (\text{Suc} \ y) = ?h \ x \ ?b \ y \) by (simp add: loc-Op2-def)
  assume \( \neg (?b < y) \)
  then have \( S2: y - ?b = 0 \) by simp
  assume \( f x y = 0 \)
  with \( S2 \) have \( ?h \ x \ ?b \ y = \text{Suc} \ y \) by (simp add: 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: loc-Op2-def 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 loc-Op2-lm-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 b-least2-aux2)
      from \( A1 \ S1 \ S2 \) show \( \text{thesis} \) 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 b-least2-le-arg)
        from \( A1 \ A3 \) this show \( \text{thesis} \) by simp
      qed
  qed

28
then show \( ?\text{thesis} \)
proof cases
  \( \text{assume } A_4 \text{: } f \ x \ y \neq 0 \)
  \( \text{with } A_3 \text{ have } S_3 \text{: loc-Op}\, f \ x \ (\text{Suc } y) = y \) by (rule \text{loc-op2-lm-2})
  \( \text{from } A_3' \ A_4 \text{ have } S_4 \text{: } \text{b-least2} f \ x \ (\text{Suc } y) = y \) by (rule \text{b-least2-aux3})
  \( \text{from } S_3 S_4 \text{ show } ?\text{thesis} \) by simp
next
  \( \text{assume } \neg f \ x \ y \neq 0 \)
  \( \text{then have } A_5 \text{: } f \ x \ y = 0 \) by simp
  \( \text{with } A_3 \text{ have } S_5 \text{: loc-Op}\, f \ x \ (\text{Suc } y) = \text{Suc } y \) by (rule \text{loc-op2-lm-3})
  \( \text{from } A_3' \ A_5 \text{ have } S_6 \text{: } \text{b-least2} f \ x \ (\text{Suc } y) = \text{Suc } y \) by (rule \text{b-least2-aux4})
  \( \text{from } S_5 S_6 \text{ show } ?\text{thesis} \) by simp
qed

have Op2-eq-b-least2: \( \text{loc-Op}\, = \text{b-least2} \) by (simp add: Op2-eq-b-least2-at-point ext)
assume A1: \( f \in \text{PrimRec2} \)
have pr-loc-Op2: \( \text{loc-Op}\, f \in \text{PrimRec2} \)
proof
  \( \text{from } A_1 \text{ have } S_1 \text{: loc-Op1}\, f \in \text{PrimRec3} \) by (simp add: loc-Op1-def, prec)
  \( \text{from } \text{pr-zero } S_1 \text{ have } S_2 \text{: PrimRecOp-last } (\lambda x. 0) \text{ (loc-Op1 } f) \in \text{PrimRec2} \)
  \( \text{by } (\text{rule } \text{pr-rec-last}) \)
  \( \text{from } this \text{ show } ?\text{thesis} \) by (simp add: loc-Op2-def)
qed

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

theorem pr-b-least: \( f \in \text{PrimRec2} \implies \text{b-least } f \in \text{PrimRec1} \)
proof
  \( \text{assume } f \in \text{PrimRec2} \)
  then have b-least2 f \in PrimRec2 by (rule pr-b-least2)
  \( \text{from } this \text{ pr-id1-1 pr-id1-1 have } (\lambda x. \text{b-least2 } f \ x \ x) \in \text{PrimRec1} \) by (rule pr-comp2-1)
  \( \text{then show } ?\text{thesis} \) by (simp add: b-least-def1)
qed

2.3 Examples

theorem c-sum-as-b-least: \( c\text{-sum} = (\lambda u. \text{b-least2 } (\lambda u z. (\text{sgn1 } (\text{sf}(z+1) - u))) u (\text{Suc } u)) \)
proof (rule ext)
  \( \text{fix } u \text{ show } c\text{-sum } u = \text{b-least2 } (\lambda u z. (\text{sgn1 } (\text{sf}(z+1) - u))) u (\text{Suc } u) \)
  proof
    \( \text{have } \text{lm-1: } (\lambda x y. (\text{sgn1 } (\text{sf}(y+1) - x) \neq 0)) = (\lambda x y. (x < \text{sf}(y+1))) \)
proof (rule ext, rule ext)
fix x y show \((\text{sgn}1 \ (sf(y+1) - x) \neq 0) = (x < sf(y+1))\)
proof
have \((\text{sgn}1 \ (sf(y+1) - x) \neq 0) = (sf(y+1) - x > 0)\) by (rule sgn1-nz-eq-arg-pos)
thus \((\text{sgn}1 \ (sf(y+1) - x) \neq 0) = (x < sf(y+1))\) by auto
qed
qed
let \(?f = \lambda u z. \ (\text{sgn}1 \ (sf(z+1) - u))\)
have \(S1: \ ?f u u \neq 0\)
proof
have \(S1-1: \ u+1 \leq sf(u+1)\) 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 \(S1-5: \ \text{sgn}1 \ (sf(u+1)-u) = 1\) by simp
from \(S1-5\) show \(\text{thesis}\) by simp
qed
have \(S3: \ u < Suc u\) by simp
from \(S3\) \(S1\) have \(S4: \ \text{b-least2} \ ?f u (Suc u) = (\text{Least} (\%z. \ (?f u z) \neq 0))\) by (rule b-least2-less-impl-eq2)
let \(?P = \lambda u z. \ ?f u z \neq 0\)
let \(?Q = \lambda u z. \ u < sf(z+1)\)
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 \(S8: (\text{Least} (\%z. \ ?P u z)) = (\text{Least} (\%z. \ ?Q u z))\) by auto
from \(S4 \ S8\) have \(S9: \ \text{b-least2} \ ?f u (Suc u) = (\text{Least} (\%z. \ u < sf(z+1)))\) by (rule trans)
thus \(\text{thesis}\) by (simp add: c-sum-def)
qed
qed

theorem c-sum-is-pr: \(c \text{-} \text{sum} \in \text{PrimRec1}\)
proof
let \(?f = \lambda u z. \ (\text{sgn}1 \ (sf(z+1) - u))\)
have \(S1: \ (\lambda u z. \ \text{sgn}1 \ ((sf(z+1) - u))) \in \text{PrimRec2}\) by prec
define \(g\) where \(g = \text{b-least2} \ ?f\)
from \(g\)-def \(S1\) have \(g \in \text{PrimRec2}\) by (simp add: pr-b-least2)
then have \(S2: \ (\lambda u. \ g u (Suc u)) \in \text{PrimRec1}\) by prec
from \(g\)-def have \(c \text{-} \text{sum} = (\lambda u. \ g u (Suc u))\) by (simp add: c-sum-as-b-least ext)
with \(S2\) show \(\text{thesis}\) by simp
qed

theorem c-fst-is-pr [prec]: \(c \text{-} \text{fst} \in \text{PrimRec1}\)
proof
have \(S1: \ (\lambda u. \ c \text{-} \text{fst} u) = (\lambda u. \ (u - sf (c \text{-} \text{sum} u)))\) by (simp add: c-fst-def ext)
from \(c \text{-} \text{sum-is-pr}\) have \(\ (\lambda u. \ (u - sf (c \text{-} \text{sum} u))) \in \text{PrimRec1}\) by prec
with \(S1\) show \(\text{thesis}\) by simp
qed
**Theorem:** \( c\text{-snd-is-pr} \) \([\text{Prec}]:\) \( c\text{-snd} \in \text{PrimRec1} \)

**Proof:**

- Have \( S1: c\text{-snd} = (\lambda u. (c\text{-sum} u) - (c\text{-fst} u)) \) by \((\text{simp add: c-snd-def ext})\)
- From \( c\text{-sum-is-pr c-fst-is-pr have S2: } (\lambda u. (c\text{-sum} u) - (c\text{-fst} u)) \in \text{PrimRec1}\) by \(\text{prec}\)

  - From \(S1\) this show \( ?\text{thesis} \) by \(\text{simp}\)

**QED**

**Theorem:** \( pr\text{-1-to-2}: f \in \text{PrimRec1} \implies (\lambda x y. f (c\text{-pair} x y)) \in \text{PrimRec2} \) by \(\text{prec}\)

**Theorem:** \( pr\text{-2-to-1}: f \in \text{PrimRec2} \implies (\lambda z. f (c\text{-fst} z) (c\text{-snd} z)) \in \text{PrimRec1} \) by \(\text{prec}\)

**Definition:** \( pr\text{-conv-1-to-2} = (\lambda f x y. f (c\text{-pair} x y)) \)

**Definition:** \( pr\text{-conv-1-to-3} = (\lambda f x y z. f (c\text{-pair} (c\text{-pair} x y) z)) \)

**Definition:** \( pr\text{-conv-2-to-1} = (\lambda f x. f (c\text{-fst} (c\text{-fst} x)) (c\text{-snd} (c\text{-fst} x))) \)

**Definition:** \( pr\text{-conv-3-to-1} = (\lambda f, pr\text{-conv-1-to-2} (pr\text{-conv-3-to-1} f)) \)

**Definition:** \( pr\text{-conv-3-to-2} = (\lambda f. pr\text{-conv-1-to-3} (pr\text{-conv-2-to-1} f)) \)

**Lemma:** \([\text{simp}]\): \( pr\text{-conv-1-to-2} (pr\text{-conv-2-to-1} f) = f\) by \((\text{simp add: pr\text{-conv-1-to-2-def pr\text{-conv-2-to-1-def}}})\)

**Lemma:** \([\text{simp}]\): \( pr\text{-conv-2-to-1} (pr\text{-conv-1-to-2} f) = f\) by \((\text{simp add: pr\text{-conv-1-to-2-def pr\text{-conv-2-to-1-def}}})\)

**Lemma:** \([\text{simp}]\): \( pr\text{-conv-1-to-3} (pr\text{-conv-3-to-1} f) = f\) by \((\text{simp add: pr\text{-conv-1-to-3-def pr\text{-conv-3-to-1-def}}})\)

**Lemma:** \([\text{simp}]\): \( pr\text{-conv-3-to-1} (pr\text{-conv-1-to-3} f) = f\) by \((\text{simp add: pr\text{-conv-1-to-3-def pr\text{-conv-3-to-1-def}}})\)

**Lemma:** \([\text{simp}]\): \( pr\text{-conv-2-to-3} = (\lambda f. pr\text{-conv-1-to-3} (pr\text{-conv-2-to-1} f)) \)

**Lemma:** \([\text{simp}]\): \( pr\text{-conv-1-to-2-lm}: f \in \text{PrimRec1} \implies pr\text{-conv-1-to-2} f \in \text{PrimRec2} \) by \((\text{simp add: pr\text{-conv-1-to-2-def, prec})\)

**Lemma:** \([\text{simp}]\): \( pr\text{-conv-1-to-3-lm}: f \in \text{PrimRec1} \implies pr\text{-conv-1-to-3} f \in \text{PrimRec3} \) by \((\text{simp add: pr\text{-conv-1-to-3-def, prec})\)

**Lemma:** \([\text{simp}]\): \( pr\text{-conv-2-to-1-lm}: f \in \text{PrimRec2} \implies pr\text{-conv-2-to-1} f \in \text{PrimRec1} \) by \((\text{simp add: pr\text{-conv-2-to-1-def, prec})\)

**Lemma:** \([\text{simp}]\): \( pr\text{-conv-3-to-1-lm}: f \in \text{PrimRec3} \implies pr\text{-conv-3-to-1} f \in \text{PrimRec1} \) by \((\text{simp add: pr\text{-conv-3-to-1-def, prec})\)

**Lemma:** \([\text{simp}]\): \( pr\text{-conv-3-to-2-lm}: f \in \text{PrimRec3} \implies pr\text{-conv-3-to-2} f \in \text{PrimRec2} \)

**Proof:**

- Assume \( f \in \text{PrimRec3} \)

  - Then have \( pr\text{-conv-3-to-1} f \in \text{PrimRec1} \) by \((\text{rule pr\text{-conv-3-to-1-lm})}\)

  - Thus \( ?\text{thesis} \) by \((\text{simp add: pr\text{-conv-3-to-2-def pr\text{-conv-1-to-2-lm}}})\)

**QED**
lemma pr-conv-2-to-3-lm: \( f \in \text{PrimRec}2 \implies \text{pr-conv-2-to-3} f \in \text{PrimRec}3 \)

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

theorem b-least2-scheme: \( \left[ f \in \text{PrimRec}2; g \in \text{PrimRec}1; \forall x. h x < g x; \forall x. f x (h x) \neq 0 ; \forall z x. z < h x \implies f x z = 0 \right] \implies h \in \text{PrimRec}1 \)

proof
- assume f-is-pr: \( f \in \text{PrimRec}2 \)
assume g-is-pr: \( g \in \text{PrimRec}1 \)
assume h-lt-g: \( \forall x. h x < g x \)
assume h-at-h-nz: \( \forall x. f x (h x) \neq 0 \)
assume h-is-min: \( \forall z x. z < h x \implies f x z = 0 \)
have 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 f-at-h-nz have \( S1: \text{b-least2} f x (g x) \leq h x \) by (simp add: nz-impl-b-least2-le)
from h-lt-g have \( h x < g x \) by auto
with \( S1 \) have \( S2: f x (h x) \neq 0 \) by (rule b-least2-less-impl-nz)
have \( S3: h x \leq \text{b-least2} f x (g x) \)
proof (rule ccontr)
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 h-is-min have \( f x (\text{b-least2} f x (g x)) = 0 \) by simp
with \( S2 \) show \( \text{False} \) by auto
qed
from \( S1, S3 \) show \(?thesis\) by (simp add: nz-impl-b-least2-le)
qed

define f1 where \( f1 = \text{b-least2} f \)
from f-is-pr f1-def have f1-is-pr: \( f1 \in \text{PrimRec}2 \) by (simp add: pr-b-least2)
with g-is-pr have \( (\lambda x. f1 x (g x)) \in \text{PrimRec}1 \) by prec
with h-def f1-def show \( h \in \text{PrimRec}1 \) by auto
qed

theorem b-least2-scheme2: \( \left[ f \in \text{PrimRec}3; g \in \text{PrimRec}2; \forall x y. h x y < g x y; \forall x y. f x y (h x y) \neq 0 ; \forall z x y. z < h x y \implies f x y z = 0 \right] \implies h \in \text{PrimRec}2 \)

proof
- assume f-is-pr: \( f \in \text{PrimRec}3 \)
assume g-is-pr: \( g \in \text{PrimRec}2 \)
assume h-lt-g: \( \forall x y. h x y < g x y \)
assume f-at-h-nz: \( \forall x y. f x y (h x y) \neq 0 \)
assume \( h\)-is-min: \( \forall \ x \ y \ z < h \ x \ y \rightarrow f \ x \ y \ z = 0 \)
define \( f1 \) where \( f1 = \text{pr-conv-3-to-2} \ f \)
define \( g1 \) where \( g1 = \text{pr-conv-2-to-1} \ g \)
define \( h1 \) where \( h1 = \text{pr-conv-2-to-1} \ h \)
from \( f\)-is-pr \( f1\)-def have \( f1\)-is-pr: \( f1 \in \text{PrimRec2} \) by (simp add: \( \text{pr-conv-3-to-2-lm} \))
from \( g\)-is-pr \( g1\)-def have \( g1\)-is-pr: \( g1 \in \text{PrimRec1} \) by (simp add: \( \text{pr-conv-2-to-1-lm} \))

\[ \begin{align*}
\text{from} & \ \text{h-lt-g} \ \text{h1-def} \ \text{have} \ h1\text{-lt-g1} \ \forall \ x. \ h1 \ x < g1 \ x \ \text{by (simp add: pr-conv-2-to-1-def)} \\
\text{from} & \ \text{f-at-h-nz} \ \text{f1-def} \ \text{have} \ f1\text{-at-h1-nz} \ \forall \ x. \ f1 \ x (h1 \ x) \neq 0 \ \text{by (simp add: pr-conv-3-to-2-lm pr-conv-3-to-1-lm pr-conv-2-to-1-lm)} \\
\text{from} & \ \text{pr-conv-2-to-1 h} \ \text{h1-is-min} \ \text{have} \ h1\text{-is-pr:} \ h1 \in \text{PrimRec1} \ \text{by (rule b-least2-scheme)} \\
\text{with} & \ \text{h1-is-pr} \ \text{show} \ h \in \text{PrimRec2} \ \text{by (simp add: pr-conv-3-to-1-lm)} \\
\end{align*} \]

qed

theorem div-is-pr: \( (\lambda \ a \ b. \ a \div b) \in \text{PrimRec2} \)
proof -
   define \( f \) where \( f \ a \ b \ z = (\text{sgn1} \ b) * (\text{sgn1} (b*(z+1) - a)) + (\text{sgn2} \ b) * (\text{sgn2} \ z) \)
   for \( a \ b \ z \)
   have \( f\)-is-pr: \( f \in \text{PrimRec3} \) unfolding \( \text{f-def} \) by \( \text{prec} \)
define \( h \) where \( h \ a \ b = a \div b \) for \( a \ b :: \text{nat} \)
define \( g \) where \( g \ a \ b = a + 1 \) for \( a \ b :: \text{nat} \)

\[ \begin{align*}
\text{have} & \ \text{g-is-pr:} \ g \in \text{PrimRec2} \ \text{unfolding} \ \text{g-def} \ \text{by} \ \text{prec} \\
\text{have} & \ \text{h-lt-g:} \ \forall \ a \ b. \ h \ a \ b < g \ a \ b \\
\text{proof} & \ \text{(rule allI, rule allII)} \\
\text{fix} & \ \ a \ b \\
\text{from} & \ \text{h-def} \ \text{have} \ h \ a \ b \leq a \ \text{by simp} \\
\text{also from} & \ \text{g-def} \ \text{have} \ a < g \ a \ b \ \text{by simp} \\
\text{ultimately show} & \ h \ a \ b < g \ a \ b \ \text{by simp} \\
\end{align*} \]

qed

have \( f\)-at-h-nz: \( \forall \ a \ b. \ f \ a \ b (h \ a \ b) \neq 0 \)
proof (rule allI, rule allII)
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 \( \text{simp} \)
   with \( f\)-def \( A \) show \( ?\text{thesis} \) by \( \text{simp} \)
next
   assume \( A : b \neq 0 \)
   then have \( S1 : b > 0 \) by \( \text{auto} \)
from \( A \) \( f\)-def have \( S2 : f \ a \ b (h \ a \ b) = \text{sgn1} (b * (h \ a \ b + 1) - a) \) by \( \text{simp} \)
then have \( ?\text{thesis} = (\text{sgn1}(b * (h \ a \ b + 1) - a) \neq 0) \) by \( \text{auto} \)
also have \( . . . \) \( = (b * (h \ a \ b + 1) - a > 0) \) by \( \text{rule sgn1-nz-eq-arg-pos} \)
also have \( . . . \) \( = (a < b * (h \ a \ b + 1)) \) by \( \text{auto} \)
also have \( . . . \) \( = (a < b * (h \ a \ b) + b) \) by \( \text{auto} \)
also from \( h\)-def have \( . . . \) \( = (a < b * (a \ div b) + b) \) by \( \text{simp} \)
finally have S3: \( \text{thesis} = (a < b \times (a \div b) + b) \) by auto

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

have h-is-min: \( \forall z a b. z < h a b \longrightarrow f a b z = 0 \)
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 \( \text{False} \) 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 \( (b \times (z+1) \leq b \times (a \div b)) = (z+1 \leq a \div b) \) by (rule nat-mult-lc-cancel)
  with S4 have S5: \( b \times (z+1) \leq b \times (a \div b) \) by simp
  moreover have \( b \times (a \div b) \leq a \)
  proof
    have \( b \times (a \div b) + (a \mod b) = a \) by (rule mult-div-mod-eq)
    moreover have \( 0 \leq a \mod b \) by auto
    ultimately show \( \text{thesis} \) by arith
    qed
  ultimately have S6: \( b \times (z+1) \leq a \)
    by (simp add: minus-mod-eq-mult-div [symmetric])
  then have \( b \times (z+1) - a = 0 \) by auto
  with S3 f-def show \( \text{thesis} \) by simp
  qed
  qed
from f-is-pr g-is-pr h-lt-g f-at-b-nz h-is-min have h-is-pr: \( h \in \text{PrimRec2} \) by (rule b-least2-scheme)
  with h-def [abs-def] show \( \text{thesis} \) by simp
  qed

theorem mod-is-pr: \( \lambda a b. a \mod b \in \text{PrimRec2} \)
proof
  have \( \lambda (a::nat) (b::nat). a \mod b = (\lambda a b. a - (a \div b) \times b) \)
  proof (rule ext, rule ext)

34
fix a b show (a::nat) mod b = a − (a div b) * b by (rule minus-div-mult-eq-mod [symmetric])
  qed
also from div-is-pr have (λ a b. a − (a div b) * b) ∈ PrimRec2 by prec
ultimately show thesis by auto
qed

theorem pr-rec-last-scheme: [ g ∈ PrimRec1; h ∈ PrimRec3; ∀ x. f x 0 = g x;
∀ x y. f x (Suc y) = h x (f x y) y ] ⇒ f ∈ PrimRec2
proof −
  assume g-is-pr: g ∈ PrimRec1
  assume h-is-pr: h ∈ PrimRec3
  assume f-at-0: ∀ x. f x 0 = g x
  assume f-at-Suc: ∀ x y. f x (Suc y) = h x (f x y) y
  from f-at-0 f-at-Suc have \( \bigwedge x y. f x y = \text{PrimRecOp-last} g x y \) by (induct-tac y, simp-all)
  then have f = PrimRecOp-last g h by (simp add: ext)
  with g-is-pr h-is-pr show thesis by (simp add: pr-rec-last)
qed

theorem power-is-pr: (λ (x::nat) (n::nat). x ^ n) ∈ PrimRec2
proof −
  define g :: nat ⇒ nat where g x = 1 for x
  define h where h a b c = a * b for a b c :: nat
  have g-is-pr: g ∈ PrimRec1 unfolding g-def by prec
  have h-is-pr: h ∈ PrimRec3 unfolding h-def by prec
  let \( \lambda f = \lambda (x::nat) (n::nat). x ^ n \)
  have f-at-0: ∀ x. if x 0 = g x
  proof
  fix x show x ^ 0 = g x by (simp add: g-def)
  qed
  have f-at-Suc: ∀ x y. if x (Suc y) = h x (if x y) y
  proof (rule allI, rule allI)
  fix x y show if x (Suc y) = h x (if x y) y by (simp add: h-def)
  qed
  from g-is-pr h-is-pr f-at-0 f-at-Suc show thesis by (rule pr-rec-last-scheme)
  qed
end

3 Primitive recursive coding of lists of natural numbers

theory PRecList
imports PRecFun
begin

We introduce a particular coding list-to-nat from lists of natural numbers
to natural numbers.

definition
  c-len :: nat ⇒ nat where
c-len = (λ (u::nat). (sgn1 u) * (c-fst(u-λ:nat)+1))

lemma c-len-1: c-len u = (case u of 0 ⇒ 0 | Suc v ⇒ c-fst(v)+1) by (unfold c-len-def, cases u, auto)

lemma c-len-is-pr: c-len ∈ PrimRec1 unfolding c-len-def by prec

lemma [simp]: c-len 0 = 0 by (simp add: c-len-def)

lemma c-len-2: u ≠ 0 ⇒ c-len u = c-fst(u-λ:nat)+1 by (simp add: c-len-def)

lemma c-len-3: u>0 ⇒ c-len u > 0 by (simp add: c-len-2)

lemma c-len-4: c-len u = 0 ⇒ u = 0
proof cases
  assume A1: u = 0
  thus ?thesis by simp
next
  assume A1: u ≠ 0
  from A1 show ?thesis by simp
qed

lemma c-len-5: c-len u > 0 ⇒ u > 0
proof cases
  assume A1: c-len u > 0 and A2: u = 0
  from A2 have c-len u = 0 by (simp add: c-len-3)
  from A1 this show u=0 by simp
next
  assume A1: u ≠ 0
  from A1 show ?thesis by simp
qed

fun c-fold :: nat list ⇒ nat where
c-fold [] = 0
| c-fold [x] = x
| c-fold (x#ls) = c-pair x (c-fold ls)

lemma c-fold-0: ls ≠ [] ⇒ c-fold (x#ls) = c-pair x (c-fold ls)
proof –
  assume A1: ls ≠ []
  then have S1: ls = (hd ls)#(tl ls) by simp
  then have S2: x#ls = x#(hd ls)#(tl ls) by simp
  have S3: c-fold (x#(hd ls)#(tl ls)) = c-pair x (c-fold ((hd ls)#(tl ls))) by simp
  from S1 S2 S3 show ?thesis by simp
qed
primrec
c-unfold :: nat ⇒ nat ⇒ 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 = (c-fst u) # (c-unfold k (c-snd 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: \([k \succ 0; (\forall u. c\text{-fold } (c\text{-unfold } k u) = u)] \implies (\forall u. c\text{-fold } (c\text{-unfold } (Suc k) u) = u)\)

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

lemma th-lm-3: \((\forall u. c\text{-fold } (c\text{-unfold } (Suc k) u) = u) \implies (\forall u. c\text{-fold } (c\text{-unfold } (Suc (Suc k) u) = u)\)

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

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

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

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

theorem c-unfold-len: \(\forall u. \text{length } (c\text{-unfold } k u) = k\)
```plaintext
apply(induct k)
apply(simp)
apply(subgoal-tac n=(0::nat) \land n>0)
apply(drule disjE)
pref
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 \neq [] 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 S2 have S4: c-unfold (length (x # ls)) (c-fold (x # ls)) = c-unfold (Suc (length ls)) (c-fold (x # ls)) by simp
  from A1 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-snd (c-fold (x#ls)))) by (rule c-unfold-4)
  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 \(\Rightarrow\) nat
where
list-to-nat = \(\lambda\) ls. if ls = [] then 0 else (c-pair ((length ls) - 1) (c-fold ls)) + 1

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

lemma nat-to-list-of-pos: \(u > 0 \Rightarrow\) 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: \(u = 0 \Rightarrow\) ?thesis by (simp add: list-to-nat-def nat-to-list-def)
  have S2: \(u > 0 \Rightarrow\) ?thesis
  proof –
    assume A1: \(u > 0\)
    define ls where ls = nat-to-list u
    from ls-def A1 have S2-1: ls = c-unfold (c-len u) (c-snd (u -(1::nat))) by (simp add: nat-to-list-def)
    let \(\kappa\) = c-len u
    from A1 have S2-2: \(\kappa > 0\) by (rule c-len-3)
    from S2-1 have S2-3: length ls = \(\kappa\) 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 \(\neq\) [] 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
    qed
    from S2-7 show ?thesis by simp
  qed
  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 S2-7 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
  from ls-def 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=[] \implies ?thesis by (simp add: nat-to-list-def list-to-nat-def)
  have S2: ls \neq [] \implies ?thesis
  proof
    assume A1: ls \neq []
    define u where u = list-to-nat ls
    from u-def 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)
    qed
    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
    qed
    from 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 u-def this show ?thesis by simp
  qed
  have S3: ls = [] \lor ls \neq [] 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 \( c\text{-len-th-1}: \) \( c\text{-len \ (list-to-nat \ ls)} = \ length \ ls \)

proof (cases)
  assume \( ls=[] \)
  from this show \( \text{thesis by simp} \)

next
  assume \( S1: \) \( ls \neq \ [] \)
  then have \( S2: \) \( \text{list-to-nat} \ ls = \ c\text{-pair (} (\length \ ls)-(1::nat) \) \ (c\text{-fold} \ ls)+1 \) by
  (simp add: \text{list-to-nat-def})
  let \( \ ?u = \text{list-to-nat} \ ls \)
  from \( S2 \) have \( u\text{-not-zero} \: \) \( ?u > 0 \) by simp
  from \( S2 \) have \( S3: \) \( ?u-(1::nat) = \ c\text{-pair (} (\length \ ls)-(1::nat) \) \ (c\text{-fold} \ ls) \)
  by simp
  then have \( S4: \) \( \text{c-fst} \ (\ ?u-(1::nat)) = \ (\length \ ls)-(1::nat) \)
  by simp
  from \( u\text{-not-zero} \ S5 \) have \( S6: \) \( c\text{-len} \ (\ ?u) = \ length \ ls \)
  by (simp add: \text{c-len-2})
  from \( S1 \ S6 \) show \( \text{thesis by simp} \)
qed

theorem \( \text{length \ (nat-to-list} \ u) = \text{c-len} \ u \)

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

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

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

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

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

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

lemma \( \text{c-hd-aux1} \): \( \text{c-len} \ u = 1 \Rightarrow \text{c-hd} \ u = \text{c-snd} \ (u-(1::nat)) \)

proof
  assume \( A1: \) \( \text{c-len} \ u = 1 \)
  then have \( S1: \) \( \text{nat-to-list} \ u = [\text{c-snd} \ (u-(1::nat))] \)
  by (simp add: \text{nat-to-list-def} \text{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: \( \text{c-len } u > 1 \implies \text{c-hd } u = \text{c-fst } (\text{c-snd } (u - (1::nat))) \)
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: \( \text{nat-to-list } u = \text{c-unfold } (\text{c-len } u) (\text{c-snd } (u - (1::nat))) \) by (rule nat-to-list-of-pos)
  from S1 S5 have S6: \( \text{nat-to-list } u = \text{c-unfold } (\text{Suc } ?k) (\text{c-snd } (u - (1::nat))) \) by simp
  from A1 have S7: \( ?k > 0 \) by simp
  from S7 have S8: \( \text{c-unfold } (\text{Suc } ?k) (\text{c-snd } (u - (1::nat))) = (\text{c-fst } (\text{c-snd } (u - (1::nat)))) \# (\text{c-unfold } ?k (\text{c-snd } (u - (1::nat)))) \) by (rule c-unfold-4)
  from S6 S8 have S9: \( \text{nat-to-list } u = (\text{c-fst } (\text{c-snd } (u - (1::nat)))) \# (\text{c-unfold } ?k (\text{c-snd } (u - (1::nat)))) \) by simp
  from S9 have S10: \( \text{hd } (\text{nat-to-list } u) = \text{c-fst } (\text{c-snd } (u - (1::nat))) \) by simp
  from S4 S10 show \( \text{thesis} \) by simp qed

lemma c-hd-aux3: \( u > 0 \implies \text{c-hd } u = (\text{if } (\text{c-len } u) = 1 \text{ then } \text{c-snd } (u - (1::nat)) \text{ else } \text{c-fst } (\text{c-snd } (u - (1::nat)))) \)
proof -
  assume A1: \( u > 0 \)
  from A1 have 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 \( ?\text{tmp} = \text{if } (\text{c-len } u) = 1 \text{ then } \text{c-snd } (u - (1::nat)) \text{ else } \text{c-fst } (\text{c-snd } (u - (1::nat))) \)
  have S2: \( \text{c-len } u = 1 \implies ?\text{thesis} \)
  proof -
    assume A2-1: \( \text{c-len } u = 1 \)
    then have S2-1: \( \text{c-hd } u = \text{c-snd } (u - (1::nat)) \) by (rule c-hd-aux1)
    from A2-1 have S2-2: ?\text{tmp} = \text{c-snd}(u - (1::nat)) by simp
    from S2-1 this show \( ?\text{thesis} \) by simp
  qed
  have S3: \( \text{c-len } u > 1 \implies ?\text{thesis} \)
  proof -
    assume A3-1: \( \text{c-len } u > 1 \)
    from A3-1 have S3-1: \( \text{c-hd } u = \text{c-fst } (\text{c-snd } (u - (1::nat))) \) by (rule c-hd-aux2)
    from A3-1 have S3-2: ?\text{tmp} = \text{c-fst } (\text{c-snd } (u - (1::nat))) by simp
    from S3-1 this show \( ?\text{thesis} \) by simp
  qed
  from S1 S2 S3 show \( \text{thesis} \) by auto
qed
lemma c-hd-aux4: c-hd \ u = (if \ u=0 \ then \ 0 \ else \ (if \ (c-len \ u) = 1 \ then \ c-snd \ (u-\{1::nat\}) \ else \ c-fst \ (c-snd \ (u-\{1::nat\})))))

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-hd \in \ PrimRec1
proof
  have c-hd = (%u. (if u=0 then 0 else (if (c-len u) = 1 then c-snd (u-\{1::nat\}) else c-fst (c-snd (u-\{1::nat\})))))) (is - = ?R) by (simp add: c-hd-aux4 ext)
  moreover have ?R \in \PrimRec1
  proof (rule if-is-pr)
    show (\lambda x. x) \in \PrimRec1 by (rule pr-id1-1)
    next show (\lambda x. 0) \in \PrimRec1 by (rule pr-zero)
    next show (\lambda x. if c-len x = 1 then c-snd (x - 1) else c-fst (c-snd (x - 1))) \in \PrimRec1
    proof (rule if-eq-is-pr)
      show c-len \in \PrimRec1 by (rule c-len-is-pr)
      next show (\lambda x. 1) \in \PrimRec1 by (rule const-is-pr)
      next show (\lambda x. c-snd (x - 1)) \in \PrimRec1 by prec
      qed
    qed
    ultimately show ?thesis by simp
  qed

lemma [simp]: c-tl 0 = 0 by (simp add: c-tl-def)

lemma c-tl-eq-tl: c-tl (list-to-nat ls) = list-to-nat (tl ls) by (simp add: c-tl-def)

lemma tl-eq-c-tl: tl (nat-to-list x) = nat-to-list (c-tl x) by (simp add: c-tl-def)

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

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

proof
  assume A1: c-len \ u > 1
  let \ ?k = (c-len \ u) - 1
  from A1 have S1: c-len \ u = Suc \ ?k by simp
  from A1 have S2: c-len \ u > 0 by simp
  from S2 have S3: u > 0 by (rule c-len-5)
  from S3 have S4: nat-to-list u = c-unfold (c-len u) (c-snd (u-\{1::nat\})) by (rule nat-to-list-of-pos)
  from A1 have S5: \ ?k > 0 by simp

44
from S5 have S6: c-unfold (Suc ?k) (c-snd (u-(1::nat))) = (c-fst (c-snd (u-(1::nat)))) # (c-unfold ?k (c-snd (c-snd (u-(1::nat))))) by (rule c-unfold-4)
from S6 have S7: tl (c-unfold (Suc ?k) (c-snd (u-(1::nat)))) = c-unfold ?k (c-snd (c-snd (u-(1::nat)))) by simp
from S2 S4 S7 have S8: (nat-to-list u) = c-unfold ?k (c-snd (c-snd (u-(1::nat)))) by simp

define ls where ls = tl (nat-to-list u)
from ls-def S8 have S9: length ls = ?k by (simp add: c-unfold-len)
from ls-def have S10: c-tl u = list-to-nat ls by (simp add: c-tl-def)
from S5 S9 have S11: length ls > 0 by simp
from S11 have S12: ls ≠ [] by simp
from S12 have S13: list-to-nat ls = (c-pair ((length ls) - 1) (c-fold ls)) + 1 by (simp add: list-to-nat-def)
from S10 S13 have S14: c-tl u = (c-pair ((length ls) - 1) (c-fold ls)) + 1 by simp
from S9 have S15: (length ls)-(1::nat) = ?k-(1::nat) by simp
from A1 have S16: ?k-(1::nat) = c-len u - (2::nat) by arith
from S15 S16 have S17: (length ls)-(1::nat) = c-len u - (2::nat) by simp
from ls-def S8 have S18: ls = c-unfold ?k (c-snd (c-snd (u-(1::nat)))) by simp
from S5 have S19: c-fold (c-unfold ?k (c-snd (c-snd (u-(1::nat))))) = c-snd (c-snd (u-(1::nat))) by (simp add: th-2)
from S18 S19 have S20: c-fold ls = c-snd (c-snd (u-(1::nat))) by simp
from S14 S17 S20 show ?thesis by simp
qed

lemma c-tl-aux3: c-tl u = (sgn1 ((c-len u) - 1))*((c-pair (c-len u - (2::nat)) (c-snd (c-snd (u-(1::nat))))) + 1) (is - ≠ R)
proof –
  have S1: u=0 => ?thesis by simp
  have S2: u>0 => ?thesis
  proof –
    assume A1: u>0
    have S2-1: c-len u = 1 => ?thesis by (simp add: c-tl-aux1)
    have S2-2: c-len u ≠ 1 => ?thesis
    proof –
      assume A2-2-1: c-len u ≠ 1
      from A1 have S2-2-1: c-len u > 0 by (rule c-len-3)
      from A2-2-1 S2-2-1 have S2-2-2: c-len u > 1 by arith
      from this have S2-2-3: c-len u - 1 > 0 by simp
      from this have S2-2-4: sgn1 (c-len u - 1)=1 by simp
      from S2-2-4 have S2-2-5: ?R = (c-pair (c-len u - (2::nat)) (c-snd (c-snd (u-(1::nat))))) + 1 by simp
      from S2-2-2 have S2-2-6: c-tl u = (c-pair (c-len u - (2::nat)) (c-snd (c-snd (u-(1::nat))))) + 1 by (rule c-tl-aux2)
      from S2-2-5 S2-2-6 show ?thesis by simp
    qed
    from S2-1 S2-2 show ?thesis by blast
  qed
  from S1 S2 show ?thesis by arith

45
proof

lemma c-tl-less: \( u > 0 \implies c\text{-}tl\ u < u \)

proof

assume \( A1: u > 0 \)
then have \( S1: c\text{-}tl\ u > 0 \) by (rule c-len-3)
then show \( \text{thesis} \)

proof cases

assume \( c\text{-}len\ u = 1 \)
from this \( A1 \) show \( \text{thesis} \) by (simp add: c-tl-aux1)

next

assume \( \neg\ c\text{-}len\ u = 1 \) with \( S1 \) have \( A2: c\text{-}len\ u > 1 \) by simp
then have \( S2: c\text{-}tl\ u = (\text{c\text{-}pair} (c\text{-}len\ u - (2::nat)) (c\text{-}snd (c\text{-}snd (u-(1::nat))))) \)

+ 1 by (rule c-tl-aux2)

from \( A1 \) have \( S3: c\text{-}len\ u = c\text{-}fst(u-(1::nat))+1 \) by (simp add: c-len-def)
from \( A2 \) \( S3 \) have \( S4: c\text{-}len\ u = (2::nat) < c\text{-}fst(u-(1::nat)) \) by simp
then have \( S5: (\text{c\text{-}pair} (c\text{-}len\ u - (2::nat)) (c\text{-}snd (c\text{-}snd (u-(1::nat))))) < 
\) (c-pair (c-fst(u-(1::nat))) (c-snd (c-snd (u-(1::nat)))))) by (rule c-pair-strict-mono1)

have \( S6: c\text{-}snd (c\text{-}snd (u-(1::nat))) \leq c\text{-}snd (u-(1::nat)) \) by (rule c-snd-le-arg)
then have \( S7: (\text{c\text{-}pair} (c\text{-}fst(u-(1::nat))) (c\text{-}snd (c\text{-}snd (u-(1::nat))))) \leq 
\) (c-pair (c-fst(u-(1::nat))) (c-snd (u-(1::nat)))) by (rule c-pair-mono2)
then have \( S8: (\text{c\text{-}pair} (c\text{-}fst(u-(1::nat))) (c\text{-}snd (c\text{-}snd (u-(1::nat))))) \leq 
\) u-(1::nat) by simp

with \( S5 \) have (c-pair (c-len u (2::nat)) (c-snd (c-snd (u-(1::nat)))))) < u
- (1::nat) by simp

with \( S2 \) have c-tl u < (u-(1::nat))+1 by simp

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

qed

lemma c-tl-le: c-tl u \leq u

proof (cases u)

assume u=0
then show \( \text{thesis} \) 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-tl-less)
with \( A1 \) show c-tl u \leq u by simp

qed

theorem c-tl-is-pr: c-tl \in PrimRec1

proof

have c-tl = (\lambda u. (sgn1 ((c-len u) - 1))*((c-pair (c-len u (2::nat)) (c-snd (c-snd (u-(1::nat))))) + 1)) (is - ?R) by (simp add: c-tl-aux3 ext)

moreover from c-len-is-pr c-pair-is-pr have ?R \in PrimRec1 by prec

ultimately show ?thesis by simp

qed

46
lemma c-cons-aux1: $c\text{-}cons\ x\ 0 = (c\text{-}pair\ 0\ x) + 1$
apply (unfold c-cons-def)
apply (simp)
apply (unfold list-to-nat-def)
apply (simp)
done

lemma c-cons-aux2: $u > 0 \implies c\text{-}cons\ x\ u = (c\text{-}pair\ (c\text{-}len\ u)\ (c\text{-}pair\ x\ (c\text{-}snd\ (u -(1\::\text{nat})))) + 1$
proof
  assume A1: $u > 0$
  from A1 have S1: $c\text{-}len\ u > 0$ by (rule c-len-3)
  from A1 have S2: $nat\text{-}to\text{-}list\ u = c\text{-}unfold\ (c\text{-}len\ u)\ (c\text{-}snd\ (u -(1\::\text{nat})))$ by (rule nat-to-list-of-pos)
  define ls where $ls = nat\text{-}to\text{-}list\ u$
  from ls-def S2 have S3: $ls = c\text{-}unfold\ (c\text{-}len\ u)\ (c\text{-}snd\ (u -(1\::\text{nat})))$ by simp
  from S3 have S4: $length\ ls = c\text{-}len\ u$ by (simp add: c-unfold-len)
  from S4 S1 have S5: $length\ ls > 0$ by simp
  from S5 have S6: $ls \neq []$ by simp
  from ls-def have S7: $c\text{-}cons\ x\ u = list\text{-}to\text{-}nat\ (x \#\ ls)$ by (simp add: c-cons-def)
  have S8: $list\text{-}to\text{-}nat\ (x \#\ ls) = (c\text{-}pair\ ((length\ (x\#ls)) -(1\::\text{nat})))\ (c\text{-}fold\ (x\#ls)) + 1$
by (simp add: list-to-nat-def)
  have S9: $(length\ (x\#ls))-(1\::\text{nat}) = length\ ls$ by simp
  from S9 S4 S8 have S10: $list\text{-}to\text{-}nat\ (x \#\ ls) = (c\text{-}pair\ (c\text{-}len\ u)\ (c\text{-}fold\ (x\#ls))) + 1$ by simp
  have S11: $c\text{-}fold\ (x\#ls) = c\text{-}pair\ x\ (c\text{-}snd\ (u -(1\::\text{nat})))$
proof
  from S6 have S11-1: $c\text{-}fold\ (x\#ls) = c\text{-}pair\ x\ (c\text{-}fold\ ls)$ by (rule c-fold-0)
  from S3 have S11-2: $c\text{-}fold\ ls = c\text{-}fold\ (c\text{-}unfold\ (c\text{-}len\ u)\ (c\text{-}snd\ (u -(1\::\text{nat}))))$ by simp
  from S11-2 have S11-3: $c\text{-}fold\ ls = c\text{-}snd\ (u -(1\::\text{nat}))$ by (simp add: th-2)
  from S11-1 S11-3 show ?thesis by simp
qed
from S7 S10 S11 show ?thesis by simp
qed

lemma c-cons-aux3: $c\text{-}cons\ = (\lambda x\ u.\ (sgn2\ u)*((c\text{-}pair\ 0\ x)+1) + (sgn1\ u)*((c\text{-}pair\ (c\text{-}len\ u)\ (c\text{-}pair\ x\ (c\text{-}snd\ (u -(1\::\text{nat})))))) + 1))$
proof (rule ext, rule ext)
  fix $x\ u$ show $c\text{-}cons\ x\ u = (sgn2\ u)*((c\text{-}pair\ 0\ x)+1) + (sgn1\ u)*((c\text{-}pair\ (c\text{-}len\ u)\ (c\text{-}pair\ x\ (c\text{-}snd\ (u -(1\::\text{nat})))))) + 1)$ (is - ?R)
proof cases
  assume A1: $u = 0$
  then have ?R = $(c\text{-}pair\ 0\ x)+1$ by simp
  moreover from A1 have $c\text{-}cons\ x\ u = (c\text{-}pair\ 0\ x)+1$ by (simp add: c-cons-aux1)
  ultimately show ?thesis by simp
next
  assume A1: $u \neq 0$
  then have S1: $?R = (c\text{-}pair\ (c\text{-}len\ u)\ (c\text{-}pair\ x\ (c\text{-}snd\ (u -(1\::\text{nat})))))) + 1$
by simp
from A1 have S2: c-cons x u = (c-pair (c-len u) (c-pair x (c-snd (u−(1::nat)))))
  + 1 by (simp add: c-cons-aux2)
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)
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

48
lemma c-drop-df: \( \forall \text{ls}. \text{drop} n \text{ls} = \text{nat-to-list} (\text{c-drop} n (\text{list-to-nat} \text{ls})) \)

proof (induct n)

  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 (induct n)

  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

c-nth :: nat \Rightarrow nat \Rightarrow nat where

c-nth = (\lambda x n. \text{c-hd} (\text{c-drop} n x))

lemma c-nth-is-pr: c-nth \in PrimRec2

proof (unfold c-nth-def)

  from c-hd-is-pr c-drop-is-pr show (\lambda x n. \text{c-hd} (\text{c-drop} n x)) \in PrimRec2 by prec

qed

lemma c-nth-at-0: c-nth x 0 = c-hd x by (simp add: c-nth-def)

lemma c-hd-c-cons [simp]: c-hd (c-cons x y) = x

proof

  have 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 c-tl-c-cons [simp]: c-tl (c-cons x y) = y by (simp add: c-tl-def c-cons-def)

definition

c-f-list :: (nat \Rightarrow nat \Rightarrow nat) \Rightarrow nat \Rightarrow nat \Rightarrow nat where
\[c-f\text{-}list = (\lambda f.\]
\[\text{let } g = (\%x. \text{c-cons } (f \ 0 \ x) \ 0); \ h = (\%a \ b \ c. \ \text{c-cons } (f \ (\text{Suc} \ a) \ c) \ b) \ \text{in}
\]
\[\text{PrimRecOp } g \ h\]

**lemma** $c-f\text{-}list\text{-}at-0$: $c-f\text{-}list \ f \ 0 \ x = c\text{-}cons \ (f \ 0 \ x) \ 0$ by \(\text{simp add: } c-f\text{-}list\text{-}def \ \text{Let-def}\)

**lemma** $c-f\text{-}list\text{-}at-Suc$: $c-f\text{-}list \ f \ (\text{Suc} \ y) \ x = c\text{-}cons \ (f \ (\text{Suc} \ y) \ x) \ (c-f\text{-}list \ f \ y \ x)$ by \((\text{simp add: } c-f\text{-}list\text{-}def \ \text{Let-def})\)

**lemma** $c-f\text{-}list\text{-}is-pr$: $f \in \text{PrimRec2} \implies c-f\text{-}list \ f \in \text{PrimRec2}$

**proof** –
\[\text{assume } A1: f \in \text{PrimRec2}\]
\[\text{let } ?g = (\%x. \ c\text{-}cons \ (f \ 0 \ x) \ 0)\]
\[\text{from } A1 \ c\text{-}cons\text{-}is\text{-}pr \ \text{have } S1: ?g \in \text{PrimRec1} \ \text{by prec}\]
\[\text{let } ?h = (\%a \ b \ c. \ c\text{-}cons \ (f \ (\text{Suc} \ a) \ c) \ b)\]
\[\text{from } A1 \ c\text{-}cons\text{-}is\text{-}pr \ \text{have } S2: ?h \in \text{PrimRec3} \ \text{by prec}\]
\[\text{from } S1 \ S2 \ \text{show } ?\text{thesis} \ \text{by } (\text{simp add: pr-rec } c-f\text{-}list\text{-}def \ \text{Let-def})\]

**qed**

**lemma** $c-f\text{-}list\text{-}to-f\text{-}0$: $f \ y \ x = c\text{-}hd \ (c-f\text{-}list \ f \ y \ x)$

**apply**(induct y)
**apply**(simp add: $c-f\text{-}list\text{-}at-0$)
**apply**(simp add: $c-f\text{-}list\text{-}at-Suc$)

**done**

**lemma** $c-f\text{-}list\text{-}to-f$: $f = (\lambda y \ x. \ c\text{-}hd \ (c-f\text{-}list \ f \ y \ x))$

**apply**(rule ext, rule ext)
**apply**(rule $c-f\text{-}list\text{-}to-f\text{-}0$)

**done**

**lemma** $c-f\text{-}list\text{-}f\text{-}is\text{-}pr$: $c-f\text{-}list \ f \in \text{PrimRec2} \implies f \in \text{PrimRec2}$

**proof** –
\[\text{assume } A1: c-f\text{-}list \ f \in \text{PrimRec2}\]
\[\text{have } S1: f = (\lambda y \ x. \ c\text{-}hd \ (c-f\text{-}list \ f \ y \ x)) \ \text{by } (\text{rule } c-f\text{-}list\text{-}to-f)\]
\[\text{from } A1 \ c\text{-}hd\text{-}is\text{-}pr \ \text{have } S2: (\lambda y \ x. \ c\text{-}hd \ (c-f\text{-}list \ f \ y \ x)) \in \text{PrimRec2} \ \text{by prec}\]
\[\text{with } S1 \ \text{show } ?\text{thesis} \ \text{by simp}\]

**qed**

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

**lemma** $c-f\text{-}list\text{-}lm\text{-}2$: $z < \text{Suc} \ n \implies c\text{-}nth \ (c-f\text{-}list \ f \ (\text{Suc} \ n) \ x) \ (\text{Suc} \ n - z) = c\text{-}nth \ (c-f\text{-}list \ f \ n \ x) \ (n - z)$

**proof** –
\[\text{assume } z < \text{Suc} \ n\]
\[\text{then have } \text{Suc} \ n - z = \text{Suc} \ (n - z) \ \text{by arith}\]
\[\text{then have } c\text{-}nth \ (c-f\text{-}list \ f \ (\text{Suc} \ n) \ x) \ (\text{Suc} \ n - z) = c\text{-}nth \ (c-f\text{-}list \ f \ (\text{Suc} \ n) \ x) \ (\text{Suc} \ (n - z)) \ \text{by simp}\]
also have \ldots \Rightarrow c\text{-nth} \ (c\text{-cons} \ (f \ (Suc \ n) \ x) \ (c\text{-f-list} \ f \ n \ x)) \ (Suc \ (n - z)) \ \text{by simp add: c-f-list-at-Suc}
also have \ldots \Rightarrow c\text{-nth} \ (c\text{-f-list} \ f \ n \ x) \ (n - z) \ \text{by simp add: c-f-list-lm-1}
finally show \ ?thesis \ \text{by simp}

qed

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

show \ z \leq 0 \Longrightarrow c\text{-nth} \ (c\text{-f-list} \ f \ 0 \ x) \ (0 - z) = f \ z \ x
proof

assume \ z \leq 0 \ \text{then have } A1: \ z = 0 \ \text{by simp}
then have c\text{-nth} \ (c\text{-f-list} \ f \ 0 \ x) \ (0 - z) = c\text{-nth} \ (c\text{-f-list} \ f \ 0 \ x) \ 0 \ \text{by simp}
also have \ \ldots = c\text{-hd} \ (c\text{-f-list} \ f \ 0 \ x) \ \text{by (simp add: c-nth-at-0)}
also have \ \ldots = c\text{-hd} \ (c\text{-cons} \ (f \ 0 \ x) \ 0) \ \text{by (simp add: c-f-list-at-0)}
also have \ \ldots = f \ 0 \ x \ \text{by simp}
finally show c\text{-nth} \ (c\text{-f-list} \ f \ 0 \ x) \ (0 - z) = f \ z \ x \ \text{by (simp add: A1)}

qed

next

fix \ n \ \text{assume } A2: \ z \leq n \ \text{\Rightarrow c\text{-nth} \ (c\text{-f-list} \ f \ n \ x) \ (n - z) = f \ z \ x} \ \text{show } z \leq Suc \ n \ \text{\Rightarrow c\text{-nth} \ (c\text{-f-list} \ f \ (Suc \ n) \ x) \ (Suc \ n - z) = f \ z \ x}

proof

assume A3: \ z \leq Suc \ n
show \ z \leq Suc \ n \ \Rightarrow c\text{-nth} \ (c\text{-f-list} \ f \ (Suc \ n) \ x) \ (Suc \ n - z) = f \ z \ x
proof cases

assume AA1: \ z \leq n
then have AA2: \ z < Suc \ n \ \text{by simp}
from A2 this have S1: \ c\text{-nth} \ (c\text{-f-list} \ f \ n \ x) \ (n - z) = f \ z \ x \ \text{by auto}
from AA2 have \ c\text{-nth} \ (c\text{-f-list} \ f \ (Suc \ n) \ x) \ (Suc \ n - z) = c\text{-nth} \ (c\text{-f-list} \ f \ n \ x) \ (n - z) \ \text{by (rule c-f-list-lm-2)}
with S1 show \ c\text{-nth} \ (c\text{-f-list} \ f \ (Suc \ n) \ x) \ (Suc \ n - z) = f \ z \ x \ \text{by simp}

next

assume \ \neg \ z \leq n
from A3 this have S1: \ z = Suc \ n \ \text{by simp}
then have S2: \ Suc \ n - z = 0 \ \text{by simp}
then have c\text{-nth} \ (c\text{-f-list} \ f \ (Suc \ n) \ x) \ (Suc \ n - z) = c\text{-nth} \ (c\text{-f-list} \ f \ (Suc \ n) \ x) \ 0 \ \text{by simp}
also have \ \ldots = c\text{-hd} \ (c\text{-f-list} \ f \ (Suc \ n) \ x) \ \text{by (simp add: c-nth-at-0)}
also have \ \ldots = c\text{-hd} \ (c\text{-cons} \ (f \ (Suc \ n) \ x) \ (c\text{-f-list} \ f \ n \ x)) \ \text{by (simp add: c-f-list-at-Suc)}
also have \ \ldots = f \ (Suc \ n) \ x \ \text{by simp}
finally show \ c\text{-nth} \ (c\text{-f-list} \ f \ (Suc \ n) \ x) \ (Suc \ n - z) = f \ z \ x \ \text{by (simp add: S1)}

qed

qed

theorem th-pr-rec: \ [ \ g \in PrimRec1; \ h \in PrimRec3; \ (\forall x. \ (f \ 0 \ x) = (g \ x)); \ (\forall x y. \ (f \ (Suc \ y) \ x) = h \ y \ (f \ y \ x) \ x) ] \Longrightarrow f \in PrimRec2
proof –
assume $g$ is-pr: \( g \in \text{PrimRec}1 \)
assume $h$ is-pr: \( h \in \text{PrimRec}3 \)
assume $f$-0: \( \forall x. f \ 0 \ x = g \ x \)
assume $f$-1: \( \forall x \ y. (f \ (\text{Suc} \ y) \ x) = h \ y \ (f \ y \ x) \ x \)
let \(?f = \text{PrimRecOp} \ g \ h\)
from $g$ is-pr $h$ is-pr have $S1$: \(?f \in \text{PrimRec}2\) by (rule pr-rec)
have $f$-2: \( \forall x. (?f \ 0 \ x) = g \ x \) by simp
have $f$-3: \( \forall x \ y. (?f \ (\text{Suc} \ y) \ x) = h \ y \ (?f \ y \ x) \ x \) by simp
have $S2$: \( f = ?f \)
proof
  have \( \lambda x \ y. f \ y \ x \ x \leq y \) by simp
  apply (induct-tac $y$)
  apply (insert $f$-0 $f$-1)
  apply (auto)
  done
  then show \( f = ?f \) by (simp add: ext)
qed
from $S1$ $S2$ show \(?thesis\) by simp
qed

theorem thr-rec: \( [g \in \text{PrimRec}1; \alpha \in \text{PrimRec}2; h \in \text{PrimRec}3; (\forall x \ y. \alpha \ x \ y \leq y); (\forall x. (f \ 0 \ x) = (g \ x)); (\forall x \ y. (f \ (\text{Suc} \ y) \ x) = h \ y \ (f \ (\alpha \ y \ x) \ x) \ x)] \implies f \in \text{PrimRec}2\)
proof
  assume $g$ is-pr: \( g \in \text{PrimRec}1 \)
  assume $\alpha$ is-pr: \( \alpha \in \text{PrimRec}2 \)
  assume $h$ is-pr: \( h \in \text{PrimRec}3 \)
  assume $a$ le: \( \forall x. \alpha \ x \ y \leq y \)
  assume $f$-0: \( \forall x. f \ 0 \ x = g \ x \)
  assume $f$-1: \( \forall x \ y. (f \ (\text{Suc} \ y) \ x) = h \ y \ (f \ (\alpha \ y \ x) \ x) \ x \)
  let \(?g' = \lambda x. \alpha \ x \ c\text{-cons} \ (g \ x) \ 0\)
  let \(?h' = \lambda a \ b \ c. \alpha \ x \ c\text{-cons} \ (h \ a \ (\alpha \ a \ c)) \ b \)
  let \(?r = \text{c-f-list} \ f\)
  from $g$ is-pr $\alpha$ is-pr $\text{c-nth-is-pr}$ have \(?g' \in \text{PrimRec}1\) by prec
  from $h$ is-pr $\alpha$ is-pr $\text{c-nth-is-pr}$ $a$ is-pr have \(?h' \in \text{PrimRec}3\) by prec
  have $S1$: \( \forall x. \ ?r \ 0 \ x = ?g' \ x \)
  proof
    fix $x$ have \(?r \ 0 \ x = \alpha \ x \ c\text{-cons} \ (f \ 0 \ x) \ 0\) by (rule c-f-list-at-0)
    with $f$-0 have \(?r \ 0 \ x = \alpha \ x \ c\text{-cons} \ (g \ x) \ 0\) by simp
    then show \(?r \ 0 \ x = ?g' \ x\) by simp
  qed
  have $S2$: \( \forall x \ y. \ ?r \ (\text{Suc} \ y) \ x = ?h' \ y \ (?r \ y \ x) \ x \)
  proof (rule allI, rule allI)
    fix $x \ y$ show \(?r \ (\text{Suc} \ y) \ x = ?h' \ y \ (?r \ y \ x) \ x\)
    proof
      have $S2$-1: \( ?r \ (\text{Suc} \ y) \ x = \alpha \ x \ y \ ?r \ y \ x \ x\) by (rule c-f-list-at-Suc)
      with $f$-1 have $S2$-2: \( f \ (\text{Suc} \ y) \ x = h \ y \ (f \ (\alpha \ y \ x) \ x) \ x\) by simp
      from $a$ le have $S2$-3: \( \alpha \ y \ x \ leq y\) by simp
  qed

52
then have \( S2-4: f (\alpha y x) x = c-nth (\?r y x) (y - (\alpha y x)) \) by (simp add: c-f-list-nth)

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

qed

from \( g'\)-is-pr \( h'\)-is-pr \( S1 \) \( S2 \) have \( S3: \?r \in \text{PrimRec2} \) by (rule th-pr-rec)
then show \( f \in \text{PrimRec2} \) by (rule c-f-list-f-is-pr)

qed

declare c-tl-less [termination-simp]

fun c-assoc-have-key :: nat \Rightarrow nat \Rightarrow nat where
  c-assoc-have-key-df [simp del]: c-assoc-have-key y x = (if c-fst (c-hd y) = x then 0 else c-assoc-have-key (c-tl y) x))

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

theorem c-assoc-have-key-is-pr: c-assoc-have-key \( \in \) PrimRec2
proof
  let \( \lambda a b c. \) if c-fst (c-hd (Suc a)) = c then 0 else b
  let \( \lambda y x. c\text{-tl} (\text{Suc} y) \)
  let \( \lambda y x. (\text{Suc} :: \text{nat}) \)

  have g-is-pr: \( \lambda y. \) PrimRec1 by (rule const-is-pr)
  from c-tl-is-pr have a-is-pr: \( \lambda a. \) PrimRec2 by prec
  have h-is-pr: \( \lambda h. \) PrimRec3
  proof (rule if-eq-is-pr3)
    from c-fst-is-pr c-hd-is-pr show (\( \lambda x y z. c\text{-fst} (c\text{-hd} (\text{Suc} x)) \)) \( \in \) PrimRec3 by prec
  next
    show (\( \lambda x y z. z \)) \( \in \) PrimRec3 by (rule pr-id3-3)
  next
    show (\( \lambda x y. 0 \)) \( \in \) PrimRec3 by prec
  next
    show (\( \lambda x y z. y \)) \( \in \) PrimRec3 by (rule pr-id3-2)
  qed

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

  have f-0: \( \forall x. c\text{-assoc-have-key} 0 x = \lambda g x \) by (simp add: c-assoc-have-key-df)
  have f-1: \( \forall x y. c\text{-assoc-have-key} (\text{Suc} y) x = \lambda h y (c\text{-assoc-have-key} (\lambda a y x) x) x \) by (simp add: c-assoc-have-key-df)
  from g-is-pr a-is-pr h-is-pr a-le f-0 f-1 show \( \text{thesis} \) by (rule th-rec)
fun c-assoc-value :: nat ⇒ nat ⇒ nat where
c-assoc-value-df [simp del]: c-assoc-value y x = (if y = 0 then 0 else
t (=) (c-fst (c-hd y) = x then c-snd (c-hd y) else c-assoc-value (c-tl y) x))

lemma c-assoc-value-lm-1: y ≠ 0 ⇒ c-assoc-value y x = (if c-fst (c-hd y) = x
then c-snd (c-hd y) else c-assoc-value (c-tl y) x) by (simp add: c-assoc-value-df)

theorem c-assoc-value-is-pr: c-assoc-value ∈ PrimRec2
proof –
  let ?h = λ a b c. if c-fst (c-hd (Suc a)) = c then c-snd (c-hd (Suc a)) else b
  let ?a = λ y x. c-tl (Suc y)
  let ?g = λ x. (0 :: 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
  from c-fst-is-pr c-hd-is-pr have h-is-pr: ?h ∈ PrimRec3
  have h-is-pr: ?g ∈ PrimRec1 by (rule const-is-pr)
  next
  from c-fst-is-pr c-hd-is-pr show (λ x y z. c-fst (c-hd (Suc x))) ∈ PrimRec3 by prec
  next
  from c-snd-is-pr c-hd-is-pr show (λ x y z. c-snd (c-hd (Suc x))) ∈ PrimRec3
  by prec
  next
  from c-assoc-value-lm-1 have a-le: ∀ x y. ?a y x ≤ y
  proof (rule allI, rule allI)
    fix x y show ?a y x ≤ y
    proof –
      have Suc y > 0 by simp
      then have ?a y x < Suc y by (rule c-tl-less)
    then show ?thesis by simp
    qed
    qed
    qed
    have f-0: ∀ x. c-assoc-value 0 x = ?g x by (simp add: c-assoc-value-df)
    have f-1: ∀ x y. c-assoc-value (Suc y) x = ?h y (c-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 ?thesis by (rule th-rec)
    qed
    lemma c-assoc-lm-1: c-assoc-have-key (c-cons (c-pair x y) z) x = 0
    apply (simp add: c-assoc-have-key-df)
    apply (simp add: c-assoc-have-key-df)
    done
    lemma c-assoc-lm-2: c-assoc-value (c-cons (c-pair x y) z) x = y

54
apply(simp add: c-assoc-value-df)
apply(rule impI)
apply(insert c-cons-pos [where x=(c-pair x y) and u=z])
apply(auto)
done

lemma c-assoc-lm-3: \( x_1 \neq x \implies c\text{-}assoc\text{-}have\text{-}key \ (c\text{-}cons \ (c\text{-}pair \ x \ y) \ z) \ x_1 = c\text{-}assoc\text{-}have\text{-}key \ z \ x_1 \)
proof -
  assume A1: \( x_1 \neq x \)
  let \( \vartheta\text{-}ls = (c\text{-}cons \ (c\text{-}pair \ x \ y) \ z) \)
  have S1: \( \vartheta\text{-}ls \neq 0 \) by (simp add: c-cons-pos)
  then have S2: c-assoc-have-key ?ls x1 = (if c-fst (c-hd ?ls) = x1 then 0 else c-assoc-have-key (c-tl ?ls) x1) (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 S4 have S5: ?R = c-assoc-have-key (c-tl ?ls) x1 by (rule if-not-P)
  from S2 S5 show ?thesis by simp
qed

lemma c-assoc-lm-4: \( x_1 \neq x \implies c\text{-}assoc\text{-}value \ (c\text{-}cons \ (c\text{-}pair \ x \ y) \ z) \ x_1 = c\text{-}assoc\text{-}value \ z \ x_1 \)
proof -
  assume A1: \( x_1 \neq x \)
  let \( \vartheta\text{-}ls = (c\text{-}cons \ (c\text{-}pair \ x \ y) \ z) \)
  have S1: \( \vartheta\text{-}ls \neq 0 \) by (simp add: c-cons-pos)
  then have S2: c-assoc-value ?ls x1 = (if c-fst (c-hd ?ls) = x1 then c-snd (c-hd ?ls) else c-assoc-value (c-tl ?ls) x1) (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 S4 have S5: ?R = c-assoc-value (c-tl ?ls) x1 by (rule if-not-P)
  from S2 S5 show ?thesis by simp
qed

end

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. pr-conv-2-to-1 (PrimRecOp g (pr-conv-1-to-3 h)))"

55
lemma unary-rec-into-pr: \[ g \in \text{PrimRec}1; h \in \text{PrimRec}1 \implies \text{UnaryRecOp} g h \in \text{PrimRec}1 \] 
by (simp add: UnaryRecOp-def pr-conv-1-to-3-lm pr-conv-2-to-1-lm pr-rec)

definition
c-f-pair :: (nat \Rightarrow nat) \Rightarrow (nat \Rightarrow nat) \Rightarrow (nat \Rightarrow nat) where
c-f-pair = (\lambda f g x. c-pair (f x) (g x))

lemma c-f-pair-to-pr: \[ f \in \text{PrimRec}1; g \in \text{PrimRec}1 \implies c-f-pair f g \in \text{PrimRec}1 \]
unfolding c-f-pair-def by prec

inductive-set PrimRec1′ :: (nat \Rightarrow nat) set
where
zero: (\lambda x. 0) \in \text{PrimRec}1′
suc: Suc \in \text{PrimRec}1′
fst: c-fst \in \text{PrimRec}1′
snd: c-snd \in \text{PrimRec}1′
comp: \[ f \in \text{PrimRec}1′; g \in \text{PrimRec}1′ \implies (\lambda x. f (g x)) \in \text{PrimRec}1′ \]
pair: \[ f \in \text{PrimRec}1′; g \in \text{PrimRec}1′ \implies c-f-pair f g \in \text{PrimRec}1′ \]
un-rec: \[ f \in \text{PrimRec}1′; g \in \text{PrimRec}1′ \implies \text{UnaryRecOp} f g \in \text{PrimRec}1′ \]

lemma primrec'-into-primrec: f \in \text{PrimRec}1′ \implies f \in \text{PrimRec}1
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′: (\lambda x. x) \in \text{PrimRec}1′
proof
  have c-f-pair c-fst c-snd \in \text{PrimRec}1′ 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′: pr-conv-2-to-1 (\lambda x y. x) \in \text{PrimRec}1′ by (simp add: pr-conv-2-to-1-def PrimRec1′.fst)

56
lemma pr-id2-2: \( pr\text{-conv-2-to-1} (\lambda x y. y) \in \text{PrimRec1}' \) by (simp add: pr\text{-conv-2-to-1-def PrimRec1}'.snd)

lemma pr-id3-1: \( pr\text{-conv-3-to-1} (\lambda x y z. x) \in \text{PrimRec1}' \)
proof
  have \( pr\text{-conv-3-to-1} (\lambda x y z. x) = (\lambda x. \text{c-fst} (\text{c-fst} x)) \) by (simp add: pr\text{-conv-3-to-1-def})
  moreover from \( \text{PrimRec1}'.\text{fst} \text{PrimRec1}'.\text{fst} \) have \( (\lambda x. \text{c-fst} (\text{c-fst} x)) \in \text{PrimRec1}' \)
  ultimately show \( ?\text{thesis} \) by simp
qed

lemma pr-id3-2: \( pr\text{-conv-3-to-1} (\lambda x y z. y) \in \text{PrimRec1}' \)
proof
  have \( pr\text{-conv-3-to-1} (\lambda x y z. y) = (\lambda x. \text{c-snd} (\text{c-fst} x)) \) by (simp add: pr\text{-conv-3-to-1-def})
  moreover from \( \text{PrimRec1}'.\text{snd} \text{PrimRec1}'.\text{fst} \) have \( (\lambda x. \text{c-snd} (\text{c-fst} x)) \in \text{PrimRec1}' \)
  ultimately show \( ?\text{thesis} \) by simp
qed

lemma pr-id3-3: \( pr\text{-conv-3-to-1} (\lambda x y z. z) \in \text{PrimRec1}' \)
proof
  have \( pr\text{-conv-3-to-1} (\lambda x y z. z) = (\lambda x. \text{c-snd} x) \) by (simp add: pr\text{-conv-3-to-1-def})
  thus \( ?\text{thesis} \) by (simp add: PrimRec1'.snd)
qed

lemma pr-comp2-1: \( \square (pr\text{-conv-2-to-1} f \in \text{PrimRec1}'; g \in \text{PrimRec1}'; h \in \text{PrimRec1}' \implies (\lambda x. f (g x) (h x)) \in \text{PrimRec1}' \)
proof
  assume A1: \( pr\text{-conv-2-to-1} f \in \text{PrimRec1}' \)
  assume A2: \( g \in \text{PrimRec1}' \)
  assume A3: \( h \in \text{PrimRec1}' \)
  let \( ?f1 = pr\text{-conv-2-to-1} f \)
  have S1: \( (\%x. ?f1 ((\text{c-fpair} g h) x)) = (\lambda x. f (g x) (h x)) \) by (simp add: c-fpair-def pr\text{-conv-2-to-1-def})
  from A2 A3 have S2: \( \text{c-fpair} g h \in \text{PrimRec1}' \) by (rule PrimRec1'.pair)
  from A1 S2 have S3: \( (\%x. ?f1 ((\text{c-fpair} g h) x)) \in \text{PrimRec1}' \) by (rule PrimRec1'.comp)
  with S1 show \( ?\text{thesis} \) by simp
qed

lemma pr-comp3-1: \( \square (pr\text{-conv-3-to-1} f \in \text{PrimRec1}'; g \in \text{PrimRec1}'; h \in \text{PrimRec1}' \implies (\lambda x. f (g x) (h x) (k x)) \in \text{PrimRec1}' \)
proof
  assume A1: \( pr\text{-conv-3-to-1} f \in \text{PrimRec1}' \)
  assume A2: \( g \in \text{PrimRec1}' \)
  assume A3: \( h \in \text{PrimRec1}' \)
  assume A4: \( k \in \text{PrimRec1}' \)
  from A2 A3 have \( \text{c-fpair} g h \in \text{PrimRec1}' \) by (rule PrimRec1'.pair)
  from this A4 have \( \text{c-fpair} (\text{c-fpair} g h) k \in \text{PrimRec1}' \) by (rule PrimRec1'.pair)
from A1 this have (%x. pr-conv-3-to-1 f) ((c-f-pair (c-f-pair g h) k) x) ∈ PrimRec1’ by (rule PrimRec1’.comp)
then show ?thesis by (simp add: c-f-pair-def pr-conv-3-to-1-def)
qed

lemma pr-comp1-2’: [ f ∈ PrimRec1’; pr-conv-2-to-1 g ∈ PrimRec1’ ] → pr-conv-2-to-1 (λ x y. f (g x y)) ∈ PrimRec1’
proof –
assume f ∈ PrimRec1’
and pr-conv-2-to-1 g ∈ PrimRec1’ (is ?g1 ∈ PrimRec1’)
then have (λ x. f (?g1 x)) ∈ PrimRec1’ by (rule PrimRec1’.comp)
then show ?thesis by (simp add: pr-conv-2-to-1-def)
qed

lemma pr-comp1-3’: [ f ∈ PrimRec1’; pr-conv-3-to-1 g ∈ PrimRec1’ ] → pr-conv-3-to-1 (λ x y z. f (g x y z)) ∈ PrimRec1’
proof –
assume f ∈ PrimRec1’
and pr-conv-3-to-1 g ∈ PrimRec1’ (is ?g1 ∈ PrimRec1’)
then have (λ x. f (?g1 x)) ∈ PrimRec1’ by (rule PrimRec1’.comp)
then show ?thesis by (simp add: pr-conv-3-to-1-def)
qed

lemma pr-comp2-2’: [ pr-conv-2-to-1 f ∈ PrimRec1’; pr-conv-2-to-1 g ∈ PrimRec1’; pr-conv-2-to-1 h ∈ PrimRec1’ ] → pr-conv-2-to-1 (λ x y. f (g x y) (h x y)) ∈ PrimRec1’
proof –
assume pr-conv-2-to-1 f ∈ PrimRec1’
and pr-conv-2-to-1 g ∈ PrimRec1’ (is ?g1 ∈ PrimRec1’)
and pr-conv-2-to-1 h ∈ PrimRec1’ (is ?h1 ∈ PrimRec1’)
then have (λ x. f (?g1 x) (?h1 x)) ∈ PrimRec1’ by (rule pr-comp2-1’)
then show ?thesis by (simp add: pr-conv-2-to-1-def)
qed

lemma pr-comp2-3’: [ pr-conv-2-to-1 f ∈ PrimRec1’; pr-conv-3-to-1 g ∈ PrimRec1’; pr-conv-3-to-1 h ∈ PrimRec1’ ] → pr-conv-3-to-1 (λ x y z. f (g x y z) (h x y z)) ∈ PrimRec1’
proof –
assume pr-conv-2-to-1 f ∈ PrimRec1’
and pr-conv-3-to-1 g ∈ PrimRec1’ (is ?g1 ∈ PrimRec1’)
and pr-conv-3-to-1 h ∈ PrimRec1’ (is ?h1 ∈ PrimRec1’)
then have (λ x. f (?g1 x) (?h1 x)) ∈ PrimRec1’ by (rule pr-comp2-1’)
then show ?thesis by (simp add: pr-conv-3-to-1-def)
qed

lemma pr-comp3-2’: [ pr-conv-3-to-1 f ∈ PrimRec1’; pr-conv-2-to-1 g ∈ PrimRec1’; pr-conv-2-to-1 h ∈ PrimRec1’; pr-conv-2-to-1 k ∈ PrimRec1’ ] → pr-conv-2-to-1 (λ x y. f (g x y) (h x y) (k x y)) ∈ PrimRec1’
proof –
assume \(pr\text{-conv-3-to-1} f \in \text{PrimRec1}'\)
and \(pr\text{-conv-2-to-1} g \in \text{PrimRec1}' \text{ (is } ?g1 \in \text{PrimRec1}')\)
and \(pr\text{-conv-2-to-1} h \in \text{PrimRec1}' \text{ (is } ?h1 \in \text{PrimRec1}')\)
and \(pr\text{-conv-2-to-1} k \in \text{PrimRec1}' \text{ (is } ?k1 \in \text{PrimRec1}')\)
then have \((\lambda x. f (\?g1 x) (\?h1 x) (\?k1 x)) \in \text{PrimRec1}' \text{ by (rule } pr\text{-comp3-1}')\)
then show \(!\text{thesis} \text{ by (simp add: } pr\text{-conv-2-to-1-def})\)
qed

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

lemma \(lm\); \((f1 \in \text{PrimRec1} \rightarrow f1 \in \text{PrimRec1}') \land (g1 \in \text{PrimRec2} \rightarrow \text{pr\text{-conv-2-to-1} } g1 \in \text{PrimRec1}') \land (h1 \in \text{PrimRec3} \rightarrow \text{pr\text{-conv-3-to-1} } h1 \in \text{PrimRec1}')\)
proof (induct rule: \text{PrimRec1-PrimRec2-PrimRec3.induct})

  case zero show \(!\text{case} \text{ by (rule } \text{PrimRec1}.zero)\)
next case suc show \(!\text{case} \text{ by (rule } \text{PrimRec1}.suc)\)
next case id1-1 show \(!\text{case} \text{ by (rule } pr\text{-id1-1}')\)
next case id2-1 show \(!\text{case} \text{ by (rule } pr\text{-id2-1}')\)
next case id3-2 show \(!\text{case} \text{ by (rule } pr\text{-id3-2}')\)
next case id3-1 show \(!\text{case} \text{ by (rule } pr\text{-id3-1}')\)
next case id3-2 show \(!\text{case} \text{ by (rule } pr\text{-id3-2}')\)
next case id3-3 show \(!\text{case} \text{ by (rule } pr\text{-id3-3}')\)
next case comp1-1 from comp1-1 show \(!\text{case} \text{ by (simp add: } \text{PrimRec1}'.comp)\)
next case comp1-2 from comp1-2 show \(!\text{case} \text{ by (simp add: } pr\text{-comp1-2}')\)
next case comp1-3 from comp1-3 show \(!\text{case} \text{ by (simp add: } pr\text{-comp1-3}')\)
next case comp2-1 from comp2-1 show \(!\text{case} \text{ by (simp add: } pr\text{-comp2-1}')\)
next case comp2-2 from comp2-2 show \(!\text{case} \text{ by (simp add: } pr\text{-comp2-2}')\)
next case comp3-2 from comp3-2 show \(!\text{case} \text{ by (simp add: } pr\text{-comp3-2}')\)
next case comp3-1 from comp3-1 show \(!\text{case} \text{ by (simp add: } pr\text{-comp3-1}')\)
next case comp3-2 from comp3-2 show \(!\text{case} \text{ by (simp add: } pr\text{-comp3-2}')\)
next case comp3-3 from comp3-3 show \(!\text{case} \text{ by (simp add: } pr\text{-comp3-3}')\)
next case prim-rec

  fix \(g h\) assume \(A1: g \in \text{PrimRec1}' \text{ and } pr\text{-conv-3-to-1} h \in \text{PrimRec1}'\)
  then have \(\text{UnaryRecOp g (pr\text{-conv-3-to-1} h) \in PrimRec1}' \text{ by (rule } \text{PrimRec1}.un-rec)\)
moreover have \(\text{UnaryRecOp g (pr\text{-conv-3-to-1} h) = pr\text{-conv-2-to-1} (PrimRecOp g h) \text{ by (simp add: } \text{UnaryRecOp-def})}\)
ultimately show \(pr\text{-conv-2-to-1} (PrimRecOp g h) \in PrimRec1}' \text{ by simp}\nqed
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 ⇒ (nat ⇒ nat)
where
| sch-to-pr Base-zero = (λ 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 (Comp-op t1 t2) = (λ x. (sch-to-pr t1) ((sch-to-pr t2) x))
| sch-to-pr (Pair-op t1 t2) = c-f-pair (sch-to-pr t1) (sch-to-pr t2)
| sch-to-pr (Rec-op t1 t2) = UnaryRecOp (sch-to-pr t1) (sch-to-pr t2)

lemma sch-to-pr-into-pr: sch-to-pr sch ∈ PrimRec1 by (simp add: pr-1-eq-1', induct sch, simp-all add: PrimRec1'.induct)

lemma sch-to-pr-srj: f ∈ PrimRec1 ⇒ (∃ sch. f = sch-to-pr sch)
proof –
assume f ∈ PrimRec1 then have A1: f ∈ PrimRec1' by (simp add: pr-1-eq-1')
from A1 show ?thesis
proof (induct f rule: PrimRec1'.induct)
have (λ x. 0) = sch-to-pr Base-zero by simp
then show ∃ sch. (λu. 0) = sch-to-pr sch by (rule exI)
next
have Suc = sch-to-pr Base-suc by simp
then show ∃ sch. Suc = sch-to-pr sch by (rule exI)
next
have c-fst = sch-to-pr Base-fst by simp
then show ∃ sch. c-fst = sch-to-pr sch by (rule exI)
next
have c-snd = sch-to-pr Base-snd by simp
then show ∃ sch. c-snd = sch-to-pr sch by (rule exI)
next
fix f1 f2 assume B1: ∃ sch. f1 = sch-to-pr sch and B2: ∃ 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 $S_1 S_2$ have $(\lambda x. f_1 (f_2 x)) = \text{sch-to-pr} (\text{Comp-op } \text{sch1 } \text{sch2})$ by simp
then show $\exists \text{sch. } (\lambda x. f_1 (f_2 x)) = \text{sch-to-pr } \text{sch}$ by (rule exI)

next
fix $f_1 f_2$ assume $B_1: \exists \text{sch. } f_1 = \text{sch-to-pr } \text{sch}$ and $B_2: \exists \text{sch. } f_2 = \text{sch-to-pr } \text{sch}$

from $B_1$ obtain $\text{sch1}$ where $S_1: f_1 = \text{sch-to-pr } \text{sch1}$ ..
from $B_2$ obtain $\text{sch2}$ where $S_2: f_2 = \text{sch-to-pr } \text{sch2}$ ..
from $S_1 S_2$ have $c\text{-f-pair } f_1 f_2 = \text{sch-to-pr} (\text{Pair-op } \text{sch1 } \text{sch2})$ by simp
then show $\exists \text{sch. } c\text{-f-pair } f_1 f_2 = \text{sch-to-pr } \text{sch}$ by (rule exI)

next
fix $f_1 f_2$ assume $B_1: \exists \text{sch. } f_1 = \text{sch-to-pr } \text{sch}$ and $B_2: \exists \text{sch. } f_2 = \text{sch-to-pr } \text{sch}$

from $B_1$ obtain $\text{sch1}$ where $S_1: f_1 = \text{sch-to-pr } \text{sch1}$ ..
from $B_2$ obtain $\text{sch2}$ where $S_2: f_2 = \text{sch-to-pr } \text{sch2}$ ..
from $S_1 S_2$ have $\text{UnaryRecOp } f_1 f_2 = \text{sch-to-pr} (\text{Rec-op } \text{sch1 } \text{sch2})$ by simp
then show $\exists \text{sch. } \text{UnaryRecOp } f_1 f_2 = \text{sch-to-pr } \text{sch}$ by (rule exI)

qed

definition $\text{loc-f} :: \text{natt } \Rightarrow \text{PrimScheme } \Rightarrow \text{PrimScheme } \Rightarrow \text{PrimScheme }$ where
$\text{loc-f } n \text{ sch1 sch2 } =$
(if $n=0$ then $\text{Base-zero}$ else
  if $n=1$ then $\text{Base-suc}$ else
    if $n=2$ then $\text{Base-fst}$ else
      if $n=3$ then $\text{Base-snd}$ else
        if $n=4$ then $(\text{Comp-op } \text{sch1 } \text{sch2})$ else
          if $n=5$ then $(\text{Pair-op } \text{sch1 } \text{sch2})$ else
            if $n=6$ then $(\text{Rec-op } \text{sch1 } \text{sch2})$ else
              $\text{Base-zero}$


definition $\text{mod7} :: \text{natt } \Rightarrow \text{natt }$ where
$\text{mod7 } = (\lambda x. x \mod 7)$

lemma $c\text{-snd-snd-lt} [\text{termination-simp}]: c\text{-snd } (c\text{-snd } (\text{Suc } (\text{Suc } x))) < \text{Suc } (\text{Suc } x)$
proof −
let $?y = \text{Suc } (\text{Suc } x)$
  have $?y > 1$ by simp
then have $c\text{-snd } ?y < ?y$ by (rule c-snd-less-arg)
moreover have $c\text{-snd } (c\text{-snd } ?y) \leq c\text{-snd } ?y$ by (rule c-snd-le-arg)
ultimately show $?\text{thesis}$ by simp
qed

lemma $c\text{-fst-snd-lt} [\text{termination-simp}]: c\text{-fst } (c\text{-snd } (\text{Suc } (\text{Suc } x))) < \text{Suc } (\text{Suc } x)$
proof −
let $?y = \text{Suc } (\text{Suc } x)$
  have $?y > 1$ by simp

61
then have \( c\text{-snd } ?y < ?y \) by (rule \( c\text{-snd-less-arg} \))
moreover have \( c\text{-fst } (c\text{-snd } ?y) \leq c\text{-snd } ?y \) by (rule \( c\text{-fst-le-arg} \))
ultimately show \(?\text{thesis by simp}\)

qed

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

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

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

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

proof –
assume \( A1: x > 1 \)
let \( ?y = x-(2::\text{nat}) \)
from \( A1 \) have \( S1: x = \text{Suc } (\text{Suc } ?y) \) by arith

have \( S2: \text{nat-to-sch } (\text{Suc } (\text{Suc } ?y)) = (\text{let } u=\text{mod7 } (c\text{-fst } (\text{Suc } (\text{Suc } ?y))); v=c\text{-snd } (\text{Suc } (\text{Suc } ?y)); v1=c\text{-fst } v; v2 = c\text{-snd } v; \text{sch1=nat-to-sch } v1; \text{sch2=nat-to-sch } v2 \text{ in loc-f } u \text{ sch1 sch2}) \) by (rule \( \text{loc-srj-lm-1} \))

from \( S1 \) \( S2 \) show \(?\text{thesis by simp}\)
qed

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

proof –
let \( ?x = \text{c-pair } 1 \ 0 \)
have \( S1: \ ?x = 2 \) by (simp add: \( \text{c-pair-def sf-def} \))
then have \( S2: \ ?x = \text{Suc } (\text{Suc } 0) \) by simp
let \( ?y = \text{Suc } (\text{Suc } 0) \)
have \( S3: \text{nat-to-sch } ?y = (\text{let } u=\text{mod7 } (c\text{-fst } ?y); v=c\text{-snd } ?y; v1=c\text{-fst } v; v2 = c\text{-snd } v; \text{sch1=nat-to-sch } v1; \text{sch2=nat-to-sch } v2 \text{ in loc-f } u \text{ sch1 sch2}) \) (is - = \( ?R \))
by (rule \( \text{loc-srj-lm-1} \))

have \( S4: c\text{-fst } ?y = 1 \)

proof –
from \( S2 \) have \( c\text{-fst } ?y = c\text{-fst } ?x \) by simp
then show \(\text{thesis by simp}\)

qed

have \(S5: \text{c-snd } ?y = 0\)

proof –
  from \(S2\) have \(\text{c-snd } ?y = \text{c-snd } ?x\) by simp
  then show \(\text{thesis by simp}\)

qed

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

qed

lemma \(\text{nat-to-sch-at-2}: \text{nat-to-sch } 2 = \text{Base-suc}\)

proof –
  have \(S1: \text{c-pair } 1 \ 0 = 2\) by (simp add: c-pair-def sf-def)
  have \(S2: \text{nat-to-sch } (\text{c-pair } 1 \ 0) = \text{Base-suc}\) by (rule loc-srj-0)
from \(S1\) \(S2\) show \(\text{thesis by simp}\)

qed

lemma \(\text{loc-srj-1}: \text{nat-to-sch } (\text{c-pair } 2 \ 0) = \text{Base-fst}\)

proof –
  let \(?x = \text{c-pair } 2 \ 0\)
  have \(S1: \ ?x = 5\) by (simp add: c-pair-def sf-def)
  then have \(S2: \ ?x = \text{Suc (Suc 3)}\) by simp
  let \(?y = \text{Suc (Suc 3)}\)
  have \(S3: \text{nat-to-sch } ?y = (\text{let } u=\text{mod7 } (\text{c-fst } ?y); v=\text{c-snd } ?y; v1=\text{c-fst } v; v2 = \text{c-snd } v; sch1=\text{nat-to-sch } v1; sch2=\text{nat-to-sch } v2 \text{ in loc-f } u \text{ sch1 sch2} ) (\text{is } = ?R)\)
  by (rule loc-srj-lm-1)
  have \(S4: \text{c-fst } ?y = 2\)
  proof –
    from \(S2\) have \(\text{c-fst } ?y = \text{c-fst } ?x\) by simp
    then show \(\text{thesis by simp}\)

qed

have \(S5: \text{c-snd } ?y = 0\)

proof –
  from \(S2\) have \(\text{c-snd } ?y = \text{c-snd } ?x\) by simp
  then show \(\text{thesis by simp}\)

qed

from \(S4\) have \(S6: \text{mod7 } (\text{c-fst } ?y) = 2\) by (simp add: mod7-def)
from \(S3\) \(S5\) \(S6\) have \(S9: ?R = \text{loc-f 2 Base-zero Base-zero}\) by (simp add: Let-def c-fst-at-0 c-snd-at-0)
  then have \(S10: ?R = \text{Base-fst}\) by (simp add: loc-f-def)
with \(S3\) have \(S11: \text{nat-to-sch } ?y = \text{Base-fst}\) by simp
from \(S2\) this show \(\text{thesis by simp}\)

qed
lemma loc-srj-2: nat-to-sch (c-pair 3 0) = Base-snd

proof –
  let ?x = c-pair 3 0
  have S1: ?x > 1 by (simp add: c-pair-def sf-def)
  from S1 have S2: 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) (is - = ?R) by (rule loc-srj-lm-2)
  have S3: c-fst ?x = 3 by simp
  have S4: c-snd ?x = 0 by simp
  from S3 have S6: mod7 (c-fst ?x) = 3 by (simp add: mod7-def)
  from S3 S4 S6 have S7: ?R = loc-f 3 Base-zero Base-zero by (simp add: Let-def c-fst-at-0 c-snd-at-0)
  then have S8: ?R = Base-snd by (simp add: loc-f-def)
  with S2 have S10: nat-to-sch ?x = Base-snd by simp
  from S2 this show ?thesis by simp
qed

lemma loc-srj-3: [nat-to-sch (sch-to-nat sch1) = sch1; nat-to-sch (sch-to-nat sch2) = sch2]
  ⇒ nat-to-sch (c-pair 4 (c-pair (sch-to-nat sch1) (sch-to-nat sch2))) =
  Comp-op sch1 sch2

proof –
  assume A1: nat-to-sch (sch-to-nat sch1) = sch1
  assume A2: nat-to-sch (sch-to-nat sch2) = sch2
  let ?x = c-pair 4 (c-pair (sch-to-nat sch1) (sch-to-nat sch2))
  have S1: ?x > 1 by (simp add: c-pair-def sf-def)
  from S1 have S2: 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) (is - = ?R) by (rule loc-srj-lm-2)
  have S3: c-fst ?x = 4 by simp
  have S4: c-snd ?x = c-pair (sch-to-nat sch1) (sch-to-nat sch2) by simp
  from S3 have S5: mod7 (c-fst ?x) = 4 by (simp add: mod7-def)
  from A1 A2 S4 S5 have ?R = Comp-op sch1 sch2 by (simp add: Let-def c-fst-at-0 c-snd-at-0 loc-f-def)
  with S2 show ?thesis by simp
qed

lemma loc-srj-3-1: nat-to-sch (c-pair 4 (c-pair n1 n2)) = Comp-op (nat-to-sch n1) (nat-to-sch n2)

proof –
  let ?x = c-pair 4 (c-pair n1 n2)
  have S1: ?x > 1 by (simp add: c-pair-def sf-def)
  from S1 have S2: 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) (is - = ?R) by (rule loc-srj-lm-2)
  have S3: c-fst ?x = 4 by simp
  have S4: c-snd ?x = c-pair n1 n2 by simp
  from S3 have S5: mod7 (c-fst ?x) = 4 by (simp add: mod7-def)
  from S4 S5 have ?R = Comp-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 ?thesis by simp
qed

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

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

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

proof -
  let ?x = c-pair 5 (c-pair n1 n2)
  have S1: ?x > 1 by (simp add: c-pair-def sf-def)
  from S1 have S2: 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) (is - = ?R) by (rule loc-srj-lm-2)
  have S3: c-fst ?x = 5 by simp
  have S4: c-snd ?x = c-pair n1 n2 by simp
  from S3 have S5: mod7 (c-fst ?x) = 5 by (simp add: mod7-def)
  from S4 S5 have ?R = 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 ?thesis by simp
qed

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

proof -
  assume A1: nat-to-sch (sch-to-nat sch1) = sch1
  assume A2: nat-to-sch (sch-to-nat sch2) = sch2
  let ?x = c-pair 6 (c-pair (sch-to-nat sch1) (sch-to-nat sch2))
  have S1: ?x > 1 by (simp add: c-pair-def sf-def)
from $S_1$ have $S_2$: \( \text{nat-to-sch } ?x = (\text{let } u = \text{mod7 } (\text{c-fst } ?x); v = \text{c-snd } ?x; \text{v1 = c-fst v}; \text{v2 = c-snd v}; \text{sch1 = nat-to-sch v1}; \text{sch2 = nat-to-sch v2} \text{ in loc-f u sch1 sch2}) \text{ (is - = } ?R \text{)} ) \) by (rule loc-srj-lm-2)

have $S_3$: \( \text{c-fst } ?x = 6 \) by simp

have $S_4$: \( \text{c-snd } ?x = \text{c-pair } (\text{sch-to-nat sch1}) (\text{sch-to-nat sch2}) \) by simp

from $S_3$ have $S_5$: \( \text{mod7 } (\text{c-fst } ?x) = 6 \) by (simp add: mod7-def)

from $A1 \ A2 \ S_4 \ S_5$ have $?R = \text{Rec-op sch1 sch2} \) by (simp add: Let-def c-fst-at-0 c-snd-at-0 loc-f-def)

with $S_2$ show $?thesis$ by simp

qed

lemma 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 $S_1$: \( ?x > 1 \) by (simp add: c-pair-def sf-def)

from $S_1$ have $S_2$: \( \text{nat-to-sch } ?x = (\text{let } u = \text{mod7 } (\text{c-fst } ?x); v = \text{c-snd } ?x; \text{v1 = c-fst v}; \text{v2 = c-snd v}; \text{sch1 = nat-to-sch v1}; \text{sch2 = nat-to-sch v2} \text{ in loc-f u sch1 sch2}) \text{ (is - = } ?R \text{)} ) \) by (rule loc-srj-lm-2)

have $S_3$: \( \text{c-fst } ?x = 6 \) by simp

have $S_4$: \( \text{c-snd } ?x = \text{c-pair n1 n2} \) by simp

from $S_3$ have $S_5$: \( \text{mod7 } (\text{c-fst } ?x) = 6 \) by (simp add: mod7-def)

from $S_4 \ S_5$ have $?R = \text{Rec-op } (\text{nat-to-sch n1}) (\text{nat-to-sch n2}) \) by (simp add: Let-def c-fst-at-0 c-snd-at-0 loc-f-def)

with $S_2$ show $?thesis$ by simp

qed

theorem nat-to-sch-srj: \( \text{nat-to-sch } (\text{sch-to-nat sch}) = \text{sch} \)

apply(induct sch, auto simp add: loc-srj-0 loc-srj-1 loc-srj-2 loc-srj-3 loc-srj-4 loc-srj-5)

apply(insert loc-srj-0)

apply(simp)

done

4.3 Indexes of primitive recursive functions of one variables

definition \( \text{nat-to-pr } :: \text{nat } \Rightarrow (\text{nat } \Rightarrow \text{nat}) \) where

\( \text{nat-to-pr } = (\lambda x. \text{sch-to-pr } (\text{nat-to-sch } x)) \)

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

lemma 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 $S_1$: \( (\exists t. f = \text{sch-to-pr } t) \) by (rule sch-to-pr-srj)

from $S_1$ obtain $t$ where $S_2$: \( f = \text{sch-to-pr } t \) ..

let $?n = \text{sch-to-nat } t$
have \( S3 \): \( \text{nat-to-pr } \exists n = \text{sch-to-pr } (\text{nat-to-sch } ?n) \) 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

then have \( f = \text{nat-to-pr } ?n \) by simp

then show \( \exists n \).

qed

lemma \( \text{nat-to-pr-at-0} \): \( \text{nat-to-pr } 0 = (\lambda x. 0) \) by (simp add: \text{nat-to-pr-def})

definition \( \text{index-of-pr} :: (\text{nat} \Rightarrow \text{nat}) \Rightarrow \text{nat} \) where
\( \text{index-of-pr } f = (\text{SOME } n. f = \text{nat-to-pr } n) \)

theorem \( \text{index-of-pr-is-real} \): \( f \in \text{PrimRec1} \Rightarrow \text{nat-to-pr } (\text{index-of-pr } f) = f \)

proof –
assume \( f \in \text{PrimRec1} \)
hence \( \exists n. f = \text{nat-to-pr } n \) by (rule \text{nat-to-pr-srj})
hence \( f = \text{nat-to-pr } (\text{SOME } n. f = \text{nat-to-pr } n) \) by (rule \text{someI-ex})
thus \( \exists n \).

qed

definition \( \text{comp-by-index} :: \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \) where
\( \text{comp-by-index } = (\lambda n1 n2. \text{c-pair } 4 (\text{c-pair } n1 n2)) \)

definition \( \text{pair-by-index} :: \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \) where
\( \text{pair-by-index } = (\lambda n1 n2. \text{c-pair } 5 (\text{c-pair } n1 n2)) \)

definition \( \text{rec-by-index} :: \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \) where
\( \text{rec-by-index } = (\lambda n1 n2. \text{c-pair } 6 (\text{c-pair } n1 n2)) \)

lemma \( \text{comp-by-index-is-pr} \): \( \text{comp-by-index} \in \text{PrimRec2} \)

unfolding \text{comp-by-index-def}

using \text{const-is-pr-2 [of 4]} by prec

lemma \( \text{comp-by-index-inj} \): \( \text{comp-by-index } x1 y1 = \text{comp-by-index } x2 y2 \Rightarrow x1 = x2 \wedge y1 = y2 \)

proof –
assume \( \text{comp-by-index } x1 y1 = \text{comp-by-index } x2 y2 \)
hence \( \text{c-pair } 4 (\text{c-pair } x1 y1) = \text{c-pair } 4 (\text{c-pair } x2 y2) \) by (unfold \text{comp-by-index-def})
hence \( \text{c-pair } x1 y1 = \text{c-pair } x2 y2 \) by (rule \text{c-pair-inj2})
thus \( \exists n \).

qed

lemma \( \text{comp-by-index-inj1} \): \( \text{comp-by-index } x1 y1 = \text{comp-by-index } x2 y2 \Rightarrow x1 = x2 \)

by (frule \text{comp-by-index-inj}, drule \text{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 \implies y1 = y2
  by (frule rec-by-index-inj, drule conjunct2)

lemma rec-by-index-main: nat-to-pr (rec-by-index n1 n2) = UnaryRecOp (nat-to-pr n1) (nat-to-pr n2)
  by (unfold rec-by-index-def, unfold nat-to-pr-def, simp add: loc-srj-5-1)

4.4 s-1-1 theorem for primitive recursive functions of one variable

definition index-of-const :: nat \Rightarrow nat where
  index-of-const = PrimRecOp1 0 (\lambda x y. c-pair 4 (c-pair 2 y))

lemma index-of-const-is-pr: index-of-const \in PrimRec1
  proof
    have (\lambda x y. c-pair (4 :: nat) (c-pair (2 :: nat) y)) \in PrimRec2
      by (insert const-is-pr-2 [\where ?n = (4 :: nat)], prec)
    then show ?thesis by (simp add: index-of-const-def pr-rec1)
  qed

lemma index-of-const-at-0: index-of-const 0 = 0
  by (simp add: index-of-const-def)

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

lemma index-of-const-main: nat-to-pr (index-of-const n) = (\lambda x. n) (is ?P n)
  proof (induct n)
    show ?P 0 by (simp add: index-of-const-at-0 nat-to-pr-at-0)
  next
    fix n assume ?P n
    then show ?P (Suc n)
      by ((simp add: index-of-const-at-suc nat-to-sch-at-2
        nat-to-pr-def loc-srj-3-1))
  qed

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

lemma index-of-const-inj: index-of-const n1 = index-of-const n2 \implies n1 = n2
  proof
    assume index-of-const n1 = index-of-const n2
    then have (nat-to-pr (index-of-const n1)) 0 = (nat-to-pr (index-of-const n2))
      0 by simp
    thus ?thesis by (simp add: index-of-const-lm-1)
  qed

definition index-of-zero = sch-to-nat Base-zero
definition index-of-suc = sch-to-nat Base-suc
definition index-of-c-fst = sch-to-nat Base-fst
definition  index-of-c-snd = sch-to-nat Base-snd

definition  index-of-id = pair-by-index index-of-c-fst index-of-c-snd

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 \implies x1 = x2

proof
  assume index-of-c-pair-n x1 = index-of-c-pair-n x2
  hence pair-by-index (index-of-const x1) index-of-id = pair-by-index (index-of-const x2) index-of-id by (unfold index-of-c-pair-n-def)
  hence index-of-const x1 = index-of-const x2 by (rule pair-by-index-inj)
  thus \ \ ?thesis by (rule index-of-const-inj)
qed

definition s1-1 :: nat \Rightarrow nat \Rightarrow nat
where
  s1-1 = (\lambda n x. \text{comp-by-index} n (\text{index-of-c-pair-n} x))

lemma s1-1-is-pr: s1-1 \in \text{PrimRec2} by (unfold s1-1-def, insert \text{comp-by-index-is-pr} \text{index-of-c-pair-n-is-pr}, \text{prec})

theorem s1-1-th: (\lambda y. (\text{nat-to-pr} n) (\text{c-pair} x y)) = \text{nat-to-pr} (s1-1 n x)

proof
  have \text{nat-to-pr} (s1-1 n x) = \text{nat-to-pr} (\text{comp-by-index} n (\text{index-of-c-pair-n} x))
  by (simp add: s1-1-def)
  also have \ldots = (\lambda z. (\text{nat-to-pr} n) ((\text{nat-to-pr} (\text{index-of-c-pair-n} x)) z))
  by (simp add: \text{comp-by-index-main})
  also have \ldots = (\lambda z. (\text{nat-to-pr} n) ((\lambda u. \text{c-pair} x u) z))
  by (simp add: \text{index-of-c-pair-n-main})
  finally show ?thesis by simp
qed

lemma s1-1-inj: s1-1 x1 y1 = s1-1 x2 y2 \implies x1 = x2 \land y1 = y2

proof
  assume s1-1 x1 y1 = s1-1 x2 y2
  then have \text{comp-by-index} x1 (\text{index-of-c-pair-n} y1) = \text{comp-by-index} x2 (\text{index-of-c-pair-n} y2)
  by (unfold s1-1-def)
  then have S1: x1 = x2 \land \text{index-of-c-pair-n} y1 = \text{index-of-c-pair-n} y2
  by (rule \text{comp-by-index-inj})
  then have S2: x1 = x2 ..
  from S1 have \text{index-of-c-pair-n} y1 = \text{index-of-c-pair-n} y2 ..
  then have y1 = y2 by (rule \text{index-of-c-pair-n-inj})
  with S2 show ?thesis ..
qed

lemma s1-1-inj1: s1-1 x1 y1 = s1-1 x2 y2 \implies x1 = x2 by (frule s1-1-inj, drule \text{conjunct1})

lemma s1-1-inj2: s1-1 x1 y1 = s1-1 x2 y2 \implies y1 = y2 by (frule s1-1-inj, drule \text{conjunct2})

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

**Theorem** pr-index-enumerator-is-pr: pr-index-enumerator ∈ PrimRec2

**Proof**
- **Define** g where g x = x for x :: nat
  - **Have** g-is-pr: g ∈ PrimRec1 by (unfold g-def, rule pr-id1-1)
- **Define** h where h a b c = comp-by-index index-of-id b for a b c :: nat
  - **From** comp-by-index-is-pr have h-is-pr: h ∈ PrimRec3 unfolding h-def by prec
- **Let** ?f = pr-index-enumerator
  - **From** g-def have f-at-0: ∀ x. ?f x 0 = g x by auto
  - **From** h-def have f-at-Suc: ∀ x y. ?f x (Suc y) = h x (?f x y) y by auto
  - **Let** ?x = pr-index-enumerator
  - **From** g-is-pr h-is-pr f-at-0 f-at-Suc show ?thesis by (rule pr-rec-last-scheme)

**Qed**

**Lemma** pr-index-enumerator-increase1: pr-index-enumerator n m < pr-index-enumerator (n+1) m

**Proof** (induct m)
- **Show** pr-index-enumerator n 0 < pr-index-enumerator (n + 1) 0 by simp
- **Next fix** na assume A: pr-index-enumerator n na < pr-index-enumerator (n + 1) na
  - **Show** pr-index-enumerator n (Suc na) < pr-index-enumerator (n + 1) (Suc na)
    - **Proof**
      - **Let** ?a = pr-index-enumerator n na
      - **Let** ?b = pr-index-enumerator (n+1) na
        - **Have** S1: pr-index-enumerator n (Suc na) = comp-by-index index-of-id ?a by simp
        - **Have** S2: comp-by-index index-of-id ?a = c-pair 4 (c-pair index-of-id ?a) by simps
          - **From** A have c-pair index-of-id ?a < c-pair index-of-id ?b by (rule c-pair-strict-mono2)
            - **Then have** c-pair 4 (c-pair index-of-id ?a) < c-pair 4 (c-pair index-of-id ?b) by (rule c-pair-strict-mono2)
          - **Then have** comp-by-index index-of-id ?a < comp-by-index index-of-id ?b by (simp add: comp-by-index-def)
            - **With** S1 S2 show ?thesis by auto
      - **Qed**

**Qed**

**Lemma** pr-index-enumerator-increase2: pr-index-enumerator n m < pr-index-enumerator n (m + 1)

**Proof**
- **Let** ?a = pr-index-enumerator n m
  - **Have** S1: pr-index-enumerator n (m + 1) = comp-by-index index-of-id ?a by simp
  - **Have** S2: comp-by-index index-of-id ?a = c-pair 4 (c-pair index-of-id ?a) by
have $S3$: $4 + \text{c-pair index-of-id} \ ?a \leq \text{c-pair} 4 (\text{c-pair index-of-id} \ ?a)$ by (rule \text{sum-le-c-pair})

then have $S4$: c-pair index-of-id \ ?a < c-pair 4 (c-pair index-of-id \ ?a) by auto

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

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

with $S1$ $S2$ show \ ?thesis by auto

qed

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

proof (rule allI, rule allI)

fix \ x \ 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: \ \land \ u \ v, \ f \ u < f \ (u + (v+1))

proof

fix u v show f u < f \ (u + (v+1))

proof (induct v)

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

next

fix v 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

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

lemma \text{pr-index-enumerator-mono1}: n1 < n2 \Longrightarrow \text{pr-index-enumerator} n1 \ m < \text{pr-index-enumerator} n2 \ m

proof

assume A: \ n1 < n2

define f where \ f x = \text{pr-index-enumerator} x \ m \ \text{for} \ x

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

proof

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

qed

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

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

qed

lemma \text{pr-index-enumerator-mono2}: m1 < m2 \Longrightarrow \text{pr-index-enumerator} n \ m1 < \text{pr-index-enumerator} n \ m2

73
proof –
assume A: m1 < m2
define f where f x = pr-index-enumerator n x for x
have f-inc: ∀ 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-inc have ∀ x y. (x < y −→ f x < f y) by (rule f-inc-mono)
with A f-def show ?thesis by auto
qed

lemma f-mono-inj: ∀ (x::nat) (y::nat). (x < y −→ (f::nat⇒nat) x < f y) −→ ∀ (x::nat) (y::nat). (f x = f y −→ x = y)
proof (rule allI, rule allI)
fix x y assume A: ∀ x y. x < y −→ f x < f y show f x = f y −→ x = y
proof
assume A1: f x = f y show x = y
proof (rule ccontr)
assume A2: x ≠ y show False
proof cases
assume A3: x < y
from A A3 have f x < f y by auto
with A1 show False by auto
next
assume ¬ x < y with A2 have A4: y < x by auto
from A A4 have f y < f x by auto
with A1 show False by auto
qed
qed
qed
qed

theorem pr-index-enumerator-inj1: pr-index-enumerator n1 m = pr-index-enumerator n2 m −→ n1 = n2
proof –
assume A: pr-index-enumerator n1 m = pr-index-enumerator n2 m
define f where f x = pr-index-enumerator n x m for x
have f-mono: ∀ x y. (x < y −→ 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-mono1)
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-inj2: pr-index-enumerator n m1 = pr-index-enumerator n m2 −→ m1 = m2
proof –
assume A: pr-index-enumerator n m1 = pr-index-enumerator n m2
define \( f \) where \( f x = \text{pr-index-enumerator} n x \) for \( x \)

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

proof (rule allI, rule allI)

fix \( x \ y \) show \( x < y \rightarrow f x < f y \) by (unfold \( f\)-def, simp add: \( \text{pr-index-enumerator-mono2} \))

qed

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

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

qed

theorem \( \text{pr-index-enumerator-main} \): \( \text{nat-to-pr} n = \text{nat-to-pr} (\text{pr-index-enumerator} n m) \)

proof (induct \( m \))

show \( \text{nat-to-pr} n = \text{nat-to-pr} (\text{pr-index-enumerator} n 0) \) by simp

next

fix \( na \) assume \( A \): \( \text{nat-to-pr} n = \text{nat-to-pr} (\text{pr-index-enumerator} n na) \)

show \( \text{nat-to-pr} n = \text{nat-to-pr} (\text{pr-index-enumerator} n (\text{Suc} \ na)) \)

proof −

let \( ?a = \text{pr-index-enumerator} n na \)

have \( S1 \): \( \text{pr-index-enumerator} n (\text{Suc} \ na) = \text{comp-by-index index-of-id} \ ?a \) by simp

have \( \text{nat-to-pr} (\text{comp-by-index index-of-id} \ ?a) = (\lambda x. (\text{nat-to-pr} \text{index-of-id}) (\text{nat-to-pr} \ ?a \ x)) \) by (rule \( \text{comp-by-index-main} \))

with \( \text{index-of-id-main} \) have \( \text{nat-to-pr} (\text{comp-by-index index-of-id} \ ?a) = \text{nat-to-pr} \ ?a \) by simp

with \( A \) \( S1 \) show \( \text{thesis} \) by simp

qed

qed

end

5 Finite sets

theory \( \text{PRecFinSet} \)
imports \( \text{PRecFun} \)
begin

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

definition \( c\text{-in} :: \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \) where

\( c\text{-in} = (\lambda x u. (u \div (2 ^ x)) \mod 2) \)

lemma \( c\text{-in-is-pr} : c\text{-in} \in \text{PrimRec2} \)

proof −

from \( \text{mod-is-pr} \) \( \text{power-is-pr} \) \( \text{div-is-pr} \) have \( (\lambda x u. (u \div (2 ^ x)) \mod 2) \in \text{PrimRec2} \) by prec

with \( c\text{-in-def} \) show \( \text{thesis} \) by auto

qed
definition

\textit{nat-to-set} :: nat ⇒ nat set where
\textit{nat-to-set} u ≡ \{ x. \text{2}^x \leq u \wedge \text{c-in} x u = 1 \}

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

lemma nat-to-set-upper-bound: x ∈ \textit{nat-to-set} u ⇒ \text{2}^x \leq u by (simp add: \textit{nat-to-set-def})

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

lemma nat-to-set-upper-bound1: x ∈ \textit{nat-to-set} u ⇒ x < u
proof –
  assume x ∈ \textit{nat-to-set} u
  then have S1: \text{2}^x \leq u by (simp add: \textit{nat-to-set-def})
  have S2: x < \text{2}^x by (rule \text{x-lt-2-x})
  from S1 S2 show \textit{?thesis} by auto
qed

lemma nat-to-set-upper-bound2: \textit{nat-to-set} u ⊆ \{ i. i < u \}
proof –
  from \textit{nat-to-set-upper-bound1} show \textit{?thesis} by blast
qed

lemma nat-to-set-is-finite: finite (\textit{nat-to-set} u)
proof –
  have S1: finite \{ i. i < u \}
  proof –
    let \textit{?B} = \{ i. i < u \}
    let \textit{?f} = \lambda (x::nat). x
    have \textit{?B} = \textit{?f \cdot ?B} by auto
    then show finite \textit{?B} by (rule \text{nat-seg-image-imp-finite})
  qed
  have S2: \textit{nat-to-set} u ⊆ \{ i. i < u \} by (rule \textit{nat-to-set-upper-bound2})
  from S2 S1 show \textit{?thesis} by (rule \text{finite-subset})

76
**lemma** $x$-in-$u$-eq: $(x \in \text{nat-to-set } u) = (c\text{-in } x u = 1)$ by (auto simp add: nat-to-set-def c-in-upper-bound)

**definition**

$log2 :: \text{nat } \Rightarrow \text{nat}$ where

$log2 = (\lambda x. \text{Least}(\%z. x < 2^z))$

**lemma** $log2$-at-$0$: $log2 0 = 0$

**proof** –

let $?v = log2 0$

have $S1$: $0 \leq ?v$ by auto

have $S2$: $?v = \text{Least}(\%z::\text{nat}. (0::\text{nat})<2^z)$ by (simp add: log2-def)

have $S3$: $(0::\text{nat})<2^0$ by auto

from $S3$ have $S4$: $\text{Least}(\%z::\text{nat}. (0::\text{nat})<2^z) \leq 0$ by (rule Least-le)

from $S2$ $S4$ have $S5$: $?v \leq 0$ by auto

from $S1$ $S5$ have $S6$: $?v = 0$ by auto

thus $?thesis$ by auto

qed

**lemma** $log2$-at-$1$: $log2 1 = 0$

**proof** –

let $?v = log2 1$

have $S1$: $0 \leq ?v$ by auto

have $S2$: $?v = \text{Least}(\%z::\text{nat}. (1::\text{nat})<2^z)$ by (simp add: log2-def)

have $S3$: $(1::\text{nat})<2^1$ by auto

from $S3$ have $S4$: $\text{Least}(\%z::\text{nat}. (1::\text{nat})<2^z) \leq 0$ by (rule Least-le)

from $S2$ $S4$ have $S5$: $?v \leq 0$ by auto

from $S1$ $S5$ have $S6$: $?v = 0$ by auto

thus $?thesis$ by auto

qed

**lemma** $log2$-le: $x > 0 \Rightarrow 2^\log2 x \leq x$

**proof** –

assume $A$: $x > 0$

show $?thesis$

proof (cases)

assume $A1$: $\log2 x = 0$

with $A$ show $?thesis$ by auto

next

assume $A1$: $\log2 x \neq 0$

then have $S1$: $\log2 x > 0$ by auto

define $y$ where $y = \log2 x - 1$

from $S1$ $y$-def have $S2$: $\log2 x = y + 1$ by auto

then have $S3$: $y < \log2 x$ by auto

have $2^y \leq x$

proof (rule ccontr)

assume $A2$: $\neg 2^y \leq x$ then have $x < 2^{y+1}$ by auto
then have \( \log_2 x \leq y \) by (simp add: log2-def Least-le)
with S3 show False by auto
qed
with S2 show ?thesis by auto
qed

lemma \( \log_2\text{-gt}: x < 2^\cdot(\log_2 x + 1) \)
proof –
  have \( x < 2^\cdot x \) by (rule x-lt-2-x)
  then have S1: \( x < 2^\cdot(x+1) \) by simp
  define \( y \) where \( y = x \)
  from S1 y-def have \( S2: x < 2^\cdot(y+1) \) by auto
  from S2 have S3: \( \forall P \cdot y \in \{ \lambda z. x < 2^\cdot(z+1) \} \) by auto
  define \( \mu \) where \( \mu = \lambda z. x < 2^\cdot(z+1) \)
  from S3 have \( S4: \mu(y) \) by (rule LeastI)
  from log2-def have S5: \( \log_2 x = \mu \) by (unfold log2-def, auto)
  from S4 S5 show ?thesis by auto
qed

lemma \( x\text{-div-x}: x > 0 \Rightarrow (x::nat) \div x = 1 \) by auto
lemma \( \text{div-ge}: (k::nat) \leq m \div n \Rightarrow n \cdot k \leq m \)
proof –
  assume A: \( k \leq m \div n \)
  have S1: \( n \cdot (m \div n) + m \mod n = m \) by (rule mult-div-mod-eq)
  have S2: \( \forall P \cdot k \leq m \mod n \) by auto
  define \( y \) where \( y = x \)
  from S1 y-def have \( S3: x < 2^\cdot(y+1) \) by auto
  define \( \mu \) where \( \mu = \lambda z. x < 2^\cdot(z+1) \)
  from S3 have \( S4: \mu(y) \) by (rule LeastI)
  from \( \forall P \cdot \mu \) have ?thesis by (rule LeastI)
  from S4 S5 show ?thesis by auto
qed

lemma \( \text{div-lt}: m < n \cdot k \Rightarrow m \div n < (k::nat) \)
proof –
  assume A: \( m < n \cdot k \)
  show ?thesis
  proof (rule ccontr)
    assume \( \neg m \div n < k \)
    then have S1: \( k \leq m \div n \) by auto
    then have S2: \( n \cdot k \leq m \) by (rule div-ge)
    show False by auto
  qed
qed

lemma \( \log_2\text{-lm1}: u > 0 \Rightarrow u \div 2^\cdot(\log_2 u) = 1 \)
proof –
  assume A: \( u > 0 \)
  then have S1: \( 2^\cdot(\log_2 u) \leq u \) by (rule log2-le)
  have S2: \( u < 2^\cdot(\log_2 u + 1) \) by (rule log2-gt)
  then have S3: \( u < (2^\cdot\log_2 u + 2) \) by simp
  have \( (2::nat) > 0 \) by simp

78
then have \((2::\text{nat}) \cdot \log_2 u > 0\) by simp
then have \(S4: (2::\text{nat}) \cdot \log_2 u \div 2 \cdot \log_2 u = 1\) by auto
  from \(S1\) have \(S5: (2::\text{nat}) \cdot \log_2 u \div 2 \cdot \log_2 u \leq u \div 2 \cdot \log_2 u\) by (rule div-le-mono)
    with \(S4\) have \(S6: 1 \leq u \div 2 \cdot \log_2 u\) by auto
  from \(S3\) have \(S7: u \div 2 \cdot \log_2 u < 2\) by (rule div-lt)
  from \(S6\) \(S7\) show \(?\)thesis by auto 
qed

lemma \(\log_2-lm2\): \(u > 0 \implies \text{c-in} (\log_2 u) u = 1\)
proof –
  assume \(A: u > 0\)
  then have \(S1: u \div 2 \cdot (\log_2 u) = 1\) by (rule \(\log_2-lm1\))
  have \(\text{c-in} (\log_2 u) u = (u \div 2 \cdot (\log_2 u)) \mod 2\) by (simp add: c-in-def)
    also from \(S1\) have \(\ldots = 1 \mod 2\) by simp
    also have \(\ldots = 1\) by auto
  finally show \(?\)thesis by auto 
qed

lemma \(\log_2-lm3\): \(\log_2 u < x \implies \text{c-in} x u = 0\)
proof –
  assume \(A: \log_2 u < x\)
  then have \(S1: (\log_2 u)+1 \leq x\) by auto
  have \(S2: 1 \leq (2::\text{nat})\) by auto
  from \(S1\) \(S2\) have \(S3: (2::\text{nat}) \cdot ((\log_2 u)+1) \leq 2 \cdot x\) by (rule power-increasing)
    have \(S4: u < (2::\text{nat}) \cdot ((\log_2 u)+1)\) by (rule \(\log_2-gt\))
    from \(S3\) \(S4\) have \(S5: u < 2 \cdot x\) by auto
    then have \(S6: u \div 2 \cdot x = 0\) by (rule div-less)
      have \(\text{c-in} x u = (u \div 2 \cdot x) \mod 2\) by (simp add: c-in-def)
      also from \(S6\) have \(\ldots = 0 \mod 2\) by simp
      also have \(\ldots = 0\) by auto
    finally have \(?\)thesis by auto
    thus \(?\)thesis by auto 
qed

lemma \(\log_2-lm4\): \(\text{c-in} x u = 1 \implies x \leq \log_2 u\)
proof –
  assume \(A: \text{c-in} x u = 1\)
  show \(?\)thesis
    proof (rule ccontr)
      assume \(\neg x \leq \log_2 u\)
      then have \(S1: \log_2 u < x\) by auto
      then have \(S2: \text{c-in} x u = 0\) by (rule \(\log_2-lm3\))
        with \(A\) show \(False\) by auto
    qed
qeda

lemma \(\text{nat-to-set-lub}\): \(x \in \text{nat-to-set} \implies x \leq \log_2 u\)
proof –

assume \( x \in \text{nat-to-set} \ u \)
then have \( S1: \ c\text{-in} \ x \ u = 1 \) by \((\text{simp add: x-in-u-eq})\)
then show \( ?\text{thesis} \) by \((\text{rule log2-lm4})\)
qed

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

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

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

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

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

lemma \( \text{ne\text{-}imp\text{-}pos:} \ \text{nat\text{-}to\text{-}set} \ u \neq {} \implies u > 0 \)
proof (rule ccontr)
  assume A1: nat-to-set u ≠ {}
  assume ¬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 ⇒ ((u + (2::nat) ∙ x) div (2::nat) ∙ y) mod 2 = (u div (2::nat) ∙ y) mod 2
proof –
  assume y-lt-x: y < x
  let ?n = (2::nat) ∙ y
  have n-pos: 0 < ?n by auto
  let ?s = 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) ∙ x = (2::nat) ∙ (y + ?s) by auto
  ultimately have (2::nat) ∙ x = 2 ∙ y ∗ 2 ∙ ?s by auto
  then have S4: u + (2::nat) ∙ x = u + (2::nat) ∙ y ∗ 2 ∙ ?s by auto
  from n-pos have S5: (u + (2::nat) ∙ y ∗ 2 ∙ ?s) div 2 ∙ y = 2 ∙ ?s + (u div 2 ∙ y) by simp
    from S4 S5 have S6: (u + (2::nat) ∙ x) div 2 ∙ y = 2 ∙ ?s + (u div 2 ∙ y) by auto
    from s-pos have S8: ?s = (?s − 1) + 1 by auto
    have (2::nat) ∙ ((?s − (1::nat)) + (1::nat)) = (2::nat) ∙ (?s − (1::nat)) ∗ 2 ∙ 1
      by (rule power-add)
    with S8 have S9: (2::nat) ∙ (?s = (2::nat) ∙ (?s − (1::nat)) ∗ 2 by auto
      then have S10: 2 ∙ ?s + (u div 2 ∙ y) = (u div 2 ∙ y) + (2::nat) ∙ (?s − (1::nat)) ∗ 2 by auto
      have S11: ((u div 2 ∙ y) + (2::nat) ∙ (u div 2 ∙ y)) mod 2 = (u div 2 ∙ y) mod 2 by (rule mod-mult-self1)
    from S6 S10 S11 show ?thesis by auto
qed

lemma add-power: u < 2 ∙ x ⇒ nat-to-set (u + 2 ∙ x) = nat-to-set u ∪ {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+1) by auto
    next
      fix z
      assume A1: u + 2 ∙ x < 2 ∙ (z+1)
      show x ≤ z
        proof (rule ccontr)
          assume ¬x ≤ z
          then have z < x by auto
          then have L1: z+1 ≤ x by auto
          have L2: 1 ≤ (2::nat) by auto
          81
from $L1 \ L2$ have $L3$: $(2::\mathtt{nat}) ^{(z+1)} \leq (2::\mathtt{nat}) ^x$ by \textit{(rule power-increasing)}

with $A1$ show $\text{False}$ by auto

qed

show $?thesis$

\textbf{proof} (\textit{rule subset-antisym})

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

\textbf{proof} fix $y$

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

\textbf{show} $y \in \text{nat-to-set} \ u \cup \{x\}$

\textbf{proof}

assume $y \notin \{x\}$ then have $SI$: $y \neq x$ by auto

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

with \textit{log2-is-x} have $y \leq x$ by auto

with $SI$ have $y$-lt-\textit{x}: $y < x$ by auto

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

then have $S2$: $(u + 2 ^ y) \mskmod 2 = 1$ by \textit{(unfold c-in-def)}

from $y$-lt-\textit{x} have $(u + (2::\mathtt{nat}) ^ x) \mskmod 2 = (u \div (2::\mathtt{nat}) ^ y) \mskmod 2$ by \textit{(rule div-mod-lm)}

with $S2$ have $u$-div-\textit{y} \mskmod 2 = 1 by \textit{(rule div-\mskmod)}

then show $y \in \text{nat-to-set} \ u$ by \textit{(simp add: x-in-u-eq)}

qed

next

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

\textbf{proof} fix $y$

assume $A1$: $y \in \text{nat-to-set} \ u \cup \{x\}$

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

\textbf{proof} cases

assume $y \in \{x\}$

then have $y$=\textit{x} by \textit{auto}

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

then show $?thesis$ by \textit{(simp add: log2-lm5)}

next

assume $y$-notin: $y \notin \{x\}$

then have $y$-ne-\textit{x}: $y \neq x$ by \textit{auto}

from $A1$ \textit{y-notin} have $y$-in: $y \in \text{nat-to-set} \ u$ by \textit{auto}

\textbf{have} $y$-lt-\textit{x}: $y < x$

\textbf{proof} \textit{(rule ccontr)}

\textbf{assume} $\neg \ y < x$

\textbf{with} $y$-ne-\textit{x} have $y$-gt-\textit{x}: $x < y$ by \textit{auto}

\textbf{have} $1 < (2::\mathtt{nat})$ by \textit{auto}

from $y$-gt-\textit{x} this have $L1$: $(2::\mathtt{nat}) ^ x < 2 ^ y$ by \textit{(rule power-strict-increasing)}

from $y$-in have $L2$: $2 ^ y \leq u$ by \textit{(rule nat-to-set-upper-bound)}

from $L1 \ L2$ have $(2::\mathtt{nat}) ^ x < u$ by \textit{arith}

\textbf{with} $A$ \textbf{show} $\text{False}$ by \textit{auto}

qed

from $y$-in have c-in \textit{y} $u = 1$ by \textit{(simp add: x-in-u-eq)}

82
then have $S2: \ (u \ div \ 2^y) \ mod \ 2 = 1$ by (unfold c-in-def)
from y-lt-x have \[ ((u + (2::nat) \ ^ \ x) \ div \ (2::nat)^y) \ mod \ 2 = (u \ div \ (2::nat)^y) \ mod \ 2 \]
by (rule div-mod-lm)
with $S2$ have \[ ((u + (2::nat) \ ^ \ x) \ div \ 2^y) \ mod \ 2 = 1 \] by auto
then have c-in y \[ (u + (2::nat) \ ^ \ x) = 1 \] by simp add: c-in-def
then show \[ y \in \ nat-to-set \ (u + (2::nat) \ ^ \ x) \] by simp add: x-in-u-eq
qed

theorem nat-to-set-inj: \[ \text{nat-to-set \ u = nat-to-set \ v} \implies u = v \]
proof
  assume A: \[ \text{nat-to-set \ u = nat-to-set \ v} \]
  let \[ ?P = \lambda \ (n::nat). \ (\forall \ D :: \text{nat set}. \ \text{finite \ D} \land \ \text{card \ D} \leq n \implies (\forall \ u \ v. \ \text{nat-to-set \ u = D} \land \ \text{nat-to-set \ v = D} \implies u = v)) \]
  have P-at-0: \[ ?P \ 0 \]
  proof fix D show \[ \text{finite \ D} \land \ \text{card \ D} \leq 0 \implies (\forall \ u \ v. \ \text{nat-to-set \ u = D} \land \ \text{nat-to-set \ v = D} \implies u = v) \]
    proof (rule impI)
      assume A1: \[ \text{finite \ D} \land \ \text{card \ D} \leq 0 \]
      from A1 have S1: \[ \text{finite \ D} \] by auto
      from A1 have S2: \[ \text{card \ D} = 0 \] by auto
      from S1 S2 have S3: \[ \text{D} = \{\} \] 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)
        fix u v show \[ \text{nat-to-set \ u = D} \land \ \text{nat-to-set \ v = D} \implies u = v \]
        proof
          assume A2: \[ \text{nat-to-set \ u = D} \land \ \text{nat-to-set \ v = D} \]
          from A2 have L1: \[ \text{nat-to-set \ u = D} \] by auto
          from A2 have L2: \[ \text{nat-to-set \ v = D} \] 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

  have P-at-Suc: \[ \forall \ n. \ ?P \ n \implies ?P \ (Suc \ n) \]
  proof - fix n
    assume A-n: \[ ?P \ n \]
    show \[ ?P \ (Suc \ n) \]
    proof fix D show \[ \text{finite \ D} \land \ \text{card \ D} \leq Suc \ n \implies (\forall \ u \ v. \ \text{nat-to-set \ u = D} \land \ \text{nat-to-set \ v = D} \implies u = v) \]
      proof (rule impI)
        assume A1: \[ \text{finite \ D} \land \ \text{card \ D} \leq Suc \ n \]
        from A1 have S1: \[ \text{finite \ D} \] by auto
        qed
    qed
  qed

83
from A1 have S2: \( \text{card } D \leq \text{Suc } n \) by auto

show (\( \forall u \ v. \text{nat-to-set } u = D \land \text{nat-to-set } v = D \longrightarrow u = v \))

proof (rule allI, rule allI, rule impI)

\[ \text{fix } u \ v \]

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 auto

from A2 have d-v-d: \( \text{nat-to-set } v = D \) by auto

show \( u = v \)

proof (cases)

assume A3: \( D = \{ \} \)

from A3 d-u-d have nat-to-set u = \{ \} by auto

then have u-z: \( u = 0 \) by (rule empty-is-zero)

from A3 d-v-d 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

next

assume A3: \( D \neq \{ \} \)

from A3 d-u-d have nat-to-set u \( \neq \) \{ \} by auto

then have u-pos: \( u > 0 \) by (rule ne-imp-pos)

from A3 d-v-d have nat-to-set v \( \neq \) \{ \} by auto

then have v-pos: \( v > 0 \) by (rule ne-imp-pos)

define \( m \) where \( m = \text{Max } D \)

from S1 m-def A3 have m-in: \( m \in D \) by auto

from d-u-d m-def have m-u: \( m = \text{Max } (\text{nat-to-set } u) \) by auto

from d-v-d m-def have m-v: \( m = \text{Max } (\text{nat-to-set } v) \) by auto

from u-pos m-u log2-is-max have m-log-u: \( m = \text{log2 } u \) by auto

from v-pos m-v log2-is-max have m-log-v: \( m = \text{log2 } v \) by auto

define D1 where D1 = \( D - \{ m \} \)

define u1 where u1 = \( u - 2^m \)

define v1 where v1 = \( v - 2^m \)

have card-D1: \( \text{card } D1 = \text{card } D - 1 \) by (simp add: card-Diff-singleton)

with S2 show ?thesis by auto

qed

have u-u1: \( u = u1 + 2^m \)

proof

from u-pos have L1: \( 2 \cdot \text{log2 } u \leq u \) by (rule log2-le)

with m-log-u have L2: \( 2 \cdot m \leq u \) by auto

with u1-def show ?thesis by auto

qed

have u1-d1: \( \text{nat-to-set } u1 = D1 \)

proof

from m-log-u log2-gt have u < \( 2^{(m+1)} \) by auto

with u-u1 have u1-lt-2-m: \( u1 < 2^m \) by auto

with u-u1 have L1: \( \text{nat-to-set } u = \text{nat-to-set } u1 \cup \{ m \} \) by (simp add: add-power)

have m-notin: \( m \notin \text{nat-to-set } u1 \)

84
proof (rule ccontr)
  assume ¬ m /∈ nat-to-set u1 then have m ∈ nat-to-set u1 by auto
  then have 2^m ≤ u1 by (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 v-v1 have 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 ≤ v1 by (rule nat-to-set-upper-bound)
  with v1-lt-2-m show False 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 v. 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

from P-at-0 P-at-Suc have main: ∃ n. ?P n by (rule nat.induct)
define D where D = nat-to-set u
from D-def A have P1: nat-to-set u = D by auto
from D-def A have P2: nat-to-set v = D by auto
from D-def nat-to-set-is-finite have d-finite: finite D by auto
define n where 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
with P1 P2 show u = v by auto
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 + 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)-Suc 1) by auto
also have ... = (2 ^ Suc (Suc n)) - 1 by auto
finally show ?thesis by auto
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 {} = 0 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

86
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
  thus ?thesis by auto
qed

next
fix n show set-to-nat \{i. i < Suc n\} = 2 ^ Suc n - 1 by (unfold set-to-nat-def, rule two-power-sum)
qed

proof -
  assume b-finite: finite B
  assume a-le-b: A ⊆ B
  let ?f = λ(x::nat). (2::nat) ^ x
  have S1: set-to-nat A = sum ?f A by (simp add: set-to-nat-def)
  have S2: set-to-nat B = sum ?f B by (simp add: set-to-nat-def)
  have S3: \x. x ∈ B - A ⇒ 0 ≤ ?f x by auto
  from b-finite a-le-b S3 have sum ?f A ≤ sum ?f B by (rule sum-mono2)
  with S1 S2 show ?thesis by auto
qed

theorem nat-to-set-srj: finite (D::nat set) ⇒ nat-to-set (set-to-nat D) = D
proof -
  assume A: finite D
  let ?P = λ(n::nat). (∀(D::nat set). finite D ∧ card D = n → nat-to-set (set-to-nat D) = D)
  have P-at-0: ?P 0
  proof (rule allI)
    fix D
    show finite D ∧ card D = 0 → nat-to-set (set-to-nat D) = D
    proof
      assume A1: finite D ∧ card D = 0
      from A1 have S1: finite D by auto
      from A1 have S2: card D = 0 by auto
      from S1 S2 have S3: D = {} by auto
      with set-to-nat-def have set-to-nat D = sum (λx. 2 ^ x) D by simp
      with S3 sum.empty have set-to-nat D = 0 by auto
      with zero-is-empty S3 show nat-to-set (set-to-nat D) = D by auto
    qed
  qed

have P-at-Suc: \n. ?P n ⇒ ?P (Suc n)
proof - fix n
  assume A-n: ?P n
  show ?P (Suc n)
  proof
    fix D show finite D ∧ card D = Suc n → nat-to-set (set-to-nat D) = D
    proof
      assume A1: finite D ∧ card D = Suc n
    qed

87
from A1 have S1: finite D by auto
from A1 have S2: card D = Suc n by auto
define m where m = Max D
from S2 have D-ne: D ≠ {} by auto
with S1 m-def have m-in: m ∈ D by auto
define D1 where 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
define u where u = set-to-nat D
define u1 where u1 = set-to-nat D1
from S1 m-in have sum (λ(x::nat). (2::nat) ^ x) D = 2 ^ m + sum (λ x. 2 ^ x) (D - {m})
  by (rule sum.remove)
with set-to-nat-def have set-to-nat D = 2 ^ m + set-to-nat (D - {m}) by auto
with u-def u1-def D1-def have u-u1: u = u1 + 2 ^ m by auto
from S3 u1-def have d1-u1: nat-to-set u1 = D1 by auto
have u1-lt: u1 < 2 ^ 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 ^ m - 1 by auto
    have 0 < (2::nat) ^ m by auto
    then have (2::nat) ^ m - 1 < (2::nat) ^ 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 ^ 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

88
from P-at-0 P-at-Suc have main: \( \forall n. ?P n \) by (rule nat.induct)
from A main show \(?thesis\) by auto
qed

theorem nat-to-set-srj1: finite \((D::nat set)\) \(\Rightarrow\) \(\exists u.\) 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 \((\text{set-to-nat } D)\) = \(D\) by (rule nat-to-set-srj)
  qed
qed

lemma sum-of-pr-is-pr: \(g \in \text{PrimRec1} \Rightarrow (\lambda n. \text{sum } g \{i. i<n\}) \in \text{PrimRec1}\)
proof
  assume g-is-pr: \(g \in \text{PrimRec1}\)
  define \(f\) where \(f n = \text{sum } g \{i. i<n\}\) for \(n\)
  from f-def have f-at-0: \(f 0 = 0\) by auto
  define \(h\) where \(h a b = g a + b\) for \(a b\)
  from g-is-pr have h-is-pr: \(h \in \text{PrimRec2}\) unfolding h-def by prec
  have f-at-Suc: \(\forall y. f (\text{Suc } y) = h y (f y)\)
  proof
    fix \(y\)
    show \(f (\text{Suc } y) = h y (f y)\)
    proof
      from f-def have S1: \(f (\text{Suc } y) = \text{sum } g \{i. i<\text{Suc } y\}\) by auto
      have S2: \(\{i. i<\text{Suc } y\} = \{i. i<y\} \cup \{y\}\) 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: \(\text{Suc } y) = \text{sum } g \{i. (i::nat)<y\} \cup \{y\}\) by auto
      also from S3 S4 sum.insert have \(\ldots = g y + \text{sum } g \{i. i<y\}\) by auto
      also from f-def have \(\ldots = g y + f y\) by auto
      also from h-def have \(\ldots = h y (f y)\) by auto
      finally show \(?thesis\) by auto
    qed
  qed
  from h-is-pr f-at-0 f-at-Suc have f-is-pr: \(f \in \text{PrimRec1}\) by (rule pr-rec1-scheme)
  with f-def \[abs-def\] show \(?thesis\) by auto
qed

lemma sum-of-pr-is-pr2: \(p \in \text{PrimRec2} \Rightarrow (\lambda n m. \text{sum } (\lambda x. p x m) \{i. i<n\}) \in \text{PrimRec2}\)
proof
  assume p-is-pr: \(p \in \text{PrimRec2}\)
  define \(f\) where \(f n m = \text{sum } (\lambda x. p x m) \{i. i<n\}\) for \(n m\)
  define \(g::\text{nat} \Rightarrow \text{nat}\) where \(g x = 0\) for \(x\)
  have g-is-pr: \(g \in \text{PrimRec1}\) by (unfold g-def, rule const-is-pr \[where \?n=0\])
  qed
have \( f\text{-at-0}: \forall x. f\ 0\ x = g\ x \)
proof
\begin{itemize}
  \item fix \( x \) from \( f\text{-def} \) \( g\text{-def} \) show \( f\ 0\ x = g\ x \) by auto
\end{itemize}
qed

define \( h \) where \( a\ b\ c = p\ a\ c + b\ ) for \( a\ b\ c \)
from \( p\text{-is-pr} \) have \( h\text{-is-pr}: h \in \text{PrimRec3} \) unfolding \( h\text{-def} \) by prec
have \( f\text{-at-Suc}: \forall x\ y. f\ (\text{Suc} y)\ x = h\ y\ (f\ y\ x)\ x \)
proof (rule allI, rule allI)
\begin{itemize}
  \item fix \( x\ y\ ) show \( f\ (\text{Suc} y)\ x = h\ y\ (f\ y\ x)\ x \)
  \item \begin{itemize}
        \item from \( f\text{-def} \) \( g\text{-def} \) have \( f\text{-at-Suc} = \) \( h\text{-def} \)
        \item have \( S1: f\ (\text{Suc} y)\ x = \text{sum}\ (\lambda z. p\ z\ x)\ \{i.\ i < \text{Suc} y\} \) by auto
        \item have \( S2: \{i.\ i < \text{Suc} y\} = \{i.\ i < y\} \cup \{y\} \) by auto
        \item have \( S3: \text{finite}\ \{i.\ i < y\} \) by (rule finite-interval)
        \item have \( S4: y \notin \{i.\ i < y\} \) by auto
        \item define \( g1\ ) where \( g1\ z = p\ z\ x\ ) for \( z \)
        \item from \( S1\ S2\ g1\text{-def} \) have \( f\ (\text{Suc} y)\ x = \text{sum}\ g1\ \{\{.\ (i::\text{nat}) < y\} \cup \{y\}\} \) by auto
  \end{itemize}
also from \( S3\ S4\ ) sum.insert have \( \ldots = g1\ y + \text{sum}\ g1\ \{i.\ i < y\} \) by auto
also from \( f\text{-def} \) \( g1\text{-def} \) have \( \ldots = g1\ y + f\ y\ x\ ) by auto
also from \( h\text{-def} \) \( g1\text{-def} \) have \( \ldots = h\ y\ (f\ y\ x)\ x\ ) by auto
  \item finally show \( \lambda\ ) \( \text{thesis} \) by auto
\end{itemize}
qed

lemma \( \text{sum-is-pr}: g \in \text{PrimRec1} \implies (\lambda u.\ \text{sum}\ (\text{nat-to-set}\ u)) \in \text{PrimRec1} \)
proof –
\begin{itemize}
  \item assume \( g\text{-is-pr}: g \in \text{PrimRec1} \)
  \item define \( g1\ ) where \( g1\ x\ u = (\text{if}\ \{\text{c-in}\ x\ u = 1\} \text{ then} (g\ x) \text{ else} 0\) ) for \( x\ u \)
  \item have \( g1\text{-is-pr}: g1 \in \text{PrimRec2} \)
  \item proof (unfold \( g1\text{-def} \), rule if-eq-is-pr2)
  \item show \( \text{c-in} \in \text{PrimRec2} \) by (rule c-in-is-pr)
\end{itemize}
next
\begin{itemize}
  \item show \( (\lambda x.\ y.\ 1) \in \text{PrimRec2} \) by (rule const-is-pr-2 [where \( ?n=1\)])
\end{itemize}
next
\begin{itemize}
  \item from \( g\text{-is-pr} \) show \( (\lambda x.\ y.\ g\ x) \in \text{PrimRec2} \) by prec
\end{itemize}
next
\begin{itemize}
  \item show \( (\lambda x.\ y.\ 0) \in \text{PrimRec2} \) by (rule const-is-pr-2 [where \( ?n=0\)])
\end{itemize}
qed

define \( f\ ) where \( f\ u = \text{sum}\ (\lambda x.\ g1\ x\ u)\ \{i.\ (i::\text{nat}) < u\} \) for \( u \)
define \( f1\ ) where \( f1\ u\ v = \text{sum}\ (\lambda x.\ g1\ x\ v)\ \{i.\ (i::\text{nat}) < u\} \) for \( u\ v \)
from \( g1\text{-is-pr} \) have \( (\lambda (u::\text{nat})\ v.\ \text{sum}\ (\lambda x.\ g1\ x\ v)\ \{i.\ (i::\text{nat}) < u\}) \in \text{PrimRec2} \)
  by (rule sum-of-pr-is-pr2)
with \( f1\text{-def} \) \( \{\text{abs-def}\} \) have \( f1\text{-is-pr}: f1 \in \text{PrimRec2} \) by auto
from \( f\text{-def} \) \( f1\text{-def} \) have \( f\ f1: f = (\lambda u.\ f1\ u\ u) \) by auto
from \( f1\text{-is-pr} \) have \( (\lambda u.\ f1\ u\ u) \in \text{PrimRec1} \) by prec
with \( f\ f1 \) have \( f\text{-is-pr}: f \in \text{PrimRec1} \) by auto
have f-is-result: \( f = (\lambda \ u. \ \text{sum} \ (\text{nat-to-set} \ u)) \)

proof

fix \( u \) show \( f \ u = \text{sum} \ (\text{nat-to-set} \ u) \)

proof –

define \( U \) where \( U = \{i. \ i < u\} \)
define \( A \) where \( A = \{x \in U. \ \text{c-in} \ x \ u = 1\} \)
define \( B \) where \( B = \{x \in U. \ \text{c-in} \ x \ u \neq 1\} \)

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 A-def B-def have A-B: \( A \cap B = \{\} \) by auto

from B-def g1-def have B-z: \( \text{sum} \ (\lambda x. \ g1 x \ u) \ B = 0 \) by auto

have u-in-U: \( \text{nat-to-set} \ u \subseteq U \) by (unfold \( U \)-def, rule nat-to-set-upper-bound2)

from u-in-U x-in-u-eq A-def have A-u: \( A = \text{nat-to-set} \ u \) by auto

from A-u x-in-u-eq g1-def have A-res: \( \text{sum} \ (\lambda x. \ g1 x \ u) \ A = \text{sum} \ (\text{nat-to-set} \ u) \) by auto

also from B-def g1-def have B-z: \( \text{sum} \ (\lambda x. \ g1 x \ u) \ B = 0 \) by auto

also from U-def A-def B-def have U-A-B: \( U = A \cup B \) by auto

also from A-finite B-finite A-B have ... = \( \text{sum} \ (\lambda x. \ g1 x \ u) \ A + \text{sum} \ (\lambda x. \ g1 x \ u) \ B \) by (rule sum.union-disjoint)

also from B-z have ... = \( \text{sum} \ (\lambda x. \ g1 x \ u) \ A + \text{sum} \ (\lambda x. \ g1 x \ u) \ B \) by auto

finally show \( \text{thesis} \) by auto

qed

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 –

define \( g \) :: nat \Rightarrow nat where \( g \ x = 1 \) for \( x \)
have g-is-pr: \( g \in \text{PrimRec}1 \) by (unfold \( g \)-def, rule const-is-pr)
have c-card = (\lambda u. \ \text{sum} \ (\text{nat-to-set} \ u))

proof

fix \( u \) show c-card \( u \) = \( \text{sum} \ (\text{nat-to-set} \ u) \) by (unfold c-card-def, unfold g-def, rule card-eq-sum)

qed

moreover from g-is-pr have (\lambda u. \ \text{sum} \ (\text{nat-to-set} \ u)) \in \text{PrimRec}1 \ by (rule sum-is-pr)

ultimately show \( \text{thesis} \) by auto

qed

definition
c-insert :: nat -> nat -> nat where
c-insert = (λ x u. if c-in x u = 1 then u else u + 2ˆx)

lemma c-insert-is-pr: c-insert ∈ 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 + 2ˆx) ∈ PrimRec2 by prec
qed

lemma [simp]: set-to-nat (nat-to-set u) = u
proof –
  define D where D = nat-to-set u
  from D-def nat-to-set-is-finite have D-finite: finite D by auto
  then have set-to-nat (set-to-nat D) = D by (rule nat-to-set-srj)
  with D-def have set-to-nat D = nat-to-set u by auto
  then have set-to-nat D = u by (rule nat-to-set-inj)
  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 + 2ˆx
proof –
  assume A: x /∈ nat-to-set u
  define D where D = nat-to-set u
  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)ˆx
  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 ... = 2ˆx + set-to-nat D by auto
  finally have set-to-nat (D ∪ {x}) = set-to-nat D + 2ˆx 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)
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
  define D where 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
next
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 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

definition
c-union :: nat ⇒ nat ⇒ nat where
c-union = (λ u v. set-to-nat (nat-to-set u ∪ nat-to-set v))

theorem c-union-is-pr: c-union ∈ PrimRec2
proof −
define f where f y x = set-to-nat ((nat-to-set (c-fst x)) ∪ \{z ∈ nat-to-set (c-snd x). z < y\})
for y x
have f-is-pr: f ∈ PrimRec2
proof −
define g where g = c-fst
from c-fst-is-pr g-def have g-is-pr: g ∈ PrimRec1 by auto
define h where h a b c = (if c-in a (c-snd c) = 1 then c-insert a b else b) for a b c
from c-in-is-pr c-insert-is-pr have h-is-pr: h ∈ PrimRec3 unfolding h-def by prec
have f-at-0: ∀ 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: ∀ 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 ∈ (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 ∈ nat-to-set (c-snd x). z < Suc y} = {z ∈ nat-to-set (c-snd x). z < y} ∪ \{y\} by auto
    from nat-to-set-is-finite have S4: finite ((nat-to-set (c-fst x)) ∪ \{z ∈ 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 ∈ 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 ∈ nat-to-set (c-snd x). z < Suc y\}) by simp
also from S3 have \ldots = \text{set-to-nat} \left( \left( \left( \text{nat-to-set} \ (c \cdot \text{fst} \ x) \right) \cup \{ \text{nat-to-set} \ (c \cdot \text{snd} \ x), \ z < y \} \right) \cup \{ y \} \right) \text{ by auto}

also from S5 have \ldots = \text{set-to-nat} \ (\text{nat-to-set} \ (f \ y \ x) \cup \{ y \}) \text{ by auto}

also have \ldots = \text{c-insert} \ y \ (f \ y \ x) \text{ by } \text{(simp add: c-insert-df)}

finally show \ ?thesis \text{ by } \text{(simp add: S2)}

next

assume A: \ \lnot \ c-in \ y \ (c \cdot \text{snd} \ x) = 1
then have S1: \ y \notin \ (\text{nat-to-set} \ (c \cdot \text{snd} \ x)) \text{ by } \text{(simp add: x-in-a-eq)}
from A \ h-def \ have S2: \ h \ y \ (f \ y \ x) = f \ y \ x \text{ by auto}

have S3: \ \{ \text{nat-to-set} \ (c \cdot \text{snd} \ x), \ z < \text{Suc} \ y \} = \{ \text{nat-to-set} \ (c \cdot \text{snd} \ x), \ z < y \}

proof

- have \{ \text{nat-to-set} \ (c \cdot \text{snd} \ x), \ z < \text{Suc} \ y \} = \{ \text{nat-to-set} \ (c \cdot \text{snd} \ x), \ z = y \}
  
  by \text{auto}

  with S1 show \ ?thesis \text{ by auto}

qed

from \text{nat-to-set-is-finite} \ have S4: \ \text{finite} \ \left( \left( \text{nat-to-set} \ (c \cdot \text{fst} \ x) \right) \cup \{ \text{nat-to-set} \ (c \cdot \text{snd} \ x), \ z < y \} \right) \text{ by auto}

with \text{nat-to-set-srj} \ f-def \ have S5: \ \text{nat-to-set} \ (f \ y \ x) = \text{nat-to-set} \ (c \cdot \text{fst} \ x) \text{ by simp}

also from S3 have \ldots = \text{set-to-nat} \left( \left( \text{nat-to-set} \ (c \cdot \text{fst} \ x) \right) \cup \{ \text{nat-to-set} \ (c \cdot \text{snd} \ x), \ z < y \} \right) \text{ by auto}

also have \ldots = f \ y \ x \text{ by simp}

finally show \ ?thesis \text{ by } \text{(simp add: S2)}

qed

from \text{g-is-pr} \ \text{h-is-pr} \ \text{f-at-0} \ \text{f-at-Suc} \ show \ ?thesis \text{ by } \text{(rule pr-rec-scheme)}

qed

\text{define union where} \ union \ u \ v = f \ v \ \text{(c-pair} \ u \ v) \text{ for} \ u \ v

\text{from} \text{f-is-pr} \ have \ \text{union-is-pr}: \ \text{union} \in \text{PrimRec2} \ \text{unfolding} \ \text{union-def} \text{ by} \ \text{prec}

have \ \bigwedge u \ v. \ \text{union} \ u \ v = \text{set-to-nat} \ (\text{nat-to-set} \ u \cup \text{nat-to-set} \ v)

proof

- fix u v show \ \text{union} \ u \ v = \text{set-to-nat} \ (\text{nat-to-set} \ u \cup \text{nat-to-set} \ v)

proof

- from \text{nat-to-set-upper-bound1} \ have \ \{ \text{nat-to-set} \ v, \ z < v \} = \text{nat-to-set} \ v

by \text{auto}

  with \text{union-def} \ f-def \ show \ ?thesis \text{ by auto}

qed

qed

then have \union = (\lambda u v. \text{set-to-nat} \ (\text{nat-to-set} \ u \cup \text{nat-to-set} \ v)) \text{ by } \text{(simp add: ext)}

with \text{c-union-def} \ have \ \text{c-union} = \text{union} \text{ by simp}

with \text{union-is-pr} \ show \ ?thesis \text{ by simp}

qed
definition
c-diff :: nat ⇒ nat ⇒ nat where
c-diff = (λ u v. set-to-nat (nat-to-set u − nat-to-set v))

definition

theorem c-diff-is-pr: c-diff ∈ PrimRec2
proof –
define f where f y x = set-to-nat ((nat-to-set (c-fst x)) − {z ∈ nat-to-set (c-snd x). z < y})
  for y x
have f-is-pr: f ∈ PrimRec2
proof –
define g where g = c-fst
from c-fst-is-pr g-def have g-is-pr: g ∈ PrimRec1 by auto
define h where h a b c = (if c-in a (c-snd c) = 1 then c-remove a b else b)
for a b c
from c-in-is-pr c-remove-is-pr have h-is-pr: h ∈ PrimRec3 unfolding h-def
by prec
have f-at-0: ∀ 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: ∀ 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 ∈ (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 ∈ nat-to-set (c-snd x). z < y} ∪ {y}) =
  ((nat-to-set (c-fst x)) − ({z ∈ nat-to-set (c-snd x). z < y}) − {y}) by auto
  then have lm1: set-to-nat (nat-to-set (c-fst x)) − ({z ∈ nat-to-set (c-snd x) . z < y} ∪ {y}) =
  set-to-nat (nat-to-set (c-fst x)) − {z ∈ nat-to-set (c-snd x). z < y} − {y}) by auto
  from S1 have S3: {z ∈ nat-to-set (c-snd x). z < Suc y} = {z ∈ nat-to-set (c-snd x). z < y} by auto
  from nat-to-set-is-finite have S4: finite ((nat-to-set (c-fst x)) − {z ∈ 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 ∈ nat-to-set (c-snd x), z < Suc y} by simp
  also from S3 have ... = set-to-nat ((nat-to-set (c-fst x)) − {z ∈ nat-to-set (c-snd x). z < y} ∪ {y}) by auto
  also have ... = set-to-nat (((nat-to-set (c-fst x)) − {z ∈ nat-to-set (c-snd x). z < y}) − {y}) by (rule lm1)
  also from S5 have ... = set-to-nat (f y x) − {y}) by auto
  also have ... = c-remove y (f y x) by (simp add: c-remove-df)

96
finally show thesis by (simp add: S2)

next

assume A: ¬ c-in y (c-snd x) = 1
then have S1: y ∉ (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 ∈ nat-to-set (c-snd x). z < Suc y} = {z ∈ nat-to-set (c-snd x). z < y}
  proof
  show {z ∈ nat-to-set (c-snd x). z < Suc y} = {z ∈ nat-to-set (c-snd x). z < y}
  proof
  show ?thesis by auto
  qed
  qed
  qed from nat-to-set-is-finite have S4: finite ((nat-to-set (c-fst x)) − {z ∈ nat-to-set (c-snd x). z < y}) by auto
  with S1 show thesis by auto
  qed
  qed
  qed
from g-is-pr h-is-pr f-at-0 f-at-Suc show thesis by (rule pr-rec-scheme)
  qed
  define diff where diff u v = f v (c-pair u v) for u v
from f-is-pr have diff-is-pr: diff ∈ PrimRec2 unfolding diff-def by prec
have ∫ u v. diff u v = set-to-nat (nat-to-set u − nat-to-set v)
  proof
  fix u v show diff u v = set-to-nat (nat-to-set u − nat-to-set v)
  proof
  show ?thesis by auto
  qed
  qed
  qed
then have diff = (λ u v. set-to-nat (nat-to-set u − nat-to-set v)) by (simp add: ext)
  with c-diff-def have c-diff = diff by simp
  with diff-is-pr show thesis by simp
  qed

definition
c-intersect :: nat ⇒ nat ⇒ nat where
c-intersect = (λ u v. set-to-nat (nat-to-set u ∩ nat-to-set v))
proof -
  define \( f \) where \( f\ u\ v = \text{c-diff}\ (\text{c-union}\ u\ v)\ (\text{c-union}\ (\text{c-diff}\ u\ v)\ (\text{c-diff}\ v\ u))\)
for \( u\ v \)
from \( \text{c-diff-is-pr}\ \text{c-union-is-pr} \) have \( f\ \in\ \text{PrimRec2} \)
unfolding \( f\)-def by pre
have \( \text{f-is-pr} \):
proof -
  fix \( u\ v \)
  show \( f\ u\ v = \text{c-intersect}\ u\ v \)
  proof -
    let \(?A\) = \( \text{nat-to-set}\ u \)
    let \(?B\) = \( \text{nat-to-set}\ v \)
    have \(?A\)-fin: \( \text{finite}\ ?A \) by (rule \( \text{nat-to-set-is-finite} \))
    have \(?B\)-fin: \( \text{finite}\ ?B \) by (rule \( \text{nat-to-set-is-finite} \))
    have \(?S1\): \( \text{c-union}\ u\ v = \text{set-to-nat}\ ((?A\ ∪\ ?B)\ \cup\ (?B\ −\ ?A)) \)
      by (simp add: \( \text{c-union-def} \))
    have \(?S2\): \( \text{c-diff}\ u\ v = \text{set-to-nat}\ ((?A\ −\ ?B)\ \cup\ (?B\ −\ ?A)) \)
      by (simp add: \( \text{c-diff-def} \))
    have \(?S3\): \( \text{c-diff}\ v\ u = \text{set-to-nat}\ ((?B\ −\ ?A)\ \cup\ (?A\ −\ ?B)) \)
      by (simp add: \( \text{c-diff-def} \))
    from \(?S2\) \(?A\)-fin \(?B\)-fin have \(?S4\): \( \text{nat-to-set}\ ((\text{c-diff}\ u\ v)) = ?A\ −\ ?B \)
      by (simp add: \( \text{nat-to-set-srj} \))
    from \(?S3\) \(?A\)-fin \(?B\)-fin have \(?S5\): \( \text{nat-to-set}\ ((\text{c-diff}\ v\ u)) = ?B\ −\ ?A \)
      by (simp add: \( \text{nat-to-set-srj} \))
    from \(?S4\) \(?S5\) have \(?S6\): \( \text{nat-to-set}\ ((\text{c-union}\ (\text{c-diff}\ u\ v)\ (\text{c-diff}\ v\ u))) = (?A\ −\ ?B)\ \cup\ (?B\ −\ ?A) \)
      by (simp add: \( \text{nat-to-set-srj} \))
    from \(?S6\) \(?S8\) have \(?S9\): \( f\ u\ v = \text{set-to-nat}\ (?(?A\ ∪\ ?B) − ((?A\ −\ ?B)\ \cup\ (?B\ −\ ?A))) \)
      by (simp add: \( \text{c-diff-def}\ ))
      with \(?S9\) have \(?S10\): \( ?A\ \cap\ ?B = (?(?A\ ∪\ ?B) − ((?A\ −\ ?B)\ \cup\ (?B\ −\ ?A)) \)
        by auto
    with \(?S9\) have \(?S11\): \( f\ u\ v = \text{set-to-nat}\ (?(?A\ ∩\ ?B) \)
      by auto
    with \(?S11\) show \(?\text{thesis} \)
      by auto
    qed
  qed
then have \( f = \text{c-intersect} \)
with \( \text{f-is-pr} \) show \(?\text{thesis} \)
  by auto
qed

6 The function which is universal for primitive recursive functions of one variable

theory \( \text{PRecUnGr} \)
imports \( \text{PRecFun2}\ \text{PRecList} \)
begin

We introduce a particular function which is universal for primitive recursive
functions of one variable.

**definition**

\[ g\text{-}comp :: \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \]  
\[ g\text{-}comp \text{ c-ls} \text{ key} = ( \]
\[ \text{let } n = \text{c-fst key}; x = \text{c-snd key}; m = \text{c-snd} n; \]
\[ m1 = \text{c-fst} m; m2 = \text{c-snd} m \text{ in} \]
\[ \quad - \text{We have key} = <n, x>; n = <?, m>; m = <m1, m2>. \]
\[ \quad \text{if c-assoc-have-key c-ls (c-pair m2 x)} = 0 \text{ then} \]
\[ \quad \text{let } y = \text{c-assoc-value c-ls (c-pair m2 x)} \text{ in} \]
\[ \quad \text{if c-assoc-have-key c-ls (c-pair m1 y)} = 0 \text{ then} \]
\[ \quad \text{let } z = \text{c-assoc-value c-ls (c-pair m1 y)} \text{ in} \]
\[ \quad \text{c-cons (c-pair key z) c-ls} \]
\[ \quad \text{else c-ls} \]
\[ \) \]
\[ \text{else c-ls} \)

**definition**

\[ g\text{-}pair :: \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \]  
\[ g\text{-}pair \text{ c-ls} \text{ key} = ( \]
\[ \text{let } n = \text{c-fst key}; x = \text{c-snd key}; m = \text{c-snd} n; \]
\[ m1 = \text{c-fst} m; m2 = \text{c-snd} m \text{ in} \]
\[ \quad - \text{We have key} = <n, x>; n = <?, m>; m = <m1, m2>. \]
\[ \quad \text{if c-assoc-have-key c-ls (c-pair m1 x)} = 0 \text{ then} \]
\[ \quad \text{let } y1 = \text{c-assoc-value c-ls (c-pair m1 x)} \text{ in} \]
\[ \quad \text{if c-assoc-have-key c-ls (c-pair m2 x)} = 0 \text{ then} \]
\[ \quad \text{let } y2 = \text{c-assoc-value c-ls (c-pair m2 x)} \text{ in} \]
\[ \quad \text{c-cons (c-pair key (c-pair y1 y2)) c-ls} \]
\[ \quad \text{else c-ls} \]
\[ \) \]
\[ \text{else c-ls} \)

**definition**

\[ g\text{-}rec :: \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \]  
\[ g\text{-}rec \text{ c-ls} \text{ key} = ( \]
\[ \text{let } n = \text{c-fst key}; x = \text{c-snd key}; m = \text{c-snd} n; \]
\[ m1 = \text{c-fst} m; m2 = \text{c-snd} m; y1 = \text{c-fst} x; x1 = \text{c-snd} x \text{ in} \]
\[ \quad - \text{We have key} = <n, x>; n = <?, m>; m = <m1, m2>; x = <y1, x1>. \]
\[ \quad \text{if } y1 = 0 \text{ then} \]
\[ \quad \text{c-cons (c-pair key (c-assoc-value c-ls (c-pair m1 x1))) c-ls} \]
\[ \quad \text{else c-ls} \]
\[ \) \]
\[ \text{else} \]
\[ \text{let } y2 = y1-(1::\text{nat}) \text{ in} \]
\[ \text{if c-assoc-have-key c-ls (c-pair n (c-pair y2 x1))} = 0 \text{ then} \]

99
let \( t1 = \text{c-assoc-value} \) \( c-\text{ls} \) \((\text{c-pair} \ n \ (\text{c-pair} \ y2 \ x1)) \); \( t2 = \text{c-pair} \ c-\text{ls} \)

let \( t1 \) in

if \( \text{c-assoc-have-key} \ c-\text{ls} \) \((\text{c-pair} \ m2 \ t2) = 0 \) then

\( \text{c-cons} \ (\text{c-pair} \ \text{key} \ (\text{c-assoc-value} \ c-\text{ls} \ (\text{c-pair} \ m2 \ t2))) \) \( c-\text{ls} \)

else \( c-\text{ls} \)

else \( c-\text{ls} \)


definition g-step :: \( \text{nat} \rightarrow \text{nat} \rightarrow \text{nat} \) where

g-step \( c-\text{ls} \) \( \text{key} \) = ( let \( n = \text{c-fst} \) \( \text{key} \); \( x = \text{c-snd} \) \( \text{key} \); \( n1 = (\text{c-fst} \ n) \mod 7 \) in

if \( n1 = 0 \) then \( \text{c-cons} \ (\text{c-pair} \ \text{key} \ 0) \) \( c-\text{ls} \) else

if \( n1 = 1 \) then \( \text{c-cons} \ (\text{c-pair} \ \text{key} \ (\text{Suc} \ x)) \) \( c-\text{ls} \) else

if \( n1 = 2 \) then \( \text{c-cons} \ (\text{c-pair} \ \text{key} \ (\text{c-fst} \ x)) \) \( c-\text{ls} \) else

if \( n1 = 3 \) then \( \text{c-cons} \ (\text{c-pair} \ \text{key} \ (\text{c-snd} \ x)) \) \( c-\text{ls} \) else

if \( n1 = 4 \) then \( \text{g-comp} \ c-\text{ls} \) \( \text{key} \) else

if \( n1 = 5 \) then \( \text{g-pair} \ c-\text{ls} \) \( \text{key} \) else

if \( n1 = 6 \) then \( \text{g-rec} \ c-\text{ls} \) \( \text{key} \) else

\( c-\text{ls} \)

)

definition pr-gr :: \( \text{nat} \rightarrow \text{nat} \) where

pr-gr-def: \( \text{pr-gr} = \text{PrimRecOp1} \) \( 0 \) \((\lambda a b. \text{g-step} \ (\text{c-fst} \ a)) \)

lemma pr-gr-at-0: \( \text{pr-gr} \) \( 0 = 0 \) by \((\text{simp add: pr-gr-def})\)

lemma pr-gr-at-Suc: \( \text{pr-gr} \) \((\text{Suc} \ x) = \text{g-step} \) \((\text{pr-gr} \) \( x \)) \((\text{c-fst} \ x) \) by \((\text{simp add: pr-gr-def})\)

definition univ-for-pr :: \( \text{nat} \rightarrow \text{nat} \) where

univ-for-pr = \text{pr-conv-2-to-1} \text{nat-to-pr}

theorem univ-is-not-pr: \( \text{univ-for-pr} \notin \text{PrimRec1} \)

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

assume \( \neg \text{univ-for-pr} \notin \text{PrimRec1} \) then have \( A1: \text{univ-for-pr} \in \text{PrimRec1} \) by \( \text{simp} \)

let \( ?f = \lambda n. \) \( \text{univ-for-pr} \) \((\text{c-pair} \ n \ n) \) + 1

let \( ?n0 = \text{index-of-pr} ?f \)

from \( A1 \) have \( S1: ?f \in \text{PrimRec1} \) by \text{prec}

then have \( S2: \text{nat-to-pr} \) \( ?n0 = ?f \) by \((\text{rule index-of-pr-is-real})\)

then have \( S3: \text{nat-to-pr} \) \( ?n0 \) \( ?n0 = ?f \) ?n0 by \( \text{simp} \)

have \( S4: ?f \) \( ?n0 = \text{univ-for-pr} \) \((\text{c-pair} ?n0 \ ?n0) \) + 1 by \( \text{simp} \)

from \( S3 \) \( S4 \) show \( \text{False} \) by \((\text{simp add: univ-for-pr-def pr-conv-2-to-1-def})\)
definition
\(c\text{-is-sub-fun} :: \text{nat} \Rightarrow (\text{nat} \Rightarrow \text{nat}) \Rightarrow \text{bool}\)
where
\[c\text{-is-sub-fun} \; ls \; f \quad\iff\quad (\forall x. \; c\text{-assoc-have-key} \; ls \; x = 0 \implies c\text{-assoc-value} \; ls \; x = f \; x)\]

lemma \(c\text{-is-sub-fun-lm-1}::\begin{array}{ll}c\text{-is-sub-fun} \; ls \; f; \; c\text{-assoc-have-key} \; ls \; x = 0 \end{array} \implies c\text{-assoc-value} \; ls \; x = f \; x\)
apply (unfold \(c\text{-is-sub-fun-def}\))
apply (auto)
done

lemma \(c\text{-is-sub-fun-lm-2}::c\text{-is-sub-fun} \; ls \; f \implies c\text{-is-sub-fun} \; c\text{-cons} \; (c\text{-pair} \; x \; (f \; x)) \; ls \; f\)
proof
  assume \(A1::c\text{-is-sub-fun} \; ls \; f\)
  show \(\text{thesis}\)
proof (unfold \(c\text{-is-sub-fun-def}\), rule allI, rule implI)
  fix \(xa\) assume \(A2::c\text{-assoc-have-key} \; (c\text{-cons} \; (c\text{-pair} \; x \; (f \; x)) \; ls) \; xa = 0\) show \(c\text{-assoc-value} \; (c\text{-cons} \; (c\text{-pair} \; x \; (f \; x)) \; ls) \; xa = f \; xa\)
proof cases
  assume \(C1::xa = x\) then show \(c\text{-assoc-value} \; (c\text{-cons} \; (c\text{-pair} \; x \; (f \; x)) \; ls) \; xa = f \; xa\) by (simp add: PRecList.c-assoc-lm-2)
next
  assume \(C2::\neg \; xa = x\)
  then have \(S1::c\text{-assoc-have-key} \; (c\text{-cons} \; (c\text{-pair} \; x \; (f \; x)) \; ls) \; xa = c\text{-assoc-have-key} \; ls \; xa\) by (rule c-assoc-lm-3)
  from \(C2\) have \(S2::c\text{-assoc-value} \; (c\text{-cons} \; (c\text{-pair} \; x \; (f \; x)) \; ls) \; xa = c\text{-assoc-value} \; ls \; xa\) by (rule c-assoc-lm-4)
  from \(A2 \; S1\) have \(S3::c\text{-assoc-have-key} \; ls \; xa = 0\) by simp
  from \(A1 \; S3\) have \(c\text{-assoc-value} \; ls \; xa = f \; xa\) by (rule c-is-sub-fun-lm-1)
  with \(S2\) show \(\text{thesis}\) by simp
qed
qed

lemma \(\text{mod7-lm}::(n::\text{nat}) \; \text{mod} \; 7 = 0 \lor (n::\text{nat}) \; \text{mod} \; 7 = 1 \lor (n::\text{nat}) \; \text{mod} \; 7 = 2 \lor (n::\text{nat}) \; \text{mod} \; 7 = 3 \lor (n::\text{nat}) \; \text{mod} \; 7 = 4 \lor (n::\text{nat}) \; \text{mod} \; 7 = 5 \lor (n::\text{nat}) \; \text{mod} \; 7 = 6\) by arith

lemma \(\text{nat-to-sch-at-pos}::x > 0 \implies \text{nat-to-sch} \; x = (\text{let} \; u = (c\text{-fst} \; x) \; \text{mod} \; 7; \; v = c\text{-snd} \; x; \; v1 = c\text{-fst} \; v; \; v2 = c\text{-snd} \; v; \; \text{sch1} = \text{nat-to-sch} \; v1; \; \text{sch2} = \text{nat-to-sch} \; v2\; \text{in} \; \text{loc-f} \; u \; \text{sch1} \; \text{sch2})\)
proof

assume $A$: $x > 0$
show $?thesis$

proof cases
  assume $A1$: $x = 1$
  then have $S1$: $c\text{-}fst \; x = 0$
  proof
    have $1 = c\text{-}pair \; 0 \; 1$ by (simp add: c\text{-}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 $S2$: $nat\text{-}to\text{-}sch \; x = Base\text{-}zero$ by simp
  from $S1 \; S2$ show $nat\text{-}to\text{-}sch \; x = (let \; u = (c\text{-}fst \; x) \; mod \; 7; \; v = c\text{-}snd \; x; \; v1 = c\text{-}fst \; v; \; v2 = c\text{-}snd \; v; \; sch1 = nat\text{-}to\text{-}sch \; v1; \; sch2 = nat\text{-}to\text{-}sch \; v2 \; in \; loc\text{-}f \; u \; sch1 \; sch2)$
    apply(insert $S1 \; S2$)
    apply(simp add: Let\_def loc\_f\_def)
    done
next
  assume $\neg \; x = 1$
  from $A$ this have $A2$: $x > 1$ by simp
  from this have $nat\text{-}to\text{-}sch \; x = (let \; u = (c\text{-}fst \; x) \; mod \; 7; \; v = c\text{-}snd \; x; \; v1 = c\text{-}fst \; v; \; v2 = c\text{-}snd \; v; \; sch1 = nat\text{-}to\text{-}sch \; v1; \; sch2 = nat\text{-}to\text{-}sch \; v2 \; in \; loc\text{-}f \; u \; sch1 \; sch2)$
    by (rule loc\_srj\_lm\_2)
  from this show $nat\text{-}to\text{-}sch \; x = (let \; u = (c\text{-}fst \; x) \; mod \; 7; \; v = c\text{-}snd \; x; \; v1 = c\text{-}fst \; v; \; v2 = c\text{-}snd \; v; \; sch1 = nat\text{-}to\text{-}sch \; v1; \; sch2 = nat\text{-}to\text{-}sch \; v2 \; in \; loc\text{-}f \; u \; sch1 \; sch2)$
    by (simp add: mod7\_def)
  qed
qed

lemma $nat\text{-}to\text{-}sch\text{-}0$: $c\text{-}fst \; n \; mod \; 7 = 0 \Rightarrow nat\text{-}to\text{-}sch \; n = Base\text{-}zero$
proof
  assume $A$: $c\text{-}fst \; n \; mod \; 7 = 0$
  show $?thesis$
  proof cases
    assume $n = 0$
    then show $nat\text{-}to\text{-}sch \; n = Base\text{-}zero$ by simp
  next
    assume $\neg \; n = 0$ then have $n > 0$ by simp
    then have $nat\text{-}to\text{-}sch \; n = (let \; u = (c\text{-}fst \; n) \; mod \; 7; \; v = c\text{-}snd \; n; \; v1 = c\text{-}fst \; v; \; v2 = c\text{-}snd \; v; \; sch1 = nat\text{-}to\text{-}sch \; v1; \; sch2 = nat\text{-}to\text{-}sch \; v2 \; in \; loc\text{-}f \; u \; sch1 \; sch2)$
      by (rule nat\text{-}to\text{-}sch\text{-}at\_pos)
    with $A$ show $nat\text{-}to\text{-}sch \; n = Base\text{-}zero$ by (simp add: Let\_def loc\_f\_def)
  qed
qed

lemma $loc\_lm\text{-}1$: $c\text{-}fst \; n \; mod \; 7 \neq 0 \Rightarrow n > 0$
proof
  assume $A$: $c\text{-}fst \; n \; mod \; 7 \neq 0$

102
have \( n = 0 \rightarrow False \)
proof 
  assume \( n = 0 \)
  then have \( c\text{-fst n mod 7 = 0} \) by (simp add: 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: \( c\text{-fst n mod 7 \neq 0} \rightarrow nat\text{-to-sch n = (let } u = (c\text{-fst n) mod 7;} \ v = c\text{-snd n}; \ v1 = c\text{-fst v}; v2 = c\text{-snd v}; \ sch1 = nat\text{-to-sch v1}; \ sch2 = nat\text{-to-sch v2 in loc-f u sch1 sch2)} \)
proof 
  assume \( c\text{-fst n mod 7 \neq 0} \)
  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: \( c\text{-fst n mod 7 = 1} \rightarrow nat\text{-to-sch n = Base-suc} \)
proof 
  assume \( A1: c\text{-fst n mod 7 = 1} \)
  then have \( nat\text{-to-sch n = (let } u = (c\text{-fst n) mod 7;} \ v = c\text{-snd n}; \ v1 = c\text{-fst v}; v2 = c\text{-snd v}; \ sch1 = nat\text{-to-sch v1}; \ sch2 = nat\text{-to-sch v2 in loc-f u sch1 sch2)} \) by (simp add: loc-lm-2)
  with \( A1 \) show \( nat\text{-to-sch n = Base-suc} \) by (simp add: Let-def loc-f-def)
qed

lemma nat-to-sch-2: \( c\text{-fst n mod 7 = 2} \rightarrow nat\text{-to-sch n = Base-fst} \)
proof 
  assume \( A1: c\text{-fst n mod 7 = 2} \)
  then have \( nat\text{-to-sch n = (let } u = (c\text{-fst n) mod 7;} \ v = c\text{-snd n}; \ v1 = c\text{-fst v}; v2 = c\text{-snd v}; \ sch1 = nat\text{-to-sch v1}; \ sch2 = nat\text{-to-sch v2 in loc-f u sch1 sch2)} \) by (simp add: loc-lm-2)
  with \( A1 \) show \( nat\text{-to-sch n = Base-fst} \) by (simp add: Let-def loc-f-def)
qed

lemma nat-to-sch-3: \( c\text{-fst n mod 7 = 3} \rightarrow nat\text{-to-sch n = Base-snd} \)
proof 
  assume \( A1: c\text{-fst n mod 7 = 3} \)
  then have \( nat\text{-to-sch n = (let } u = (c\text{-fst n) mod 7;} \ v = c\text{-snd n}; \ v1 = c\text{-fst v}; v2 = c\text{-snd v}; \ sch1 = nat\text{-to-sch v1}; \ sch2 = nat\text{-to-sch v2 in loc-f u sch1 sch2)} \) by (simp add: loc-lm-2)
  with \( A1 \) show \( nat\text{-to-sch n = Base-snd} \) by (simp add: Let-def loc-f-def)
qed

lemma nat-to-sch-4: \( c\text{-fst n mod 7 = 4} \rightarrow nat\text{-to-sch n = Comp-op (nat-to-sch (c-fst (c-snd n))) (nat-to-sch (c-snd (c-snd n)))} \)
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

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 $S_1$: nat-to-pr $n \times =\text{sch-to-pr (nat-to-sch n)} \times \text{by (simp add: nat-to-pr-def)}$
from $A$ have $S_2$: nat-to-sch $n = \text{Base-fst}$ by (rule nat-to-sch-2)
from $S_1$ $S_2$ show ?thesis by simp
qed

lemma nat-to-pr-lm-3: c-fst $n \times \mod 7 = 3 \Rightarrow \text{nat-to-pr n x = c-snd x}$
proof
  assume $A$: c-fst $n \times \mod 7 = 3$
  have $S_1$: nat-to-pr $n \times =\text{sch-to-pr (nat-to-sch n)} \times \text{by (simp add: nat-to-pr-def)}$
  from $A$ have $S_2$: nat-to-sch $n = \text{Comp-op (nat-to-sch (c-fst (c-snd n)) (nat-to-sch (c-snd (c-snd n))) (nat-to-sch (c-snd (c-snd (c-snd n))))) \times \text{by simp}$
  from $S_1$ $S_2$ have $S_3$: nat-to-pr $n \times =\text{sch-to-pr (Comp-op (nat-to-sch (c-fst (c-snd n)) (nat-to-sch (c-snd (c-snd n))) (nat-to-sch (c-snd (c-snd (c-snd n)))) \times \text{by simp}$
  from $S_3$ show ?thesis by (simp add: nat-to-pr-def)
qed

lemma nat-to-pr-lm-4: c-fst $n \times \mod 7 = 4 \Rightarrow \text{nat-to-pr n x = (c-fst (c-snd (c-snd n))) (nat-to-pr (c-snd (c-snd n)))) x}$
proof
  assume $A$: c-fst $n \times \mod 7 = 4$
  have $S_1$: nat-to-pr $n \times =\text{sch-to-pr (nat-to-sch n)} \times \text{by (simp add: nat-to-pr-def)}$
  from $A$ have $S_2$: nat-to-sch $n = \text{Pair-op (nat-to-sch (c-fst (c-snd n)) (nat-to-sch (c-snd (c-snd n))) (nat-to-sch (c-snd (c-snd (c-snd n)))) \times \text{by simp}$
  from $S_1$ $S_2$ have $S_3$: nat-to-pr $n \times =\text{sch-to-pr (Pair-op (nat-to-sch (c-fst (c-snd n)) (nat-to-sch (c-snd (c-snd n))) (nat-to-sch (c-snd (c-snd (c-snd n)))) \times \text{by simp}$
  from $S_3$ show ?thesis by (simp add: nat-to-pr-def)
qed

lemma nat-to-pr-lm-5: c-fst $n \times \mod 7 = 5 \Rightarrow \text{nat-to-pr n x = (c-f-pair (c-fst (c-snd n))) (nat-to-pr (c-snd (c-snd n)))) x}$
proof
  assume $A$: c-fst $n \times \mod 7 = 5$
  have $S_1$: nat-to-pr $n \times =\text{sch-to-pr (nat-to-sch n)} \times \text{by (simp add: nat-to-pr-def)}$
  from $A$ have $S_2$: nat-to-sch $n = \text{Rec-op (nat-to-sch (c-fst (c-snd n)) (nat-to-sch (c-snd (c-snd n))) by (rule nat-to-sch-6)$
  from $S_1$ $S_2$ have $S_3$: nat-to-pr $n \times =\text{sch-to-pr (Rec-op (nat-to-sch (c-fst (c-snd n)) (nat-to-sch (c-snd (c-snd n)))) \times \text{by simp}$
  from $S_3$ show ?thesis by (simp add: nat-to-pr-def)
qed

lemma nat-to-pr-lm-6: c-fst $n \times \mod 7 = 6 \Rightarrow \text{nat-to-pr n x = (UnaryRecOp (nat-to-pr (c-fst (c-snd n))) (nat-to-pr (c-snd (c-snd n)))) x}$
proof
  assume $A$: c-fst $n \times \mod 7 = 6$
  have $S_1$: nat-to-pr $n \times =\text{sch-to-pr (nat-to-sch n)} \times \text{by (simp add: nat-to-pr-def)}$
  from $A$ have $S_2$: nat-to-sch $n = \text{Rec-op (nat-to-sch (c-fst (c-snd n)) by (rule nat-to-sch-6)$
  from $S_1$ $S_2$ have $S_3$: nat-to-pr $n \times =\text{sch-to-pr (Rec-op (nat-to-sch (c-fst (c-snd n)) (nat-to-sch (c-snd (c-snd n)))) \times \text{by simp}$
  from $S_3$ show ?thesis by (simp add: nat-to-pr-def)
qed
lemma univ-for-pr-lm-0: \( \text{c-fst (c-fst key) \ mod 7 = 0} \implies \text{univ-for-pr key = 0} \)
proof
  assume A: \( \text{c-fst (c-fst key) \ mod 7 = 0} \)
  have S1: \( \text{univ-for-pr key = nat-to-pr (c-fst key) (c-snd key)} \) 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-0)
qed

lemma univ-for-pr-lm-1: \( \text{c-fst (c-fst key) \ mod 7 = 1} \implies \text{univ-for-pr key = Suc (c-snd key)} \)
proof
  assume A: \( \text{c-fst (c-fst key) \ mod 7 = 1} \)
  have S1: \( \text{univ-for-pr key = nat-to-pr (c-fst key) (c-snd key)} \) 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-1)
qed

lemma univ-for-pr-lm-2: \( \text{c-fst (c-fst key) \ mod 7 = 2} \implies \text{univ-for-pr key = c-fst (c-snd key)} \)
proof
  assume A: \( \text{c-fst (c-fst key) \ mod 7 = 2} \)
  have S1: \( \text{univ-for-pr key = nat-to-pr (c-fst key) (c-snd key)} \) 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-2)
qed

lemma univ-for-pr-lm-3: \( \text{c-fst (c-fst key) \ mod 7 = 3} \implies \text{univ-for-pr key = c-snd (c-snd key)} \)
proof
  assume A: \( \text{c-fst (c-fst key) \ mod 7 = 3} \)
  have S1: \( \text{univ-for-pr key = nat-to-pr (c-fst key) (c-snd key)} \) 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-3)
qed

lemma univ-for-pr-lm-4: \( \text{c-fst (c-fst key) \ mod 7 = 4} \implies \text{univ-for-pr key = (nat-to-pr (c-fst (c-snd (c-fst key))) (nat-to-pr (c-snd (c-snd (c-fst key))))) (c-snd key))) \)
proof
  assume A: \( \text{c-fst (c-fst key) \ mod 7 = 4} \)
  have S1: \( \text{univ-for-pr key = nat-to-pr (c-fst key) (c-snd key)} \) 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-4)
qed

lemma univ-for-pr-lm-4-1: \( \text{c-fst (c-fst key) \ mod 7 = 4} \implies \text{univ-for-pr key = univ-for-pr (c-pair (c-fst (c-snd (c-fst key)))) (univ-for-pr (c-pair (c-snd (c-snd key)))))} \)
proof
  assume A: \( \text{c-fst (c-fst key) \ mod 7 = 4} \)
  have S1: \( \text{univ-for-pr key = nat-to-pr (c-fst key) (c-snd key)} \) 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-4-1)
proof
  assume A: c-fst (c-fst key) mod 7 = 4
  have S1: univ-for-pr key = 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 ?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: c-fst (c-fst key) mod 7 = 5 =⇒ univ-for-pr key = c-pair (c-fst (c-fst key)) (c-snd (c-snd key))
proof
  assume A: c-fst (c-fst key) mod 7 = 5
  have S1: univ-for-pr key = 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 ?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: [c-fst (c-fst key) mod 7 = 6; c-fst (c-snd key) = 0] =⇒ univ-for-pr key = univ-for-pr (c-pair (c-fst (c-snd key))) (c-snd (c-snd key))
proof
  assume A1: c-fst (c-fst key) mod 7 = 6
  assume A2: c-fst (c-snd key) = 0
  have S1: univ-for-pr key = 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 ?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: [c-fst (c-fst key) mod 7 = 6; c-fst (c-snd key) = Suc u] =⇒ univ-for-pr key = univ-for-pr (c-pair (c-snd key) (c-fst (c-snd key)))
proof
  assume A1: c-fst (c-fst key) mod 7 = 6
  assume A2: c-fst (c-snd key) = Suc u
  have S1: univ-for-pr key = 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 ?thesis
apply(simp add: nat-to-pr-lm-6 UnaryRecOp-def univ-for-pr-def pr-conv-2-to-1-def)
apply(simp add: pr-conv-1-to-3-def)
done
qed

lemma univ-for-pr-lm-6-3: [c-fst (c-fst key) mod 7 = 6; c-fst (c-snd key) ≠ 0] =⇒ univ-for-pr key = univ-for-pr

107
(c-pair (c-snd (c-snd (c-fst key))))
(c-pair (c-pair (c-fst (c-snd key)) − 1) (univ-for-pr (c-pair (c-fst key))
(c-pair (c-fst (c-snd key)) − 1) (c-snd (c-snd key))))
(c-snd (c-snd key)))

proof −
assume A1: c-fst (c-fst key) mod 7 = 6
assume A2: c-fst (c-snd key) ≠ 0 then have
A3: c-fst (c-snd key) > 0 by simp
let ?u = c-fst (c-snd key) − (1::nat)
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-snd (c-fst key)))
(c-pair (c-pair ?u (univ-for-pr (c-fst key)) (c-pair ?u (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: [ [ c-fst (c-fst key) mod 7 = 4; c-is-sub-fun ls univ-for-pr;
g-comp ls key ≠ ls] ] ⇒ 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 univ-for-pr
assume A3: g-comp ls key ≠ 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 ≠ 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 ≠ 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)
lemma g-comp-lm-1: \[ \[ c-fst (c-fst \text{ key}) \mod 7 = 4; \ c-is-sub-fun \ ls \ \text{univ-for-pr} \] \implies c-is-sub-fun \ (g-comp \ ls \ \text{key}) \ \text{univ-for-pr} \]
proof -
  assume A1: c-fst (c-fst \text{ key}) \mod 7 = 4
  assume A2: c-is-sub-fun \ ls \ \text{unie-for-pr}
  show ?thesis
  proof cases
    assume g-comp \ ls \ \text{key} = \ ls
    with A2 show c-is-sub-fun \ (g-comp \ ls \ \text{key}) \ \text{univ-for-pr} \ by \ simp
  next
    assume g-comp \ ls \ \text{key} \neq \ ls
    from A1 A2 this have S1: g-comp \ ls \ \text{key} = c-cons \ (c-pair \ \text{key} \ (\text{univ-for-pr} \ \text{key})) \ \ls \ by \ (rule \ g-comp-lm-0)
    with A2 show c-is-sub-fun \ (g-comp \ ls \ \text{key}) \ \text{univ-for-pr} \ by \ (simp \ add: \ c-is-sub-fun-lm-2)
  qed
qed

lemma g-pair-lm-0: \[ \[ c-fst (c-fst \text{ key}) \mod 7 = 5; \ c-is-sub-fun \ ls \ \text{univ-for-pr}; \ g-pair \ ls \ \text{key} \neq \ ls \] \implies g-pair \ ls \ \text{key} = c-cons \ (c-pair \ \text{key} \ (\text{univ-for-pr} \ \text{key})) \ \ls \]
proof -
  assume A1: c-fst (c-fst \text{ key}) \mod 7 = 5
  assume A2: c-is-sub-fun \ ls \ \text{unie-for-pr}
  assume A3: g-pair \ ls \ \text{key} \neq \ ls
  let \(?n = c-fst \text{ key}\)
  let \(?x = c-snd \text{ key}\)
  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 \ \text{key} = \ls \ by (simp add: g-pair-def)
    with A3 show False by simp
    qed
  let \(?y1 = c-assoc-value \ls \ ?k1\)
  from A2 S1 have S2: \(?y1 = \text{univ-for-pr} \ ?k1\) 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 \ \text{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: \(?y2 = \text{univ-for-pr} \ ?k2\) by (rule c-is-sub-fun-lm-1)
  let \(?z = c-pair \ ?y1 \ ?y2\)

qed
from \( S2 \) \( S4 \) have \( S5 \): \(?z = \text{c-pair} \ (\text{univ-for-pr} \ ?k1) \ (\text{univ-for-pr} \ ?k2) \) by simp
from \( A1 \) \( S5 \) have \( S6 \): \(?z = \text{univ-for-pr} \ key \) by (simp add: univ-for-pr-lm-5)
from \( S1 \) \( S3 \) \( S6 \) show ?thesis by (simp add: g-pair-def Let-def)

qed

lemma \( g\text{-pair-lm-1} \): \[ \begin{array}{l}
\text{c-fst} \ (\text{c-fst} \ key) \mod 7 = 5; \ c\text{-is-sub-fun} \ ls \ \text{univ-for-pr}
\end{array} \]
\( \Rightarrow \ c\text{-is-sub-fun} \ (g\text{-pair} \ ls \ key) \ \text{univ-for-pr} \)
proof –
assume \( A1 \): \( \text{c-fst} \ (\text{c-fst} \ key) \mod 7 = 5 \)
assume \( A2 \): \( \text{c-is-sub-fun} \ ls \ \text{univ-for-pr} \)
show ?thesis
proof cases
  assume \( g\text{-pair} \ ls \ key = \text{ls} \)
  with \( A2 \) show \( \text{c-is-sub-fun} \ (g\text{-pair} \ ls \ key) \ \text{univ-for-pr} \) by simp
next
  assume \( g\text{-pair} \ ls \ key \neq \text{ls} \)
  from \( A1 \) \( A2 \) this have \( S1 \): \( g\text{-pair} \ ls \ key = \text{c-cons} \ (\text{c-pair} \ key \ (\text{univ-for-pr} \ key)) \)
  ls by (rule g-pair-lm-0)
  with \( A2 \) show \( \text{c-is-sub-fun} \ (g\text{-pair} \ ls \ key) \ \text{univ-for-pr} \) by (simp add: c-is-sub-fun-lm-2)
qed

qed

lemma \( g\text{-rec-lm-0} \): \[ \begin{array}{l}
\text{c-fst} \ (\text{c-fst} \ key) \mod 7 = 6; \ c\text{-is-sub-fun} \ ls \ \text{univ-for-pr}; \ g\text{-rec} \ ls \ key \neq \text{ls}
\end{array} \]
\( \Rightarrow \ g\text{-rec} \ ls \ key = \text{c-cons} \ (\text{c-pair} \ key \ (\text{univ-for-pr} \ key)) \ \text{ls} \)
proof –
assume \( A1 \): \( \text{c-fst} \ (\text{c-fst} \ key) \mod 7 = 6 \)
assume \( A2 \): \( \text{c-is-sub-fun} \ ls \ \text{univ-for-pr} \)
assume \( A3 \): \( \text{g-rec} \ ls \ key \neq \text{ls} \)
let \(?n = \text{c-fst} \ key \)
let \(?x = \text{c-snd} \ key \)
let \(?m = \text{c-snd} \ ?n \)
let \(?m1 = \text{c-fst} \ ?m \)
let \(?m2 = \text{c-snd} \ ?m \)
let \(?y1 = \text{c-fst} \ ?x \)
let \(?x1 = \text{c-snd} \ ?x \)
show ?thesis
proof cases
  assume \( A1-1 \): \(?y1 = 0 \)
  let \(?k1 = \text{c-pair} \ ?m1 \ ?x1 \)
  have \( S1-1 \): \( \text{c-assoc-have-key} \ ls \ ?k1 = 0 \)
  proof (rule ccontr)
    assume \( \text{c-assoc-have-key} \ ls \ ?k1 \neq 0 \)
    with \( A1-1 \) have \( \text{g-rec} \ ls \ key = \text{ls} \) by (simp add: g-rec-def)
    with \( A3 \) show False by simp
  qed
  let \(?v = \text{c-assoc-value} \ ls \ ?k1 \)
  from \( A2 \) \( S1-1 \) have \( S1-2 \): \(?v = \text{univ-for-pr} \ ?k1 \) by (rule c-is-sub-fun-lm-1)
from \( A1 \) \( A1-1 \) \( S1-2 \) have \( S1-3 \): \(?v = \text{univ-for-pr} \ key \) by (simp add: univ-for-pr-lm-6-1)
from \( A1-1 \) \( S1-1 \) \( S1-3 \) show ?thesis by (simp add: g-rec-def Let-def)

110
next

assume A2-1: \(?y_1 \neq 0\) then have A2-2: \(?y_1 > 0\) by simp

let \(?y_2 = \?y_1 - (1 :: nat)\)

let \(?k_2 = \text{c-pair ?n (c-pair ?y_2 ?x_1)}\)

have S2-1: c-assoc-have-key ls ?k_2 = 0

proof (rule ccontr)
  assume c-assoc-have-key ls ?k_2 \(\neq 0\)
  with A2-1 have g-rec ls key = ls by (simp add: g-rec-def Let-def)
  with A3 show False by simp
qed

let \(?y_2 = \text{c-assoc-value ls ?k_2}\)

from A2 S2-1 have S2-2: \(?y_1 > 0\) by simp

proof (rule ccontr)
  assume c-assoc-have-key ls ?k_2 \(\neq 0\)
  with A2-1 have g-rec ls key = ls by (simp add: g-rec-def Let-def)
  with A3 show False by simp
qed

lemma g-rec-lm-1: \([\text{c-fst (c-fst key)} \mod 7 = 6; \text{c-is-sub-fun ls univ-for-pr}] \implies \text{c-is-sub-fun (g-rec ls key) univ-for-pr}\)

proof –
  assume A1: c-fst (c-fst key) mod 7 = 6
  assume A2: c-is-sub-fun ls univ-for-pr
  show \(\text{thesis}\)
  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 \(\neq ls\)
  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)
qeda
lemma g-step-lm-1: c-fst (c-fst key) mod 7 = 1 \implies 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 \implies 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 \implies 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 \(A\): \(?n1 = 3\)
  then have \(S2-1\): g-step ls key = c-cons (c-pair key (c-snd (c-snd key))) ls by (rule g-step-lm-3)
  from \(A\) have \(S2-2\): univ-for-pr key = c-snd (c-snd key) by (rule univ-for-pr-lm-3)
  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 \(S5\): \(?n1 = 4\) \(\implies \) \(?thesis\)
proof –
  assume \(A\): \(?n1 = 4\)
  then have \(S2-1\): g-step ls key = g-comp ls key by (rule g-step-lm-4)
  from \(A\) \(A1\) \(S2-1\) show \(?thesis\) by (simp add: g-comp-lm-1)
qed
have \(S6\): \(?n1 = 5\) \(\implies \) \(?thesis\)
proof –
  assume \(A\): \(?n1 = 5\)
  then have \(S2-1\): g-step ls key = g-pair ls key by (rule g-step-lm-5)
  from \(A\) \(A1\) \(S2-1\) show \(?thesis\) by (simp add: g-pair-lm-1)
qed
have \(S7\): \(?n1 = 6\) \(\implies \) \(?thesis\)
proof –
  assume \(A\): \(?n1 = 6\)
  then have \(S2-1\): g-step ls key = g-rec ls key by (rule g-step-lm-6)
  from \(A\) \(A1\) \(S2-1\) show \(?thesis\) by (simp add: g-rec-lm-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 mod7-lm)
with \(S1\) \(S2\) \(S3\) \(S4\) \(S5\) \(S6\) \(S7\) show \(?thesis\) by fast
qed

theorem pr-gr-1: c-is-sub-fun (pr-gr \(x\)) univ-for-pr
apply(induct \(x\))
apply(simp add: pr-gr-at-0 c-is-sub-fun-def c-assoc-have-key-df)
apply(simp add: pr-gr-at-Suc)
apply(simp add: g-step-lm-7)
done

lemma comp-next: g-comp ls key = ls \(\lor\) c-tl (g-comp ls key) = ls by(simp add: g-comp-def Let-def)
lemma pair-next: g-pair ls key = ls \(\lor\) c-tl (g-pair ls key) = ls by(simp add: g-pair-def Let-def)
lemma rec-next: \( g\text{-rec }ls \text{ key } = ls \lor c\text{-tl } (g\text{-rec }ls \text{ key }) = ls \) by(simp add: g-rec-def Let-def)

lemma step-next: \( g\text{-step }ls \text{ key } = ls \lor c\text{-tl } (g\text{-step }ls \text{ key }) = ls \) apply(simp add: g-step-def comp-next pair-next rec-next Let-def)
done

lemma lm1: \( \text{pr-gr } (\text{Suc }x) = \text{pr-gr } x \lor c\text{-tl } (\text{pr-gr } (\text{Suc }x)) = \text{pr-gr } x \) by(simp add: pr-gr-at-Suc step-next)

lemma c-assoc-have-key-pos: \( c\text{-assoc-have-key }ls \text{ x } = 0 \Rightarrow ls > 0 \)
proof –
  assume A1: \( c\text{-assoc-have-key }ls \text{ x } = 0 \)
  thus \(?thesis \)
  proof (cases)
    assume A2: \( ls = 0 \)
    then have S1: \( c\text{-assoc-have-key }ls \text{ x } = 1 \) by simp add: c-assoc-have-key-df
    with A1 have S2: False by auto
    then show \( ls > 0 \) by auto
  next
    assume A3: \( \neg ls = 0 \)
    then show \( ls > 0 \) by auto
  qed
qed

lemma lm2: \( c\text{-assoc-have-key } (c\text{-tl }ls) \text{ key } = 0 \Rightarrow c\text{-assoc-have-key }ls \text{ key } = 0 \)
proof –
  assume A1: \( c\text{-assoc-have-key } (c\text{-tl }ls) \text{ key } = 0 \)
  from A1 have S1: \( c\text{-tl }ls > 0 \) by (rule c-assoc-have-key-pos)
  have S2: \( c\text{-tl }ls \leq ls \) by (rule c-tl-le)
  from S1 S2 have S3: \( ls \neq 0 \) by auto
  from A1 S3 show \(?thesis \) by (auto simp add: c-assoc-have-key-lm-1)
qed

lemma lm3: \( c\text{-assoc-have-key } (\text{pr-gr }x) \text{ key } = 0 \Rightarrow c\text{-assoc-have-key } (\text{pr-gr } (\text{Suc }x)) \text{ key } = 0 \)
proof –
  assume A1: \( c\text{-assoc-have-key } (\text{pr-gr }x) \text{ key } = 0 \)
  have S1: \( \text{pr-gr } (\text{Suc }x) = \text{pr-gr } x \lor c\text{-tl } (\text{pr-gr } (\text{Suc }x)) = \text{pr-gr } x \) by (rule lm1)
  from A1 have S2: \( \text{pr-gr } (\text{Suc }x) = \text{pr-gr } x \Rightarrow \(?thesis \) by auto
  have S3: \( c\text{-tl } (\text{pr-gr } (\text{Suc }x)) = \text{pr-gr } x \Rightarrow \(?thesis \)
  proof –
    assume c-tl \( (\text{pr-gr } (\text{Suc }x)) = \text{pr-gr } x \) (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 \( \text{lm}_4 \): \[
\text{c-assoc-have-key} (\text{pr-gr} \ x) \ \text{key} = 0; \ 0 \leq y \] \[\implies\] \text{c-assoc-have-key} (\text{pr-gr} \ (x+y)) \ \text{key} = 0
apply(induct-tac y)
apply(auto)
apply(simp add: lm3)
done

lemma \( \text{lm}_5 \): \[
\text{c-assoc-have-key} (\text{pr-gr} \ x) \ \text{key} = 0; \ x \leq y \] \[\implies\] \text{c-assoc-have-key} (\text{pr-gr} \ y) \ \text{key} = 0
proof
\begin{align*}
\text{assume } A1 &: \text{c-assoc-have-key} (\text{pr-gr} \ x) \ \text{key} = 0 \\
\text{assume } A2 &: x \leq y \\
\text{let } ?z = y - x \\
\text{from } A2 \ 
\text{have } S1 &: 0 \leq ?z \text{ by auto} \\
\text{from } A2 \ 
\text{have } S2 &: y = x + ?z \text{ by auto} \\
\text{from } A1 \ S1 \ 
\text{have } S3 &: \text{c-assoc-have-key} (\text{pr-gr} \ (x+?z)) \ \text{key} = 0 \text{ by (rule lm}_4) \\
\text{from } S2 \ S3 \ 
\text{show } \text{thesis} \text{ by auto} 
\end{align*}
qed

lemma \( \text{loc-upb-lm-1} \): \( n = 0 \implies (\text{c-fst} \ n) \ \text{mod} \ 7 = 0 \)
apply(simp add: c-fst-at-0)
done

lemma \( \text{loc-upb-lm-2} \): \( (\text{c-fst} \ n) \ \text{mod} \ 7 > 1 \implies c-snd \ n < n \)
proof
\begin{align*}
\text{assume } A1 &: (\text{c-fst} \ n) \ \text{mod} \ 7 > 1 \\
\text{from } A1 \ 
\text{have } S1 &: 1 < (\text{c-fst} \ n) \text{ by simp} \\
\text{have } S2 &: (\text{c-fst} \ n) \leq n \text{ by (rule c-fst-le-arg)} \\
\text{from } S1 \ S2 \ 
\text{have } S3 &: 1 < n \text{ by simp} \\
\text{from } S3 \ 
\text{have } S4 &: n>1 \text{ by simp} \\
\text{from } S4 \ 
\text{show } \text{thesis} \text{ by (rule c-snd-less-arg)} 
\end{align*}
qed

lemma \( \text{loc-upb-lm-2-0} \): \( (\text{c-fst} \ n) \ \text{mod} \ 7 = 4 \implies (\text{c-fst} \ (\text{c-snd} \ n)) < n \)
proof
\begin{align*}
\text{assume } A1 &: (\text{c-fst} \ n) \ \text{mod} \ 7 = 4 \\
\text{then have } S0 &: (\text{c-fst} \ n) \ \text{mod} \ 7 > 1 \text{ by auto} \\
\text{then have } S1 &: (\text{c-snd} \ n) < n \text{ by (rule loc-upb-lm-2)} \\
\text{have } S2 &: (\text{c-fst} \ (\text{c-snd} \ n)) \leq (\text{c-snd} \ n) \text{ by (rule c-fst-le-arg)} \\
\text{from } S1 \ S2 \ 
\text{show } \text{c-fst} \ (\text{c-snd} \ n) < n \text{ by auto} 
\end{align*}
qed

lemma \( \text{loc-upb-lm-2-2} \): \( (\text{c-fst} \ n) \ \text{mod} \ 7 = 4 \implies (\text{c-snd} \ (\text{c-snd} \ n)) < n \)
proof
\begin{align*}
\text{assume } A1 &: (\text{c-fst} \ n) \ \text{mod} \ 7 = 4 \\
\text{then have } S0 &: (\text{c-fst} \ n) \ \text{mod} \ 7 > 1 \text{ by auto} \\
\text{then have } S1 &: (\text{c-snd} \ n) < n \text{ by (rule loc-upb-lm-2)} \\
\text{have } S2 &: (\text{c-snd} \ (\text{c-snd} \ n)) \leq (\text{c-snd} \ n) \text{ by (rule c-snd-le-arg)} 
\end{align*}
qed
from \( S1 \) \( S2 \) show \( c-snd \ (c-snd \ n) < n \) by auto

qed

lemma loc-upb-lm-2-3: \( (c-fst \ n) \mod 7 = 5 \longrightarrow c-fst \ (c-snd \ n) < n \)
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 \( \text{rule loc-upb-lm-2} \)
  have \( S2: \ c-fst \ (c-snd \ n) \leq c-snd \ n \) by \( \text{rule c-fst-le-arg} \)
  from \( S1 \) \( S2 \) show \( c-fst \ (c-snd \ n) < n \) by auto

qed

lemma loc-upb-lm-2-4: \( (c-fst \ n) \mod 7 = 5 \longrightarrow c-snd \ (c-snd \ n) < n \)
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 \( \text{rule loc-upb-lm-2} \)
  have \( S2: \ c-snd \ (c-snd \ n) \leq c-snd \ n \) by \( \text{rule c-snd-le-arg} \)
  from \( S1 \) \( S2 \) show \( c-snd \ (c-snd \ n) < n \) by auto

qed

lemma loc-upb-lm-2-5: \( (c-fst \ n) \mod 7 = 6 \longrightarrow c-fst \ (c-snd \ n) < n \)
proof
  assume \( A1: \ c-fst \ n \mod 7 = 6 \)
  then have \( S0: \ c-fst \ n \mod 7 > 1 \) by auto
  then have \( S1: \ c-snd \ n < n \) by \( \text{rule loc-upb-lm-2} \)
  have \( S2: \ c-fst \ (c-snd \ n) \leq c-snd \ n \) by \( \text{rule c-fst-le-arg} \)
  from \( S1 \) \( S2 \) show \( c-fst \ (c-snd \ n) < n \) by auto

qed

lemma loc-upb-lm-2-6: \( (c-fst \ n) \mod 7 = 6 \longrightarrow c-snd \ (c-snd \ n) < n \)
proof
  assume \( A1: \ c-fst \ n \mod 7 = 6 \)
  then have \( S0: \ c-fst \ n \mod 7 > 1 \) by auto
  then have \( S1: \ c-snd \ n < n \) by \( \text{rule loc-upb-lm-2} \)
  have \( S2: \ c-snd \ (c-snd \ n) \leq c-snd \ n \) by \( \text{rule c-snd-le-arg} \)
  from \( S1 \) \( S2 \) show \( c-snd \ (c-snd \ n) < n \) by auto

qed

lemma loc-upb-lm-2-7: \( [y2 = y1 - (1::nat); 0 < y1; x1 = c-snd \ x; y1 = c-fst \ x] \Rightarrow c-pair \ y2 \ x1 < x \)
proof –
  assume \( A1: y2 = y1 - (1::nat) \) and \( A2: 0 < y1 \) and \( A3: x1 = c-snd \ x \) and 
  \( A4: y1 = c-fst \ x \)
  from \( A1 \) \( A2 \) have \( S1: y2 < y1 \) by auto
  from \( S1 \) have \( S2: c-pair \ y2 \ x1 < c-pair \ y1 \ x1 \) by \( \text{rule c-pair-strict-mono1} \)
  from \( A3 \) \( A4 \) have \( S3: c-pair \ y1 \ x1 = x \) by auto
  from \( S2 \) \( S3 \) show \( c-pair \ y2 \ x1 < x \) by auto

qed
function \text{loc-upb} :: \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \text{ where}
\begin{align*}
\text{loc-upb} \ n \ x \ (= \ \\
\text{let} \ n1 = \text{c-fst} \ n \text{ mod} 7 \text{ in} \ \\
\text{if} \ n1 = 0 \text{ then} \ (\text{c-pair} \ (\text{c-pair} \ n \ x) \ 0) + 1 \ \text{ else} \ \\
\text{if} \ n1 = 1 \text{ then} \ (\text{c-pair} \ (\text{c-pair} \ n \ x) \ 0) + 1 \ \text{ else} \ \\
\text{if} \ n1 = 2 \text{ then} \ (\text{c-pair} \ (\text{c-pair} \ n \ x) \ 0) + 1 \ \text{ else} \ \\
\text{if} \ n1 = 3 \text{ then} \ (\text{c-pair} \ (\text{c-pair} \ n \ x) \ 0) + 1 \ \text{ else} \ \\
\text{if} \ n1 = 4 \text{ then} ( \\
\text{let} \ m = \text{c-snd} \ n; \ m1 = \text{c-fst} \ m; \ m2 = \text{c-snd} \ m; \ \\
y = \text{c-assoc-value} \ (\text{pr-gr} \ (\text{loc-upb} \ m2 \ x)) \ (\text{c-pair} \ m2 \ x) \text{ in} \ \\
(\text{c-pair} \ (\text{c-pair} \ n \ x) \ (\text{loc-upb} \ m2 \ x + \text{loc-upb} \ m1 \ y)) + 1 \\
) \ \text{ else} \ \\
\text{if} \ n1 = 5 \text{ then}( \\
\text{let} \ m = \text{c-snd} \ n; \ m1 = \text{c-fst} \ m; \ m2 = \text{c-snd} \ m \text{ in} \ \\
(\text{c-pair} \ (\text{c-pair} \ n \ x) \ (\text{loc-upb} \ m1 \ x + \text{loc-upb} \ m2 \ x)) + 1 \\
) \ \text{ else} \ \\
\text{if} \ n1 = 6 \text{ then}( \\
\text{let} \ m = \text{c-snd} \ n; \ m1 = \text{c-fst} \ m; \ m2 = \text{c-snd} \ m; \ y1 = \text{c-fst} \ x; \ x1 = \text{c-snd} \ x \text{ in} \ \\
\text{if} \ y1 = 0 \text{ then} ( \\
(\text{c-pair} \ (\text{c-pair} \ n \ x) \ (\text{loc-upb} \ m1 \ x1)) + 1 \\
) \ \text{ else} ( \\
\text{let} \ y2 = y1 - (1::\text{nat}); \ \\
t1 = \text{c-assoc-value} \ (\text{pr-gr} \ (\text{loc-upb} \ n \ (\text{c-pair} \ y2 \ x1))) \ (\text{c-pair} \ n \ (\text{c-pair} \ y2 \ x1)); \ t2 = \text{c-pair} \ (\text{c-pair} \ y2 \ t1) \ x1 \text{ in} \ \\
(\text{c-pair} \ (\text{c-pair} \ n \ x) \ (\text{loc-upb} \ n \ (\text{c-pair} \ y2 \ x1) + \text{loc-upb} \ m2 \ t2)) + 1 \\
) \ \\
) \ \text{ else} 0 \\
) \ 
\text{by auto}
\end{align*}

termination
apply \ (\text{relation} \ \text{measure} \ (\lambda \ m. \ m) <\text{x\text{-}lex}> \ \text{measure} \ (\lambda \ n. \ n))
apply \ (\text{simp-all add: loc-upb-lm\text{-}2\text{-}0 \ loc-upb-lm\text{-}2\text{-}2 \ loc-upb-lm\text{-}2\text{-}3 \ loc-upb-lm\text{-}2\text{-}4 \ loc-upb-lm\text{-}2\text{-}5 \ loc-upb-lm\text{-}2\text{-}6 \ loc-upb-lm\text{-}2\text{-}7})
apply \ \text{auto}
done

definition
\text{lex-p} :: ((\text{nat} \times \text{nat}) \times \text{nat} \times \text{nat}) \text{ set} \text{ where}
\text{lex-p} = ((\text{measure} \ (\lambda \ m. \ m)) <\text{x\text{-}lex}> \ (\text{measure} \ (\lambda \ n. \ n)))

lemma \text{wf-lex-p}: \text{wf}(\text{lex-p})
apply(\text{simp add: lex-p-def})
apply(\text{auto})
done
lemma lex-p-eq: \((n', x'), (n, x)\) \(\in\) lex-p = \((n' < n \lor n' = n \land x' < x)\)
apply(simp add: lex-p-def)
done

lemma loc-upb-lex-0: c-fst n mod 7 = 0 \(\Longrightarrow\) c-assoc-have-key (pr-gr (loc-upb n x)) (c-pair n x) = 0
proof |
  assume A1: c-fst n mod 7 = 0
  let ?key = c-pair n x
  let ?s = c-pair ?key 0
  let ?ls = pr-gr ?s
  from A1 have loc-upb n x = ?s + 1 by simp
  then have S1: pr-gr (loc-upb n x) = g-step (pr-gr ?s) (c-fst ?s) by (simp add: pr-gr-at-Suc)
  from A1 have S2: g-step ?ls ?key = c-cons (c-pair ?key 0) ?ls by (simp add: g-step-def)
  from S1 S2 have pr-gr (loc-upb n x) = c-cons (c-pair ?key 0) ?ls by auto
  thus ?thesis by (simp add: c-assoc-lm-1)
qed

lemma loc-upb-lex-1: c-fst n mod 7 = 1 \(\Longrightarrow\) c-assoc-have-key (pr-gr (loc-upb n x)) (c-pair n x) = 0
proof |
  assume A1: c-fst n mod 7 = 1
  let ?key = c-pair n x
  let ?s = c-pair ?key 0
  let ?ls = pr-gr ?s
  from A1 have loc-upb n x = ?s + 1 by simp
  then have S1: pr-gr (loc-upb n x) = g-step (pr-gr ?s) (c-fst ?s) by (simp add: pr-gr-at-Suc)
  from A1 have S2: g-step ?ls ?key = c-cons (c-pair ?key (Suc x)) ?ls by (simp add: g-step-def)
  from S1 S2 have pr-gr (loc-upb n x) = c-cons (c-pair ?key (Suc x)) ?ls by auto
  thus ?thesis by (simp add: c-assoc-lm-1)
qed

lemma loc-upb-lex-2: c-fst n mod 7 = 2 \(\Longrightarrow\) c-assoc-have-key (pr-gr (loc-upb n x)) (c-pair n x) = 0
proof |
  assume A1: c-fst n mod 7 = 2
  let ?key = c-pair n x
  let ?s = c-pair ?key 0
  let ?ls = pr-gr ?s
  from A1 have loc-upb n x = ?s + 1 by simp
  then have S1: pr-gr (loc-upb n x) = g-step (pr-gr ?s) (c-fst ?s) by (simp add: pr-gr-at-Suc)
  from A1 have S2: g-step ?ls ?key = c-cons (c-pair ?key (c-fst x)) ?ls by (simp add: g-step-def)
  from S1 S2 have pr-gr (loc-upb n x) = c-cons (c-pair ?key (c-fst x)) ?ls by
lemmas *}{}{proof -}
assume A1: c-fst n mod 7 = 3
let ?key = c-pair n x
let ?s = c-pair ?key 0
let ?ls = pr-gr ?s

from A1 have loc-upb n x = ?ls + 1 by simp

then have S1: pr-gr (loc-upb n x) = g-step (pr-gr ?ls) (c-fst ?ls) by (simp add: pr-gr-at-Suc)

from A1 have S2: g-step ?ls ?key = c-cons (c-pair ?key (c-snd x)) ?ls by (simp add: g-step-def)

from S1 S2 have pr-gr (loc-upb n x) = c-cons (c-pair ?key (c-snd x)) ?ls by auto

thus ?thesis by (simp add: c-assoc-lm-1)
qed

lemma loc-upb-lex-4: \[ \forall n \cdot (n, x) \in \text{lex-p} \implies c\text{-assoc-have-key} (pr-gr (loc-upb n x)) (c-pair n x) = 0 \]

proof -
assume A1: \[ \forall n \cdot (n, x) \in \text{lex-p} \implies c\text{-assoc-have-key} (pr-gr (loc-upb n x)) (c-pair n x) = 0 \]

let ?m1 = c-fst (c-snd n)

let ?m2 = c-snd (c-snd n)

define upb1 where upb1 = loc-upb ?m2 x

from A2 have m2-lt-n: ?m2 < n by (simp add: loc-upb-lm-2-2)

then have M2: (\(\langle ?m2, x \rangle, (n,x) \rangle \in \text{lex-p} \) by (simp add: lex-p-eq)

with A1 upb1-def have S1: c-assoc-have-key (pr-gr upb1) (c-pair ?m2 x) = 0

by auto

from M2 have M2': (\(\langle ?m2, x \rangle, (n,x) \rangle \in \text{measure} (\lambda n. m) <*lex*> \text{ measure} (\lambda n. m) \) by (simp add: lex-p-def)

have T1: c-is-sub-fun (pr-gr upb1) univ-for-pr by (rule pr-gr-1)

from T1 S1 have T2: 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)

define y where y = c-assoc-value (pr-gr upb1) (c-pair ?m2 x)

from T2 y-def have T3: y = univ-for-pr (c-pair ?m2 x) by auto

define upb2 where upb2 = loc-upb ?m1 y

from A2 have ?m1 < n by (simp add: loc-upb-lm-2-0)

then have M1: (\(\langle ?m1, y \rangle, (n,y) \rangle \in \text{lex-p} \) by (simp add: lex-p-eq)

with A1 have S2: c-assoc-have-key (pr-gr (loc-upb ?m1 y)) (c-pair ?m1 y) = 0

119
by auto
from M1 have M1*: ((?m1, y), n, x) ∈ measure (λm. m) <*lex*> measure (λn. n) by (simp add: lex-p-def)
from S1 upb1-def have S3: c-assoc-have-key (pr-gr upb1) (c-pair ?m2 x) = 0
by auto
from S2 upb2-def have S4: c-assoc-have-key (pr-gr upb2) (c-pair m1 y) = 0
by auto

let ?s = c-pair ?key (upb1 + upb2)
let ?sum-upb = upb1 + upb2
from A2 have ?m1 < n by (simp add: loc-upb-lm-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 M1 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)
by (simp add: Let-def)
define upb where upb = loc-upb n x
from S11 y-def upb1-def upb2-def 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 ?s) (c-fst ?s) 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-lm-1)

from T3 T-ls2 have T-y: c-assoc-value ?ls (c-pair ?m2 x) = y by auto

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 (c-pair ?m1 y)) by

by (unfold g-comp-def) (simp del: loc-upb.simps add: Let-def)

from S18 S21 have pr-gr upb = c-cons (c-pair ?key (c-assoc-value ?ls (c-pair ?m1 y))) ?ls by auto

with upb-def have pr-gr (loc-upb n x) = c-cons (c-pair ?key (c-assoc-value ?ls (c-pair ?m1 y))) ?ls by auto

thus ?thesis by (simp add: c-assoc-lm-1)

qed

lemma loc-upb.lex-5: \[ \forall n' x'. ((n',x'), (n,x)) \in \text{lex-p} \implies c-assoc-have-key \text{ (pr-gr (loc-upb n' x'))} (c-pair n' x') = 0 \]

c-fst n mod 7 = 5 \implies c-assoc-have-key \text{ (pr-gr (loc-upb n x))} (c-pair n x) = 0

proof –

assume A1: \[ \forall n' x'. ((n',x'), (n,x)) \in \text{lex-p} \implies c-assoc-have-key \text{ (pr-gr (loc-upb n' x'))} (c-pair n' x') = 0 \]

assume A2: c-fst n mod 7 = 5

let ?key = c-pair n x

let ?m1 = c-fst (c-snd n)

let ?m2 = c-snd (c-snd n)

from A2 have ?m1 < n by (simp add: loc-upb-lm-2-3)

then have \[ ((?m1, x), (n,x)) \in \text{lex-p} \text{ by (simp add: lex-p-eq)} \]

with A1 have S1: c-assoc-have-key \text{ (pr-gr (loc-upb m1 x))} (c-pair ?m1 x) = 0 by auto

from A2 have ?m2 < n by (simp add: loc-upb-lm-2-4)

then have \[ ((?m2, x), (n,x)) \in \text{lex-p} \text{ by (simp add: lex-p-eq)} \]

with A1 have S2: c-assoc-have-key \text{ (pr-gr (loc-upb m2 x))} (c-pair ?m2 x) = 0 by auto

define upb1 where upb1 = loc-upb ?m1 x

define upb2 where upb2 = loc-upb ?m2 x

from upb1-def S1 have S3: c-assoc-have-key \text{ (pr-gr upb1) (c-pair ?m1 x) = 0}

by auto

from upb2-def S2 have S4: c-assoc-have-key \text{ (pr-gr upb2) (c-pair ?m2 x) = 0}

by auto

let ?sum-upb = upb1 + upb2

have S5: upb1 \leq ?sum-upb by (rule Nat.le-add1)

have S6: upb2 \leq ?sum-upb by (rule Nat.le-add2)

let ?s = (c-pair ?key ?sum-upb)

have S7: ?sum-upb \leq ?s by (rule arg2-le-c-pair)

from S5 S7 have S8: upb1 \leq ?s by auto

from S6 S7 have S9: upb2 \leq ?s by auto

let ?ls = pr-gr ?s

from A2 upb1-def upb2-def have S10: loc-upb n x = ?s + 1 by (simp add: Let-def)
define upb where upb = loc-upb n x
from upb-def S10 have S11: upb = ?s + 1 by auto
from S11 have S12: pr-gr upb = g-step (pr-gr ?s) (c-fst ?s) by (simp add: pr-gr-at-Suc)
from S8 S10 upb-def have S13: upb1 ≤ upb by (simp only:)
from S9 S10 upb-def have S14: upb2 ≤ upb by (simp only:)
from S3 S13 have S15: c-assoc-have-key (pr-gr upb) (c-pair ?m1 x) = 0 by (rule lm5)
from S4 S14 have S16: c-assoc-have-key (pr-gr upb) (c-pair ?m2 x) = 0 by (rule lm5)
from A2 have S17: g-step ?ls ?key = g-pair ?ls ?key by (simp add: g-step-def)
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 upb-def 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 (c-pair n x) (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 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 (simp add: Let-def)

lemma loc-upb-lex-6: [\land n' x'. ((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: \land n' x'. ((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

122
let \(?m1 = c-fst (c-snd n)\)
let \(?m2 = c-snd (c-snd n)\)
let \(?y1 = c-fst x\)
let \(?x1 = c-snd x\)

define upb where 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)
define upb1 where 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 A S6 have g-rec ?ls \(?key\) = c-cons (c-pair \(?key\) (c-assoc-value \(?ls\) (c-pair \(?m1 \(?x1\)))) \(?ls\) by (simp add: g-rec-def Let-def)
with S5 show \(?\)thesis by (simp add: c-assoc-lm-1)
next
assume A: c-fst x \(\neq 0\) 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 m (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))) (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
\[ (\text{pr-gr}\ ?t2) + 1 \text{ by (simp del: loc-upb.simps add: Let-def)} \]

let \( ?t1 = \text{univ-for-pr} (\text{c-pair n} (\text{c-pair} \ ?y2 \ ?x1)) \)

let \( ?t2 = \text{c-pair} (\text{c-pair} \ ?y2 \ ?t1) \ ?x1 \)

have \( S1-1: \text{c-assoc-have-key} (\text{pr-gr} (\text{loc-upb n} (\text{c-pair} \ ?y2 \ ?x1))) (\text{c-pair n} (\text{c-pair} \ ?y2 \ ?x1)) = 0 \)

proof -
  from \( A \) have \( ?y2 < ?y1 \) by auto
  then have \( \text{c-pair} \ ?y2 \ ?x1 < \text{c-pair} \ ?y1 \ ?x1 \) by (rule c-pair-strict-mono1)
  then have \( ((n, \text{c-pair} \ ?y2 \ ?x1), n, x) \in \text{lex-p} \) by (simp add: lex-p-eq)
  with \( \text{A1 show} \ \?t1 \) by auto

qed

have \( S2: \text{c-assoc-value} (\text{pr-gr} (\text{loc-upb n} (\text{c-pair} \ ?y2 \ ?x1))) (\text{c-pair n} (\text{c-pair} \ ?y2 \ ?x1)) = \text{univ-for-pr} (\text{c-pair n} (\text{c-pair} \ ?y2 \ ?x1)) \)

proof -
  have \( \text{c-is-sub-fun} (\text{pr-gr} (\text{loc-upb n} (\text{c-pair} \ ?y2 \ ?x1))) \) univ-for-pr by (rule pr-gr-1)
  with \( \text{S1-1 show} \ \?thesis \) by (simp add: c-is-sub-fun-lm-1)

qed

from \( S1 \ S2 \) have \( S3: \text{loc-upb n} \ n = \text{c-pair} (\text{c-pair n} \ n) (\text{loc-upb n} (\text{c-pair} \ ?y2 \ ?x1)) + \text{loc-upb} \ ?m2 \ ?t2 + 1 \) by (simp del: loc-upb.simps add: Let-def)

let \( ?s = \text{c-pair} (\text{c-pair n} \ n) (\text{loc-upb n} (\text{c-pair} \ ?y2 \ ?x1) + \text{loc-upb} \ ?m2 \ ?t2) \)

from \( S3 \) have \( S4: \text{pr-gr} (\text{loc-upb n} \ n) = \text{pr-gr} (\text{Suc} \ ?s) \) by (simp del: loc-upb.simps)

have \( \text{pr-gr} (\text{Suc} \ ?s) = \text{g-step} (\text{pr-gr} \ ?s) \) (c-fst \ ?s) by (rule pr-gr-at-Suc)

with \( \text{S4 have} \ S5: \text{pr-gr} (\text{loc-upb n} \ n) = \text{g-step} (\text{pr-gr} \ ?s) \ ?key \) by (simp del: loc-upb.simps)

let \( ?ls = \text{pr-gr} \ ?s \)

from \( A2 \) have \( g-step \ ?ls \ ?key = \text{g-rec} \ ?ls \ ?key \) by (simp add: g-step-def)

with \( S5 \) have \( S6: \text{pr-gr} (\text{loc-upb n} \ n) = \text{g-rec} \ ?ls \ ?key \) by (simp del: loc-upb.simps)

have \( S7: \text{c-assoc-have-key} ?ls (\text{c-pair n} (\text{c-pair} \ ?y2 \ ?x1)) = 0 \)

proof -
  have \( \text{loc-upb n} (\text{c-pair} \ ?y2 \ ?x1) \leq \text{loc-upb n} (\text{c-pair} \ ?y2 \ ?x1) + \text{loc-upb} \ ?m2 \ ?t2 \) by (auto simp del: loc-upb.simps)
  also have \( \text{loc-upb n} (\text{c-pair} \ ?y2 \ ?x1) + \text{loc-upb} \ ?m2 \ ?t2 \leq ?s \) by (rule arg2-le-c-pair)
  ultimately have \( S7-1: \text{loc-upb n} (\text{c-pair} \ ?y2 \ ?x1) \leq ?s \) by (auto simp del: loc-upb.simps)

from \( S1-1 \ S7-1 \) show \( \?thesis \) by (rule lm5)

qed

have \( S8: \text{c-assoc-value} ?ls (\text{c-pair n} (\text{c-pair} \ ?y2 \ ?x1)) = \?t1 \)

proof -
  have \( \text{c-is-sub-fun} ?ls \text{univ-for-pr} \) by (rule pr-gr-1)
  with \( \text{S7 show} \ \?thesis \) by (simp add: c-is-sub-fun-lm-1)

qed

have \( S9: \text{c-assoc-have-key} ?ls (\text{c-pair} \ ?m2 \ ?t2) = 0 \)

proof -
  from \( A2 \) have \( ?m2 < n \) by (simp add: loc-upb-lm-2-6)
  then have \( ((?m2, ?t2), n, x) \in \text{lex-p} \) by (simp add: lex-p-eq)
  with \( \text{A1 have} \ \text{c-assoc-have-key} (\text{pr-gr} (\text{loc-upb} \ ?m2 \ ?t2)) (\text{c-pair} \ ?m2 \ ?t2) = 0 \) by auto
also have $\text{loc-upb} \ ?m2 \ ?t2 \leq \ ?s$
proof 
  have $\text{loc-upb} \ ?m2 \ ?t2 \leq \text{loc-upb} \ n \ (\text{c-pair} \ ?y2 \ ?x1) + \text{loc-upb} \ ?m2 \ ?t2$ by (auto simp del: loc-upb.simps)
also have $\text{loc-upb} \ n \ (\text{c-pair} \ ?y2 \ ?x1) + \text{loc-upb} \ ?m2 \ ?t2 \leq \ ?s$ by (rule arg2-le-c-pair)
  ultimately show $\text{thesis}$ by (auto simp del: loc-upb.simps)
qede

ultimately show $\text{thesis}$ by (rule lm5)
qede

from $A \ S7 \ S8 \ S9$ have $\text{g-rec} \ ?ls \ ?key = \text{c-cons} \ (\text{c-pair} ?key \ (\text{c-assoc-value} \ ?ls \ (\text{c-pair} \ ?m2 \ ?t2))) \ ?ls$ by (simp del: loc-upb.simps add: g-rec-def Let-def)
with $S6$ show $\text{thesis}$ by (simp add: c-assoc-lm-1)
qede

lemma $\text{wf-upb-step-0}$:
\[\forall n \ x. ((n',x'),(n,x)) \in \text{lex-p} \implies \text{c-assoc-have-key} \ (\text{pr-gr} \ (\text{loc-upb} \ n' \ x')) \ (\text{c-pair} n' x') = 0 \implies \text{c-assoc-have-key} \ (\text{pr-gr} \ (\text{loc-upb} \ n \ x)) \ (\text{c-pair} n \ x) = 0\]
proof 
  assume $A1: \forall n \ x. ((n',x'),(n,x)) \in \text{lex-p} \implies \text{c-assoc-have-key} \ (\text{pr-gr} \ (\text{loc-upb} \ n' \ x')) \ (\text{c-pair} n' x') = 0$\n  let $?n1 = (\text{c-fst} \ n) \mod 7$
  have $S1$: $?n1 = 0 \implies \text{thesis}$
proof 
  assume $A$: $?n1 = 0$
  thus $\text{thesis}$ by (rule loc-upb-lex-0)
qede

have $S2$: $?n1 = 1 \implies \text{thesis}$
proof 
  assume $A$: $?n1 = 1$
  thus $\text{thesis}$ by (rule loc-upb-lex-1)
qede

have $S3$: $?n1 = 2 \implies \text{thesis}$
proof 
  assume $A$: $?n1 = 2$
  thus $\text{thesis}$ by (rule loc-upb-lex-2)
qede

have $S4$: $?n1 = 3 \implies \text{thesis}$
proof 
  assume $A$: $?n1 = 3$
  thus $\text{thesis}$ by (rule loc-upb-lex-3)
qede

have $S5$: $?n1 = 4 \implies \text{thesis}$
proof 
  assume $A$: $?n1 = 4$
  from $A1 \ A \ A$ show $\text{thesis}$ by (rule loc-upb-lex-4)
qede

125
have S6: \(\forall n1 = 5 \implies \text{thesis}\)
proof –
assume A: \(\forall n1 = 5\)
from A1 A show \(\text{thesis}\) by (rule loc-upb-lex-5)
qed
have S7: \(\forall n1 = 6 \implies \text{thesis}\)
proof –
assume A: \(\forall n1 = 6\)
from A1 A show \(\text{thesis}\) by (rule loc-upb-lex-6)
qed
have S8: \(\forall n1 = 0 \lor \forall n1 = 1 \lor \forall n1 = 2 \lor \forall n1 = 3 \lor \forall n1 = 4 \lor \forall n1 = 5 \lor \forall n1 = 6\)
by (rule mod7-lm)
from S1 S2 S3 S4 S5 S6 S7 S8 show \(\text{thesis}\) by fast
qed

lemma wf-upb-step:
assumes A1: \(\forall p2. (p2, p1) \in \text{lex-p} \implies c\text{-assoc-have-key} (\text{pr-gr} (\text{loc-upb} (\text{fst} p2) (\text{snd} p2))) (\text{c-pair} (\text{fst} p2) (\text{snd} p2)) = 0\)
shows \(c\text{-assoc-have-key} (\text{pr-gr} (\text{loc-upb} (\text{fst} p1) (\text{snd} p1))) (\text{c-pair} (\text{fst} p1) (\text{snd} p1)) = 0\)
proof –
let \(\forall n = \text{fst} p1\)
let \(\forall x = \text{snd} p1\)
from A1 have S1: \(\forall n \in \text{lex-p}\implies c\text{-assoc-have-key} (\text{pr-gr} (\text{loc-upb} (\text{fst} p2) (\text{snd} p2))) (\text{c-pair} (\text{fst} p2) (\text{snd} p2)) = 0\)
by auto
have S2: \((\forall n' \in \text{lex-p} \implies c\text{-assoc-have-key} (\text{pr-gr} (\text{loc-upb} n' (\text{fst} p1) (\text{snd} p1))) (\text{c-pair} (\text{fst} p1) (\text{snd} p1)) = 0)\)
by (rule wf-upb-step-0)
then have S3: \((\forall n' \in \text{lex-p} \implies c\text{-assoc-have-key} (\text{pr-gr} (\text{loc-upb} n' x')) (\text{c-pair} n' x') = 0)\)
by auto
have S4: \((\forall n' \in \text{lex-p} \implies c\text{-assoc-have-key} (\text{pr-gr} (\text{loc-upb} n' x')) (\text{c-pair} n' x') = 0)\)
by (rule A1)
then show \(c\text{-assoc-have-key} (\text{pr-gr} (\text{loc-upb} (n' x')) (\text{c-pair} n' x') = 0)\)
by auto
qed
from S4 S3 show ?thesis by auto

qed

theorem loc-upb-main: c-assoc-have-key (pr-gr (loc-upb n x)) (c-pair n x) = 0

proof –
  have loc-upb-lm: \( \bigwedge p. \) c-assoc-have-key (pr-gr (loc-upb (fst p) (snd p))) (c-pair (fst p) (snd p)) = 0

proof –
  fix p show c-assoc-have-key (pr-gr (loc-upb (fst p) (snd p))) (c-pair (fst p) (snd p)) = 0

proof –
  have S1: wf lex-p by (auto simp add: lex-p-def)

  from S1 wf-upb-step show ?thesis by (rule wf-induct-rule)

qed

let ?p = (n, x)

have c-assoc-have-key (pr-gr (loc-upb ?p)) (c-pair ?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 \in PrimRec2

proof –
  from c-assoc-have-key-is-pr c-assoc-value-is-pr c-cons-is-pr have \( \lambda x y. \) g-comp x y \in PrimRec2

  unfolding g-comp-def Let-def by prec

  thus ?thesis by auto

qed

theorem g-pair-is-pr: g-pair \in PrimRec2

proof –
  from c-assoc-have-key-is-pr c-assoc-value-is-pr c-cons-is-pr have \( \lambda x y. \) g-pair x y \in PrimRec2

  unfolding g-pair-def Let-def by prec

  thus ?thesis by auto

qed

theorem g-rec-is-pr: g-rec \in PrimRec2

proof –
  from c-assoc-have-key-is-pr c-assoc-value-is-pr c-cons-is-pr have \( \lambda x y. \) g-rec x y \in PrimRec2

  unfolding g-rec-def Let-def by prec

  thus ?thesis by auto

qed

theorem g-step-is-pr: g-step \in PrimRec2
proof 
  from g-comp-is-pr g-pair-is-pr g-rec-is-pr mod-is-pr c-associative-key-is-pr c-associative-value-is-pr
  c-cons-is-pr have 
    (λ ls key. g-step ls key) ∈ PrimRec2 unfolding g-step-def Let-def by preceq
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
    fix x
    show pr-gr x = ?f x by (induct x) (simp add: pr-gr-at-0, simp add: pr-gr-at-Suc)
  qed
  have S2 : PrimRecOp1 0 (λ x y. g-step y (c-fst x)) ∈ PrimRec1
  proof (rule pr-rec1)
    from g-step-is-pr show (λ x y. g-step y (c-fst x)) ∈ PrimRec2 by preceq
  qed
from S1 S2 show ?thesis by auto
qed

end

7  Computably enumerable sets of natural numbers

theory RecEnSet
imports PRRecList PRRecFun2 PRRecFinSet PRRecUnGr
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 $\forall x. (x \in A) = (\exists y. p \ x \ y = 0)$
then have \( A = \text{fn-to-set } p \) by (unfold \text{fn-to-set-def}, auto)
with \( \text{p-is-pr} \) show \( A \in \text{ce-sets} \) by (simp add: \text{ce-set-lm-1})
qed

lemma \text{ce-set-lm-3}: \( A \in \text{ce-sets} \Longrightarrow \exists p \in \text{PrimRec2}. A = \text{fn-to-set } p \)
proof –
assume \( A \in \text{ce-sets} \)
then have \( A \in \{ (\text{fn-to-set } p) \mid p. p \in \text{PrimRec2} \} \) by (simp add: \text{ce-sets-def})
thus \( \text{thesis} \) by auto
qed

lemma \text{ce-set-lm-4}: \( A \in \text{ce-sets} \Longrightarrow \exists p \in \text{PrimRec2}. \forall x. (x \in A) = (\exists y. p x y = 0) \)
proof –
assume \( A \in \text{ce-sets} \)
then have \( \exists p \in \text{PrimRec2}. A = \text{fn-to-set } p \) by (rule \text{ce-set-lm-3})
then obtain \( p \) where \( \text{p-is-pr} \): \( p \in \text{PrimRec2} \) and \( L1: A = \text{fn-to-set } p \) ..
from \( \text{p-is-pr} \) \( L1 \) show \( \text{thesis} \) by (unfold \text{fn-to-set-def}, auto)
qed

lemma \text{ce-set-lm-5}: \( \{ A \in \text{ce-sets}; p \in \text{PrimRec1} \} \Longrightarrow \{ x . p x \in A \} \in \text{ce-sets} \)
proof –
assume \( A1: A \in \text{ce-sets} \)
assume \( A2: p \in \text{PrimRec1} \)
from \( A1 \) have \( \exists p.A \in \text{PrimRec2}. A = \text{fn-to-set } pA \) by (rule \text{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: \text{fn-to-set-def})
define \( q \) where \( q x y = pA (p \ x \ y) \) for \( x \ y \)
from \( pA-is-pr \) \( A2 \) have \( q-is-pr: q \in \text{PrimRec2} \) unfolding \text{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: \text{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: \text{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: \text{fn-to-set-def})
moreover from \( \text{q-is-pr} \) have \( \text{fn-to-set } q \in \text{ce-sets} \) by (rule \text{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 \mid 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)
  then obtain a where a-in: a \in A by auto
  define q where q z = (if pA (c-fst z) (c-snd z) = 0 then c-fst z else a) for z
  from S2 have S3: \forall z. \exists y. pA x y = 0 by simp
  with a-in show \exists u. x = q u \land u \in \text{UNIV} by auto
  qed

qed

then have S4: \{ q x \mid x. x \in \text{UNIV} \} \subseteq A by auto
have S5: A \subseteq \{ q x \mid x. x \in \text{UNIV} \}
proof
  fix x assume A: x \in A show x \in \{ q x \mid x. x \in \text{UNIV} \}
  proof
    from A S2 obtain y where L1: pA x y = 0 by auto
    let ?z = c-pair x y
    from L1 have q ?z = x by (simp add: q-def)
    then have \exists u. q u = x by blast
    then show \exists u. x = q u \land u \in \text{UNIV} by auto
  qed

qed

from S4 S5 have S6: A = \{ q x \mid x. x \in \text{UNIV} \} by auto
with q-is-pr show \?thesis by blast

qed

lemma ce-set-lm-7: \[ A \in \text{ce-sets}; p \in \text{PrimRec1} \] \implies \{ p x \mid x. x \in A \} \in \text{ce-sets}
proof –
  assume A1: A \in \text{ce-sets}
  assume A2: p \in \text{PrimRec1}
  let ?B = \{ p x \mid x. x \in A \}
  fix y have S1: (y \in ?B) = (\exists x. x \in A \land (y = p x)) by auto

130
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 \( S2: A = \text{fn-to-set} \ pA \).
from \( S2 \) have \( S3: A = \{ x. \ \exists \ y. \ pA \ x \ y = 0 \} \) by (simp add: fn-to-set-def)
define \( q \) where \( q \ y \ t = (\text{if } y = p \ (c\text{-snd} \ t) \ \text{then } pA \ (c\text{-snd} \ t) \ (c\text{-fst} \ t) \ \text{else } 1) \)
for \( y \ t \)
from \( pA\)-is-pr A2 have \( q\)-is-pr: \( q \in \text{PrimRec2} \) unfolding \( q\)-def by prec

define \( q \) where \( q \ y \ t = (\text{if } y = p \ (c\text{-snd} \ t) \ \text{then } pA \ (c\text{-snd} \ t) \ (c\text{-fst} \ t) \ \text{else } 1) \)
for \( y \ t \)
from \( pA\)-is-pr A2 have \( q\)-is-pr: \( q \in \text{PrimRec2} \) unfolding \( q\)-def by prec

proof –

assume \( AA1: y \in ?B \)
then obtain \( x0 \) where \( LL2: x0 \in A \) and \( LL3: y = p \ x0 \) by auto
from \( S3 \) have \( LL4: (x0 \in A) = (\exists \ z. \ pA \ x0 \ z = 0) \) by auto
from \( LL2 \ LL4 \) obtain \( z0 \) where \( LL5: \ pA \ x0 \ z0 = 0 \) by auto
define \( t \) where \( t = c\text{-pair} \ z0 \ x0 \)
from \( t\)-def \( q\)-def \( LL2 \ LL5 \) have \( q \ y \ t = 0 \) by simp
then show \( \exists \ z. \ q \ y \ z = 0 \) by auto
next
assume \( A1: \exists \ z. \ q \ y \ z = 0 \)
then obtain \( z0 \) where \( LL1: q \ y \ z0 = 0 \) ..
have \( LL2: y = p \ (c\text{-snd} \ z0) \) (rule ccontr)
proof

assume \( y \neq p \ (c\text{-snd} \ z0) \)
with \( q\)-def \( LL1 \) have \( q \ y \ z0 = 1 \) by auto
with \( LL1 \) show False by auto
qed
from \( LL2 \ LL1 \) q-def have \( LL3: pA \ (c\text{-snd} \ z0) \ (c\text{-fst} \ z0) = 0 \) by auto
with \( S3 \) have \( LL4: c\text{-snd} \ z0 \in A \) by auto
with \( LL2 \) show \( y \in \{ p \ x | x. \ x \in A \} \) by auto
qed
then have \( LL2 \): \( ?B = \{ y | y. \exists \ z. \ q \ y \ z = 0 \} \) by auto
with \( \text{fn-to-set-def} \) have \( ?B = \text{fn-to-set} \ q \) by auto
with \( q\)-is-pr \ ce-set-lm-1 \ show \( ?\thesis \) by auto
qed


proof
  then have $\forall \ a. \ ?f \ x \ a = 0$ by simp
  then have $\{x. \ ?f \ x \ a = 0\} = \text{UNIV}$ by auto
  also have $\text{fn-to-set} ?f = \ldots$ by (simp add: fn-to-set-def)
  with $S1$ show $\vdash \text{thesis}$ by (auto simp add: ce-sets-def)
qed

theorem ce-singleton: $\{a\} \in \text{ce-sets}$
proof
  let $\lambda x y. (\text{abs-of-diff} x a) + y$
  have $S1$: $\lambda x y. (\text{abs-of-diff} x a) + y$
    by (simp add: ce-sets-def)
  then obtain $?p$ from $S2$ $S5$
  have $\lambda x y. (\text{abs-of-diff} x a) + y$
    by (simp add: fn-to-set-def)
  with $S1$ show $\vdash \text{thesis}$ by (auto simp add: ce-sets-def)
qed

theorem ce-union: $[ A \in \text{ce-sets}; B \in \text{ce-sets} ] \Longrightarrow A \cup B \in \text{ce-sets}$
proof
  assume $A1$: $A \in \text{ce-sets}$
  then obtain $p-a$ where $S2$: $p-a \in \text{PrimRec2}$ and $S3$: $A = \text{fn-to-set} p-a$
    by (auto simp add: ce-sets-def)
  assume $A2$: $B \in \text{ce-sets}$
  then obtain $p-b$ where $S5$: $p-b \in \text{PrimRec2}$ and $S6$: $B = \text{fn-to-set} p-b$
    by (auto simp add: ce-sets-def)
  assume $A1$: $A \in \text{ce-sets}$
  then obtain $p-a$ where $S2$: $p-a \in \text{PrimRec2}$ and $S3$: $A = \text{fn-to-set} p-a$
    by (auto simp add: ce-sets-def)
  assume $A2$: $B \in \text{ce-sets}$
  then obtain $p-b$ where $S5$: $p-b \in \text{PrimRec2}$ and $S6$: $B = \text{fn-to-set} p-b$
    by (auto simp add: ce-sets-def)
  show $\vdash \text{thesis}$ by simp
qed

theorem ce-intersect: $[ A \in \text{ce-sets}; B \in \text{ce-sets} ] \Longrightarrow A \cap B \in \text{ce-sets}$
proof
  assume $A1$: $A \in \text{ce-sets}$
  then obtain $p-a$ where $S2$: $p-a \in \text{PrimRec2}$ and $S3$: $A = \text{fn-to-set} p-a$
    by (auto simp add: ce-sets-def)
  assume $A2$: $B \in \text{ce-sets}$
  then obtain $p-b$ where $S5$: $p-b \in \text{PrimRec2}$ and $S6$: $B = \text{fn-to-set} p-b$
    by (auto simp add: ce-sets-def)
  show $\vdash \text{thesis}$ by simp
qed
fix \( x \) show \( (\exists \ y. \ ?p \ x \ y = 0) = ((\exists \ z. \ p-a \ x \ z = 0) \land (\exists \ z. \ p-b \ x \ z = 0)) \)

proof –

have 1: \( (\exists \ y. \ ?p \ x \ y = 0) \implies ((\exists \ z. \ p-a \ x \ z = 0) \land (\exists \ z. \ p-b \ x \ z = 0)) \)

by blast

have 2: \( ((\exists \ z. \ p-a \ x \ z = 0) \land (\exists \ z. \ p-b \ x \ z = 0)) \implies (\exists \ y. \ ?p \ x \ y = 0) \)

proof –

assume \( (\exists \ z. \ p-a \ x \ z = 0) \land (\exists \ z. \ p-b \ x \ z = 0) \)

then obtain \( z_1 \) \( z_2 \) where \( \langle s-23 \rangle: \ p-a \ x \ z_1 = 0 \) and \( \langle s-24 \rangle: \ p-b \ x \ z_2 = 0 \)

by auto

let \( ?y_1 = \text{c-pair} \ z_1 \ z_2 \)

from \( \langle s-23 \rangle \) have \( \langle s-25 \rangle: \ p-a \ x \ (\text{c-fst} \ ?y_1) = 0 \)

by simp

from \( \langle s-24 \rangle \) have \( \langle s-26 \rangle: \ p-b \ x \ (\text{c-snd} \ ?y_1) = 0 \)

by simp

from \( \langle s-25 \rangle, \langle s-26 \rangle \) have \( \langle s-27 \rangle: \ p-a \ x \ (\text{c-fst} \ ?y_1) + p-b \ x \ (\text{c-snd} \ ?y_1) = 0 \)

by simp

then show \( \langle \text{thesis} \rangle .. \)

qed

from 1 2 have \( (\exists \ y. \ ?p \ x \ y = 0) = ((\exists \ z. \ p-a \ x \ z = 0) \land (\exists \ z. \ p-b \ x \ z = 0)) \)

by (rule iffI)

then show \( \langle \text{thesis} \rangle \) by auto

qed

lemma \( \text{nat-to-ce-set-lm-1} \): \( \text{nat-to-ce-set} \ n = \{ \ x . \ \exists \ y. \ (\text{nat-to-pr} \ n) \ (\text{c-pair} \ x \ y) = 0 \} \)

proof –

have \( \langle S1 \rangle: \ \text{nat-to-ce-set} \ n = \text{fn-to-set} \ (\text{pr-conv-1-to-2} \ (\text{nat-to-pr} \ n)) \)

by (simp add: \( \text{nat-to-ce-set-def} \))

then have \( \langle S2 \rangle: \ \text{nat-to-ce-set} \ n = \{ \ x . \ \exists \ y. \ (\text{pr-conv-1-to-2} \ (\text{nat-to-pr} \ n)) \ x \ y = 0 \} \)

by (simp add: \( \text{fn-to-set-def} \))

have \( \langle S3 \rangle: \ \lambda \ x. \ y. \ (\text{pr-conv-1-to-2} \ (\text{nat-to-pr} \ n)) \ x \ y = (\text{nat-to-pr} \ n) \ (\text{c-pair} \ x \ y) \)

by (simp add: \( \text{pr-conv-1-to-2-def} \))

from \( \langle S2 \rangle, \langle S3 \rangle \) show \( \langle \text{thesis} \rangle \) by auto

qed

lemma \( \text{nat-to-ce-set-into-ce} \): \( \text{nat-to-ce-set} \ n \in \text{ce-sets} \)

7.3 Enumeration of computably enumerable sets

definition \( \text{nat-to-ce-set} :: \text{nat} \Rightarrow \text{nat set} \) where

\( \text{nat-to-ce-set} = (\lambda \ n. \ \text{fn-to-set} \ (\text{pr-conv-1-to-2} \ (\text{nat-to-pr} \ n))) \)
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-is-pr: p ∈ PrimRec2 and S1: A = fn-to-set p ..
  define q where q = pr-conv-2-to-1 p
  from p-is-pr have q-is-pr: q ∈ PrimRec1 by (unfold q-def, rule pr-conv-2-to-1-lm)
  from q-def have S2: pr-conv-1-to-2 q = p by simp
  let ?n = index-of-pr q
  from q-is-pr have nat-to-pr ?n = q by (rule index-of-pr-is-real)
  with S2 S1 have A = nat-to-ce-set ?n by (simp add: nat-to-ce-set-def)
  thus ?thesis ..
qed

7.4 Characteristic functions

definition
  chf :: nat set ⇒ (nat ⇒ nat) — Characteristic function where
  chf = (λ A x. if x ∈ A then 0 else 1 )

definition
  zero-set :: (nat ⇒ nat) ⇒ nat set where
  zero-set = (λ f. { x. f x = 0})

lemma chf-lm-1 [simp]: zero-set (chf A) = A by (unfold chf-def, unfold zero-set-def, simp)

lemma chf-lm-2: (x ∈ A) = (chf A x = 0) by (unfold chf-def, simp)

lemma chf-lm-3: (x /∈ A) = (chf A x = 1) by (unfold chf-def, simp)

lemma chf-lm-4: chf A ∈ PrimRec1 ⇒ A ∈ ce-sets
proof
  assume A: chf A ∈ PrimRec1
  define p where p = chf A
  from A p-def have p-is-pr: p ∈ PrimRec1 by auto
  define q where q x y = p x for x y :: nat
  from p-is-pr have q-is-pr: q ∈ PrimRec2 unfolding q-def by prec
  have S1: A = {x. p(x) = 0}
  proof
    ..
  qed
have zero-set \( p = A \) by (unfold p-def, simp)
thus \( \)?thesis by (simp add: zero-set-def)
qed

have \( S2 \): fn-to-set \( q = \{ x. \exists y. q x y = 0 \} \) by (simp add: fn-to-set-def)

have \( S3 \): \( \bigwedge x. (p x = 0) = (\exists y. q x y = 0) \) by (unfold q-def, auto)
then have \( S4 \): \( \{ x. p x = 0 \} = \{ x. \exists y. q x y = 0 \} \) by auto

with \( S1 \) \( S2 \) have \( S5 \): fn-to-set \( q = A \) by auto

from \( q-is-pr \) have fn-to-set \( q \in ce-sets \) by (rule ce-set-lm-1)

lemma \( chf-lm-5 \): \( \text{finite A } \Rightarrow \text{chf A } \in \text{PrimRec1} \)

proof –
assume \( A \): \( \text{finite A} \)
define \( u \) where \( u = \text{set-to-nat A} \)
from \( A \) have \( S1 \): \( \text{nat-to-set u } = A \) by (unfold u-def, rule nat-to-set-srj)
have \( \text{chf A } = (\lambda x. \text{sgn2} (c-in x u)) \)
proof
fix \( x \) show \( \text{chf A x } = \text{sgn2} (c-in x u) \)
proof cases
assume \( A \): \( x \in A \)
then have \( S1-1 \): \( \text{chf A x } = 0 \) by (simp add: chf-lm-2)
from \( A \) \( S1 \) have \( x \in \text{nat-to-set u } \) 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 \( A \): \( x \notin A \)
then have \( S1-1 \): \( \text{chf A x } = 1 \) by (simp add: chf-def)
from \( A \) \( S1 \) have \( x \notin \text{nat-to-set u } \) 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 \( (\lambda x. \text{sgn2} (c-in x u)) \in \text{PrimRec1} \) by prec
ultimately show \( \)?thesis by auto
qed

theorem \( ce-finite \): \( \text{finite A } \Rightarrow \text{A } \in \text{ce-sets} \)
proof –
assume \( A \): \( \text{finite A} \)
then have \( \text{chf A } \in \text{PrimRec1} \) by (rule chf-lm-5)
then show \( \)?thesis by (rule chf-lm-4)
qed

7.5 Computably enumerable relations

definition \( ce-set-to-rel :: \text{nat set } \Rightarrow (\text{nat } * \text{ nat}) \text{ set where} \)
\( ce-set-to-rel = (\lambda A. \{ (c-fst x, c-snd x) | x. x \in A \}) \)
definition
\[ ce\text{-}rel\text{-}to\text{-}set :: (\text{n}at \ast \text{n}at) \text{ set} \Rightarrow \text{n}at \text{ set} \text{ where} \]
\[ ce\text{-}rel\text{-}to\text{-}set = (\lambda R. \{ \text{c}\text{-}pair \ x \ y \mid x, y. (x,y) \in R \}) \]

definition
\[ ce\text{-}rels :: ((\text{n}at \ast \text{n}at) \text{ set}) \text{ set} \text{ where} \]
\[ ce\text{-}rels = \{ R \mid R. \text{ce}\text{-}rel\text{-}to\text{-}set R \in \text{ce}\text{-}sets \} \]

lemma ce-rel-lm-1 [simp]: ce-set-to-rel (ce-rel-to-set r) = r
proof
\[
\begin{align*}
\text{show ce-set-to-rel (ce-rel-to-set r) } &\subseteq r \\
\text{proof fix } z &\text{ assume } A: z \in \text{ce-set-to-rel (ce-rel-to-set r)} \\
\text{then obtain } u \text{ where } L1: u \in (\text{ce-rel-to-set r}) \text{ and } L2: z = (\text{c-fst } u, \text{ c-snd } u) \\
\text{unfolding ce-set-to-rel-def by auto} &\text{ from } L1 \text{ obtain } x, y \text{ where } L3: (x, y) \in r \text{ and } L4: u = \text{c-pair } x \ y \\
\text{unfolding ce-rel-to-set-def by auto} &\text{ from } L4 \text{ have } L5: \text{ c-fst } u = x \text{ by simp} \\
\text{from } L4 \text{ have } L6: \text{ c-snd } u = y \text{ by simp} \\
\text{from } L5 \ L6 \ L2 \text{ have } z = (x, y) \text{ by simp} \\
\text{with } L3 \text{ show } z \in r \text{ by auto} \\
\text{qed}
\end{align*}
\]
next
\[\text{show } r \subseteq \text{ce-set-to-rel (ce-rel-to-set r)} \]
\text{proof fix } z \text{ show } z \in r \Rightarrow z \in \text{ce-set-to-rel (ce-rel-to-set r)}
\text{proof –}
\[
\begin{align*}
\text{assume } A: z \in r &\text{ define } x \text{ where } x = \text{fst } z \\
\text{define } y \text{ where } y = \text{snd } z &\text{ from } x\text{-}def \ y\text{-}def \text{ have } L1: z = (x,y) \text{ by simp} \\
\text{define } u \text{ where } u = \text{c-pair } x \ y &\text{ from } A \ L1 \ u\text{-}def \text{ have } L2: u \in \text{ce-rel-to-set r} \text{ by (unfold ce-rel-to-set-def, auto)} \\
\text{from } L1 \ u\text{-}def \text{ have } L3: z = (\text{c-fst } u, \text{ c-snd } u) \text{ by simp} \\
\text{from } L2 \ L3 \text{ show } z \in \text{ce-set-to-rel (ce-rel-to-set r)} \text{ by (unfold ce-set-to-rel-def, auto)} \\
\text{qed}
\end{align*}
\]
\text{qed}
\text{qed}

lemma ce-rel-lm-2 [simp]: ce-rel-to-set (ce-set-to-rel A) = A
proof
\[
\begin{align*}
\text{show ce-rel-to-set (ce-set-to-rel A) } &\subseteq A \\
\text{proof fix } z \text{ show } z \in \text{ce-rel-to-set (ce-set-to-rel A)} \Rightarrow z \in A \\
\text{proof –}
\text{assume } A: z \in \text{ce-rel-to-set (ce-set-to-rel A)} &\text{ then obtain } x, y \text{ where } L1: z = \text{c-pair } x \ y \text{ and } L2: (x, y) \in \text{ce-set-to-rel A} \\
\text{unfolding ce-rel-to-set-def by auto}
\end{align*}
\]
from L2 obtain u where L3: \((x,y) = (\text{c-fst } u, \text{c-snd } u)\) and L4: \(u \in A\)
unfolding 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 ce-set-to-rel-def, auto)
define x where \(x = \text{c-fst } z\)
define y where \(y = \text{c-snd } z\)
from L1 x-def y-def have L2: \((x,y) \in \text{ce-set-to-rel } A\) by simp
then have L3: \(\text{c-pair } x \ y \ \in \text{ce-rel-to-set } (\text{ce-set-to-rel } A)\) by (unfold ce-rel-to-set-def, auto)
with x-def y-def show \(z \in \text{ce-rel-to-set } (\text{ce-set-to-rel } A)\) by simp
qed
qed
qed

lemma ce-rels-def1: \(\text{ce-rels} = \{ \text{ce-set-to-rel } A \mid A. A \in \text{ce-sets} \}\)
proof
show \(\text{ce-rels} \subseteq \{ \text{ce-set-to-rel } A \mid A. A \in \text{ce-sets} \}\)
proof fix r show \(r \in \text{ce-rels} \implies r \in \{ \text{ce-set-to-rel } A \mid A. A \in \text{ce-sets} \}\)
proof
assume A: \(r \in \text{ce-rels}\)
then have L1: \(\text{ce-rel-to-set } r \in \text{ce-sets}\) by (unfold ce-rels-def, auto)
define A where \(A = \text{ce-rel-to-set } r\)
from A-def L1 have L2: \(A \in \text{ce-sets}\) by auto
from A-def have L3: \(\text{ce-set-to-rel } A = r\) by simp
with L2 show \(r \in \{ \text{ce-set-to-rel } A \mid A. A \in \text{ce-sets} \}\) by auto
qed
qed
qed
next
show \(\{ \text{ce-set-to-rel } A \mid A. A \in \text{ce-sets} \}\) \(\subseteq \text{ce-rels}\)
proof fix r show \(r \in \{ \text{ce-set-to-rel } A \mid A. A \in \text{ce-sets} \} \implies r \in \text{ce-rels}\)
proof
assume A: \(r \in \{ \text{ce-set-to-rel } A \mid A. A \in \text{ce-sets} \}\)
then obtain A where \(L1: r = \text{ce-set-to-rel } A\) and \(L2: A \in \text{ce-sets}\) by auto
from L1 have \(\text{ce-rel-to-set } r = A\) by simp
with L2 show \(r \in \text{ce-rels}\) unfolding ce-rels-def by auto
qed
qed
qed

lemma ce-rel-to-set-inj: inj \(\text{ce-rel-to-set}\)
proof (rule inj-on-inverse1)
  fix x assume A: (x::(nat×nat) set) ∈ 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)
  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-inverse1)
  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 $A \in \text{ce-sets}$ then show $\text{ce-set-to-rel } A \in \text{ce-rels}$ by (rule ce-rel-lm-3)
next
assume $\text{ce-set-to-rel } A \in \text{ce-rels}$ then show $A \in \text{ce-sets}$ by (rule ce-rel-lm-4)
qed

lemma ce-rel-lm-6: $r \in \text{ce-rels} \implies \text{ce-rel-to-set } r \in \text{ce-sets}$
proof –
  assume $A: r \in \text{ce-rels}$
  then show $\text{thesis}$ by (unfold ce-rels-def, auto)
qed

lemma ce-rel-lm-7: $\text{ce-rel-to-set } r \in \text{ce-sets} \implies r \in \text{ce-rels}$
proof –
  assume $\text{ce-rel-to-set } r \in \text{ce-sets}$
  then show $\text{thesis}$ by (unfold ce-rels-def, auto)
qed

lemma ce-rel-lm-8: $(r \in \text{ce-rels}) = (\text{ce-rel-to-set } r \in \text{ce-sets})$ by (unfold ce-rels-def, auto)

lemma ce-rel-lm-9: $(x, y) \in r \implies \text{c-pair } x \ y \in \text{ce-rel-to-set } r$ by (unfold ce-rel-to-set-def, auto)

lemma ce-rel-lm-10: $x \in A \implies (\text{c-fst } x, \text{c-snd } x) \in \text{ce-set-to-rel } A$ by (unfold ce-set-to-rel-def, auto)

lemma ce-rel-lm-11: $\text{c-pair } x \ y \in \text{ce-rel-to-set } r \implies (x, y) \in r$
proof –
  assume $A: \text{c-pair } x \ y \in \text{ce-rel-to-set } r$
  let $?z = \text{c-pair } x \ y$
  from $A$ have $(\text{c-fst } ?z, \text{c-snd } ?z) \in \text{ce-set-to-rel } (\text{ce-rel-to-set } r)$ by (rule ce-rel-lm-10)
  then show $(x, y) \in r$ by simp
qed

lemma ce-rel-lm-12: $(\text{c-pair } x \ y \in \text{ce-rel-to-set } r) = ((x, y) \in r)$
proof
  assume $\text{c-pair } x \ y \in \text{ce-rel-to-set } r$ then show $(x, y) \in r$ by (rule ce-rel-lm-11)
next
  assume $(x, y) \in r$ then show $\text{c-pair } x \ y \in \text{ce-rel-to-set } r$ by (rule ce-rel-lm-9)
qed

lemma ce-rel-lm-13: $(x, y) \in \text{ce-set-to-rel } A \implies \text{c-pair } x \ y \in A$
proof –
  assume $(x, y) \in \text{ce-set-to-rel } A$
  then have $\text{c-pair } x \ y \in \text{ce-set-to-rel } (\text{ce-set-to-rel } A)$ by (rule ce-rel-lm-9)
  then show $\text{thesis}$ by simp
qed

lemma ce-rel-lm-14: $\text{c-pair } x \ y \in A \implies (x, y) \in \text{ce-set-to-rel } A$
proof –
assume \( c\text{-pair } x \ y \in A \)
then have \( c\text{-pair } x \ y \in \text{ce-set-to-rel (ce-set-to-rel } A) \) by simp
then show \( ?\text{thesis} \) by (rule ce-rel-lm-11)
qed

lemma ce-rel-lm-15: \((x,y) \in \text{ce-set-to-rel } A) = (c\text{-pair } x \ y \in A)\)
proof –
assume \((x, y) \in \text{ce-set-to-rel } A \) 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 \((x, y) \in \text{ce-set-to-rel } A \) by (rule ce-rel-lm-14)
qed

lemma ce-rel-lm-16: \( x \in \text{ce-rel-to-set } r \implies (c\text{-fst } x, c\text{-snd } x) \in r \)
proof –
assume \( x \in \text{ce-rel-to-set } r \)
then have \( (c\text{-fst } x, c\text{-snd } x) \in \text{ce-set-to-rel (ce-set-to-rel r) \) by (rule ce-rel-lm-10)
then show \( ?\text{thesis} \) by simp
qed

lemma ce-rel-lm-17: \( c\text{-fst } x, c\text{-snd } x) \in \text{ce-set-to-rel } A \implies x \in A \)
proof –
assume \( (c\text{-fst } x, c\text{-snd } x) \in \text{ce-set-to-rel } A \)
then have \( c\text{-pair } (c\text{-fst } x) (c\text{-snd } x) \in A \) by (rule ce-rel-lm-13)
then show \( ?\text{thesis} \) by simp
qed

lemma ce-rel-lm-18: \((c\text{-fst } x, c\text{-snd } x) \in \text{ce-set-to-rel } A \) = \( x \in A \)
proof –
assume \( (c\text{-fst } x, c\text{-snd } x) \in \text{ce-set-to-rel } A \)
then show \( x \in A \) by (rule ce-rel-lm-17)
next
assume \( x \in A \) then show \( (c\text{-fst } x, c\text{-snd } x) \in \text{ce-set-to-rel } A \) by (rule ce-rel-lm-10)
qed

lemma ce-rel-lm-19: \( c\text{-fst } x, c\text{-snd } x) \in r \implies x \in \text{ce-rel-to-set } r \)
proof –
assume \( (c\text{-fst } x, c\text{-snd } x) \in r \)
then have \( (c\text{-fst } x, c\text{-snd } x) \in \text{ce-set-to-rel (ce-rel-to-set r) \) by simp
then show \( ?\text{thesis} \) by (rule ce-rel-lm-17)
qed

lemma ce-rel-lm-20: \((c\text{-fst } x, c\text{-snd } x) \in r \) = \( x \in \text{ce-rel-to-set } r \)
proof
assume \( (c\text{-fst } x, c\text{-snd } x) \in r \) 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 \( (c\text{-fst } x, c\text{-snd } x) \in r \) by (rule ce-rel-lm-16)
qed

lemma ce-rel-lm-21: \( r \in \text{rels} \implies \exists \ p \in \text{PrimRec3 }, \forall x, y, ((x,y) \in r) = (\exists
proof
  assume r-ce: r ∈ ce-rels
  define A where A = ce-rel-to-set r
  from r-ce have A-ce: A ∈ ce-sets by (unfold A-def, rule ce-rel-lm-6)
  then obtain q where q-is-pr: q ∈ PrimRec2 and A-def1: A = fn-to-set q ..
  from A-def1 have A-def2: A = \{ x. \exists y. q x y = 0 \} by (unfold fn-to-set-def)
  define p where p x y u = q (c-pair x y) u for x y u
  from q-is-pr have p-is-pr: p ∈ PrimRec3 unfolding p-def by prec
  have (x,y) ∈ r = (\exists u. p x y u = 0) by auto
  proof
    assume A: (x,y) ∈ r
    define z where 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-def: z ∈ \{ x. \exists y. q x y = 0 \} by auto
    then obtain u where q u = 0 by auto
    with z-def have p x y u = 0 by (simp add: z-def p-def)
    then show \exists u. p x y u = 0 by auto
  next
    assume A: \exists u. p x y u = 0
    define z where z = c-pair x y
    from A obtain u where p x y u = 0 by auto
    then have q-z: q z = 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

lemma ce-rel-lm-22: r ∈ ce-rels ⇒ \exists p \in PrimRec3. r = { (x,y). \exists u. p x y u = 0 }
proof
  assume r-ce: r ∈ ce-rels
  then have \exists p \in PrimRec3. \forall x y. ((x,y) ∈ r) = (\exists u. p x y u = 0) by (rule ce-rel-lm-21)
  then obtain p where p-is-pr: p ∈ PrimRec3 and L1: \forall x y. ((x,y) ∈ r) = (\exists u. p x y u = 0) by auto
  from p-is-pr L1 show \?thesis by blast
  qed
lemma ce-rel-lm-23: [ p \in PrimRec3; \forall x y. ((x,y) ∈ r) = (\exists u. p x y u = 0) ]
⇒ r ∈ ce-rels
proof
  assume p-is-pr: p ∈ PrimRec3

141
\textbf{assume} \( A \) \( : \forall \ x \ y. \ ((x,y) \in r) = (\exists \ u. \ p \ x \ y \ u = 0) \)

\textbf{define} \( q \) \textbf{where} \( q \ z \ u = p \ c\text{-fst} \ z \ (c\text{-snd} \ z) \ u \) \textbf{for} \( z \ u \)

\textbf{from} \( p\text{-is-pr} \) \textbf{have} \( q\text{-is-pr} : q \in \text{PrimRec}2 \) \textbf{unfolding} \( q\text{-def} \) \textbf{by} \( \text{prec} \)

\textbf{define} \( A \) \textbf{where} \( A = \{ x. \exists \ y. \ q x y = 0 \} \)

\textbf{then have} \( A\text{-def1} : A \in \text{ce-sets} \) \textbf{by} \( \text{simp add: ce-set-lm-1} \)

\textbf{proof}\n
\textbf{show} \( A \subseteq \text{ce-rel-to-set} \ r \)

\textbf{proof}\n
\textbf{fix} \( z \) \textbf{assume} \( z\text{-in-A} : z \in A \)

\textbf{show} \( z \in \text{ce-rel-to-set} \ r \)

\textbf{proof} −

\textbf{define} \( x \) \textbf{where} \( x = c\text{-fst} \ z \)

\textbf{define} \( y \) \textbf{where} \( y = c\text{-snd} \ z \)

\textbf{from} \( z\text{-in-A} \) \textbf{A-def obtain} \( u \) \textbf{where} \( L2 : q z u = 0 \) \textbf{by} \( \text{auto} \)

\textbf{with} \( x\text{-def} \ y\text{-def} \ q\text{-def} \) \textbf{have} \( L3 : p x y u = 0 \) \textbf{by} \( \text{simp} \)

\textbf{then have} \( \exists \ u. \ p x y u = 0 \) \textbf{by} \( \text{auto} \)

\textbf{with} \( A \) \textbf{have} \( (x,y) \in r \) \textbf{by} \( \text{auto} \)

\textbf{then have} \( c\text{-pair} \ x \ y \in \text{ce-rel-to-set} \ r \) \textbf{by} \( \text{rule ce-rel-lm-9} \)

\textbf{with} \( x\text{-def} \ y\text{-def} \) \textbf{show} \( ?\text{thesis} \) \textbf{by} \( \text{simp} \)

\textbf{qed}\n
\textbf{qed}\n
\textbf{next}\n
\textbf{show} \( \text{ce-rel-to-set} \ r \subseteq A \)

\textbf{proof}\n
\textbf{fix} \( z \) \textbf{assume} \( z\text{-in-r} : z \in \text{ce-rel-to-set} \ r \)

\textbf{show} \( z \in A \)

\textbf{proof} −

\textbf{define} \( x \) \textbf{where} \( x = c\text{-fst} \ z \)

\textbf{define} \( y \) \textbf{where} \( y = c\text{-snd} \ z \)

\textbf{from} \( z\text{-in-r} \) \textbf{have} \( (c\text{-fst} \ z, c\text{-snd} \ z) \in r \) \textbf{by} \( \text{rule ce-rel-lm-16} \)

\textbf{with} \( x\text{-def} \ y\text{-def} \) \textbf{have} \( (x,y) \in r \) \textbf{by} \( \text{simp} \)

\textbf{with} \( A \) \textbf{obtain} \( u \) \textbf{where} \( L1 : p x y u = 0 \) \textbf{by} \( \text{auto} \)

\textbf{with} \( x\text{-def} \ y\text{-def} \ q\text{-def} \) \textbf{have} \( q z u = 0 \) \textbf{by} \( \text{simp} \)

\textbf{with} \( A\text{-def} \) \textbf{show} \( z \in A \) \textbf{by} \( \text{auto} \)

\textbf{qed}\n
\textbf{qed}\n
\textbf{with} \( A\text{-ce} \) \textbf{have} \( \text{ce-rel-to-set} \ r \in \text{ce-sets} \) \textbf{by} \( \text{auto} \)

\textbf{then show} \( r \in \text{ce-rels} \) \textbf{by} \( \text{rule ce-rel-lm-7} \)

\textbf{qed}\n
\textbf{lemma} \( \text{ce-rel-lm-24} : [ r \in \text{ce-rels}; s \in \text{ce-rels} ] \implies s \ O \ r \in \text{ce-rels} \)

\textbf{proof} −

\textbf{assume} \( r\text{-ce} : r \in \text{ce-rels} \)

\textbf{assume} \( s\text{-ce} : s \in \text{ce-rels} \)

\textbf{from} \( r\text{-ce} \) \textbf{have} \( \exists \ p \in \text{PrimRec}3. \forall \ x \ y. \ ((x,y) \in r) = (\exists \ u. \ p x y u = 0) \) \textbf{by} \( \text{rule ce-rel-lm-21} \)

\textbf{142}
then obtain $p-r$ where $p-r$-is-pr: $p-r \in \text{PrimRec3}$ and $R1: \forall \ x \ y. ((x,y) \in r)=(\exists \ u. \ p-r x y u = 0)$

by auto

from $s-ce$ have $\exists \ p \in \text{PrimRec3}. \forall \ x \ y. ((x,y) \in s)=(\exists \ u. \ p x y u = 0)$ by (rule ce-rel-lm-21)

then obtain $p-s$ where $p-s$-is-pr: $p-s \in \text{PrimRec3}$ and $S1: \forall \ x \ y. ((x,y) \in s)=(\exists \ u. \ p-s x y u = 0)$

by auto

define $p$ where $p x z u = (p-s x (c-fst u) (c-fst (c-snd u))) + (p-r (c-fst u) z (c-snd (c-snd u)))$

for $x \ z \ u$

from $p-r$-is-pr $p-s$-is-pr have $p$-is-pr: $p \in \text{PrimRec3}$ unfolding $p-def$ by prec

define $sr$ where $sr = s O r$

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

from $L2$ $R1$ obtain $u$-r where $L4: p-r y z u-r = 0$ by auto

define $u$ where $u = c$-pair $y$ ($c$-pair $u$-s $u$-r)

from $L3$ $L4$ have $p x z u = 0$ by (unfold $p-def$, unfold $u-def$, simp)

then show $\exists \ u. \ p x z u = 0$ by auto

qed

next

assume $A: \exists \ u. \ p x z u = 0$

show $(x, z) \in sr$

proof –

from $A$ obtain $u$ where $L1: p x z u = 0$ by auto

then have $L2: (p-s x (c-fst u) (c-fst (c-snd u))) + (p-r (c-fst u) z (c-snd (c-snd u))) = 0$ by (unfold $p-def$)

from $L2$ have $L3: p-s x (c-fst u) (c-fst (c-snd u)) = 0$ by auto

from $L2$ have $L4: p-r (c-fst u) z (c-snd (c-snd u)) = 0$ by auto

from $L3$ $S1$ have $L5: (x,(c-fst u)) \in s$ by auto

from $L4$ $R1$ have $L6: ((c-fst u),z) \in r$ by auto

from $L5$ $L6$ have $(x,z) \in s O r$ by auto

with $sr$-def show $\exists \ u. \ p x z u = 0$ by auto

qed

qed

from $p$-is-pr main have $sr \in ce-rels$ by (rule $ce-rel-lm-23$)

then show $\exists \ u. \ p x z u = 0$ by (unfold $sr$-def)

qed

lemma $ce-rel-lm-25$: $r \in ce-rels \implies r^-1 \in ce-rels$
proof

assume r-ce: r ∈ ce-rels
have r′−1 = {(y,x). (x,y) ∈ r} by auto
then have LI: ∀ x y. ((x,y) ∈ r) = ((y,x) ∈ r′−1) by auto
from r-ce 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 R1: ∀ x y. ((x,y) ∈ r) = (∃ u. p x y u = 0)

proof

let obtain p x y u

from p-is-pr have q x y u for x y u

from LI R1 have L2: ∀ x y. ((x,y) ∈ r′−1) = (∃ u. p y x u = 0)
with q-def have L3: ∀ x y. ((x,y) ∈ r′−1) = (∃ u. q x y u = 0)

with q-is-pr show ?thesis by (rule ce-rel-lm-23)

qed

lemma ce-rel-lm-26: r ∈ ce-rels ⇒ Domain r ∈ ce-sets

proof

assume r-ce: r ∈ ce-rels
have LI: ∀ x. (x ∈ Domain r) = (∃ y. (x,y) ∈ r) by auto

define A where A = ce-rel-to-set r
from r-ce have have A-ce: A ∈ ce-sets by (rule ce-rel-lm-6)
then have A-ce: A ∈ ce-sets by (unfold A-def)

have ∀ x y. ((x,y) ∈ r) = (c-pair x y ∈ ce-rel-to-set r) by (simp add: ce-rel-lm-12)
then have L2: ∀ x y. ((x,y) ∈ r) = (c-pair x y ∈ A) by (unfold A-def)
from A-ce c-fst-is-pr have L3: { c-fst z | z. z ∈ A } ∈ ce-sets by (rule ce-set-lm-7)

have L4: ∀ x. (x ∈ { c-fst z | z. z ∈ A }) = (∃ y. c-pair x y ∈ A)
proof fix x show (x ∈ { c-fst z | z. z ∈ A }) = (∃ y. c-pair x y ∈ A)

proof

assume A: x ∈ { c-fst z | z. z ∈ A }
then obtain z where z-in-A: z ∈ A and x-z: x = c-fst z by auto
from x-z have z = c-pair x (c-snd z) by simp
with z-in-A have c-pair x (c-snd z) ∈ A by auto
then show ∃ y. c-pair x y ∈ A by auto

next

assume A: ∃ y. c-pair x y ∈ A
then obtain y where y-1: c-pair x y ∈ A by auto

define z where z = c-pair x y
from y-1 have z-in-A: z ∈ A by (unfold z-def)
from z-def have (x-z: x = c-fst z ∈ ce-rels by (unfold z-def, simp)
from z-in-A x-z show x ∈ { c-fst z | z. z ∈ A } by auto

qed

qed

from LI L2 have L5: ∀ x. (x ∈ Domain r) = (∃ y. c-pair x y ∈ A) by auto
from LI L2 have L6: ∀ x. (x ∈ Domain r) = (x ∈ { c-fst z | z. z ∈ A }) by auto
then have Domain r = { c-fst z | z. z ∈ A } by auto
with L3 show Domain r ∈ ce-sets by auto

qed
lemma ce-rel-lm-27: \( r \in \text{ce-rels} \implies \text{Range } r \in \text{ce-sets} \)
proof
  assume r-ce: \( r \in \text{ce-rels} \)
  then have \( r^{-1} \in \text{ce-rels} \) by (rule ce-rel-lm-25)
  then have \( \text{Domain } (r^{-1}) \in \text{ce-sets} \) by (rule ce-rel-lm-26)
  then show \(?thesis\) by (unfold Domain-converse [symmetric])
qed

lemma ce-rel-lm-28: \( r \in \text{ce-rels} \implies \text{Field } r \in \text{ce-sets} \)
proof
  assume r-ce: \( r \in \text{ce-rels} \)
  from r-ce have \( L1: \text{Domain } r \in \text{ce-sets} \) by (rule ce-rel-lm-26)
  from r-ce have \( L2: \text{Range } r \in \text{ce-sets} \) by (rule ce-rel-lm-27)
  from \( L1 \ L2 \) have \( L3: \text{Domain } r \cup \text{Range } r \in \text{ce-sets} \) by (rule ce-union)
  then show \(?thesis\) by (unfold Field-def)
qed

lemma ce-rel-lm-29: \( [\ A \in \text{ce-sets}; \ B \in \text{ce-sets} \] \implies A \times B \in \text{ce-rels} \)
proof
  assume A-ce: \( A \in \text{ce-sets} \)
  assume B-ce: \( B \in \text{ce-sets} \)
  define \( r-a \) where \( r-a = \{(x,(\theta::\text{nat})) | x. x \in A\} \)
  define \( r-b \) where \( r-b = \{(\theta::\text{nat}),z | z. z \in B\} \)
  have \( L1: r-a \cap r-b = A \times B \) by (unfold r-a-def, unfold r-b-def, auto)
  have \( r-a-ce: r-a \in \text{ce-rels} \)
  proof
    have loc1: \( \text{ce-rel-to-set } r-a = \{ \text{c-pair } x 0 | x. x \in A\} \) by (unfold r-a-def, unfold ce-rel-to-set-def, auto)
    define \( p \) where \( p x = \text{c-pair } x 0 \) for \( x \)
    have \( p-is-pr: p \in \text{PrimRec1} \) unfolding p-def by prec
    from A-ce p-is-pr have \( \{ \text{c-pair } x 0 | x. x \in A\} \in \text{ce-sets} \)
      unfolding p-def by (simp add: ce-set-lm-7)
    with \( loc1 \) have \( \text{ce-rel-to-set } r-a \in \text{ce-sets} \) by auto
    then show \(?thesis\) by (rule ce-rel-lm-7)
  qed
  have \( r-b-ce: r-b \in \text{ce-rels} \)
  proof
    have loc1: \( \text{ce-rel-to-set } r-b = \{ \text{c-pair } \theta z | z. z \in B\} \)
      by (unfold r-b-def, unfold ce-rel-to-set-def, auto)
    define \( p \) where \( p z = \text{c-pair } \theta z \) for \( z \)
    have \( p-is-pr: p \in \text{PrimRec1} \) unfolding p-def by prec
    from B-ce p-is-pr have \( \{ \text{c-pair } \theta z | z. z \in B\} \in \text{ce-sets} \)
      unfolding p-def by (simp add: ce-set-lm-7)
    with \( loc1 \) have \( \text{ce-rel-to-set } r-b \in \text{ce-sets} \) by auto
    then show \(?thesis\) by (rule ce-rel-lm-7)
  qed
  from \( r-b-ce \ r-a-ce \) have \( r-a \cap r-b \in \text{ce-rels} \) by (rule ce-rel-lm-24)
  with \( L1 \) show \(?thesis\) by auto

145
\textbf{qed}

\textbf{lemma ce-rel-lm-30}: \{\} \in ce-rels
\textbf{proof} –
\begin{itemize}
  \item \textbf{have} ce-rel-to-set \{\} = \{\} \textbf{by} (unfold ce-rel-to-set-def, auto)
  \item \textbf{with} ce-empty \textbf{have} ce-rel-to-set \{\} \in ce-sets \textbf{by} auto
\end{itemize}
\textbf{then show} \textit{thesis} \textbf{by} (rule ce-rel-lm-7)
\textbf{qed}

\textbf{lemma ce-rel-lm-31}: UNIV \in ce-rels
\textbf{proof} –
\textbf{from} ce-univ ce-univ \textbf{have} UNIV \times UNIV \in ce-rels \textbf{by} (rule ce-rel-lm-29)
\textbf{then show} \textit{thesis} \textbf{by} auto
\textbf{qed}

\textbf{lemma ce-rel-lm-32}: \textit{ce-rel-to-set} (r \cup s) = (ce-rel-to-set r) \cup (ce-rel-to-set s) \textbf{by} (unfold ce-rel-to-set-def, auto)
\textbf{proof} –
\begin{itemize}
  \item \textbf{assume} r \in ce-rels
  \item \textbf{then have} r-ce: ce-rel-to-set r \in ce-sets \textbf{by} (rule ce-rel-lm-6)
  \item \textbf{assume} s \in ce-rels
  \item \textbf{then have} s-ce: ce-rel-to-set s \in ce-sets \textbf{by} (rule ce-rel-lm-6)
  \item \textbf{have} ce-rel-to-set (r \cup s) = (ce-rel-to-set r) \cup (ce-rel-to-set s) \textbf{by} (unfold ce-rel-to-set-def, auto)
  \item \textbf{moreover from} r-ce s-ce \textbf{have} (ce-rel-to-set r) \cup (ce-rel-to-set s) \in ce-sets \textbf{by} (rule ce-union)
  \item \textbf{ultimately have} ce-rel-to-set (r \cup s) \in ce-sets \textbf{by} auto
  \item \textbf{then show} \textit{thesis} \textbf{by} (rule ce-rel-lm-7)
\end{itemize}
\textbf{qed}

\textbf{lemma ce-rel-lm-34}: \textit{ce-rel-to-set} (r \cap s) = (ce-rel-to-set r) \cap (ce-rel-to-set s)
\textbf{proof} –
\begin{itemize}
  \item \textbf{show} ce-rel-to-set (r \cap s) \subseteq ce-rel-to-set r \cap ce-rel-to-set s \textbf{by} (unfold ce-rel-to-set-def, auto)
\end{itemize}
\textbf{next}
\textbf{show} ce-rel-to-set r \cap ce-rel-to-set s \subseteq ce-rel-to-set (r \cap s)
\textbf{proof fix} x \textbf{assume} A: x \in ce-rel-to-set r \cap ce-rel-to-set s
\textbf{from} A \textbf{have} L1: x \in ce-rel-to-set r \textbf{by} auto
\textbf{from} A \textbf{have} L2: x \in ce-rel-to-set s \textbf{by} auto
\textbf{from} L1 \textbf{obtain} u v \textbf{where} L3: (u,v) \in r \textbf{and} L4: x = c-pair u v
\textbf{unfolding} ce-rel-to-set-def \textbf{by} auto
\textbf{from} L2 \textbf{obtain} u1 v1 \textbf{where} L5: (u1,v1) \in s \textbf{and} L6: x = c-pair u1 v1
\textbf{unfolding} ce-rel-to-set-def \textbf{by} auto
\textbf{from} L4 L6 \textbf{have} L7: c-pair u1 v1 = c-pair u v \textbf{by} auto
\textbf{then have} u1 = u \textbf{by} (rule c-pair-inj1)
\textbf{moreover from} L7 \textbf{have} v1 = v \textbf{by} (rule c-pair-inj2)
\textbf{ultimately have} (u,v) = (u1,v1) \textbf{by} auto

\textbf{qed}
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 \text{ce-rel-to-set-def}, auto)

qed

lemma ce-rel-lm-35: \[ r \in \text{ce-rels}; s \in \text{ce-rels} \] \(\implies r \cap s \in \text{ce-rels}\)

proof –
assume \(r \in \text{ce-rels}\)
then have \(r-\text{ce}: \text{ce-rel-to-set} r \in \text{ce-sets}\) by (rule ce-rel-lm-6)
assume \(s \in \text{ce-rels}\)
then have \(s-\text{ce}: \text{ce-rel-to-set} s \in \text{ce-sets}\) by (rule ce-rel-lm-6)

have \(\text{ce-rel-to-set} (r \cap s) = (\text{ce-rel-to-set} r) \cap (\text{ce-rel-to-set} s)\) by (rule ce-rel-lm-34)
moreover from \(r-\text{ce} s-\text{ce}\) 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 \text{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 –
define \(f\) where \(f x = (\text{c-fst} x, \text{c-snd} x)\) for \(x\)

have \(f-\text{inj}\): \(\text{inj} f\)

proof (unfold \text{f-def}, rule inj-on-inverseI \[\text{where} \ ?g=\lambda (u,v). \text{c-pair} u v]\)
fix \(x::\text{nat}\)
assume \(x \in \text{UNIV}\)
show \(\text{case-prod c-pair} (\text{c-fst} x, \text{c-snd} x) = x\) by simp

qed

from \(f-\text{inj}\) have \(f^{-1} (A \cap B) = f^{-1} A \cap f^{-1} B\) by (rule image-Int)
then show \(?\text{thesis}\) by (unfold \text{f-def}, unfold \text{ce-set-to-rel-def}, auto)

qed

lemma ce-rel-lm-38: \[ r \in \text{ce-rels}; A \in \text{ce-sets} \] \(\implies r\text{"}A \in \text{ce-sets}\)

proof –
assume \(r-\text{ce}: r \in \text{ce-rels}\)
assume \(A-\text{ce}: A \in \text{ce-sets}\)

have L1: \(r\text{"}A = \text{Range} (r \cap A \times \text{UNIV})\) 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 \(A \times \text{UNIV} \in \text{ce-sets}\)
proof (rule ce-rel-lm-29)

show \(A \in \text{ce-sets}\) by (rule A-ce)

next


show $UNIV \in ce$-sets by (rule ce-univ)

qed

qed

from $L1 \ L2$ show $\text{thesis}$ by auto

qed

7.6 Total computable functions

definition

c-graph :: $(nat \Rightarrow nat) \Rightarrow (nat \times nat)$ set where
c-graph $= (\lambda f. \{ (x, f x) | x. x \in UNIV \})$

 lemma graph-lm-1: $(x, y) \in graph f \Longrightarrow y = f x$ by (unfold graph-def, auto)

 lemma graph-lm-2: $y = f x \Longrightarrow (x, y) \in graph f$ by (unfold graph-def, auto)

 lemma graph-lm-3: $((x, y) \in graph f) = (y = f x)$ by (unfold graph-def, auto)

 lemma graph-lm-4: $graph (f \circ g) = (graph g) O (graph f)$ by (unfold graph-def, auto)

definition

c-graph :: $(nat \Rightarrow nat) \Rightarrow nat$ set where
c-graph $= (\lambda f. \{ c\text{-pair} x (f x) | x. x \in UNIV \})$

 lemma c-graph-lm-1: $c\text{-pair} x y \in c$-graph $f \Longrightarrow y = f x$

 proof
  assume $A$: $c\text{-pair} x y \in c$-graph $f$
  have $S1$: $c$-graph $f = \{ c\text{-pair} x (f x) | x. x \in UNIV \}$ by (simp add: c-graph-def)
  from $A$ $S1$ obtain $z$ where $S2$: $c\text{-pair} x y = c\text{-pair} z (f z)$ 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 $\text{thesis}$ by auto

 qed

 lemma c-graph-lm-2: $y = f x \Longrightarrow c\text{-pair} x y \in c$-graph $f$

 proof
  assume $c$-pair $x y \in c$-graph $f$
  then show $y = f x$ by (rule c-graph-lm-1)

 next
  assume $y = f x$
  then show $c$-pair $x y \in c$-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 c-c-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)
next
  assume c-graph \( f \) ∈ ce-sets
  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
  define \( p \) where \( p \ x = c\text{-pair} \ x \ (f \ x) \) for \( x \)
  from A have p-is-pr: \( p \ ∈ PrimRec1 \) unfolding p-def by prec
  let ?U = \{ \( p \ x \ | \ x \ ∈ UNIV \} \)
  from cc-univ p-is-pr have U-ce: \( ?U \ ∈ ce\text{-sets} \) by (rule cc-set-lm-7)
  have U-1: \( ?U = \{ c\text{-pair} \ x \ (f \ x) \ | \ x \ ∈ UNIV \} \) by (simp add: p-def)
  with U-ce have S1: \( \{ c\text{-pair} \ x \ (f \ x) \ | \ x \ ∈ UNIV \} \ ∈ ce\text{-sets} \) by simp
  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 \( g \ O \ f \) ∈ 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 –
define \( A \) where \( A = c\text{-}graph \ univ\text{-}for\text{-}pr \)
from \( A\text{-}def \) have \( S1 \colon A = \{ \text{c\text{-}pair} x \ (\text{univ}\text{-}for\text{-}pr x) \mid x. x \in UNIV \} \)
  by (simp add: c\text{-}graph\text{-}def)
from \( S1 \) have \( S2 \colon A = \{ z. \exists x. z = \text{c\text{-}pair} x \ (\text{univ}\text{-}for\text{-}pr x) \} \) by auto
have \( S3 \colon \bigwedge z. (\exists x. (z = \text{c\text{-}pair} x (\text{univ}\text{-}for\text{-}pr x))) = (\text{univ}\text{-}for\text{-}pr (\text{c\text{-}fst} z) = \text{c\text{-}snd} z) \)
proof –
  fix \( z \) show \((\exists x. (z = \text{c\text{-}pair} x (\text{univ}\text{-}for\text{-}pr x))) = (\text{univ}\text{-}for\text{-}pr (\text{c\text{-}fst} z) = \text{c\text{-}snd} z) \)
proof
  assume \( A \colon \exists x. z = \text{c\text{-}pair} x (\text{univ}\text{-}for\text{-}pr x) \)
  then obtain \( x \) where \( S3\text{-}1 \colon z = \text{c\text{-}pair} x (\text{univ}\text{-}for\text{-}pr x) \)
  then show \( \text{univ\text{-}for\text{-}pr (c\text{-}fst} z) = \text{c\text{-}snd} z \) by simp
next
  assume \( A \colon \text{univ\text{-}for\text{-}pr (c\text{-}fst} z) = \text{c\text{-}snd} z \)
from \( A \) have \( z = \text{c\text{-}pair} (\text{c\text{-}fst} z) (\text{univ}\text{-}for\text{-}pr (\text{c\text{-}fst} z)) \) by simp
thus \( \exists x. z = \text{c\text{-}pair} x (\text{univ}\text{-}for\text{-}pr x) \)
qed
qed
with \( S2 \) have \( S4 \colon A = \{ z. \text{univ\text{-}for\text{-}pr (c\text{-}fst} z) = \text{c\text{-}snd} z \} \) by auto

define \( p \) where \( p \ x \ y = \)
  (if \( \text{c\text{-}assoc\text{-}have\text{-}key} (\text{pr\text{-}gr} y) (\text{c\text{-}fst} x) = 0 \) then
  (if \( \text{c\text{-}assoc\text{-}value} (\text{pr\text{-}gr} y) (\text{c\text{-}fst} x) = \text{c\text{-}snd} x \) then \( 0 :: \text{nat} \) else \( 1 \))
  else \( 1 \)) for \( x \ y \)
from \( \text{c\text{-}assoc\text{-}have\text{-}key\text{-}is\text{-}pr} \ \text{c\text{-}assoc\text{-}value\text{-}is\text{-}pr} \ \text{pr\text{-}gr\text{-}is\text{-}pr} \) have \( \text{p\text{-}is\text{-}pr} \colon p \in \text{PrimRec} \)

unfolding \( p\text{-}def \) by \( \text{prec} \)
have \( S5 \colon \bigwedge z. (\text{univ\text{-}for\text{-}pr (c\text{-}fst} z) = \text{c\text{-}snd} z) = (\exists y. p \ y \ y = 0) \)
proof –
  fix \( z \) show \( (\text{univ\text{-}for\text{-}pr (c\text{-}fst} z) = \text{c\text{-}snd} z) = (\exists y. p \ y \ y = 0) \)
proof
  assume \( A \colon \text{univ\text{-}for\text{-}pr (c\text{-}fst} z) = \text{c\text{-}snd} z \)
  let \( ?n = \text{c\text{-}fst} (\text{c\text{-}fst} z) \)
  let \( ?x = \text{c\text{-}snd} (\text{c\text{-}fst} z) \)
  let \( ?y = \text{loc\text{-}upb} ?n ?x \)
  have \( S5\text{-}1 \colon \text{c\text{-}assoc\text{-}have\text{-}key (pr\text{-}gr} ?y) (\text{c\text{-}pair} ?n ?x) = 0 \) by (rule loc\text{-}apb\text{-}main)
  have \( S5\text{-}2 \colon \text{c\text{-}assoc\text{-}value (pr\text{-}gr} ?y) (\text{c\text{-}pair} ?n ?x) = \text{univ\text{-}for\text{-}pr (c\text{-}pair} ?n ?x \)
by (rule pr\text{-}gr\text{-}value)
from \( S5\text{-}1 \) have \( S5\text{-}3 \colon \text{c\text{-}assoc\text{-}have\text{-}key (pr\text{-}gr} ?y) (\text{c\text{-}fst} z) = 0 \) by simp
from \( S5\text{-}2 \) \( A \) have \( S5\text{-}4 \colon \text{c\text{-}assoc\text{-}value (pr\text{-}gr} ?y) (\text{c\text{-}fst} z) = \text{c\text{-}snd} z \) by simp
from \( S5\text{-}3 \) \( S5\text{-}4 \) have \( p \ z \ y = 0 \) by (simp add: p\text{-}def)
thus \( \exists y. p \ y \ y = 0 \)
next
  assume \( A \colon \exists y. p \ y \ y = 0 \)
  then obtain \( y \) where \( S5\text{-}1 \colon p \ y \ y = 0 \)
  have \( S5\text{-}2 \colon \text{c\text{-}assoc\text{-}have\text{-}key (pr\text{-}gr} y) (\text{c\text{-}fst} z) = 0 \)
  proof \( \text{(rule ccontr)} \)
    assume \( A\text{-}1 \colon \text{c\text{-}assoc\text{-}have\text{-}key (pr\text{-}gr} y) (\text{c\text{-}fst} z) \neq 0 \)
    then have \( p \ y \ y = 1 \) by (simp add: p\text{-}def)
with S5-1 show False by auto 
qed

then have S5-3: p z y = (if c-assoc-value (pr-gr y) (c-fst z) = c-snd z then (0::nat) else 1) by (simp add: p-def)
have S5-4: c-assoc-value (pr-gr y) (c-fst z) = c-snd z
proof (rule ccontr)
  assume A-2: c-assoc-value (pr-gr y) (c-fst z) ≠ c-snd z
  then have p z y = 1 by (simp add: p-def)
  with S5-1 show False by auto
qed

have S5-5: c-is-sub-fun (pr-gr y) univ-for-pr by (rule pr-gr-1)
from S5-5 S5-2 have S5-6: c-assoc-value (pr-gr y) (c-fst z) = univ-for-pr (c-fst z) by (rule c-is-sub-fun-lm-1)
  with S5-4 show univ-for-pr (c-fst z) = c-snd z by auto
qed

from S5 S4 have A = {z. ∃ y. p z y = 0} by auto
then have A = 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 have A ∈ ce-sets by auto
with A-def show ?thesis by auto
qed

theorem univ-for-pr-tot-rec: total-recursive univ-for-pr
proof –
  have c-graph univ-for-pr ∈ 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 = (A 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 $A \rightarrow$ total-recursive ($chf A$)
proof
  assume $A$: computable $A$
  from $A$ have $A1$: $A \in ce$-sets by (unfold computable-def, simp)
  from $A$ have $A2$: $-A \in ce$-sets by (unfold computable-def, simp)
  define $p$ where $p x = c$-pair $x \, 0$ for $x$
  define $q$ where $q x = c$-pair $x \, 1$ for $x$
  from $p$-def have $p$-is-pr: $p \in$ PrimRec1 unfolding $p$-def by prec
  from $q$-def have $q$-is-pr: $q \in$ PrimRec1 unfolding $q$-def by prec
  define $U0$ where $U0 = \{ p x | x, x \in A \}$
  define $U1$ where $U1 = \{ q x | x, x \in - A \}$
  from $A1$ $p$-is-pr have $U0$-ce: $U0 \in ce$-sets by (unfold $U0$-def, rule ce-set-lm-7)
  from $A2$ $q$-is-pr have $U1$-ce: $U1 \in ce$-sets by (unfold $U1$-def, rule ce-set-lm-7)
  define $U$ where $U = U0 \cup U1$
  from $U0$-ce $U1$-ce have $U$-ce: $U \in ce$-sets by (unfold $U$-def, rule ce-union)
  define $V$ where $V = c$-graph ($chf A$)
  have $V$-1: $V = \{ c$-pair $x \ (chf A 
  x) | x, x \in UNIV \}$ by (simp add: $V$-def c-graph-def)
  from $U0$-def $p$-def have $U0$-1: $U0 = \{ c$-pair $x 
  y | x, y, x \in A \wedge y=0 \}$ by auto
  from $U1$-def $q$-def have $U1$-1: $U1 = \{ c$-pair $x \n  y | x, y, x \notin A \wedge y=1 \}$ by auto
  from $U0$-1 $U1$-1 $U$-def have $U$-1: $U = \{ c$-pair $x \n  y | x, y, (x \in A \wedge y=0) \vee (x \notin A \wedge y=1) \}$ by auto
  from $V$-1 have $V$-2: $V = \{ c$-pair $x \n  y | x, y \in chf A \}$ by auto
  have $L1$: $\forall x, y. ((x \in A \wedge y=0) \vee (x \notin A \wedge y=1)) = (y = chf A 
  x)$
  proof
    fix $x, y$
    show $((x \in A \wedge y=0) \vee (x \notin A \wedge y=1)) = (y = chf A 
  x)$
      by (unfold chf-def, auto)
    qed
  from $V$-2 $U$-1 $L1$ have $U=V$ by simp
  with $U$-ce have $V$-ce: $V \in ce$-sets by auto
  with $V$-def have $c$-graph ($chf A$) $\in$ ce-sets by auto
  then show $\Psi$thesis by (unfold total-recursive-def1)
  qed

theorem tot-rec-impl-comp: total-recursive ($chf A$) $\rightarrow$ computable $A$
proof
  assume $A$: total-recursive ($chf A$)
  then have $A1$: $c$-graph ($chf A$) $\in$ ce-sets by (unfold total-recursive-def1)
  let $?U = c$-graph ($chf A$)
  have $L1$: $?U = \{ c$-pair $x \ (chf A 
  x) | x, x \in UNIV \}$ by (simp add: $c$-graph-def)
  have $L2$: $\forall x, y. ((x \in A \wedge y=0) \vee (x \notin A \wedge y=1)) = (y = chf A 
  x)$
  proof
    fix $x, y$
    show $((x \in A \wedge y=0) \vee (x \notin A \wedge y=1)) = (y = chf A 
  x)$
      by (unfold chf-def, auto)
    qed
  from $L1$ $L2$ have $L3$: $?U = \{ c$-pair $x y | x, y, (x \in A \wedge y=0) \vee (x \notin A \wedge 
  y=1) \}$ by auto
  define $p$ where $p x = c$-pair $x \, 0$ for $x$
define q where q x = c-pair x 1 for x
have p-is-pr: p ∈ PrimRec1 unfolding p-def by prec
have q-is-pr: q ∈ PrimRec1 unfolding q-def by prec
define V where 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: ∃ 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: ∃ x. q x ∈ V
proof -
assume A: q x ∈ V
from A have L5-1: ∃ x1 y1. c-pair x 1 = c-pair x1 y1 ∧ ((x1 ∈ A ∧ y1 = 0) ∨ (x1 ∈ A ∧ y1 = 1)) by auto
next
assume A: x ∈ A
from A have L5-1: ∃ x1 y1. c-pair x 1 = c-pair x1 y1 ∧ ((x1 ∈ A ∧ y1 = 0) ∨ (x1 ∈ 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 \textit{thesis} by auto

qed

from $V$: $\forall x. q \in V \in \text{ce-sets}$ by (rule ce-set-lm-5)

with $CA$: $\forall x. A \in \text{ce-sets}$ by simp

from $A$: $\forall x. A \in \text{ce-sets}$ by simp add: computable-def

qed

\begin{theorem}
\textit{post-th-0:} $(\text{computable } A) = (\text{total-recursive } (\text{chf } A))$
\end{theorem}

\begin{proof}
assume $\text{computable } A$ then show $\text{total-recursive } (\text{chf } A)$ by (rule comp-impl-tot-rec)
\end{proof}

\begin{proof}
assume $\text{total-recursive } (\text{chf } A)$ then show $\text{computable } A$ by (rule tot-rec-impl-comp)
\end{proof}

\section{7.8 Universal computably enumerable set}

\begin{definition}
\textit{univ-ce} :: \textit{nat set}
where
$\text{univ-ce} = \{ \text{c-pair } n \mid n \in \text{nat-to-ce-set } n \}$
\end{definition}

\begin{lemma}
\textit{univ-for-pr-lm:} $(\text{univ-for-pr } (\text{c-pair } n x)) = (\text{nat-to-pr } n) x$
\end{lemma}

\begin{proof}
\begin{proof}
define $A$ where $A = \text{graph univ-for-pr}$
then have $A \in \text{ce-sets}$ by (simp add: univ-for-pr-tot-rec-lm)
\end{proof}
\begin{proof}
define $p$ where $p z y = p (\text{c-pair } (\text{c-pair } (\text{c-fst } z) (\text{c-snd } y)) 0) (\text{c-snd } y)$
\end{proof}
\begin{proof}
define $p$ where $p \in \text{PrimRec2}$ unfolding $p$-def by prec
have $\forall z. (\exists n x. z = \text{c-pair } n x \land x \in \text{nat-to-ce-set } n) = (\text{c-snd } z \in \text{nat-to-ce-set } (\text{c-fst } z))$
\end{proof}
\begin{proof}
define $A$ where $A = \exists n x. z = \text{c-pair } n x \land x \in \text{nat-to-ce-set } n$
then obtain $n x$ where $L1: z = \text{c-pair } n x \land x \in \text{nat-to-ce-set } n$ by auto
\end{proof}
\begin{proof}
define $L2: z = \text{c-pair } n x$ by auto
\end{proof}
\begin{proof}
define $L3: x \in \text{nat-to-ce-set } n$ by auto
\end{proof}
\begin{proof}
define $L4: \text{c-fst } z = n$ by simp
\end{proof}
\begin{proof}
define $L5: \text{c-snd } z = x$ by simp
\end{proof}
\begin{proof}
define $L3 L4 L5$ show $\text{c-snd } z \in \text{nat-to-ce-set } (\text{c-fst } z)$ by auto
\end{proof}
\end{proof}

154
assume $A$: $c$-snd $z \in \text{nat-to-ce-set} (c\text{-fst } z)$

let $\exists n = c\text{-fst } z$

let $\exists x = c$-snd $z$

have $L1$: $z = c$-pair $\exists n \exists x$ by simp

from $L1$ $A$ have $z = c$-pair $\exists n \exists x \land \exists x \in \text{nat-to-ce-set} \exists n$ by auto

thus $\exists n. x \cdot z = c$-pair $n \cdot x \land x \in \text{nat-to-ce-set} \exists n$ by blast

qed

qed

then have $\{ \text{c-pair } n \cdot x \mid n \cdot x \in \text{nat-to-ce-set } n \} = \{ z. \text{c-snd } z \in \text{nat-to-ce-set} (c\text{-fst } z) \}$ by auto

then have $S3$: $\text{univ-ce} = \{ z. \text{c-snd } z \in \text{nat-to-ce-set} (c\text{-fst } z) \}$ by (simp add: univ-ce-def)

have $S4$: $\forall z. (\text{c-snd } z \in \text{nat-to-ce-set} (c\text{-fst } z)) = (\exists y. p \cdot z \cdot y = 0)$

proof

fix $z$ show $(\text{c-snd } z \in \text{nat-to-ce-set} (c\text{-fst } z)) = (\exists y. p \cdot z \cdot y = 0)$

proof

assume $A$: $\exists n = c$-snd $z \in \text{nat-to-ce-set} (c\text{-fst } z)$

have univ-ce-set $(c\text{-fst } z) = (\exists x. \exists y. (\text{nat-to-pr} (c\text{-fst } z)) (c\text{-pair } x y) = 0 \}$ by (simp add: nat-to-ce-set-lm-1)

with $A$ obtain $u$ where $S4-1$: $(\text{nat-to-pr} (c\text{-fst } z)) (c\text{-pair } (c\text{-snd } z) u) = 0$ by auto

then have $S4-2$: $\text{univ-for-pr} (c\text{-pair } (c\text{-fst } z) (c\text{-pair } (c\text{-snd } z) u)) = 0$ by (simp add: univ-for-pr-lm)

from $A$-def have $S4-3$: $\exists n = c$-pair $x \cdot (\text{univ-for-pr } x) \mid x \cdot x \in \text{UNIV }$ by (simp add: c-graph-def)

then have $S4-4$: $\forall x. \text{c-pair } x \cdot (\text{univ-for-pr } x) \in A$ by auto

then have $c$-pair $(c\text{-pair } (c\text{-fst } z) (c\text{-pair } (c\text{-snd } z) u)) (\text{univ-for-pr} (c\text{-pair } (c\text{-fst } z) (c\text{-pair } (c\text{-snd } z) u))) \in A$ by auto

with $S4-2$ have $S4-5$: $\text{c-pair } (c\text{-pair } (c\text{-fst } z) (c\text{-pair } (c\text{-snd } z) u)) \in A$ by auto

with $S2$ obtain $v$ where $S4-6$: $pA (c\text{-pair } (c\text{-pair } (c\text{-fst } z) (c\text{-pair } (c\text{-snd } z) u)) 0) \cdot v = 0$

by auto

define $y$ where $y = c$-pair $u \cdot v$

from $y$-def have $S4-7$: $u = c\text{-fst } y$ by simp

from $y$-def have $S4-8$: $v = c\text{-snd } y$ by simp

from $S4-6$ $S4-7$ $S4-8$ $p$-def have $p \cdot z \cdot y = 0$ by simp

thus $\exists y. p \cdot z \cdot y = 0$ ..

next

assume $A$: $\exists y. p \cdot z \cdot y = 0$

then obtain $y$ where $S4-1$: $p \cdot z \cdot y = 0$ ..

from $S4-1$ $p$-def have $S4-2$: $pA (c\text{-pair } (c\text{-pair } (c\text{-fst } z) (c\text{-pair } (c\text{-snd } z) (c\text{-fst } y))) 0) (c\text{-snd } y) = 0$ by simp

with $S2$ have $S4-3$: $c$-pair $(c\text{-pair } (c\text{-fst } z) (c\text{-pair } (c\text{-snd } z) (c\text{-fst } y))) 0 \in A$ by auto

with $A$-def have $c$-pair $(c\text{-pair } (c\text{-fst } z) (c\text{-pair } (c\text{-snd } z) (c\text{-fst } y))) 0 \in c$-graph univ-for-pr by simp

then have $S4-4$: $0 = \text{univ-for-pr} (c\text{-pair } (c\text{-fst } z) (c\text{-pair } (c\text{-snd } z) (c\text{-fst } y)))$

by (rule c-graph-lm-1)
then have S4-5: \( \text{univ-for-pr} \ (\text{c-pair} \ (\text{c-fst} \ z) \ (\text{c-pair} \ (\text{c-snd} \ z) \ (\text{c-fst} \ y))) = 0 \) by auto
then have S4-6: \( \text{nat-to-pr} \ (\text{c-fst} \ z) \) (\text{c-pair} \ (\text{c-snd} \ z) \ (\text{c-fst} \ y)) = 0 \) by (simp add: univ-for-pr-lm)
then have S4-7: \( \exists y. \ (\text{nat-to-pr} \ (\text{c-fst} \ z)) \ (\text{c-pair} \ (\text{c-snd} \ z) y) = 0 \) by auto
with S4-8 show \( \text{c-snd} \ z \in \text{nat-to-ce-set} \ (\text{c-fst} \ z) \) by auto
qed

lemma univ-ce-lm-1: \( (\text{c-pair} \ n \ x \in \text{univ-ce}) = (x \in \text{nat-to-ce-set} \ n) \)
proof -
from univ-ce-def have S1: \( \text{univ-ce} = \{ z . \exists n \ x. \ z = \text{c-pair} \ n \ x \land x \in \text{nat-to-ce-set} \ n \} \) by auto
have S2: \( \exists n1 \ x1. \ \text{c-pair} \ n \ x = \text{c-pair} \ n1 \ x1 \land x1 \in \text{nat-to-ce-set} \ n1 \) = \( (x \in \text{nat-to-ce-set} \ n) \)
proof
assume \( \exists n1 \ x1. \ \text{c-pair} \ n \ x = \text{c-pair} \ n1 \ x1 \land x1 \in \text{nat-to-ce-set} \ n1 \)
then obtain \( n1 \ x1 \) where L1: \( \text{c-pair} \ n \ x = \text{c-pair} \ n1 \ x1 \land x1 \in \text{nat-to-ce-set} \ n1 \) by auto
next
assume A: \( x \in \text{nat-to-ce-set} \ n \)
then have \( \text{c-pair} \ n \ x = \text{c-pair} \ n \ x \land x \in \text{nat-to-ce-set} \ n \) 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

definition p where p x = c-pair x x for x
define p-is-pr: \( p \in \text{PrimRec1} \)
proof (rule p_is_pr)
assume \( \neg \text{univ-ce} \notin \text{ce-sets} \)
then have A: \( \neg \text{univ-ce} \in \text{ce-sets} \) by auto
define A1 where A = \( \{ x, p \in \neg \text{univ-ce} \} \)
from A p-is-pr have \( \{ x, p \in \neg \text{univ-ce} \} \in \text{ce-sets} \) 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 \( A\text{-def} \) have \( (n \in A) = (p \ n \in - \ \text{univ-ce}) \) by \( \text{auto} \)

with \( p\text{-def} \) have \( (n \in A) = (c\text{-pair } n \ n \notin \text{univ-ce}) \) by \( \text{auto} \)

with \( \text{univ-ce-def} \ \text{univ-ce-lm-1} \) have \( (n \in A) = (n \notin \text{nat-to-ce-set } n) \) by \( \text{auto} \)

thus \( \text{False} \) by \( \text{auto} \)

qed

**theorem** \( \text{univ-ce-is-not-comp2} \): \( \neg \text{total-recursive (chf univ-ce)} \)

**proof**

assume \( \text{total-recursive (chf univ-ce)} \)

then have computable univ-ce by (rule \( \text{tot-rec-impl-comp} \))

then have \( \neg \text{univ-ce} \in \text{ce-sets} \) by (unfold computable-def, \( \text{auto} \))

with \( \text{univ-ce-is-not-comp1} \) show \( \text{False} \) by \( \text{auto} \)

qed

**theorem** \( \text{univ-ce-is-not-comp3} \): \( \neg \text{computable univ-ce} \)

**proof** (rule \( \text{ccontr} \))

assume \( \neg \neg \text{computable univ-ce} \)

then have computable univ-ce by \( \text{auto} \)

then have total-recursive (chf univ-ce) by (rule \( \text{comp-impl-tot-rec} \))

with \( \text{univ-ce-is-not-comp2} \) show \( \text{False} \) by \( \text{auto} \)

qed

### 7.9 s-1-1 theorem, one-one and many-one reducibilities

**definition**

\( \text{index-of-r-to-l} :: \text{nat} \)

\( \text{index-of-r-to-l} = \)

\( \text{pair-by-index} \)

\( (\text{pair-by-index} \ \text{index-of-c-fst} \ (\text{comp-by-index} \ \text{index-of-c-fst} \ \text{index-of-c-snd} )) \)

\( (\text{comp-by-index} \ \text{index-of-c-snd} \ \text{index-of-c-snd}) \)

**lemma** \( \text{index-of-r-to-l-lm: nat-to-pr index-of-r-to-l } \ (c\text{-pair } x \ (c\text{-pair } y \ z)) = c\text{-pair } \)

\( (c\text{-pair } x \ y \ z) \)

apply (unfold \( \text{index-of-r-to-l-def} \))

apply (simp add: \( \text{pair-by-index-main} \))

apply (unfold c-f-pair-def)

apply (simp add: \( \text{index-of-c-fst-main} \))

apply (simp add: \( \text{comp-by-index-main} \))

apply (simp add: \( \text{index-of-c-fst-main} \))

apply (simp add: \( \text{index-of-c-snd-main} \))

done

**definition**

\( \text{s-ce} :: \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \)

\( \text{s-ce} = (\lambda e. \ \text{s1-1 (comp-by-index } e \ \text{index-of-r-to-l}) \ x) \)

**lemma** \( \text{s-ce-is-pr: s-ce } \in \text{PrimRec2} \)
unfolding \texttt{s-ce-def} using \texttt{comp-by-index-is-pr s1-1-is-pr by prec}

\textbf{lemma s-ce-inj:} \texttt{s-ce e1 x1 = s-ce e2 x2 }\implies e1=e2 \land x1=x2

\textbf{proof} –
\begin{itemize}
\item let \(\texttt{?n1} = \text{index-of-r-to-l}\)
\item assume \texttt{s-ce e1 x1 = s-ce e2 x2}
\item then have \texttt{s1-1 (comp-by-index e1 ?n1) x1 = s1-1 (comp-by-index e2 ?n1) x2}
\end{itemize}
\begin{itemize}
\item by (unfold \texttt{s-ce-def})
\item then have \texttt{L1: comp-by-index e1 ?n1 = comp-by-index e2 ?n1 \land x1=x2 by (rule s1-1-inj)}
\item from \texttt{L1} have \texttt{comp-by-index e1 ?n1 = comp-by-index e2 ?n1 ..}
\item then have \texttt{e1=e2 by \text{rule comp-by-index-inj1}}
\item moreover from \texttt{L1} have \texttt{x1=x2 by auto}
\item ultimately show \(\texttt{?thesis by auto}\)
\end{itemize}
\textbf{qed}

\textbf{lemma s-ce-inj1:} \texttt{s-ce e1 x = s-ce e2 x }\implies e1=e2

\textbf{proof} –
\begin{itemize}
\item assume \texttt{s-ce e1 x = s-ce e2 x}
\item then have \texttt{e1=e2 \land x=x by \text{rule s-ce-inj}}
\item then show \texttt{e1=e2 by auto}
\end{itemize}
\textbf{qed}

\textbf{lemma s-ce-inj2:} \texttt{s-ce e x1 = s-ce e x2 }\implies x1=x2

\textbf{proof} –
\begin{itemize}
\item assume \texttt{s-ce e x1 = s-ce e x2}
\item then have \texttt{e=e \land x1=x2 by \text{rule s-ce-inj}}
\item then show \texttt{x1=x2 by auto}
\end{itemize}
\textbf{qed}

\textbf{theorem s1-1-th1:} \(\forall\ n x y. ((\text{nat-to-pr n} (\text{c-pair x y})) = (\text{nat-to-pr} (s1-1 n x)) y\)

\textbf{proof} (rule allI, rule allI, rule allI)
\begin{itemize}
\item fix \texttt{n x y} show \texttt{nat-to-pr n (c-pair x y) = nat-to-pr (s1-1 n x) y}
\end{itemize}
\begin{itemize}
\item proof –
\item have \(\lambda y. (\text{nat-to-pr n} (\text{c-pair x y})) = \text{nat-to-pr} (s1-1 n x)\ by \text{rule s1-1-th})
\item then show \(\texttt{thesis by (simp add: fun-eq-iff)}\)
\end{itemize}
\textbf{qed}
\textbf{qed}

\textbf{lemma s-ln:} \((\text{nat-to-pr} (\text{s-ce e x})) (\text{c-pair y z}) = (\text{nat-to-pr e}) (\text{c-pair} (\text{c-pair x y}) z)\)

\textbf{proof} –
\begin{itemize}
\item let \(\texttt{?n1 = index-of-r-to-l}\)
\item have \((\text{nat-to-pr} (\text{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 \texttt{s-ce-def, simp})
\item 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)
\item 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)
\end{itemize}

158
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 \( (\text{nat-to-pr } e) (\text{c-pair } x \ y) z) = 0 \) 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 :: \( (\text{nat set}) \Rightarrow (\text{nat set}) \Rightarrow (\text{nat} \Rightarrow \text{nat}) \Rightarrow \text{bool} \)
where
one-reducible-to-via = \( (\lambda \ A \ B \ f. \ \text{total-recursive } f \wedge \text{inj } f \wedge (\forall \ x. (x \in A) = (f x \in B))) \)

definition one-reducible-to :: \( (\text{nat set}) \Rightarrow (\text{nat set}) \Rightarrow \text{bool} \)
where
one-reducible-to = \( (\lambda \ A \ B. \ \exists \ f. \ \text{one-reducible-to-via } A \ B \ f) \)

definition many-reducible-to-via :: \( (\text{nat set}) \Rightarrow (\text{nat set}) \Rightarrow (\text{nat} \Rightarrow \text{nat}) \Rightarrow \text{bool} \)
where
many-reducible-to-via = \( (\lambda \ A \ B \ f. \ \text{total-recursive } f \wedge (\forall \ x. (x \in A) = (f x \in B))) \)

definition many-reducible-to :: \( (\text{nat set}) \Rightarrow (\text{nat set}) \Rightarrow \text{bool} \)
where
many-reducible-to = \( (\lambda \ A \ B. \ \exists \ f. \ \text{many-reducible-to-via } A \ B \ f) \)

lemma one-reducible-to-via-trans: \[ \text{one-reducible-to-via } A \ B \ f; \text{one-reducible-to-via } B \ C \ g \ \Rightarrow \text{one-reducible-to-via } A \ C \ (g \circ f) \]
proof –
assume A1: \( \text{one-reducible-to-via } A \ B \ f \)
assume A2: \( \text{one-reducible-to-via } B \ C \ g \)
from A1 have f-tr: \( \text{total-recursive } f \) by (unfold one-reducible-to-via-def, auto)
from A1 have f-inj: \( \text{inj } f \) by (unfold one-reducible-to-via-def, auto)
from A1 have L1: \( \forall \ x. (x \in A) = (f x \in B) \) by (unfold one-reducible-to-via-def, auto)
from A2 have g-tr: \( \text{total-recursive } g \) by (unfold one-reducible-to-via-def, auto)
from A2 have g-inj: \( \text{inj } g \) by (unfold one-reducible-to-via-def, auto)
from A2 have L2: \( \forall \ x. (x \in B) = (g x \in C) \) by (unfold one-reducible-to-via-def, auto)
lemma one-reducible-to-trans: \[ [ \text{one-reducible-to } A B; \text{one-reducible-to } B C ] ] \Rightarrow \text{one-reducible-to } A C
proof –
assume \( A1: \text{one-reducible-to-via } A B f \) unfolding \text{one-reducible-to-def}
by auto
assume \( A2: \text{one-reducible-to-via } B C g \) unfolding \text{one-reducible-to-def}
by auto
from \( A1 \) \( A2 \) have \( \text{one-reducible-to-via } A C (g o f) \) by \text{rule one-reducible-to-via-trans}
then show \( \text{thesis} \) unfolding \text{one-reducible-to-def} by auto
qed

lemma one-reducible-to-refl: \text{one-reducible-to } A A (\lambda x. x)
proof –
have \( \text{is-pr}: (\lambda x. x) \in \text{PrimRec1} \) by \text{rule pr-id1-1}
then have \( \text{is-tr}: \text{total-recursive } (\lambda x. x) \) by \text{rule pr-is-total-rec}
have \( \text{is-inj}: \text{inj } (\lambda x. x) \) by simp
with \( \text{is-tr is-inj} \) show \( \text{thesis} \) by \text{unfold one-reducible-to-via-def, auto}
qed

lemma many-reducible-to-via-trans: \[ [ \text{many-reducible-to-via } A B f; \text{many-reducible-to-via } B C g ] ] \Rightarrow \text{many-reducible-to-via } A C (g o f)
proof –
assume \( A1: \text{many-reducible-to-via } A B f \)
assume \( A2: \text{many-reducible-to-via } B C g \)
from \( A1 \) have \( f\text{-tr}: \text{total-recursive } f \) by \text{unfold many-reducible-to-via-def, auto}
from \( A1 \) have \( L1: \forall x. (x \in A) = (f x \in B) \) by \text{unfold many-reducible-to-via-def, auto}
from \( A2 \) have \( g\text{-tr}: \text{total-recursive } g \) by \text{unfold many-reducible-to-via-def, auto}
from \( A2 \) have \( L2: \forall x. (x \in B) = (g x \in C) \) by \text{unfold many-reducible-to-via-def, auto}
from \( g\text{-tr f-tr} \) have \( fg\text{-tr}: \text{total-recursive } (g o f) \) by \text{rule comp-tot-rec}
from \( L1 \) \( L2 \) have \( L3: \forall x. (x \in A) = ((g o f) x \in C) \) by auto
with \( fg\text{-tr} \) show \( \text{thesis} \) by \text{unfold many-reducible-to-via-def, auto}
qed

lemma many-reducible-to-trans: \[ \text{many-reducible-to } A B; \text{many-reducible-to } B C \implies \text{many-reducible-to } A C \]
proof –
  assume \text{many-reducible-to } A B
  then obtain \(f\) where \text{many-reducible-to-via } A B f
    unfolding \text{many-reducible-to-def} by auto
  assume \text{many-reducible-to } B C
  then obtain \(g\) where \text{many-reducible-to-via } B C g
    unfolding \text{many-reducible-to-def} by auto
  from \(A1\) \(A2\) have \text{many-reducible-to-via } A C (\(g \circ f\)) by (rule \text{many-reducible-to-via-trans})
  then show \?thesis unfolding \text{many-reducible-to-def} by auto
qed

lemma one-reducibility-via-is-many: \text{one-reducible-to-via } A B f \implies \text{many-reducible-to-via } A B f
proof –
  assume \(A:\) \text{one-reducible-to-via } A B f
  from \(A\) have \(f\text{-tr: total-recursive } f\) by (unfold \text{one-reducible-to-via-def}, auto)
  from \(A\) have \(\forall \ x. (x \in A) = (f x \in B)\) by (unfold \text{one-reducible-to-via-def}, auto)
    with \(f\text{-tr}\) show \?thesis by (unfold \text{many-reducible-to-via-def}, auto)
qed

lemma one-reducibility-is-many: \text{one-reducible-to } A B \implies \text{many-reducible-to } A B
proof –
  assume \text{one-reducible-to } A B
  then obtain \(f\) where \text{one-reducible-to-via } A B f
    unfolding \text{one-reducible-to-def} by auto
  then have \text{many-reducible-to-via } A B f by (rule \text{one-reducibility-via-is-many})
  then show \?thesis unfolding \text{many-reducible-to-def} by auto
qed

lemma many-reducible-to-via-refl: \text{many-reducible-to-via } A A (\(\lambda x. x\))
proof –
  have \text{one-reducible-to-via } A A (\(\lambda x. x\)) by (rule \text{one-reducible-to-via-refl})
  then show \?thesis by (rule \text{one-reducibility-via-is-many})
qed

lemma many-reducible-to-refl: \text{many-reducible-to } A A
proof –
  have \text{one-reducible-to } A A by (rule \text{one-reducible-to-refl})
  then show \?thesis by (rule \text{one-reducibility-is-many})
qed

theorem m-red-to-comp: \[ \text{many-reducible-to } A B; \text{computable } B \implies \text{computable } A \]
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-via-def, auto)
from A1 have L1: ∀ x. (x ∈ A) ⇒ (f x ∈ B) by (unfold many-reducible-to-via-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
then show chf A x = (chf B ◦ f) x by auto
qed
from L2 f-tr have total-recursive (chf B ◦ 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 =⇒ ¬ computable A
proof (rule ccontr)
  assume A1: many-reducible-to univ-ce A
  assume ¬ ¬ computable A
  then have A2: computable A by auto
  from A1 A2 have computable univ-ce by (rule m-red-to-comp)
  with univ-ce-is-not-comp3 show False by auto
qed
lemma one-reducible-lm-1: one-reducible-to univ-ce A =⇒ ¬ 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 ?thesis by (rule many-reducible-lm-1)
qed
lemma one-reducible-lm-2: one-reducible-to-via (nat-to-ce-set n) univ-ce (λ x. c-pair n x)
proof –
define \( f \) where \( f \) is a \( \lambda \)-expression for \( n \)

have \( f \)-is-pr: \( f \in \text{PrimRec1} \) unfolding \( f \)-def by prec
then have \( f \)-tr: total-recursive \( f \) by (rule pr-is-total-rec)

have \( f \)-inj: inj \( f \)
proof (rule injI)
fix \( x \), \( y \)
assume \( A \)
then have \( c \)-pair \( n \) \( x \) = \( c \)-pair \( n \) \( y \) unfolding \( f \)-def by (rule \( c \)-pair-inj2)
then show \( x \) = \( y \) by (rule \( c \)-pair-inj2)
qed

have \( \forall \) \( x \). \( (x \in (\text{nat-to-ce-set} \ n)) = (f \ x \in \text{univ-ce}) \)
proof fix \( x \)
show \( (x \in \text{nat-to-ce-set} \ n) = (f \ x \in \text{univ-ce}) \)
by (unfold \( f \)-def, simp add: \text{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 \( (\text{nat-to-ce-set} \ n) \) \( \text{univ-ce} \)
proof –

have one-reducible-to-via \( (\text{nat-to-ce-set} \ n) \) \( \text{univ-ce} \) \( (\lambda \ x. \ \text{c-pair} \ n \ x) \) 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 \( A \) \( \text{univ-ce} \)
proof –

assume \( A \in \text{ce-sets} \)
then have \( \exists \) \( n \). \( A = \text{nat-to-ce-set} \ n \) by (rule \text{nat-to-ce-set-srj})
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 = \( (\lambda A. \ A \in \text{ce-sets} \land (\forall B. \ B \in \text{ce-sets} \rightarrow \text{one-reducible-to} \ B \ A)) \)

theorem univ-is-complete: one-complete \( \text{univ-ce} \)
proof (unfold one-complete-def)

show \( \text{univ-ce} \in \text{ce-sets} \land (\forall B. \ B \in \text{ce-sets} \rightarrow \text{one-reducible-to} \ B \ \text{univ-ce}) \)
proof
show \( \text{univ-ce} \in \text{ce-sets} \) by (rule \text{univ-is-ce})
next
show \( \forall B. \ B \in \text{ce-sets} \rightarrow \text{one-reducible-to} \ B \ \text{univ-ce} \)
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

163
7.11 Index sets, Rice’s theorem

definition
index-set :: nat set ⇒ bool where
index-set = (λ A. ∀ n m. n ∈ A ∧ (nat-to-ce-set n = nat-to-ce-set m) → m ∈ A)

lemma index-set-lm-1: [ index-set A; n ∈ A; nat-to-ce-set n = nat-to-ce-set m ]
⇒ m ∈ A
proof −
  assume A1: index-set A
  assume A2: n ∈ A
  assume A3: nat-to-ce-set n = nat-to-ce-set m
  from A2 A3 have L1: n ∈ A ∧ (nat-to-ce-set n = nat-to-ce-set m) by auto
  from A1 L1 have L2: n ∈ A ∧ (nat-to-ce-set n = nat-to-ce-set m) → m ∈ A by (unfold index-set-def)
  from L2 show ?thesis by auto
qed

lemma index-set-lm-2: index-set A = index-set (−A)
proof −
  assume A: index-set A
  show index-set (−A)
  proof (unfold index-set-def)
    show ∀ n m. n ∈ − A ∧ nat-to-ce-set n = nat-to-ce-set m → m ∈ − A
    proof (rule allI, rule allI, rule impI)
      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
  qed
  qed

lemma Rice-lm-1: [ index-set A; A ≠ {}; A ≠ UNIV; ∃ n ∈ A. nat-to-ce-set n = {}; ] ⇒ one-reducible-to univ-ce (− A)
proof −
  assume A1: index-set A
  assume A2: A ≠ {}
assume $A3$: $A \neq \text{UNIV}$
assume $\exists n \in A. \text{nat-to-ce-set } n = \{\}$
then obtain $e'\emptyset$ where $e'0$-in-$A$: $e'0 \in A$ and $e$-0-empty: $\text{nat-to-ce-set } e'0 = \{\}$ by auto
from $e$-0-in-$A$ $A3$ obtain $e1$ where $e$-1-not-in-$A$: $e1 \notin (-A)$ by auto
with $e$-0-in-$A$ have $e$-0-neq-e-1: $e0 \neq e1$ by auto
have $\text{nat-to-ce-set } e0 \neq \text{nat-to-ce-set } e1$
proof
assume $\text{nat-to-ce-set } e0 = \text{nat-to-ce-set } e1$
with $A1$ $e0$-in-$A$ have $e0 \in A$ by (rule index-set-lm-1)
with $e$-0-not-in-$A$ show $\text{False}$ by auto
qed
with $e'$-0-empty have $e'$-0-empty: $\text{nat-to-ce-set } e0 \neq \{\}$ by auto
define $\text{we}$-1 where $\text{we}$-1 = $\text{nat-to-ce-set } e1$
from $\text{we}$-1-not-empty have $\text{we}$-1-not-empty: $\text{we}$-1 $\neq \{\}$ by (unfold $\text{we}$-1-def)$
define $r$ where $r = \text{univ-ce} \times \text{we}$-1
have $\text{loc-lm}$-1: $\forall x. x \in \text{univ-ce} \implies \forall y. (y \in \text{we}$-1) = $((x, y) \in r)$ by (unfold $\text{r-def}$, auto)
have $\text{loc-lm}$-2: $\forall x. x \notin \text{univ-ce} \implies \forall y. (y \in \{\}) = $ $((x, y) \in r)$ by (unfold $\text{r-def}$, auto)
have $\text{r-ce}$: $r \in \text{ce-rels}$
proof (unfold $\text{r-def}$, rule ce-rel-lm-29)
show $\text{univ-ce} \in \text{ce-sets}$ by (rule univ-is-ce)
show $\text{we}$-1 $\in \text{ce-sets}$ by (unfold $\text{we}$-1-def, rule nat-to-ce-set-into-ce)
qed
define $\text{we}$-n where $\text{we}$-n = $\text{ce-rel-to-set } r$
from $\text{r-ce}$ have $\text{we}$-n-cc: $\text{we}$-n $\in \text{ce-sets}$ by (unfold $\text{we}$-n-def, rule ce-rel-lm-6)$
then have $\exists n. \text{we}$-n $= \text{nat-to-ce-set } n$ by (rule nat-to-ce-set-srj)$
then obtain $n$ where $\text{we}$-n-def1: $\text{we}$-n $= \text{nat-to-ce-set } n$ by auto
define $f$ where $f x = \text{s-ce} n x$ for $x$
from $\text{s-ce}$-is-pr have $f$-is-pr: $f \in \text{PrimRec1}$ unfolding $\text{f-def}$ by prec
then have $f$-tr: $\text{total-recursive } f$ by (rule pr-is-total-rec)
thus $f$-inj: $\text{inj } f$
proof (rule injI)
fix $x$ $y$
assume $f x = f y$
then have $\text{s-ce } n x = \text{s-ce } n y$ by (unfold $\text{f-def}$)
then show $x = y$ by (rule s-ce-inj2)
qed
have $\text{loc-lm}$-3: $\forall x y. (\text{c-pair } x y \in \text{we}$-n) = $(y \in \text{nat-to-ce-set } (f x))$
proof (rule allI, rule allI)
fix $x$ $y$ show $(\text{c-pair } x y \in \text{we}$-n) = $(y \in \text{nat-to-ce-set } (f x))$ by (unfold $\text{f-def}$, unfold $\text{we}$-n-def1, simp add: s-ce-1-1-th)
qed
from $A1$ have $\text{loc-lm}$-4: $\text{index-set } (-A)$ by (rule index-set-lm-2)
have $\text{loc-lm}$-5: $\forall x. (x \in \text{univ-ce}) = (f x \in -A)$
proof fix $x$ show $(x \in \text{univ-ce}) = (f x \in -A)$
proof
assume $A$: $x \in \text{univ-ce}$

165
then have $S1: \forall y. ((y \in we-1) = ((x,y) \in r))$ by (rule loc-lm-1)
from $ce-rel-lm-12$ have $\forall y. (c\text{-pair } x y \in ce\text{-rel}-to-set r) = ((x,y) \in r)$ by auto
then have $\forall y. ((x,y) \in r) = (c\text{-pair } x y \in we-n)$ by (unfold we-n-def, auto)
with $S1$ have $\forall y. (y \in we-1) = (c\text{-pair } x y \in we-n)$ by auto
with $loc-lm-3$ have $\forall y. (y \in we-1) = (y \in nat\text{-to-ce-set } (f x))$ by auto
then have $S2: we-1 = nat\text{-to-ce-set } (f x)$ by auto
then have $nat\text{-to-ce-set } e-1 = nat\text{-to-ce-set } (f x)$ by (unfold we-1-def)
with $loc-lm-4$ $e\text{-1-nat-in-A}$ have $f x \in -A$ by (rule index-set-lm-1)

next

show $f x \in -A \implies x \in univ-ce$

proof (rule ccontr)
assume $fx\text{-in-A}: f x \in - A$
assume $x\text{-not-in-univ}: x \notin univ-ce$
then have $S1: \forall y. ((y \in \{\}) = ((x,y) \in r))$ by (rule loc-lm-2)
from $ce-rel-lm-12$ have $\forall y. (c\text{-pair } x y \in ce\text{-rel}-to-set r) = ((x,y) \in r)$ by auto
then have $\forall y. ((x,y) \in r) = (c\text{-pair } x y \in we-n)$ by (unfold we-n-def, auto)
with $S1$ have $\forall y. (y \in \{\}) = (c\text{-pair } x y \in we-n)$ by auto
with $loc-lm-3$ have $\forall y. (y \in \{\}) = (y \in nat\text{-to-ce-set } (f x))$ by auto
then have $S2: \{\} = nat\text{-to-ce-set } (f x)$ by auto
then have $nat\text{-to-ce-set } e\text{-0} = nat\text{-to-ce-set } (f x)$ by (unfold e-0-empty)
with $A1$ $e\text{-0-in-A}$ have $f x \in A$ by (rule index-set-lm-1)
with $fx\text{-in-A}$ show False by auto
qed

lemma $Rice-lm-2: [\{ index\text{-set } A; A \neq \{\}; A \neq UNIV; n \in A; nat\text{-to-ce-set } n = \{\} \} \implies one\text{-reducible-to } univ-ce (-A)$

proof –
assume $A1: index\text{-set } A$
assume $A2: A \neq \{\}$
assume $A3: A \neq UNIV$
assume $A4: n \in A$
assume $A5: nat\text{-to-ce-set } n = \{\}$
from $A4 A5$ have $S1: \exists n \in A. nat\text{-to-ce-set } n = \{\}$ by auto
from $A1 A2 A3 S1$ show $\{ \text{thesis} \}$ by (rule Rice-lm-1)

lemma $Rice-lm-3: [\{ index\text{-set } A; A \neq \{\}; A \neq UNIV \} \implies one\text{-reducible-to } univ-ce (A \lor one\text{-reducible-to } univ-ce (-A))$

proof –
assume $A1: index\text{-set } A$

166
assume \( A_2: A \neq \{\} \)
assume \( A_3: A \neq \text{UNIV} \)
from \( \text{ce-empty} \) have \( \exists \ n. \{\} = \text{nat-to-ce-set} \ n \) by (rule \( \text{nat-to-ce-set-srj} \))
then obtain \( n \) where \( \text{n-empty}: \text{nat-to-ce-set} \ n = \{\} \) by auto
show \(?thesis\)
proof cases
  assume \( A: n \in A \)
  from \( A_1 \) \( A_2 \) \( A_3 \) \( A \) \( \text{n-empty} \) have one-reducible-to univ-ce \(( - A)\) by (rule \( \text{Rice-lm-2} \))
  then show \(?thesis\) by auto
next
  assume \( n \notin A \) then have \( A: n \in - A \) by auto
  from \( A_1 \) have \( S_1: \text{index-set} \ (- A) \) by (rule \( \text{index-set-lm-2} \))
  from \( A_3 \) have \( S_2: - A \neq \{\} \) by auto
  from \( A_2 \) have \( S_3: - A \neq \text{UNIV} \) by auto
  from \( S_1 \) \( S_2 \) \( S_3 \) \( A \) \( \text{n-empty} \) have one-reducible-to univ-ce \(( - ( - A))\) by (rule \( \text{Rice-lm-2} \))
  then have one-reducible-to univ-ce \( A \) by simp
  then show \(?thesis\) by auto
qed

theorem \( \text{Rice-2}: [\ [ \text{index-set} \ A; A \neq \{\}; A \neq \text{UNIV} \ ] \implies \neg \text{computable} \ A \)
proof
  assume \( A_1: \text{index-set} \ A \)
  assume \( A_2: A \neq \{\} \)
  assume \( A_3: A \neq \text{UNIV} \)
  from \( A_1 \) \( A_2 \) \( A_3 \) have one-reducible-to univ-ce \( A \lor \text{one-reducible-to univ-ce} \ (- A) \) by (rule \( \text{Rice-1} \))
  then have \( S_1: \neg \text{one-reducible-to univ-ce} \ A \implies \text{one-reducible-to univ-ce} \ (- A) \) by auto
  show \(?thesis\)
  proof cases
    assume \( \neg \text{one-reducible-to univ-ce} \ A \)
    then show \(?thesis\) by (rule \( \text{one-reducible-lm-1} \))
  next
    assume \( \neg \text{one-reducible-to univ-ce} \ A \)
    with \( S_1 \) have \( \neg \text{computable} \ (- A) \) by auto
    then have \( \neg \text{computable} \ (- A) \) by (rule \( \text{one-reducible-lm-1} \))
    with \( \text{computable-complement-3} \) show \( \neg \text{computable} \ A \) by auto
  qed
qed

theorem \( \text{Rice-3}: [\ [ C \subseteq \text{ce-sets}; \text{computable} \ \{ \ n. \text{nat-to-ce-set} \ n \in C \} \ ] \implies C = \{\} \lor C = \text{ce-sets} \)
proof (rule \( \text{ccontr} \))
  assume \( A_1: C \subseteq \text{ce-sets} \)
  assume \( A_2: \text{computable} \ \{ \ n. \text{nat-to-ce-set} \ n \in C \} \)
  assume \( A_3: \neg (C = \{\} \lor C = \text{ce-sets}) \)
from A3 have A4: C ≠ {} by auto
from A3 have A5: C ≠ ce-sets by auto
define A where A = { n. nat-to-ce-set n ∈ C}
have S1: index-set A
proof (unfold index-set-def)
  show ∀ n m. n ∈ A ∧ nat-to-ce-set n = nat-to-ce-set m → m ∈ A
proof (rule allI, rule allI, rule impI)
  fix n m assume A1-1: n ∈ A ∧ nat-to-ce-set n = nat-to-ce-set m
  from A1-1 have n ∈ A by auto
  then have S1-1: nat-to-ce-set n ∈ C by (unfold A-def, auto)
  from A1-1 have nat-to-ce-set m ∈ C by auto
  then show m ∈ A by (unfold A-def, auto)
qed
qed
have S2: A ≠ {}
proof –
  from A4 obtain B where S2-1: B ∈ C by auto
  with A1 have B ∈ ce-sets by auto
  then have ∃ n. B = nat-to-ce-set n by (rule nat-to-ce-set-srj)
  then obtain n where B = nat-to-ce-set n ..
  with S2-1 have nat-to-ce-set n ∈ C by auto
  then show ?thesis by (unfold A-def, auto)
qed
have S3: A ≠ UNIV
proof –
  from A1 A5 obtain B where S2-1: B ⊈ C and S2-2: B ∈ ce-sets by auto
  from S2-2 have ∃ n. B = nat-to-ce-set n by (rule nat-to-ce-set-srj)
  then obtain n where B = nat-to-ce-set n ..
  with S2-1 have nat-to-ce-set n ⊈ C by auto
  then show ?thesis by (unfold A-def, auto)
qed
from S1 S2 S3 have ¬ computable A by (rule Rice-2)
with A2 show False unfolding A-def by auto
qed
end

References
