Recursion Theory I

Michael Nedzelsky

September 13, 2023

Abstract

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

Contents

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

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

3 Primitive recursive coding of lists of natural numbers 35

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

5 Finite sets 75

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

7.1 Basic definitions .................................................. 128
7.2 Basic properties of computably enumerable sets .......... 128
7.3 Enumeration of computably enumerable sets ............... 133
7.4 Characteristic functions ......................................... 134
7.5 Computably enumerable relations ............................... 135
7.6 Total computable functions ..................................... 148
7.7 Computable sets, Post’s theorem .............................. 151
7.8 Universal computably enumerable set ......................... 154
7.9 s-1-1 theorem, one-one and many-one reducibilities ....... 157
7.10 One-complete sets ................................................. 163
7.11 Index sets, Rice’s theorem ...................................... 163

1 Cantor pairing function

theory CPair
imports Main
begin

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

1.1 Pairing function

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

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

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

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

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

proof –
  have S1: sf(x+1) = ((x+1)∗(x+2)) div 2 by (simp add: sf-def)
  have S2: (x+1)∗(x+2) = x∗(x+1) + 2∗(x+1) by (auto)
  have S2-1: x y. x=y ⟹ x div 2 = y div 2 by auto
  from S2 have S3: (x+1)∗(x+2) div 2 = (x∗(x+1) + 2∗(x+1)) div 2 by (rule S2-1)
  have S4: (0::nat) < 2 by (auto)
  from S4 have S5: (x∗(x+1) + 2∗(x+1)) div 2 = (x+1) + x∗(x+1) div 2 by simp
from \( S1 \) \( S3 \) \( S5 \) show \( \text{?thesis} \) by (simp add: sf-def)

qed

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

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

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

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

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

lemma sf-eq-arg: \( sf \ x = x \implies x \leq 1 \)


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 $\negthesis$ by (auto)
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 $\negthesis$ by (auto)
qed

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

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

\begin{lemma}
\textit{sf-aux3: } \( sf(z) + m < sf(z+1) \implies m \leq z \)
\end{lemma}
\begin{proof}
assume \( A1: sf(z) + m < sf(z+1) \)
\begin{itemize}
\item have \( S1: sf(z+1) = sf(z) + z + 1 \) by (rule \textit{sf-at-Suc})
\item from \( A1 \) have \( S2: sf(z) + m < sf(z) + z + 1 \) by (auto)
\item from \( S2 \) have \( S3: m < z + 1 \) by (auto)
\item from \( S3 \) show \( ?\text{thesis} \) by (auto)
\end{itemize}
\end{proof}

\begin{lemma}
\textit{sf-aux4: } \( (s::nat) < t \implies (sf \ s) + s < sf \ t \)
\end{lemma}
\begin{proof}
assume \( A1: (s::nat) < t \)
\begin{itemize}
\item have \( s*(s+1) + 2*(s+1) \leq t*(t+1) \)
\item from \( A1 \) have \( S1: (s::nat) + 1 \leq t \) by (auto)
\item from \( A1 \) have \( (s::nat) + 2 \leq t+1 \) by (auto)
\item with \( S1 \) have \( ((s::nat)+1)*(s+2) \leq t*(t+1) \) by (rule \textit{mult-le-mono})
\item thus \( ?\text{thesis} \) by (auto)
\end{itemize}
\end{proof}

Basic properties of \( \textit{c_pair} \) function

\begin{lemma}
\textit{sum-le-c-pair: } \( x + y \leq \textit{c_pair} \ x \ y \)
\end{lemma}
\begin{proof}
\begin{itemize}
\item have \( x+y \leq sf(x+y) \) by (rule \textit{arg-le-sf})
\item thus \( ?\text{thesis} \) by (simp add: \textit{c_pair-def})
\end{itemize}
\end{proof}

\begin{lemma}
\textit{arg1-le-c-pair: } \( x \leq \textit{c_pair} \ x \ y \)
\end{lemma}
\begin{proof}
\begin{itemize}
\item have \( (x::nat) \leq x + y \) by (simp)
\item moreover have \( x + y \leq \textit{c_pair} \ x \ y \) by (rule \textit{sum-le-c-pair})
\item ultimately show \( ?\text{thesis} \) by (simp)
\end{itemize}
\end{proof}

\begin{lemma}
\textit{arg2-le-c-pair: } \( y \leq \textit{c_pair} \ x \ y \)
\end{lemma}
\begin{proof}
\begin{itemize}
\item have \( (y::nat) \leq x + y \) by (simp)
\item moreover have \( x + y \leq \textit{c_pair} \ x \ y \) by (rule \textit{sum-le-c-pair})
\item ultimately show \( ?\text{thesis} \) by (simp)
\end{itemize}
\end{proof}
lemma c-pair-sum-mono: \( (x1::\text{nat}) + y1 < x2 + y2 \Rightarrow \text{c-pair } x1 \text{ } y1 < \text{c-pair } x2 \text{ } y2 \)

proof –
  assume \((x1::\text{nat}) + y1 < x2 + y2\)
  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: \( \text{c-pair } x1 \text{ } y1 = \text{c-pair } x2 \text{ } y2 \Rightarrow x1 + y1 = x2 + y2 \)

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

qed

lemma c-pair-inj: \( \text{c-pair } x1 \text{ } y1 = \text{c-pair } x2 \text{ } y2 \Rightarrow x1 = x2 \text{ } \land \text{ } y1 = y2\)

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

qed

lemma c-pair-strict-mono1: \( x1 < x2 \Rightarrow \text{c-pair } x1 \text{ } y < \text{c-pair } x2 \text{ } y\)

proof –
  assume \(x1 < x2\)
  then have \(x1 + y < x2 + y\) by simp
  then show ?thesis by (rule c-pair-sum-mono)

qed

lemma c-pair-mono1: \( x1 \leq x2 \Rightarrow \text{c-pair } x1 \text{ } y \leq \text{c-pair } x2 \text{ } y\)

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

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

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

1.2 Inverse mapping

c-fst and c-snd are the functions which yield the inverse mapping to c-pair.

definition
  c-sum :: nat ⇒ nat where
c-sum u = (LEAST z. u < sf (z+1))

definition
  c-fst :: nat ⇒ nat where
c-fst u = u - sf (c-sum u)

definition
  c-snd :: nat ⇒ nat where
c-snd u = c-sum u - c-fst u
lemma \textit{arg-less-sf-at-Suc-of-c-sum}: \( u < \text{sf} \ ((\text{c-sum } u) + 1) \)

proof –
\begin{itemize}
  \item have \( u + 1 \leq \text{sf}(u+1) \) by \textit{(rule arg-le-sf)}
  \item hence \( u < \text{sf}(u+1) \) by \textit{simp}
  \item thus \( ?\text{thesis} \) by \textit{(unfold c-sum-def, rule LeastI)}
\end{itemize}
qed

lemma \textit{arg-less-sf-imp-c-sum-less-arg}: \( u < \text{sf}(x) \implies \text{c-sum } u < x \)

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

lemma \textit{sf-c-sum-le-arg}: \( u \geq \text{sf} \ (\text{c-sum } u) \)

proof –
\begin{itemize}
  \item let \( ?z = \text{c-sum } u \)
  \item from \textit{arg-less-sf-at-Suc-of-c-sum} have \( S1: u < \text{sf } (?z+1) \) by \textit{(auto)}
  \item have \( S2: \neg \text{c-sum } u < \text{c-sum } u \) by \textit{(auto)}
  \item from \textit{arg-less-sf-imp-c-sum-less-arg} \( S2 \) have \( S3: \neg u < \text{c-sum } u \) by \textit{(auto)}
  \item from \( S3 \) show \( ?\text{thesis} \) by \textit{(auto)}
\end{itemize}
qed

lemma \textit{c-sum-le-arg}: \( \text{c-sum } u \leq u \)

proof –
\begin{itemize}
  \item have \( \text{c-sum } u \leq \text{sf } (\text{c-sum } u) \) by \textit{(rule arg-le-sf)}
  \item moreover have \( \text{sf}(\text{c-sum } u) \leq u \) by \textit{(rule sf-c-sum-le-arg)}
  \item ultimately show \( ?\text{thesis} \) by \textit{simp}
\end{itemize}
qed

lemma \textit{c-sum-of-c-pair} \( [simp] \): \( \text{c-sum } (\text{c-pair } x y) = x + y \)

proof –
\begin{itemize}
  \item let \( ?u = \text{c-pair } x y \)
  \item let \( ?z = \text{c-sum } ?u \)
  \item have \( S1: ?u < \text{sf}(?z+1) \) by \textit{(rule arg-less-sf-at-Suc-of-c-sum)}
  \item have \( S2: \text{sf}(?z) \leq ?u \) by \textit{(rule sf-c-sum-le-arg)}
\end{itemize}

qed
from S1 have S3: \( sf(x+y) + x < sf(\tilde{z}+1) \) by (simp add: c-pair-def)
from S2 have S4: \( sf(\tilde{z}) \leq sf(x+y) + x \) by (simp add: c-pair-def)
from S3 have S5: \( sf(x+y) < sf(\tilde{z}+1) \) by (auto)
from S5 have S6: \( x+y < \tilde{z}+1 \) by (rule sf-less-sfD)
from S4 have S7: \( x+y \leq \tilde{z} \) by (auto)
from S7 S8 have S9: \( \tilde{z} = x+y \) by (auto)
from S9 show \(?thesis\) by (simp)
qed

lemma c-fst-of-c-pair[simp]: c-fst (c-pair x y) = x
proof –
  let \(?u = c\)-pair x y
  have c-sum \(?u = x + y\) by simp
  hence c-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 \(?thesis\) by (simp)
qed

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

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

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

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

lemma sf-c-sum-plus-c-fst: sf(c-sum u) + c-fst u = u
proof –
  have S1: sf(c-sum u) \leq u by (rule sf-c-sum-le-arg)
  have S2: c-fst u = u - sf(c-sum u) by (simp add: c-fst-def)
  from S1 S2 show \(?thesis\) 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

10
assume \( A1: \text{c-sum } x = x \)

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

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

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

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

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

qed

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

proof
assume \( A1: \text{c-sum } x = x \)

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

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

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

from \( S2 \) \( S3 \) show ?thesis
by simp

qed

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

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

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

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

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

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

with \( A1 \) show ?thesis
by simp

qed

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

proof
assume \( A1: x > 0 \)

show ?thesis
proof cases
assume \( c-fst x < x \)
then show ?thesis
by simp

next
assume \( \neg \text{c-fst } x < x \)
then have \( S1: \text{c-fst } x \geq x \)
by simp

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

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

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

with \( A1 \) show ?thesis
by simp

qed

qed

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

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

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

have \( S2: \text{c-sum } x \leq x \)
by (rule \text{c-sum-le-arg})
from A1 S1 S2 have c-sum x = x by simp
then show ?thesis by (rule c-sum-eq-arg)
qed

lemma c-snd-less-arg: x > 1 =⇒ c-snd x < x
proof
- assume A1: x > 1
  show ?thesis
  proof cases
  assume c-snd x < x
  then show ?thesis .
  next
  assume ¬ c-snd x < x
  then have S1: c-snd x ≥ x by auto
  have c-snd x ≤ x by (rule c-snd-le-arg)
  with S1 have c-snd x = x by simp
  then have x ≤ 1 by (rule c-snd-eq-arg)
  with A1 show ?thesis by simp
  qed
  qed

end

2 Primitive recursive functions

theory PRecFun imports CPair
begin

This theory contains definition of the primitive recursive functions.

2.1 Basic definitions

primrec
PrimRecOp :: (nat ⇒ nat) ⇒ (nat ⇒ nat ⇒ nat ⇒ nat) ⇒ (nat ⇒ nat ⇒ nat)
where
PrimRecOp g h 0 x = g x
| PrimRecOp g h (Suc y) x = h x (PrimRecOp g h y x) x

primrec
PrimRecOp-last :: (nat ⇒ nat) ⇒ (nat ⇒ nat ⇒ nat ⇒ nat) ⇒ (nat ⇒ nat ⇒ nat)
where
PrimRecOp-last g h x 0 = g x
| PrimRecOp-last g h x (Suc y)= h x (PrimRecOp-last g h y) y

primrec
PrimRecOp1 :: nat ⇒ (nat ⇒ nat ⇒ nat) ⇒ (nat ⇒ nat)
where
PrimRecOp1 a h 0 = a
\[ \text{ PrimRecOp1 } a \; h \; (\text{ Suc } y) = h \; y \; (\text{ PrimRecOp1 } a \; h \; y) \]

\textbf{inductive-set}

\begin{align*}
\text{ PrimRec1 } &:: (\text{ nat } \Rightarrow \text{ nat }) \text{ set and } \\
\text{ PrimRec2 } &:: (\text{ nat } \Rightarrow \text{ nat } \Rightarrow \text{ nat }) \text{ set and } \\
\text{ PrimRec3 } &:: (\text{ nat } \Rightarrow \text{ nat } \Rightarrow \text{ nat } \Rightarrow \text{ nat }) \text{ set } \\
\text{ where } \\
\text{ zero} : (\lambda x. \; 0) &\in \text{ PrimRec1} \\
\text{ suc} : &\text{ Suc } \in \text{ PrimRec1} \\
\text{id1-1} : (\lambda x. \; x) &\in \text{ PrimRec1} \\
\text{id2-1} : (\lambda x y. x) &\in \text{ PrimRec2} \\
\text{id2-2} : (\lambda x y. y) &\in \text{ PrimRec2} \\
\text{id3-1} : (\lambda x y z. x) &\in \text{ PrimRec3} \\
\text{id3-2} : (\lambda x y z. y) &\in \text{ PrimRec3} \\
\text{id3-3} : (\lambda x y z. z) &\in \text{ PrimRec3} \\
\text{ comp1-1} : [ f \in \text{ PrimRec1}; g \in \text{ PrimRec1} ] \Rightarrow (\lambda x. \; f \; (g \; x)) &\in \text{ PrimRec1} \\
\text{ comp1-2} : [ f \in \text{ PrimRec1}; g \in \text{ PrimRec2} ] \Rightarrow (\lambda x y. \; f \; (g \; x \; y)) &\in \text{ PrimRec2} \\
\text{ comp1-3} : [ f \in \text{ PrimRec1}; g \in \text{ PrimRec3} ] \Rightarrow (\lambda x y z. \; f \; (g \; x \; y \; z)) &\in \text{ PrimRec3} \\
\text{ comp2-1} : [ f \in \text{ PrimRec2}; g \in \text{ PrimRec1}; h \in \text{ PrimRec1} ] \Rightarrow (\lambda x. \; f \; (g \; x) \; (h \; x) \; y) &\in \text{ PrimRec2} \\
\text{ comp2-2} : [ f \in \text{ PrimRec2}; g \in \text{ PrimRec2}; h \in \text{ PrimRec2} ] \Rightarrow (\lambda x y. \; f \; (g \; x \; y) \; (h \; x \; y) \; (k \; x \; y)) &\in \text{ PrimRec3} \\
\text{ comp2-3} : [ f \in \text{ PrimRec2}; g \in \text{ PrimRec3}; h \in \text{ PrimRec3}; k \in \text{ PrimRec3} ] \Rightarrow (\lambda x y z. \; f \; (g \; x \; y \; z) \; (h \; x \; y \; z) \; (k \; x \; y \; z)) &\in \text{ PrimRec3} \\
\text{ prim-rec} : [ g \in \text{ PrimRec1}; h \in \text{ PrimRec3} ] \Rightarrow \text{ PrimRecOp } g \; h \; \in \text{ PrimRec2} \\
\end{align*}

\textbf{lemmas}

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

ML-file \( \langle \text{Utils.ML} \rangle \)

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

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

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

theorem \( \text{pr-rec-scheme} \): \[ [ 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 \( \forall x y. f y x = \text{PrimRecOp} g h y x \) by \( \text{induct-tac y, simp-all} \)
  then have \( f = \text{PrimRecOp} g h \) by \( \text{simp add: ext} \)
  with \( g\text{-is-pr h\text{-is-pr}} \) show \( \exists f \) by \( \text{simp add: pr-rec} \)

qed

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

proof \( \text{rule pr-swap} \)

show \( (\lambda x y. x + y) \in \text{PrimRec2} \)

proof –
  have \( S1\): \( \text{PrimRecOp} (\lambda x. x) (\lambda x y z. \text{Suc} y) \in \text{PrimRec2} \)
  proof \( \text{rule pr-rec} \)
    show \( (\lambda x. x) \in \text{PrimRec1} \) by \( \text{rule pr-id1-1} \)
  next
    show \( (\lambda x y z. \text{Suc} y) \in \text{PrimRec3} \) by \( \text{prec0} \)
  qed
  have \( (\lambda x y. x + y) = \text{PrimRecOp} (\lambda x. x) (\lambda x y z. \text{Suc} y) \) \( \text{(is - = ?f)} \)
  proof –
    have \( \forall x y. (\text{Suc} y x y x + z) \) by \( \text{induct-tac y, auto} \)
    thus \( \exists f \) by \( \text{simp add: ext} \)
  qed
  with \( S1 \) show \( \exists f \) by \( \text{simp} \)

qed
lemma op-mult-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: \(\text{PrimRecOp} (\lambda x. 0) (\lambda x y z. y + z) \in \text{PrimRec2}\)
      proof (rule pr-rec)
    next
      show \((\lambda x y z. y + z) \in \text{PrimRec3}\) by prec0
    qed
    have \((\lambda x y. y + x) = \text{PrimRecOp} (\lambda x. 0) (\lambda x y z. y + z) \in \text{PrimRec2}\)
      proof
        have \(\exists y x. (y = y * x)\) by (induct-tac y, auto)
        thus ?thesis by (simp add: ext)
      qed
    with S1 show ?thesis by simp
  qed
  qed

lemma const-is-pr: \((\lambda x. (\text{n::nat})) \in \text{PrimRec1}\)
proof (induct n)
  show \((\lambda x. 0) \in \text{PrimRec1}\) 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 const-is-pr-2: \((\lambda x y. (\text{n::nat})) \in \text{PrimRec2}\)
proof (rule pr-comp1-2 [where \(?f=%x.(\text{n::nat})\) and \(?g=%x y. x\)])
  show \((\lambda x. n) \in \text{PrimRec1}\) by (rule const-is-pr)
next
  show \((\lambda x y. x) \in \text{PrimRec2}\) by (rule pr-id2-1)
qed

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

theorem 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 \(?h1 = \lambda x y z. h z y x\)
  from A2 pr-id3-3 pr-id3-2 pr-id3-1 have \(?h1\)-is-pr: \(?h1 \in \text{PrimRec3}\) by (rule pr-comp3-3)
let \( ?f1 = \text{PrimRecOp} g ?h1 \)
from \( A1 \) \( h1 \)-is-pr have \( f1 \)-is-pr: \( ?f1 \in \text{PrimRec2} \)
by (rule \text{pr-rec})
let \( ?f = \lambda x y. ?f1 y x \)
from \( f1 \)-is-pr have \( f \)-is-pr: \( ?f \in \text{PrimRec2} \)
by (rule \text{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: \text{ext})
with \( f \)-is-pr show \( \text{?thesis} \)
by simp

qed

theorem \( \text{pr-rec1} \): \( h \in \text{PrimRec2} \Rightarrow \text{PrimRecOp}_{\text{1}} (a::\text{nat}) \)
\( h \in \text{PrimRec1} \)
proof –
assume \( A1: \ h \in \text{PrimRec2} \)
let \( ?g = (\lambda x. a) \)
have \( g \)-is-pr: \( ?g \in \text{PrimRec1} \)
by (rule \text{const-is-pr})
let \( ?h1 = (\lambda x y z. h x y) \)
from \( A1 \) have \( h1 \)-is-pr: \( ?h1 \in \text{PrimRec3} \)
by \text{pre0}
let \( ?f1 = \text{PrimRecOp} ?g ?h1 \)
from \( g \)-is-pr \( h1 \)-is-pr have \( f1 \)-is-pr: \( ?f1 \in \text{PrimRec2} \)
by (rule \text{pr-rec})
let \( ?f = (\lambda x. ?f1 x 0) \)
from \( f1 \)-is-pr \( \text{pr-id1-1} \) \( \text{pr-zero} \) have \( f \)-is-pr: \( ?f \in \text{PrimRec1} \)
by (rule \text{pr-comp2-1})

have \( \forall y. \ ?f y = \text{PrimRecOp}_{\text{1}} a h y \)
by (induct-tac \( y \), \text{auto})
then have \( ?f = \text{PrimRecOp}_{\text{1}} a h \)
by (simp add: \text{ext})
with \( f \)-is-pr show \( \text{?thesis} \)
by (auto)
qed

theorem \( \text{pr-rec1-scheme} \): \[ \begin{align*}
 & h \in \text{PrimRec2} \quad f 0 = a \; ; \\
 & \forall y. f (\text{Suc} y) = h y (f y)
\end{align*} \] 
\( \Rightarrow f \in \text{PrimRec1} \)
proof –
assume \( h \)-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 \( f\text{-at-0} \) \( f\text{-at-Suc} \) have \( \forall y. f y = \text{PrimRecOp}_{\text{1}} a h y \)
by (induct-tac \( y \), simp-all)
then have \( f = \text{PrimRecOp}_{\text{1}} a h \)
by (simp add: \text{ext})
with \( h \)-is-pr show \( \text{?thesis} \)
by (simp add: \text{pr-rec1})
qed

lemma \( \text{pred-is-pr} \): \( (\lambda x. x - (1::\text{nat})) \in \text{PrimRec1} \)
proof –
have \( S1: \text{PrimRecOp}_{\text{1}} 0 (\lambda x y. x) \in \text{PrimRec1} \)
by (rule \text{pr-rec1})
show \( (\lambda x y. x) \in \text{PrimRec2} \)
by (rule \text{pr-id2-1})
qed

have \( \lambda x. x - (1::\text{nat}) = \text{PrimRecOp}_{\text{1}} 0 (\lambda x y. x) \)
(is - = \( ?f \))
proof –
have \( \forall x. (\text{if } x = x - (1::\text{nat})) \)
by (induct-tac \( x \), \text{auto})
thus \( \text{?thesis} \)
by (simp add: \text{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 \((\lambda x. x) \in \text{PrimRec1}\) by (rule pr-id1-1)
next
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 \((\lambda x y. y-x) = \text{PrimRecOp} (\lambda x. x) (\lambda x y z. y-(1::nat))\) (is - = ?f)
proof
\(\lambda x y. (\text{if } y x = x - y \text{ by (induct-tac y, auto)}\)
thus ?thesis by (simp add: ext)
qed
with \(S1\) show ?thesis by simp
qed

lemmas [prec] =
  const-is-pr [of 0] const-is-pr-2 [of 0] const-is-pr-3 [of 0]
  const-is-pr [of 1] const-is-pr-2 [of 1] const-is-pr-3 [of 1]
  const-is-pr [of 2] const-is-pr-2 [of 2] const-is-pr-3 [of 2]

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

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

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

lemma [simp]: \(sgn1 0 = 0\) by (simp add: sgn1-def)
lemma [simp]: \(sgn1 (\text{Suc } y) = 1\) by (simp add: sgn1-def)
lemma [simp]: \(sgn2 0 = 1\) by (simp add: sgn2-def)
lemma [simp]: \(sgn2 (\text{Suc } y) = 0\) 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 ⇒ x > 0 by (cases x) auto

lemma sgn1-nz-eq-arg-pos: (sgn1 x ≠ 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 ≠ 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 =⇒ sgn1 x = 1 by (cases x) auto
lemma sgn2-pos-eq-one: sgn2 x > 0 =⇒ sgn2 x = 1 by (cases x) auto

lemma sgn2-eq-1-sub-arg: sgn2 = (λ x. 1 - x)
proof (rule ext)
  fix x show sgn2 x = 1 - x by (cases x) auto
qed

lemma sgn1-eq-1-sub-sgn2: sgn1 = (λ x. 1 - (sgn2 x))
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

lemma sgn2-is-pr [prec]: sgn2 ∈ PrimRec1
proof
  have (λ x. 1 - x) ∈ PrimRec1 by prec0
  thus thesis by (simp add: sgn2-eq-1-sub-arg)
qed

lemma sgn1-is-pr [prec]: sgn1 ∈ PrimRec1
proof
  from sgn2-is-pr have (λ x. 1 - (sgn2 x)) ∈ PrimRec1 by prec0
  thus thesis by (simp add: sgn1-eq-1-sub-sgn2)
qed

lemma abs-of-diff-is-pr [prec]: abs-of-diff ∈ 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 ∈ PrimRec1
proof
  have S1: PrimRecOp1 0 (λ x y. y + x + 1) ∈ PrimRec1
  proof (rule pr-rec1)
    show (λ x y. y + x + 1) ∈ PrimRec2 by prec0
  qed
  have (λ x. sf x) = PrimRecOp1 0 (λ x y. y + x + 1) (is - = ?f)
proof –
  have \( \forall x. (\exists f \; sf \; x) \)
  proof (induct-tac x)
    show \( \exists f \; 0 = sf \; 0 \) by (simp add: sf-at-0)
  next
    fix \( x \) assume \( \exists f \; x = sf \; x \)
    with sf-at-Suc show \( \exists f \; (Suc \; x) = sf \; (Suc \; x) \) by auto
  qed
  thus \?thesis by (simp add: ext)
  qed

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

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

lemma if-eq-is-pr [prec]: \[ \begin{array}{l}
  \; p1 \in PrimRec1; \; p2 \in PrimRec1; \; q1 \in PrimRec1; \; q2 \in PrimRec1
\end{array} \] \implies \( (\lambda x. \; \text{if} \; (p1 \; x = p2 \; x) \; \text{then} \; q1 \; x \; \text{else} \; q2 \; x) \) \in PrimRec1
  proof –
    have S1: \( (\lambda x. \; \text{if} \; (p1 \; x = p2 \; x) \; \text{then} \; q1 \; x \; \text{else} \; q2 \; x) \) = \( (\lambda x. \; (\text{abs-of-diff} \; (p1-p2) \; (q1-q2))) \)

lemma if-is-pr3 [prec]:  [ p ∈ PrimRec3; q1 ∈ PrimRec3; q2 ∈ PrimRec3] ⇒ (λ x. (abs-of-diff (p1 x) (p2 x))) ∈ PrimRec3
proof
  assume A1: p1 ∈ PrimRec1 and A2: p2 ∈ PrimRec1
  with abs-of-diff-is-pr have S2: (λ x. abs-of-diff (p1 x) (p2 x)) ∈ PrimRec1 by prec0
  assume q1 ∈ PrimRec1 and q2 ∈ PrimRec1
  with S2 have ?R ∈ PrimRec1 by (rule if-is-pr)
  with S1 show ?thesis by simp
qed

lemma if-is-pr2 [prec]:  [ p ∈ PrimRec2; q1 ∈ PrimRec2; q2 ∈ PrimRec2] ⇒ (λ x. (abs-of-diff (p1 x) (p2 x))) ∈ PrimRec2
proof
  have if-as-pr: (λ x. y. if (p x y = 0) then (q1 x y) else (q2 x y)) = (λ 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 ?left = ?right)
    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 ≠ 0
      then have S3: p x y > 0 by simp
      then show ?thesis by simp
    qed
  qed
  assume p ∈ PrimRec2 and q1 ∈ PrimRec2 and q2 ∈ PrimRec2
  then have (λ x y. (sgn2 (p x y)) * (q1 x y) + (sgn1 (p x y)) * (q2 x y)) ∈ PrimRec2 by prec0
  with if-as-pr show ?thesis by simp
qed

lemma if-eq-is-pr2:  [ p1 ∈ PrimRec2; p2 ∈ PrimRec2; q1 ∈ PrimRec2; q2 ∈ PrimRec2] ⇒ (λ x y. if (p1 x y = p2 x y) then (q1 x y) else (q2 x y)) ∈ PrimRec2
proof
  have S1: (λ x y. if (p1 x y = p2 x y) then (q1 x y) else (q2 x y)) = (λ 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 ∈ PrimRec2 and A2: p2 ∈ PrimRec2
  with abs-of-diff-is-pr have S2: (λ x y. abs-of-diff (p1 x y) (p2 x y)) ∈ PrimRec2 by prec0
  assume q1 ∈ PrimRec2 and q2 ∈ PrimRec2
  with S2 have ?R ∈ PrimRec2 by (rule if-is-pr)
  with S1 show ?thesis by simp
qed

lemma if-is-pr3 [prec]:  [ p ∈ PrimRec3; q1 ∈ PrimRec3; q2 ∈ PrimRec3] ⇒ (λ x y. (abs-of-diff (p1 x) (p2 x))) ∈ PrimRec3
proof
  have if-as-pr: (λ x. y. if (p x y = 0) then (q1 x y) else (q2 x y)) = (λ 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 ?left = ?right)
    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 ≠ 0
      then have S3: p x y > 0 by simp
      then show ?thesis by simp
    qed
  qed
  assume p ∈ PrimRec3 and q1 ∈ PrimRec3 and q2 ∈ PrimRec3
  then have (λ x y. (sgn2 (p x y)) * (q1 x y) + (sgn1 (p x y)) * (q2 x y)) ∈ PrimRec3 by prec0
  with if-as-pr show ?thesis by simp
qed
if \((p \ x \ y \ z = 0)\) then \((q_1 \ x \ y \ z)\) else \((q_2 \ x \ y \ z)\) ∈ PrimRec3

**proof**

- have if-as-pr: \((\lambda \ x \ y \ z. \ if \ (p \ x \ y \ z = 0) \ then \ (q_1 \ x \ y \ z) \ else \ (q_2 \ x \ y \ z))\) = \((\lambda \ x \ y \ z. \ (sgn_2 \ (p \ x \ y \ z)) * (q_1 \ x \ y \ z) + (sgn_1 \ (p \ x \ y \ z)) * (q_2 \ x \ y \ z))\)

**proof** (rule ext, rule ext, rule ext)

fix \(x\) fix \(y\) fix \(z\) show \((\lambda \ x \ y \ z. \ if \ (p \ x \ y \ z = 0) \ then \ (q_1 \ x \ y \ z) \ else \ (q_2 \ x \ y \ z)) = (\lambda \ x \ y \ z. \ sgn_2 \ (p \ x \ y \ z)) * (q_1 \ x \ y \ z) + (sgn_1 \ (p \ x \ y \ z)) * (q_2 \ x \ y \ z))\)

**proof** cases

- assume \(A_1: p \ x \ y \ z = 0\)
  - then have \(S_1: lleft = q_1 \ x \ y \ z\) by simp
  - from \(A_1\) have \(S_2: \lright = q_1 \ x \ y \ z\) by simp
  - from \(S_1\) \(S_2\) show \(?thesis\) by simp
- next
  - assume \(A_2: p \ x \ y \ z \neq 0\)
  - then have \(S_3: p \ x \ y \ z > 0\) by simp
  - then show \(?thesis\) by simp
qed

**lemma** if-eq-is-pr3: \([p_1 \in \text{PrimRec3}; \ p_2 \in \text{PrimRec3}; \ q_1 \in \text{PrimRec3}; \ q_2 \in \text{PrimRec3}] \implies (\lambda \ x \ y \ z. \ if \ (p_1 \ x \ y \ z = p_2 \ x \ y \ z) \ then \ (q_1 \ x \ y \ z) \ else \ (q_2 \ x \ y \ z)) \in \text{PrimRec3}\)

**proof**

- have \(S_1: (\lambda \ x \ y \ z. \ if \ (p_1 \ x \ y \ z = p_2 \ x \ y \ z) \ then \ (q_1 \ x \ y \ z) \ else \ (q_2 \ x \ y \ z)) = (\lambda \ x \ y \ z. \ if \ (abs-of-diff \ (p_1 \ x \ y \ z) \ (p_2 \ x \ y \ z) = 0) \ then \ (q_1 \ x \ y \ z) \ else \ (q_2 \ x \ y \ z))\) (is \(?L = ?R\)) by (simp add: abs-of-diff-eq)
  - assume \(A_1: p_1 \in \text{PrimRec3}\) and \(A_2: p_2 \in \text{PrimRec3}\)
  - with \(abs-of-diff-is-pr\) have \(S_2: (\lambda \ x \ y \ z. \ abs-of-diff \ (p_1 \ x \ y \ z) \ (p_2 \ x \ y \ z)) \in \text{PrimRec3}\)
  - by prec0
  - assume \(q_1 \in \text{PrimRec3}\) and \(q_2 \in \text{PrimRec3}\)
  - with \(S_2\) have \(?R \in \text{PrimRec3}\) by (rule if-is-pr3)
  - with \(S_1\) show \(?thesis\) by simp
qed

**ML**

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

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

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

method-setup prec = ('
Attrib.thms >> (fn ths => fn ctxt => Method.METHOD (fn facts =>
    HEADGOAL (prec-tac ctxt (facts @ Named-Theorems.get ctxt @\{named-theorems prec\})))
) > apply primitive recursive functions

2.2 Bounded least operator

definition
  \(b\text{-}least :: (\text{nat} \Rightarrow \text{nat} \Rightarrow \text{(nat} \Rightarrow \text{nat}) \Rightarrow (\text{nat} \Rightarrow \text{nat}))\) where
  \(b\text{-}least \ f \ x \equiv (\text{Least } (\% y. \ y = x \lor ( y < x \land (f \ x \ y) \neq 0))\))

definition
  \(b\text{-}least2 :: (\text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat}) \Rightarrow (\text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat})\) where
  \(b\text{-}least2 \ f \ y \equiv (\text{Least } (\% z. \ z = y \lor ( z < y \land (f \ x \ z) \neq 0))\))

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

lemma b-least-le-arg: \(b\text{-least } f\ x \leq x\)
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 less-b-least-impl-zero: \(y < b\text{-least } f\ x \Rightarrow f\ x\ y = 0\)
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 nz-impl-b-least-le: \((f\ x\ y) \neq 0 \Rightarrow (b\text{-least } f\ x) \leq y\)
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 False by (simp add: less-b-least-impl-zero)
qed

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

lemma nz-impl-b-least-less: \[ y < x; (f x y) \neq 0 \] \implies (b\text{-}\text{least} f x) < x

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

qed

lemma b-least-aux2: \[ y < x; (f x y) \neq 0 \] \implies (b\text{-}\text{least} f x) = (Least (\%y. (f x y) \neq 0))

proof –
assume A1: y < x and A2: f x y \neq 0
from A1 A2 have S1: b\text{-}\text{least} f x < x by (rule nz-impl-b-least-less)
thus ?thesis by (rule b-least-less-impl-eq)

qed

lemma b-least2-aux1: b\text{-}\text{least2} f x y = y \lor (b\text{-}\text{least2} f x y < y \land (f x (b\text{-}\text{least2} f x y)) \neq 0)

proof –
let ?P = \%z. z = y \lor (z < y \land (f x z) \neq 0)
have ?P y by simp
then have ?P (Least ?P) by (rule LeastI)
thus ?thesis by (simp add: b-least2-def)

qed

lemma b-least2-le-arg: b\text{-}\text{least2} f x y \leq y

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

qed

lemma less-b-least2-impl-zero: z < b\text{-}\text{least2} f x y \implies f x z = 0

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

qed

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

proof –
assume $A_1$: $f \ x \ z \neq 0$

have $S_1$: $z < b\text{-}\text{least2} \ f \ x \ y \implies f \ x \ z = 0$ by (rule less-b\text{-}\text{least2-impl-zero})

from $A_1$ $S_1$ show $\text{thesis}$ by arith

qed

lemma $b\text{-}\text{least2-less-impl-nz}$: $b\text{-}\text{least2} \ f \ x \ y < y \implies f \ x \ (b\text{-}\text{least2} \ f \ x \ y) \neq 0$

proof –

assume $A_1$: $b\text{-}\text{least2} \ f \ x \ y < y$

have $b\text{-}\text{least2} \ f \ x \ y = y \lor (b\text{-}\text{least2} \ f \ x \ y < y \land (f \ x \ (b\text{-}\text{least2} \ f \ x \ y)) \neq 0)$ by (rule b\text{-}\text{least2-aux1})

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

qed

lemma $b\text{-}\text{least2-less-impl-eq}$: $b\text{-}\text{least2} \ f \ x \ y < y \implies (\text{Least} \ (%z. (f \ x \ z)) \neq 0))$

proof –

assume $A_1$: $b\text{-}\text{least2} \ f \ x \ y < y$ (is $\ ?b < -$)

let $\ ?B = (\text{Least} \ (%z. (f \ x \ z) \neq 0))$

from $A_1$ have $S_1$: $f \ x \ ?b \neq 0$ by (rule b\text{-}\text{least2-less-impl-nz})

from $S_1$ have $S_2$: $\ ?B \leq \ ?b$ by (rule Least-le)

from $S_1$ have $S_3$: $f \ x \ ?B \neq 0$ by (rule LeastI)

from $S_3$ have $S_4$: $?b \leq ?B$ by (rule nz-impl-b\text{-}\text{least2-le})

from $S_2$ $S_4$ show $\text{thesis}$ by simp

qed

lemma $\text{less-b}\text{-}\text{least2-impl-zero2}$: $[z < y; \ b\text{-}\text{least2} \ f \ x \ y = y] \implies f \ x \ z = 0$

proof –

assume $z < y$ and $b\text{-}\text{least2} \ f \ x \ y = y$

hence $z < b\text{-}\text{least2} \ f \ x \ y$ by simp

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

qed

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

proof (rule ccontr)

assume $A_1$: $z < y$

assume $A_2$: $f \ x \ z \neq 0$

assume $\neg (b\text{-}\text{least2} \ f \ x \ y) < y$ then have $A_3$: $y \leq (b\text{-}\text{least2} \ f \ x \ y)$ by simp

have $b\text{-}\text{least2} \ f \ x \ y \leq y$ by (rule b\text{-}\text{least2-le-arg})

with $A_3$ have $b\text{-}\text{least2} \ f \ x \ y = y$ by simp

with $A_1$ have $f \ x \ z = 0$ by (rule less-b\text{-}\text{least2-impl-zero2})

with $A_2$ show False by simp

qed

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

proof –

assume $A_1$: $z < y$ and $A_2$: $f \ x \ z \neq 0$

from $A_1$ $A_2$ have $S_1$: $b\text{-}\text{least2} \ f \ x \ y < y$ by (rule nz-b\text{-}\text{least2-impl-less})

thus $\text{thesis}$ by (rule b\text{-}\text{least2-less-impl-eq})
lemma b-least2-aux2: b-least2 f x y < y ⇒ b-least2 f x (Suc y) = b-least2 f x y
proof -
  let ?B = b-least2 f x y
  assume A1: ?B < y
  from A1 have S1: f x ?B ≠ 0 by (rule b-least2-less-impl-nz)
  from S1 have S2: b-least2 f x (Suc y) ≤ ?B by (simp add: nz-impl-b-least2-le)
  from A1 S2 have S3: b-least2 f x (Suc y) < Suc y by (simp)
  from S3 have S4: f x (b-least2 f x (Suc y)) ≠ 0 by (rule b-least2-less-impl-nz)
  from S4 have S5: ?B ≤ b-least2 f x (Suc y) by (rule nz-impl-b-least2-le)
  from S2 S5 show ?thesis by simp
qed

lemma b-least2-aux3: [ b-least2 f x y = y; f x y ≠ 0 ] ⇒ b-least2 f x (Suc y) = y
proof -
  assume A1: b-least2 f x y = y
  assume A2: f x y ≠ 0
  from A2 have S1: b-least2 f x (Suc y) ≤ y by (rule nz-impl-b-least2-le)
  have S2: b-least2 f x (Suc y) < y ⇒ False
  proof -
    assume A2-1: b-least2 f x (Suc y) < y
    have S2-1: z < Suc y by simp
    from S2-1 have S2-2: f x z ≠ 0 by (rule b-least2-less-impl-nz)
    from A2-1 S2-2 have S2-3: b-least2 f x y < y by (rule nz-b-least2-impl-less)
    from S2-3 A1 show ?thesis by simp
  qed
  from S2 have S3: ¬ (b-least2 f x (Suc y) < y) by auto
  from S1 S3 show ?thesis by simp
qed

lemma b-least2-mono: y1 ≤ y2 ⇒ b-least2 f x y1 ≤ b-least2 f x y2
proof (rule contr)
  assume A1: y1 ≤ y2
  let ?b1 = b-least2 f x y1 and ?b2 = b-least2 f x y2
  assume ¬ (?b1 ≤ ?b2 then have A2: ?b2 < ?b1 by simp
  have S1: ?b1 ≤ y1 by (rule b-least2-le-arg)
  have S2: ?b2 ≤ y2 by (rule b-least2-le-arg)
  from A1 A2 S1 S2 have S3: ?b2 < y2 by simp
  then have S4: f x ?b2 ≠ 0 by (rule b-least2-less-impl-nz)
  from A2 have S5: f x ?b2 = 0 by (rule less-b-least2-impl-zero)
  from S4 S5 show False by simp
qed

lemma b-least2-aux4: [ b-least2 f x y = y; f x y = 0 ] ⇒ b-least2 f x (Suc y) = Suc y
proof -
  assume A1: b-least2 f x y = y
  assume A2: f x y = 0
have $S1$: $\text{b-least2 } f x \ (\text{Suc } y) \leq \text{Suc } y$ by (rule \text{b-least2-le-arg})

have $S2$: $y \leq \text{b-least2 } f x \ (\text{Suc } y)$

proof

- have $y \leq \text{Suc } y$ by simp
- then have $\text{b-least2 } f x \ y \leq \text{b-least2 } f x \ (\text{Suc } y)$ by (rule \text{b-least2-mono}) with $A1$ show ?thesis by simp

qed

from $S1 \ S2$ have $\text{b-least2 } f x \ (\text{Suc } y) = y \lor \text{b-least2 } f x \ (\text{Suc } y) = \text{Suc } y$ by arith

moreover

{ assume $A3$: $\text{b-least2 } f x \ (\text{Suc } y) = y$
- have $f x y \neq 0$
  proof
    - have $y < \text{Suc } y$ by simp
    with $A3$ have $\text{b-least2 } f x \ (\text{Suc } y) < \text{Suc } y$ by simp
    from this have $f x \ (\text{b-least2 } f x \ (\text{Suc } y)) \neq 0$ by (simp add: \text{b-least2-less-impl-nz})
    with $A3$ show $f x y \neq 0$ by simp
  qed
  with $A2$ have ?thesis by simp
}

moreover

{ assume $\text{b-least2 } f x \ (\text{Suc } y) = \text{Suc } y$
- then have ?thesis by simp
}

ultimately show ?thesis by blast

qed

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

proof

- have $S1$: $\text{b-least2 } f x \ 0 \leq 0$ by (rule \text{b-least2-le-arg})
  from $S1$ show ?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} = $(\lambda f::\text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} ) \ x \ y \ z \ ((\text{sgn1 } (z - y)) \ast y + (\text{sgn2 } (z - y)) \ast ((\text{sgn1 } f x z) \ast z + (\text{sgn2 } f x z) \ast (\text{Suc } z)))$
- 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}: $\forall f x y. \text{loc-Op2 } f x y < y \implies \text{loc-Op2 } f x \ (\text{Suc } y) = \text{loc-Op2 } f x y$

proof

- fix $f \ x \ y$
- let $?b = \text{loc-Op2 } f x y$
- have $S1$: $\text{loc-Op2 } f x \ (\text{Suc } y) = ((\text{loc-Op1 } f) \ x) \ ?b \ y$ 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: \text{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 \( \text{loc-op2-lm-2} \): \( \forall f \ x \ y. [\neg (\text{loc-Op2} \ f \ x \ y < y); \ f \ x \ y \neq 0] \implies \text{loc-Op2} \ f \ x \ (\text{Suc} \ y) = y \)
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: \text{loc-Op2-def})
  assume \( \neg (?b < y) \)
  then have S2: \( y - ?b = 0 \) by simp
  assume \( f \ x \ y \neq 0 \)
  with S2 have \( ?h \ x \ ?b \ y = y \) by (simp add: \text{loc-Op1-def})
  with S1 show \( \text{loc-Op2} \ f \ x \ (\text{Suc} \ y) = y \) by simp
qed

have \( \text{loc-op2-lm-3} \): \( \forall f \ x \ y. [\neg (\text{loc-Op2} \ f \ x \ y < y); \ f \ x \ y = 0] \implies \text{loc-Op2} \ f \ x \ (\text{Suc} \ y) = \text{Suc} \ y \)
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: \text{loc-Op2-def})
  assume \( \neg (?b < y) \)
  then have S2: \( y - ?b = 0 \) by simp
  assume \( f \ x \ y = 0 \)
  with S2 have \( ?h \ x \ ?b \ y = \text{Suc} \ y \) by (simp add: \text{loc-Op1-def})
  with S1 show \( \text{loc-Op2} \ f \ x \ (\text{Suc} \ y) = \text{Suc} \ y \) by simp
qed

have Op2-eq-b-least2-at-point: \( \forall f \ x. \text{loc-Op2} \ f \ x = \text{b-least2} \ f \ x \)
proof -
  fix \( f \ x \ y \)
  assume \( \text{loc-Op2} \ f \ x \ y = \text{b-least2} \ f \ x \ y \)
  show \( \text{loc-Op2} \ f \ x = \text{b-least2} \ f \ x \) by (simp add: \text{loc-Op2-def} \text{b-least2-at-zero})
next
  fix \( y \)
  assume A1: \( \text{loc-Op2} \ f \ x \ y = \text{b-least2} \ f \ x \ y \)
  then show \( \text{loc-Op2} \ f \ x = \text{b-least2} \ f \ x \) by (rule \text{loc-op2-lm-1})
next
  assume A2: \( \text{loc-Op2} \ f \ x \ y < y \)
  then have S1: \( \text{loc-Op2} \ f \ x \ (\text{Suc} \ y) = \text{loc-Op2} \ f \ x \ y \) by (rule \text{loc-op2-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 \text{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 \text{b-least2-le-arg})
    from A1 A3 this show \( \text{thesis} \) by simp
  qed
  then show \( \text{thesis} \)
proof cases
  assume A4: \( f \cdot x \cdot y \neq 0 \)
  with A3 have S3: \( \text{loc-\text{Op2}} f x (\text{Suc} y) = y \) by (rule \text{loc-op2-lm-2})
  from A3' A4 have S4: \( \text{b-least2} f x (\text{Suc} y) = y \) by (rule \text{b-least2-aux3})
  from S3 S4 show ?thesis by simp
next
  assume \( \neg f \cdot x \cdot y \neq 0 \)
  then have A5: \( f \cdot x \cdot y = 0 \) by simp
  with A3 have S5: \( \text{loc-\text{Op2}} f x (\text{Suc} y) = \text{Suc} y \) by (rule \text{loc-op2-lm-3})
  from A3' A5 have S6: \( \text{b-least2} f x (\text{Suc} y) = \text{Suc} y \) by (rule \text{b-least2-aux4})
  from S5 S6 show ?thesis by simp
qed

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

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

theorem \( \text{pr-b-least} = f \in \text{PrimRec2} \Rightarrow \text{b-least} f \in \text{PrimRec1} \)

2.3 Examples

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

proof (rule \text{ext})
  fix u show c\text{-sum} u = b\text{-least2} (\lambda \ u \ z. (\text{sgn1} (\text{sf} (z+1) - u))) u (\text{Suc} u)
  proof
    have lm-1: \( (\lambda \ x \ y. (\text{sgn1} (\text{sf} (y+1) - x) \neq 0)) = (\lambda \ x \ y. (x < \text{sf}(y+1))) \)
    proof (rule \text{ext}, \text{rule \text{ext}})
      qed
\begin{proof}
\begin{thm}
\begin{case}
\[ c \text{-snd-is-pr} \quad (\text{sgn} (sf(y+1) - x) \neq 0) = (x < sf(y+1)) \]
\end{case}
\end{thm}
\end{proof}

\begin{proof}
\begin{thm}
\begin{case}
\[ c \text{-fst-is-pr} \quad (\text{sgn} (sf(y+1) - x) \neq 0) = (sf(y+1) - x > 0) \quad \text{by} \quad (\text{rule} \quad \text{sgn1-nz-arg-pos}) \]
\end{case}
\end{thm}
\end{proof}

\begin{proof}
\begin{thm}
\begin{case}
\[ c \text{-sum-is-pr} \quad (\text{sgn} (sf(y+1) - x) \neq 0) = (x < sf(y+1)) \quad \text{by} \quad \text{auto} \]
\end{case}
\end{thm}
\end{proof}

\begin{proof}
\begin{thm}
\begin{case}
\[ \text{fix} \ x \ y \ \text{show} \quad (\text{sgn} (sf(y+1) - x) \neq 0) = (x < sf(y+1)) \]
\end{case}
\end{thm}
\end{proof}

\begin{proof}
\begin{thm}
\begin{case}
\[ \text{let} \ \lambda \ u \ z. (\text{sgn} (sf(z+1) - u)) \]
\end{case}
\end{thm}
\end{proof}

\begin{proof}
\begin{thm}
\begin{case}
\[ \text{from} \ \text{g-def} \ \text{have} \quad S1: (\lambda \ u \ z. \text{sgn} (sf(z+1) - u)) \]
\end{case}
\end{thm}
\end{proof}

\begin{proof}
\begin{thm}
\begin{case}
\[ \text{from} \ \text{g-def} \ \text{have} \quad S2: (\lambda \ u. \ g \ u \ (\text{Suc} \ u)) \in \text{PrimRec1} \quad \text{by} \quad \text{prec} \]
\end{case}
\end{thm}
\end{proof}

\begin{proof}
\begin{thm}
\begin{case}
\[ \text{from} \ \text{c-sum-is-pr} \ \text{have} \quad (\lambda \ u. \ (u - sf (c\text{-sum} \ u))) \in \text{PrimRec1} \quad \text{by} \quad \text{prec} \]
\end{case}
\end{thm}
\end{proof}

\begin{proof}
\begin{thm}
\begin{case}
\[ \text{from} \ \text{c-fst-is-pr} \ [\text{prec}] \quad \text{c-fst} \in \text{PrimRec1} \]
\end{case}
\end{thm}
\end{proof}

\begin{proof}
\begin{thm}
\begin{case}
\[ \text{from} \ \text{c-sum-is-pr} \ \text{have} \quad (\lambda \ u. \ (u - sf (c\text{-sum} \ u))) \in \text{PrimRec1} \quad \text{by} \quad \text{prec} \]
\end{case}
\end{thm}
\end{proof}

\begin{proof}
\begin{thm}
\begin{case}
\[ \text{from} \ \text{c-snd-is-pr} \ [\text{prec}] \quad \text{c-snd} \in \text{PrimRec1} \]
\end{case}
\end{thm}
\end{proof}
proof
  have \( S1: \text{c-snd} = (\lambda \ u. (\text{c-sum} \ u) - (\text{c-fst} \ u)) \) by \((\text{simp add: c-snd-def ext})\)
  from c-sum-is-pr c-fst-is-pr have \( S2: (\lambda \ u. (\text{c-sum} \ u) - (\text{c-fst} \ u)) \in \text{PrimRec1} \)
  by \(\text{prec}\)
  from \(S1\) this show ?thesis by \(\text{simp}\)
qed

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

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

definition pr-conv-1-to-2 = \( (\lambda \ f \ x \ y. f \ (\text{c-pair} \ x \ y)) \)
definition pr-conv-1-to-3 = \( (\lambda \ f \ x \ y. f \ (\text{c-pair} \ x \ (\text{c-snd} \ y))) \)
definition pr-conv-2-to-1 = \( (\lambda \ f \ x. f \ (\text{c-fst} \ x) \ (\text{c-snd} \ x)) \)
definition pr-conv-3-to-1 = \( (\lambda \ f. \text{pr-conv-1-to-2} \ (\text{pr-conv-3-to-1} \ f)) \)
definition pr-conv-3-to-2 = \( (\lambda \ f. \text{pr-conv-1-to-3} \ (\text{pr-conv-2-to-1} \ f)) \)
definition pr-conv-3-to-3 = \( (\lambda \ f. \text{pr-conv-1-to-3} \ (\text{pr-conv-2-to-1} \ f)) \)

lemma \(\text{simp}\): \(\text{pr-conv-1-to-2} \ (\text{pr-conv-2-to-1} \ f) = f\) by \((\text{simp add: pr-conv-1-to-2-def} \ \text{pr-conv-2-to-1-def})\)
lemma \(\text{simp}\): \(\text{pr-conv-2-to-1} \ (\text{pr-conv-1-to-2} \ f) = f\) by \((\text{simp add: pr-conv-1-to-2-def} \ \text{pr-conv-2-to-1-def})\)
lemma \(\text{simp}\): \(\text{pr-conv-1-to-3} \ (\text{pr-conv-3-to-1} \ f) = f\) by \((\text{simp add: pr-conv-1-to-3-def} \ \text{pr-conv-3-to-1-def})\)
lemma \(\text{simp}\): \(\text{pr-conv-3-to-1} \ (\text{pr-conv-1-to-3} \ f) = f\) by \((\text{simp add: pr-conv-1-to-3-def} \ \text{pr-conv-3-to-1-def})\)
lemma \(\text{simp}\): \(\text{pr-conv-3-to-2} \ (\text{pr-conv-2-to-3} \ f) = f\) by \((\text{simp add: pr-conv-3-to-2-def} \ \text{pr-conv-2-to-3-def})\)
lemma \(\text{simp}\): \(\text{pr-conv-2-to-3} \ (\text{pr-conv-3-to-2} \ f) = f\) by \((\text{simp add: pr-conv-3-to-2-def} \ \text{pr-conv-2-to-3-def})\)

lemma pr-conv-1-to-2-lm: \( f \in \text{PrimRec1} \implies \text{pr-conv-1-to-2} \ f \in \text{PrimRec2} \) by \((\text{simp add: pr-conv-1-to-2-def}, \text{prec})\)
lemma pr-conv-1-to-3-lm: \( f \in \text{PrimRec1} \implies \text{pr-conv-1-to-3} \ f \in \text{PrimRec3} \) by \((\text{simp add: pr-conv-1-to-3-def}, \text{prec})\)
lemma pr-conv-2-to-1-lm: \( f \in \text{PrimRec2} \implies \text{pr-conv-2-to-1} \ f \in \text{PrimRec1} \) by \((\text{simp add: pr-conv-2-to-1-def}, \text{prec})\)
lemma pr-conv-3-to-1-lm: \( f \in \text{PrimRec3} \implies \text{pr-conv-3-to-1} \ f \in \text{PrimRec1} \) by \((\text{simp add: pr-conv-3-to-1-def}, \text{prec})\)
lemma pr-conv-3-to-2-lm: \( f \in \text{PrimRec3} \implies \text{pr-conv-3-to-2} \ f \in \text{PrimRec2} \)
proof
  assume \( f \in \text{PrimRec3} \)
  then have \( \text{pr-conv-3-to-1} \ f \in \text{PrimRec1} \) by \((\text{rule pr-conv-3-to-1-lm})\)
  thus ?thesis by \((\text{simp add: pr-conv-3-to-2-def}, \text{prec}, \text{pr-conv-1-to-2-lm})\)
qed

lemma pr-conv-2-to-3-lm: \( f \in \text{PrimRec2} \implies \text{pr-conv-2-to-3} \ f \in \text{PrimRec3} \)
proof
  assume \( f \in \text{PrimRec2} \)

31
then have \( pr\text{-}conv\text{-}2\text{-}to\text{-}1 \ f \in PrimRec1 \) by (rule \( pr\text{-}conv\text{-}2\text{-}to\text{-}1\text{-}lm \))
thus \( \text{thesis} \) by (simp add: \( pr\text{-}conv\text{-}2\text{-}to\text{-}3\text{-}def \) \( pr\text{-}conv\text{-}1\text{-}to\text{-}3\text{-}lm \))
qed

**Theorem b-least2-scheme** \[ [ f \in PrimRec2; \ g \in PrimRec1; \ \forall \ x. \ h \ x < g \ x; \ \forall \ x. \ f \ (h \ x) \neq 0; \ \forall \ z \ x \ z < h \ x \longrightarrow f \ x \ z = 0 ] \implies h \in PrimRec1 \]
proof –
assume f-is-pr: \( f \in PrimRec2 \)
assume g-is-pr: \( g \in PrimRec1 \)
assume h-lt-g: \( \forall \ x. \ h \ x < g \ x \)
assume f-at-h-nz: \( \forall \ x. \ f \ (h \ x) \neq 0 \)
assume h-is-min: \( \forall \ z \ x \ z < h \ x \longrightarrow f \ x \ z = 0 \)
have h-def: \( h = (\lambda x. \ b\text{-}least2 f \ (g \ x)) \)
proof
fix \( x \) show \( h \ x = b\text{-}least2 f \ (g \ x) \)
proof
from f-at-h-nz have \( S1: \ b\text{-}least2 f \ (g \ x) \leq h \ x \) by (simp add: nz-impl-b-least2-le)
from h-lt-g have \( h \ x < g \ x \) by auto
with \( S1 \) have \( b\text{-}least2 f \ (g \ x) < g \ x \) by simp
then have \( S2: \ f \ (b\text{-}least2 f \ (g \ x)) \neq 0 \) by (rule b-least2-less-impl-nz)
have \( S3: \ h \ x \leq b\text{-}least2 f \ (g \ x) \)
proof (rule ccontr)
assume \( \neg \ h \ x \leq b\text{-}least2 f \ (g \ x) \) then have \( b\text{-}least2 f \ (g \ x) < h \ x \) by auto
with h-is-min have \( f \ (b\text{-}least2 f \ (g \ x)) = 0 \) by simp
with \( S2 \) show \( False \) by auto
qed
from \( S1 \) \( S3 \) show \( \text{thesis} \) by auto
qed
qed

**Theorem b-least2-scheme2** \[ [ f \in PrimRec3; \ g \in PrimRec2; \ \forall \ x \ y. \ h \ x \ y < g \ x \ y; \ \forall \ x \ y. \ f \ x \ y \ (h \ x \ y) \neq 0; \ \forall \ z \ x \ y. \ z < h \ x \ y \longrightarrow f \ x \ y \ z = 0 ] \implies h \in PrimRec2 \]
proof –
assume f-is-pr: \( f \in PrimRec3 \)
assume g-is-pr: \( g \in PrimRec2 \)
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: \( \forall \ z \ x \ y. \ z < h \ x \ y \longrightarrow f \ x \ y \ z = 0 \)
define \( f1 \) where \( f1 = b\text{-}least2 f \)
define \( g1 \) where \( g1 = pr\text{-}conv\text{-}3\text{-}to\text{-}2 \)

32
define h1 where h1 = pr-conv-2-to-1 h
from f-is-pr f1-def have f1-is-pr: f1 ∈ PrimRec2 by (simp add: pr-conv-3-to-2-lm)
from g-is-pr g1-def have g1-is-pr: g1 ∈ PrimRec1 by (simp add: pr-conv-2-to-1-lm)
from h-lt-g h1-def g1-def have h1-lt-g1: ∀ x. h1 x < g1 x by (simp add: pr-conv-2-to-1-def)
from f-at-h-nz f1-def h1-def have f1-at-h1-nz: ∀ a b. f a b (h a b) ≠ 0 by (simp add: pr-conv-3-to-2-def pr-conv-1-to-2-def)
from f-at-h-nz f1-def h1-def have f1-at-h1-nz: ∀ x. f1 x (h1 x) ≠ 0 by (simp add: pr-conv-2-to-1-def pr-conv-3-to-2-def pr-conv-1-to-2-def)
qed

theorem div-is-pr: (λ a b. a div b) ∈ PrimRec2
proof
  define f where f a b z = (sgn1 b) * (sgn1 (b*(z+1)) - a) + (sgn2 b)*sgn2 z
  for a b z
  have f-is-pr: f ∈ PrimRec3 unfolding f-def by prec
  define h where h a b = a div b for a b :: nat
  define g where g a b = a + 1 for a b :: nat
  have g-is-pr: g ∈ PrimRec2 unfolding g-def by prec
  have h-lt-g: ∀ a b. h a b < g a b
  proof (rule allI, rule allI)
    fix a b
    from h-def have h a b ≤ a by simp
    also from g-def have a < g a b by simp
    ultimately show h a b < g a b by simp
  qed
  have f-at-h-nz: ∀ a b. f a b (h a b) ≠ 0
  proof (rule allI, rule allI)
    fix a b show f a b (h a b) ≠ 0
    proof cases
      assume A: b = 0
      with h-def have h a b = 0 by simp
      with f-def A show ?thesis by simp
    next
      assume A: b ≠ 0
      then have S1: b > 0 by auto
      from A f-def have S2: f a b (h a b) = sgn1 (b * (b a b + 1) - a) by simp
      then have ?thesis = (sgn1(b * (b a b + 1) - a) ≠ 0) by auto
      also have ... = (b * (h a b + 1) - a > 0) by (rule sgn1-nz-eq-arg-pos)
      also have ... = (a < b * (h a b + 1)) by auto
      also have ... = (a < b * (h a b) + b) by auto
      also from h-def have ... = (a < b * (a div b) + b) by simp
      finally have S3: ?thesis = (a < b * (a div b) + b) by auto
      have S4: a < b * (a div b) + b
      proof
from $S1$ have $S4\text{-}1$: $a \mod b < b$ by (rule mod-less-divisor)
also have $S4\text{-}2$: $b \ast (a \div b) + a \mod b = a$ by (rule mult-div-mod-eq)
from $S4\text{-}1$ have $S4\text{-}3$: $b \ast (a \div b) + a \mod b < b \ast (a \div b) + b$ by arith
from $S4\text{-}2$ $S4\text{-}3$ show $\lnot$thesis by auto
qed
from $S3$ $S4$ show $\lnot$thesis by auto
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 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
proof
assume $\neg b \neq 0$
then have $b = 0$ by auto
with $S2$ show False by auto
qed
from $S3$ have $b$-pos: $0 < b$ by auto
from $S1$ have $S4$: $z+1 \leq a \div b$ by auto
from $b$-pos have $(b \ast (z+1) \leq b \ast (a \div b)) = (z+1 \leq a \div b)$ by (rule
nat-mult-le-cancel1)
with $S4$ have $S5$: $b \ast (z+1) \leq b \ast (a \div b)$ by simp
moreover have $b \ast (a \div b) \leq a$
proof
have $b \ast (a \div b) + (a \mod b) = a$ by (rule mult-div-mod-eq)
moreover have $0 \leq a \mod b$ by auto
ultimately show $\lnot$thesis by arith
qed
ultimately have $S6$: $b \ast (z+1) \leq a$
by (simp add: minus-mod-eq-mult-div [symmetric])
then have $b \ast (z+1) - a = 0$ by auto
with $S3$ $f$-def show $\lnot$thesis by simp
qed
qed

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

theorem $\text{mod}$-is-pr: $(\lambda a b. a \mod b) \in \text{PrimRec2}$
proof
have $(\lambda (a::\text{nat}) (b::\text{nat}). a \mod b) = (\lambda a b. a - (a \div b) \ast b)$
proof (rule ext, rule ext)
fix $a b$
show $(a::\text{nat}) \ mod \ b = a - (a \div b) \ast b$ by (rule minus-div-mult-eq-mod
[symmetric])
qed

34
also from div-is-pr have \((\lambda \ a \ b. \ a - (a \ div \ b) * b) \in \PrimRec2\) by prec
ultimately show ?thesis by aut
qed

theorem pr-rec-last-scheme: \[ g \in \PrimRec1; \ h \in \PrimRec3; \ \forall \ x. \ f \ x \ 0 = g \ x; \ \forall \ x \ y. \ f \ x \ (Suc \ y) = h \ x \ (f \ x \ y) \ y \] \implies \ f \in \PrimRec2
proof –
assume g-is-pr: g \in \PrimRec1
assume h-is-pr: h \in \PrimRec3
assume f-at-0: \ \forall \ x. \ f \ x \ 0 = g \ x
assume f-at-Suc: \ \forall \ x \ y. \ f \ x \ (Suc \ y) = h \ x \ (f \ x \ y) \ y
from f-at-0 f-at-Suc have \ \forall \ x \ y. \ f \ x \ y = PrimRecOp-last \ g \ h \ 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: (\lambda (\ x::nat) \ (\ n::nat). \ x ^ n) \in \PrimRec2
proof –
define g :: nat \Rightarrow 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 \in \PrimRec1 \ unfolding \ g-def \ by \ prec
have h-is-pr: h \in \PrimRec3 \ unfolding \ h-def \ by \ prec
let \ ?f = \lambda (\ x::nat) \ (\ n::nat). \ x ^ n
have f-at-0: \ \forall \ x. \ ?f \ x \ 0 = g \ x
proof
fix x show ?x 0 \ = \ g \ x \ by \ \ (simp \ add: \ g-def)
qed
have f-at-Suc: \ \forall \ x \ y. \ ?f \ x \ (Suc \ y) = h \ x \ (?f \ x \ y) \ y
proof (rule allI, rule allI)
fix x y show ?f \ x \ (Suc \ y) = h \ x \ (?f \ 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 \Rightarrow nat \ where
\[ c\text{-len} = (\lambda (u::\text{nat}). (\text{sgn}1 \ u) \ast (c\text{-fst}(u-(1::\text{nat}))+1)) \]

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

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

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

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

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

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

**Proof cases**
- Assume \( A1: u = 0 \)
  - Thus ?thesis by simp
- Next
  - Assume \( A1: c\text{-len} \ u = 0 \) and \( A2: u \neq 0 \)
  - From \( A2 \) have \( c\text{-len} \ u > 0 \) by (simp add: c-len-3)
  - From \( A1 \) this show \( u=0 \) by simp

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

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

**QED**

**Fun c-fold :: nat list ⇒ nat where**
- \( c\text{-fold} \ [] = 0 \)
- \( c\text{-fold} \ [x] = x \)
- \( c\text{-fold} \ (x#ls) = c\text{-pair} \ x \ (c\text{-fold} \ ls) \)

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

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

**QED**

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

\textbf{lemma} \ c\text{-fold-1}: \ c\text{-unfold} \ 1 \ (\text{c\text{-}fold} \ [x]) = [x] \ \textbf{by simp}

\textbf{lemma} \ c\text{-fold-2}: \ c\text{-fold} \ (\text{c\text{-}unfold} \ 1 \ u) = u \ \textbf{by simp}

\textbf{lemma} \ c\text{-unfold-1}: \ c\text{-unfold} \ 1 \ u = [u] \ \textbf{by simp}

\textbf{lemma} \ c\text{-unfold-2}: \ c\text{-unfold} \ (\text{Suc} \ 1) \ u = (\text{c\text{-}fst} \ u) \ # \ (\text{c\text{-}unfold} \ 1 \ (\text{c\text{-}snd} \ u)) \ \textbf{by simp}

\textbf{lemma} \ c\text{-fold-3}: \ c\text{-fold} \ (\text{c\text{-}unfold} \ 2 \ u) = u \ \textbf{by simp}

\textbf{lemma} \ c\text{-fold-4}: \ k > 0 \Rightarrow \ c\text{-unfold} \ (\text{Suc} \ k) \ u = (\text{c\text{-}fst} \ u) \ # \ (\text{c\text{-}unfold} \ k \ (\text{c\text{-}snd} \ u)) \ \textbf{by simp}

\textbf{lemma} \ c\text{-fold-4-1}: \ k > 0 \Rightarrow \ c\text{-unfold} \ (\text{Suc} \ k) \ u \ # \ (\text{c\text{-}snd} \ u) \ \textbf{by simp add: c\text{-unfold-4}}

\textbf{lemma} \ two: \ (2::nat) = \text{Suc} \ 1 \ \textbf{by simp}

\textbf{lemma} \ c\text{-unfold-5}: \ c\text{-unfold} \ 2 \ u = (\text{c\text{-}fst} \ u) \ # \ (\text{c\text{-}snd} \ u) \ \textbf{by simp add: two}

\textbf{lemma} \ c\text{-unfold-6}: \ k > 0 \Rightarrow \ c\text{-unfold} \ k \ u \ # \ (\text{c\text{-}snd} \ u) \ # \ u \ \textbf{by simp}

\textbf{proof} –
\hspace{1em} \textbf{assume} \ A1: \ k > 0
\hspace{1em} \textbf{let} \ ?k1 = k - \text{(1::nat)}
\hspace{1em} \textbf{from} \ A1 \ \textbf{have} \ S1: \ k = \text{Suc} \ ?k1 \ \textbf{by simp}
\hspace{1em} \textbf{have} \ S2: ?k1 = 0 \Rightarrow \ ?thesis
\hspace{1em} \textbf{proof} –
\hspace{2em} \textbf{assume} \ A2-1: \ ?k1 = 0
\hspace{2em} \textbf{from} \ A1 \ A2-1 \ \textbf{have} \ S2-1: \ k = 1 \ \textbf{by simp}
\hspace{2em} \textbf{from} \ S2-1 \ \textbf{show} \ ?thesis \ \textbf{by simp add: c\text{-unfold-1}}
\hspace{1em} \textbf{qed}
\hspace{1em} \textbf{have} \ S3: ?k1 > 0 \Rightarrow \ ?thesis
\hspace{1em} \textbf{proof} –
\hspace{2em} \textbf{assume} \ A3-1: \ ?k1 > 0
\hspace{2em} \textbf{from} \ A3-1 \ \textbf{have} \ S3-1: \ c\text{-unfold} \ (\text{Suc} \ ?k1) \ u \ # \ (\text{c\text{-}snd} \ u) \ \textbf{by rule c\text{-unfold-4-1}}
\hspace{2em} \textbf{from} \ S1 \ S3-1 \ \textbf{show} \ ?thesis \ \textbf{by simp}
\hspace{1em} \textbf{qed}
\hspace{1em} \textbf{from} \ S2 \ S3 \ \textbf{show} \ ?thesis \ \textbf{by arith}
\hspace{1em} \textbf{qed}

\textbf{lemma} \ th\text{-}lm-1: \ k = 1 \Rightarrow (\forall \ u. \ c\text{-fold} \ (\text{c\text{-}unfold} \ k \ u) = u) \ \textbf{by simp add: c\text{-fold-2}}

\textbf{lemma} \ th\text{-}lm-2: \ [k > 0; (\forall \ u. \ c\text{-fold} \ (\text{c\text{-}unfold} \ k \ u) = u)] \Rightarrow (\forall \ u. \ c\text{-fold} \ (\text{c\text{-}unfold} \ (\text{Suc} \ k) \ u) = u)

\textbf{proof}
assume \( A1: k > 0 \)
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 \( c\text{-unfold-4} \))
  let \( ?ls = c\text{-unfold} \; k \; (c\text{-snd} \; u) \)
from \( A1 \) have \( S2: \; ?ls \neq [] \) by (rule \( c\text{-unfold-6} \))
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 \( c\text{-fold-0} \))
from \( A2 \) 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 \( \text{thesis} \) by simp
qed

lemma \( th\text{-lm-3}: \; (\forall u. \; c\text{-fold} \; (c\text{-unfold} \; (Suc \; k) \; u) = u) \implies (\forall u. \; c\text{-fold} \; (c\text{-unfold} \; (Suc \; (Suc \; k)) \; u) = u) \)
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 \( A1 \) have \( S2: \; \forall u. \; c\text{-fold} \; (c\text{-unfold} \; (Suc \; ?k1) \; u) = u \) by (rule \( th\text{-lm-2} \))
  thus \( \text{thesis} \) by simp
qed

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

theorem \( th\text{-2}: \; k > 0 \implies (\forall u. \; c\text{-fold} \; (c\text{-unfold} \; k \; u) = u) \)
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 \( th\text{-1} \))
  from \( S1 \; S2 \) show \( \text{thesis} \) by simp
qed

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

theorem \( c\text{-unfold-len}: \; \forall u. \; length \; (c\text{-unfold} \; k \; u) = k \)
apply(induct \( k \))
apply(simp)
apply(subgoal-tac \( n=(0::nat) \lor n>0 \))
apply (drule disjE)
prefer 3
apply (simp-all)
apply (auto)
done

lemma th-3-lm-0: \[c\text{-unfold}\ (\text{length } ls)\ (c\text{-fold } ls) = ls; ls = a \# ls1; ls1 = aa \# list\] \(\implies\) \(c\text{-unfold}\ (\text{length } (x \# ls))\ (c\text{-fold } (x \# ls)) = x \# ls\)
proof –
assume A1: \(c\text{-unfold}\ (\text{length } ls)\ (c\text{-fold } ls) = ls\)
assume A2: \(ls = a \# ls1\)
assume A3: \(ls1 = aa \# list\)
from A2 have S1: \(ls \neq []\) by simp
from S1 have S2: \(c\text{-fold } (x \# ls) = c\text{-pair } x (c\text{-fold } ls)\) by (rule c-fold-0)
have S3: \(\text{length } (x \# ls) = \text{Suc } (\text{length } ls)\) by simp
from S3 have S4: \(c\text{-unfold}\ (\text{length } (x \# ls))\ (c\text{-fold } (x \# ls)) = c\text{-unfold}\ (\text{Suc } (\text{length } ls))\ (c\text{-fold } (x \# ls))\) by simp
from A2 have S5: \(\text{length } ls > 0\) by simp
from S5 have S6: \(c\text{-unfold}\ (\text{Suc } (\text{length } ls))\ (c\text{-fold } (x \# ls)) = c\text{-fst } (c\text{-fold } (x \# ls))\) by (rule c-unfold-4)
from S2 have S7: \(c\text{-fst } (c\text{-fold } (x \# ls)) = x\) by simp
from S2 have S8: \(c\text{-snd } (c\text{-fold } (x \# ls)) = c\text{-fold } ls\) by simp
from S6 S7 S8 have S9: \(c\text{-unfold}\ (\text{Suc } (\text{length } ls))\ (c\text{-fold } (x \# ls)) = x \# (\text{c-unfold } (\text{length } ls)\ (c\text{-fold } ls))\) by simp
from A1 have S10: \(x \# (\text{c-unfold } (\text{length } ls)\ (c\text{-fold } ls)) = x \# ls\) by simp
from S9 S10 have S11: \(c\text{-unfold}\ (\text{Suc } (\text{length } ls))\ (c\text{-fold } (x \# ls)) = (x \# ls)\) by simp
thus ?thesis by simp
qed

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

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

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

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

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

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

theorem list-to-nat-th [simp]: list-to-nat (nat-to-list u) = u
proof –
  have S1: u=0 ⇒ ?thesis by (simp add: list-to-nat-def nat-to-list-def)
  have S2: u>0 ⇒ ?thesis
proof –
  assume A1: u>0
  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 ?k = c-len u
  from A1 have S2-2: ?k > 0 by (rule c-len-3)
  from S2-1 have S2-3: length ls = ?k by (simp add: c-unfold-len)
  from S2-2 S2-3 have S2-4: length ls > 0 by simp
  from S2-4 have S2-5: ls ≠ [] by simp
  from S2-5 have S2-6: list-to-nat ls = c-pair ((length ls)−(1::nat)) (c-fold ls)+1
  by (simp add: list-to-nat-def)
  have S2-7: c-fold ls = c-snd(u−(1::nat))
  proof –
  from S2-1 have S2-7-1: c-fold ls = c-fold (c-unfold (c-len u) (c-snd (u−(1::nat))))
  by simp
  from S2-2 S2-7-1 show ?thesis by (simp add: th-2)
  qed
  have S2-8: (length ls)−(1::nat) = c-fst (u−(1::nat))
  proof –
  from S2-3 have S2-8-1: length ls = c-len u by simp
  from A1 S2-8-1 have S2-8-2: length ls = c-fst(u−(1::nat)) + 1 by (simp add: c-len-2)
  from S2-8-2 show ?thesis by simp
  qed
  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=[] ==> ?thesis by (simp add: nat-to-list-def list-to-nat-def)
  have S2: ls ≠ [] ==> ?thesis
  proof –
    assume A1: ls ≠ []
    define u where u = list-to-nat ls
    from u-def A1 have S2-1: u = c-pair ((length ls)-(1::nat)) (c-fold bs)+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)
    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 = [] ∨ ls ≠ [] by simp
  from S1 S2 S3 show ?thesis by auto
  qed

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

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

theorem c-len-th-1: c-len (list-to-nat ls) = length ls
proof (cases)
  assume ls=[]

41
from this show ?thesis by simp
next
  assume S1: ls ≠ []
  then have S2: list-to-nat ls = c-pair ((length ls)−(1::nat)) (c-fold ls)+1 by (simp add: list-to-nat-def)
  let ?u = list-to-nat ls
  from S2 have u-not-zero: ?u > 0 by simp
  from S2 have S3: ?u−(1::nat) = c-pair ((length ls)−(1::nat)) (c-fold ls) by simp
  then have S4: c-fst(?u−(1::nat)) = (length ls)−(1::nat) by simp
  from S1 this have S5: c-fst(?u−(1::nat))+(1=length ls) by simp
  from u-not-zero S5 have S6: c-fst (?u) = length ls by (simp add: c-len-2)
  from S1 S6 show ?thesis by simp
qed

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

definition c-hd :: nat ⇒ nat where
c-hd = (λ u. if u=0 then 0 else hd (nat-to-list u))

definition c-tl :: nat ⇒ nat where
c-tl = (λ u. list-to-nat (tl (nat-to-list u)))

definition c-cons :: nat ⇒ nat ⇒ nat where
c-cons = (λ x u. list-to-nat (x # (nat-to-list u)))

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

lemma c-hd-aux0: c-len u = 1 ⇒ nat-to-list u = [c-snd (u−(1::nat))] by (simp add: nat-to-list-def c-len-5)

lemma c-hd-aux1: c-len u = 1 ⇒ c-hd u = c-snd (u−(1::nat))
proof −
  assume A1: c-len u = 1
  then have S1: nat-to-list u = [c-snd (u−(1::nat))] by (simp add: nat-to-list-def c-len-5)
  from A1 have u > 0 by (simp add: c-len-5)
  with S1 show ?thesis by (simp add: c-hd-def)
qed

42
lemma c-hd-aux2: c-len u > 1 \implies c-hd u = c-fst (c-snd (u-(1::nat)))
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: c-hd u = hd (nat-to-list u) by (simp add: c-hd-def)
  from S3 have S5: nat-to-list u = c-unfold (c-len u) (c-snd (u-(1::nat))) by
  (rule nat-to-list-of-pos)
  from S1 S5 have S6: nat-to-list u = c-unfold (Suc \(?k\)) (c-snd (u-(1::nat))) by simp
  from S1 have S7: \(?k > 0\) by simp
  from S7 have S8: c-unfold (Suc \(?k\)) (c-snd (u-(1::nat))) = (c-fst (c-snd (u-(1::nat))))
  (c-unfold \(?k\) (c-snd (u-(1::nat)))) by (rule c-unfold-4)
  from S6 S8 have S9: nat-to-list u = (c-fst (c-snd (u-(1::nat)))) # (c-unfold \(?k\)
  (c-snd (u-(1::nat)))) by simp
  from S9 have S10: hd (nat-to-list u) = c-fst (c-snd (u-(1::nat))) by simp
  from S4 S10 show \(?thesis\) by simp
qed

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

next
assume \( u \neq 0 \) then have \( A1: \ u > 0 \) by simp
then show \( \textit{thesis} \) by (simp add: c-hd-aux3)
qed

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

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

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

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

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

lemma \( \text{c-tl-aux2} \): \( \text{c-len } u > 1 \implies \text{c-tl } u = (\text{c-pair } (\text{c-len } u - (2::nat)) \text{ (c-snd } (\text{c-snd } (u-(1::nat)))) + 1 \)
proof –
  assume \( A1: \text{c-len } u > 1 \)
  let \( ?k = (\text{c-len } u) - 1 \)
  from \( A1 \) have \( S1: \text{c-len } u = \text{Suc } ?k \) by simp
  from \( A1 \) have \( S2: \text{c-len } u > 0 \) by simp
  from \( S2 \) have \( S3: u > 0 \) by (rule c-len-5)
  from \( S3 \) have \( S4: \text{nat-to-list } u = \text{c-unfold } (\text{c-len } u) \text{ (c-snd } (u-(1::nat)))) \) by (rule nat-to-list-of-pos)
  from \( A1 \) have \( S5: ?k > 0 \) by simp
  from \( S5 \) have \( S6: \text{c-unfold } (\text{Suc } ?k) \text{ (c-snd } (u-(1::nat)))) = (\text{c-fst } (\text{c-snd } (u-(1::nat)))) \text{ (c-snd } (\text{c-snd } (u-(1::nat)))) \) \# (\text{c-unfold } ?k \text{ (c-snd } (c-snd } (u-(1::nat)))) \) by (rule c-unfold-4)
  from \( S6 \) have \( S7: \text{tl } \text{c-unfold } (\text{Suc } ?k) \text{ (c-snd } (u-(1::nat)))) = \text{c-unfold } ?k \)
(c-snd (c-snd (u−(1::nat)))) by simp
from S2 S4 S7 have S8: tl (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
qed

lemma c-tl-less: u > 0 ⇒ c-tl u < u

45
proof

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

lemma $c$-tl-le: $c$-tl $u \leq u$
proof (cases $u$
assume $u = 0$
then show ?thesis by simp
next
fix $v$ assume $A1$: $u = \text{Suc} \; v$
then have $S1$: $u > 0$ by simp
then have $S2$: $c$-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 $\in \text{PrimRec1}$
proof
have $c$-tl $= (\lambda \; u. (\text{snd} \; 1 ((\text{c-len} \; u) - 1)) * (\text{c-pair} (\text{c-len} \; u - (2 :: \text{nat})) (\text{c-snd} (\text{c-snd} (u - (1 :: \text{nat})))))) + 1)$ by (simp add: $c$-tl-aux3 ext)
moreover from $\text{c-len-is-pr} \; c$-pair-is-pr have $?R \in \text{PrimRec1}$ by prec
ultimately show ?thesis by simp
qed

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

lemma c-cons-aux2: \( u > 0 \implies c\text{-}\text{cons} \, x \, u = (c\text{-pair} \, (c\text{-len} \, u) \, (c\text{-pair} \, x \, (c\text{-snd} \, (u-(1::nat)))) + 1 \)
proof
  assume A1: \( u > 0 \)
  from A1 have S1: c-len u > 0 by (rule c-len-3)
  from A1 have S2: nat-to-list u = c-unfold (c-len u) (c-snd (u-(1::nat))) by (rule nat-to-list-of-pos)
    define ls where \( ls = \text{nat-to-list} \, u \)
  from ls-def S2 have S3: ls = c-unfold (c-len u) (c-snd (u-(1::nat))) by simp
  from S3 have S4: length ls = c-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 S7 have S8: c-cons x u = list-to-nat (x \# ls) by (simp add: c-cons-def)
    have S9: list-to-nat (x \# ls) = (c\text{-pair} ((\text{length} \, (x\#ls))-(1::nat)) \, (c\text{-fold} \, (x\#ls))) + 1
      by (simp add: list-to-nat-def)
    have S10: list-to-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::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::nat))))
          by simp
        from S1 S11-2 have S11-3: c\text{-fold} \, ls = c\text{-snd} \, (u-(1::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)\ast((c\text{-pair} \, 0 \, x) + 1) + (sgn1 \, u)\ast((c\text{-pair} \, (c\text{-len} \, u) \, (c\text{-pair} \, x \, (c\text{-snd} \, (u-(1::nat)))) + 1))
proof (rule ext, rule ext)
  fix x u show c\text{-cons} \, x \, u = (sgn2 \, u)\ast((c\text{-pair} \, 0 \, x) + 1) + (sgn1 \, u)\ast((c\text{-pair} \, (c\text{-len} \, u) \, (c\text{-pair} \, x \, (c\text{-snd} \, (u-(1::nat)))) + 1) \, (\text{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: \( u\neq0 \)
    then have S1: \(?R = (c\text{-pair} \, (c\text{-len} \, u) \, (c\text{-pair} \, x \, (c\text{-snd} \, (u-(1::nat)))) + 1 \) by simp
    from A1 have S2: c\text{-cons} \, x \, u = (c\text{-pair} \, (c\text{-len} \, u) \, (c\text{-pair} \, x \, (c\text{-snd} \, (u-(1::nat)))) + 1 by (simp add: c-cons-aux2)

47
from S1 S2 have c-cons x u = ?R by simp 
then show ?thesis . 
qed
lemma c-cons-pos: c-cons x u > 0 
proof cases 
  assume u=0 
  then show c-cons x u > 0 by (simp add: c-cons-aux1) 
next 
  assume ¬ u=0 then have u>0 by simp 
  then show c-cons x u > 0 by (simp add: c-cons-aux2) 
qed
theorem c-cons-is-pr: c-cons ∈ PrimRec2 
proof – 
  have c-cons = (λ x u. (sgn2 u)*((c-pair 0 x)+1) + (sgn1 u)*((c-pair (c-len u) 
  (c-pair x (c-snd (u−(1::nat)))))) + 1)) (ls = ?R) by (simp add: c-cons-aux3) 
  moreover from c-pair-is-pr c-len-is-pr have ?R ∈ PrimRec2 by prec 
  ultimately show ?thesis by simp 
qed
definition c-drop :: nat ⇒ nat ⇒ nat where 
c-drop = PrimRecOp (λ x. x) (λ x y z. c-tl y) 
lemma c-drop-at-0 [simp]: c-drop 0 x = x by (simp add: c-drop-def) 
lemma c-drop-at-Suc: c-drop (Suc y) x = c-tl (c-drop y x) by (simp add: c-drop-def) 
theorem c-drop-is-pr: c-drop ∈ PrimRec2 
proof – 
  have (λ x. x) ∈ PrimRec1 by (rule pr-id1-1) 
  moreover from c-tl-is-pr have (λ x y z. c-tl y) ∈ PrimRec3 by prec 
  ultimately show ?thesis by (simp add: c-drop-def pr-rec) 
qed
lemma c-tl-c-drop: c-tl (c-drop y x) = c-drop y (c-tl x) 
apply(induct y) 
apply(simp) 
apply(simp add: c-drop-at-Suc) 
done
lemma c-drop-at-Suc1: c-drop (Suc y) x = c-drop y (c-tl x) 
apply(simp add: c-drop-at-Suc c-tl-c-drop) 
done
lemma c-drop-df: ∀ ls. drop n ls = nat-to-list (c-drop n (list-to-nat ls)) 
proof (induct n)
show \( \forall \) ls. drop 0 ls = nat-to-list (c-drop 0 (list-to-nat ls)) by (simp add: c-drop-def)

next

fix n assume A1: \( \forall \) ls. drop n ls = nat-to-list (c-drop n (list-to-nat ls))
then show \( \forall \) ls. drop (Suc n) ls = nat-to-list (c-drop (Suc n) (list-to-nat ls))
proof –

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

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

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

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

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

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

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

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

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

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

lemma c-f-list-to-f: \( f = (\lambda y \ x. \text{c-hd } (\text{c-f-list } f \ y \ x)) \)
apply (rule ext, rule ext)
apply (rule c-f-list-to-f-0)
done

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

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

lemma c-f-list-lm-2: \( z < \text{Suc } n \implies \text{c-nth } (\text{c-f-list } f \ (\text{Suc } n) \ x) \ (\text{Suc } n - z) = \text{c-nth } (\text{c-f-list } f \ x) \ (n - z) \)
proof –
  assume z < Suc n
  then have Suc n - z = Suc (n-z) by arith
  then have c-nth (c-f-list f (Suc n) x) (Suc n - z) = c-nth (c-f-list f (Suc n) x) (Suc (n - z)) by simp
  also have \( \ldots = \text{c-nth } (\text{c-cons } (f \ (\text{Suc } n) \ x) \ (\text{c-f-list } f \ n \ x)) \ (\text{Suc } (n - z)) \) by (simp add: c-f-list-at-Suc)
  also have \( \ldots = \text{c-nth } (\text{c-f-list } f \ n \ x) \ (n - z) \) by (simp add: c-f-list-lm-1)
  finally show \( ?\text{thesis} \) by simp
qed
lemma \( \text{c-f-list-nth: } z \leq y \rightarrow \text{c-nth } (\text{c-f-list } f y x) (y-z) = f z x \)

proof (induct y)

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

proof

assume \( z \leq 0 \) then have \( A1: z = 0 \) by simp

then have \( \text{c-nth } (\text{c-f-list } f 0 x) (0 - z) = \text{c-nth } (\text{c-f-list } f 0 x) 0 \) by simp

also have \( \ldots = \text{c-hd } (\text{c-f-list } f 0 x) \) by (simp add: c-nth-at-0)

also have \( \ldots = \text{c-hd } (\text{c-cons } (f 0 x) 0) \) by (simp add: c-f-list-0)

also have \( \ldots = f 0 x \) by simp

finally show \( \text{c-nth } (\text{c-f-list } f 0 x) (0 - z) = f z x \) by (simp add: A1)

qed

next

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

Suc \( n \rightarrow \text{c-nth } (\text{c-f-list } f (\text{Suc } n) x) (\text{Suc } n - z) = f z x \)

proof

assume \( A3: z \leq \text{Suc } n \)

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

proof cases

assume \( \text{AA1: } z \leq n \)

then have \( \text{AA2: } z < \text{Suc } n \) by simp

from \( A2 \) this have \( S1: \text{c-nth } (\text{c-f-list } f n x) (n - z) = f z x \) by auto

from \( AA2 \) have \( \text{c-nth } (\text{c-f-list } f (\text{Suc } n) x) (\text{Suc } n - z) = \text{c-nth } (\text{c-f-list } f n \)

\( x \) (\( n - z \)) by (rule c-f-list-tm-2)

with \( S1 \) show \( \text{c-nth } (\text{c-f-list } f (\text{Suc } n) x) (\text{Suc } n - z) = f z x \) by simp

next

assume \( \neg z \leq n \)

from \( A3 \) this have \( S1: \text{z = Suc } n \) by simp

then have \( S2: \text{Suc } n - z = 0 \) by simp

then have \( \text{c-nth } (\text{c-f-list } f (\text{Suc } n) x) (\text{Suc } n - z) = \text{c-nth } (\text{c-f-list } f (\text{Suc } n) \)

\( x \) 0 \) by simp

also have \( \ldots = \text{c-hd } (\text{c-f-list } f (\text{Suc } n) x) \) by (simp add: c-nth-at-0)

also have \( \ldots = \text{c-hd } (\text{c-cons } (f (\text{Suc } n) x) (\text{c-f-list } f n \)

\( x) \) by (simp add: c-f-list-at-Suc)

also have \( \ldots = f (\text{Suc } n) x \) by simp

finally show \( \text{c-nth } (\text{c-f-list } f (\text{Suc } n) x) (\text{Suc } n - z) = f z x \) by (simp add: \( S1) \)

qed

qed

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

proof

assume \( g\text{-is-pr: } g \in \text{PrimRec1} \)

assume \( h\text{-is-pr: } h \in \text{PrimRec3} \)

assume \( f 0: \forall x. f 0 x = g x \)

assume \( f 1: \forall x y. (f (\text{Suc } y) x) = h y (f y x) x \)

let \( \forall f = \text{PrimRec1} g h \)
from g-is-pr h-is-pr have S1: ?f ∈ PrimRec2 by (rule pr-rec)
have f-2:∀ x. ?f 0 x = g x by simp
have f-3: ∀ x y. (?f (Suc y) x) = h y (?f y x) x by simp
have S2: f = ?f
proof
  have ∃ x y. ?f x y = ?f x
  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 th-rec: [ g ∈ PrimRec1; α ∈ PrimRec2; h ∈ PrimRec3; (∀ x y. α y x ≤ y); (∀ x. (f 0 x) = (g x)); (∀ x y. (f (Suc y) x) = h y (f (α y x) x)) ] ⇒ f ∈ PrimRec2
proof
  assume g-is-pr: g ∈ PrimRec1
  assume a-is-pr: α ∈ PrimRec2
  assume h-is-pr: h ∈ PrimRec3
  assume a-le: (∀ x y. α y x ≤ y)
  assume f-0: ∀ x. f 0 x = g x
  assume f-1: ∀ x y. (f (Suc y) x) = h y (f (α y x) x)
  let ?g′ = λ x. c-cons (g x) 0
  let ?h′ = λ a b c. c-cons (h a (c-nth b (a − (α a c)))) c) b
  let ?r = c-f-list f
from g-is-pr c-cons-is-pr have g′-is-pr: ?g′ ∈ PrimRec1 by prec
from h-is-pr c-cons-is-pr c-nth-is-pr a-is-pr have h′-is-pr: ?h′ ∈ PrimRec3 by prec
have S1: ∀ x. ?r 0 x = ?g′ x
proof
  fix x have ?r 0 x = c-cons (f 0 x) 0 by (rule c-f-list-at-0)
  with f-0 have ?r 0 x = c-cons (g x) 0 by simp
  then show ?r 0 x = ?g′ x by simp
qed
have S2: ∀ x y. ?r (Suc y) x = ?h′ y (?r y x) x
proof (rule allI, rule allI)
  fix x y show ?r (Suc y) x = ?h′ y (?r y x) x
  proof
    have S2-1: ?r (Suc y) x = c-cons (f (Suc y) x) (?r y x) by (rule c-f-list-at-Suc)
    with f-1 have S2-2: f (Suc y) x = h y (f (α y x) x) x by simp
    from a-le have S2-3: α y x ≤ y by simp
    then have S2-4: f (α y x) x = c-nth (?r y x) (y−(α y x)) by (simp add: c-f-list-nth)
    from S2-1 S2-2 S2-4 show ?thesis by simp
  qed
qed

52
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 :: \( \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \) where
  c-assoc-have-key-df [simp del]: c-assoc-have-key \( y \ x = (\text{if } y = 0 \text{ then } 1 \text{ else } \text{c-assoc-have-key} \ (\text{c-tl } y) \ x) \)
lemma c-assoc-have-key-lm-1: \( y \neq 0 \Rightarrow \text{c-assoc-have-key} \ y \ x = (\text{if } \text{c-fst} \ (\text{c-hd } y) = x \text{ then } 0 \text{ else } \text{c-assoc-have-key} \ (\text{c-tl } y) \ x) \) by (simp add: c-assoc-have-key-df)
theorem c-assoc-have-key-is-pr: c-assoc-have-key \( \in \text{PrimRec2} \)
proof
  let \( ?h = \lambda a \ b \ c. \text{if } \text{c-fst} \ (\text{c-hd} \ (\text{Suc } a)) = c \text{ then } 0 \text{ else } b \)
  let \( ?a = \lambda y \ x. \text{c-tl} \ (\text{Suc } y) \)
  let \( ?q = \lambda x. \ (1::\text{nat}) \)
  have g-is-pr: \( ?g \in \text{PrimRec1} \) by (rule const-is-pr)
from c-tl-is-pr have a-is-pr: \( ?a \in \text{PrimRec2} \) by prec
have h-is-pr: \( ?h \in \text{PrimRec3} \)
proof (rule if-eq-is-pr3)
from c-fst-is-pr c-hd-is-pr show \( \lambda x \ y \ z. \text{c-fst} \ (\text{c-hd} \ (\text{Suc } x)) \in \text{PrimRec3} \) by prec
next
  show \( \lambda x \ y \ z. \in \text{PrimRec3} \) by (rule pr-id3-3)
next
  show \( \lambda x \ y \ 0. \in \text{PrimRec3} \) by prec
next
  show \( \lambda x \ y \ z. \in \text{PrimRec3} \) by (rule pr-id3-2)
qed
have a-le: \( \forall \ x \ y. \ ?a \ y \ x \leq y \)
proof (rule allI, rule allI)
  fix \( x \ y \) show \( ?a \ y \ x \leq y \)
  proof
    have Suc y > 0 by simp
    then have \( ?a \ y \ x < \text{Suc } y \) by (rule c-tl-less)
    then show \( \text{thesis} \) by simp
  qed
qed
have f-0: \( \forall \ x. \ \text{c-assoc-have-key} \ 0 \ x = \ ?g \ x \) by (simp add: c-assoc-have-key-df)
have f-1: \( \forall \ x \ y. \ \text{c-assoc-have-key} \ (\text{Suc } y) \ x = \ ?h \ y \ (\text{c-assoc-have-key} \ (\ ?a \ y \ x) \ x) \) 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)
qed

fun c-assoc-value :: \( \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \) where
  c-assoc-value-df [simp del]: c-assoc-value \( y \ x = (\text{if } y = 0 \text{ then } 0 \text{ else } \text{c-assoc-value} \ (\text{c-tl } y) \ x) \)
\[ y \neq 0 \implies c\text{-assoc-value} \ x \ y = (\text{if } c\text{-fst} (c\text{-hd} \ y) = x \ \text{then } c\text{-snd} (c\text{-hd} \ y) \ \text{else } c\text{-assoc-value} (c\text{-tl} \ y) \ x) \] by (simp add: c-assoc-value-df)

**Theorem** c-assoc-value-is-pr: c-assoc-value \( \in \text{PrimRec2} \)

**Proof**
- let \( ?h = \lambda a \ b \ c. \ \text{if } c\text{-fst} (c\text{-hd} (\text{Suc} \ a)) = c \ \text{then } c\text{-snd} (c\text{-hd} (\text{Suc} \ a)) \ \text{else } b \)
- let \( ?a = \lambda y \ x. \ c\text{-tl} (\text{Suc} \ y) \)
- let \( ?g = \lambda x. (0::\text{nat}) \)

- have g-is-pr: \( ?g \in \text{PrimRec1} \) by (rule const-is-pr)
- from c-tl-is-pr have a-is-pr: \( ?a \in \text{PrimRec2} \) by prec
- have h-is-pr: \( ?h \in \text{PrimRec3} \) proof (rule if-eq-is-pr3)

- from c-fst-is-pr c-hd-is-pr show \( (\lambda x \ y \ z. \ c\text{-fst} (c\text{-hd} (\text{Suc} \ x))) \in \text{PrimRec3} \) by prec
- next
  - show \( (\lambda x \ y \ z. \ c\text{-snd} (c\text{-hd} (\text{Suc} \ x))) \in \text{PrimRec3} \) by prec
- next
  - show \( (\lambda x \ y. \ c\text{-assoc-value} 0 \ x) \in \text{PrimRec2} \) by (simp add: c-assoc-value-df)

**Lemma** c-assoc-lm-1: \( c\text{-assoc-value} (c\text{-cons} (c\text{-pair} \ x \ y) \ z) \ x = 0 \)
apply (simp add: c-assoc-value-df)
done

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

54
lemma \(c\text{-assoc-lm-3}: x_1 \neq x \implies c\text{-assoc-have-key} (c\text{-cons} (c\text{-pair} x y) z) x_1 = c\text{-assoc-have-key} z x_1\)

proof –

assume \(A_1: x_1 \neq x\)

let \(\ell s = (c\text{-cons} (c\text{-pair} x y) z)\)

have \(S_1: \ell s \neq 0\) by (simp add: c-cons-pos)

then have \(S_2: c\text{-assoc-have-key} \ell s x_1 = \text{if} c\text{-fst} (c\text{-hd} \ell s) = x_1 \text{ then } 0 \text{ else } c\text{-assoc-have-key} (c\text{-tl} \ell s) x_1\) (is - = ?R) by (rule c-assoc-have-key-lm-1)

have \(S_3: c\text{-fst} (c\text{-hd} \ell s) = x\) by simp

with \(A_1\) have \(S_4: \neg (c\text{-fst} (c\text{-hd} \ell s) = x_1)\) by simp

from \(S_4\) have \(S_5: \ell s = c\text{-assoc-have-key} (c\text{-tl} \ell s) x_1\) by (rule if-not-P)

from \(S_2\) \(S_5\) show ?thesis by simp

qed

lemma \(c\text{-assoc-lm-4}: x_1 \neq x \implies c\text{-assoc-value} (c\text{-cons} (c\text{-pair} x y) z) x_1 = c\text{-assoc-value} z x_1\)

proof –

assume \(A_1: x_1 \neq x\)

let \(\ell s = (c\text{-cons} (c\text{-pair} x y) z)\)

have \(S_1: \ell s \neq 0\) by (simp add: c-cons-pos)

then have \(S_2: c\text{-assoc-value} \ell s x_1 = \text{if} c\text{-fst} (c\text{-hd} \ell s) = x_1 \text{ then } c\text{-snd} (c\text{-hd} \ell s) \text{ else } c\text{-assoc-value} (c\text{-tl} \ell s) x_1\) (is - = ?R) by (rule c-assoc-value-lm-1)

have \(S_3: c\text{-fst} (c\text{-hd} \ell s) = x\) by simp

with \(A_1\) have \(S_4: \neg (c\text{-fst} (c\text{-hd} \ell s) = x_1)\) by simp

from \(S_4\) have \(S_5: \ell s = c\text{-assoc-value} (c\text{-tl} \ell s) x_1\) by (rule if-not-P)

from \(S_2\) \(S_5\) 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

\[\text{UnaryRecOp} :: (\text{nat} \Rightarrow \text{nat}) \Rightarrow (\text{nat} \Rightarrow \text{nat}) \Rightarrow (\text{nat} \Rightarrow \text{nat}) \text{ where}\]

\[\text{UnaryRecOp} = (\lambda g h. \text{pr-conv-2-to-1} (\text{PrimRecOp} g (\text{pr-conv-1-to-3} h)))\]

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

definition
c-f-pair :: (nat ⇒ nat) ⇒ (nat ⇒ nat) ⇒ (nat ⇒ nat) where
  c-f-pair = (λ f g x. c-pair (f x) (g x))

lemma c-f-pair-to-pr: [
  f ∈ PrimRec1; g ∈ PrimRec1
] ⇒ c-f-pair f g ∈ PrimRec1
unfolding c-f-pair-def by prec

inductive-set PrimRec1' :: (nat ⇒ nat) set
where
  zero: (λ x. 0) ∈ PrimRec1'
  suc: Suc ∈ PrimRec1'
  fst: c-fst ∈ PrimRec1'
  snd: c-snd ∈ PrimRec1'
  comp: [ f ∈ PrimRec1'; g ∈ PrimRec1' ] ⇒ (λ x. f (g x)) ∈ PrimRec1'
  pair: [ f ∈ PrimRec1'; g ∈ PrimRec1' ] ⇒ c-f-pair f g ∈ PrimRec1'
  un-rec: [ f ∈ PrimRec1'; g ∈ PrimRec1' ] ⇒ UnaryRecOp f g ∈ PrimRec1'

lemma primrec'-into-primrec: f ∈ PrimRec1' ⇒ f ∈ PrimRec1
proof (induct f rule: PrimRec1'.induct)
case zero show ?case by (rule pr-zero)
next
case suc show ?case by (rule pr-suc)
next
case fst show ?case by (rule c-fst-is-pr)
next
case snd show ?case by (rule c-snd-is-pr)
next
case comp from comp show ?case by (simp add: pr-comp1-1)
next
case pair from pair show ?case by (simp add: c-f-pair-to-pr)
next
case un-rec from un-rec show ?case by (simp add: unary-rec-into-pr)
qed

lemma pr-id1-1': (λ x. x) ∈ PrimRec1'
proof –
have c-f-pair c-fst c-snd ∈ PrimRec1' by (simp add: PrimRec1'.fst PrimRec1'.snd PrimRec1'.pair)
moreover have c-f-pair c-fst c-snd = (λ x. x) by (simp add: c-f-pair-def)
ultimately show ?thesis by simp
qed

lemma pr-id2-1': pr-conv-2-to-1 (λ x y. x) ∈ PrimRec1' by (simp add: pr-conv-2-to-1-def
PrimRec1'.fst)

lemma pr-id2-2': pr-conv-2-to-1 (λ x y. y) ∈ PrimRec1' by (simp add: pr-conv-2-to-1-def
PrimRec1'.snd)

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

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

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

lemma pr-comp3-1': [ pr-conv-2-to-1 f \in PrimRec1'; g \in PrimRec1'; h \in PrimRec1' ] \implies (\lambda x. f (g x) (h x)) \in PrimRec1'
proof –
assume A1: pr-conv-2-to-1 f \in PrimRec1'
assume A2: g \in PrimRec1'
assume A3: h \in PrimRec1'
let \(?f1 = pr\text{-}conv\text{-}2\text{-}to\text{-}1 f\)
have S1: \((\%x. ?f1 ((c\text{-}f\text{pair} g h) x))\) = \((\lambda x. f (g x) (h x))\) by (simp add: c-f-pair-def pr-conv-2-to-1-def)
from A2 A3 have S2: c-f-pair g h \in PrimRec1' by (rule PrimRec1',pair)
from A1 S2 have S3: \((\%x. ?f1 ((c\text{-}f\text{pair} g h) x))\) \in PrimRec1' by (rule PrimRec1',comp)
with S1 show \(?\)thesis by simp
qed

lemma pr-comp3-1'': [ pr-conv-3-to-1 f \in PrimRec1'; g \in PrimRec1'; h \in PrimRec1'; k \in PrimRec1' ] \implies (\lambda x. f (g x) (h x) (k x)) \in PrimRec1'
proof –
assume A1: pr-conv-3-to-1 f \in PrimRec1'
assume A2: g \in PrimRec1'
assume A3: h \in PrimRec1'
assume A4: k \in PrimRec1'
from A2 A3 have c-f-pair g h \in PrimRec1' by (rule PrimRec1',pair)
from this A4 have c-f-pair (c-f-pair g h) k \in PrimRec1' by (rule PrimRec1',pair)
from A1 this have \((\%x. (pr\text{-}conv\text{-}3\text{-}to\text{-}1 f)) ((c\text{-}f\text{pair} (c\text{-}f\text{pair} g h) k) x))\) \in PrimRec1' by (rule PrimRec1',comp)
then show \(?\)thesis by (simp add: c-f-pair-def pr-conv-3-to-1-def)
qed
lemma \( \text{pr-comp1-2}' \): \([ f \in \text{PrimRec}' ; \text{pr-conv-2-to-1} g \in \text{PrimRec}' \] \implies \text{pr-conv-2-to-1} (\lambda x y z. f (g x y z)) \in \text{PrimRec}'

proof

assume \( f \in \text{PrimRec}' \)
and \( \text{pr-conv-2-to-1} g \in \text{PrimRec}' \) (is \(?g1 \in \text{PrimRec}'\))
then have \( (\lambda x. f \ (?g1 x)) \in \text{PrimRec}' \) by (rule PrimRec',comp)
then show \(?\text{thesis}\) by (simp add: pr-conv-2-to-1-def)
qed

lemma \( \text{pr-comp1-3}' \): \([ f \in \text{PrimRec}' ; \text{pr-conv-3-to-1} g \in \text{PrimRec}' \] \implies \text{pr-conv-3-to-1} (\lambda x y z. f (g x y z)) \in \text{PrimRec}'

proof

assume \( f \in \text{PrimRec}' \)
and \( \text{pr-conv-3-to-1} g \in \text{PrimRec}' \) (is \(?g1 \in \text{PrimRec}'\))
then have \( (\lambda x. f \ (?g1 x)) \in \text{PrimRec}' \) by (rule PrimRec',comp)
then show \(?\text{thesis}\) by (simp add: pr-conv-3-to-1-def)
qed

lemma \( \text{pr-comp2-2}' \): \([ \text{pr-conv-2-to-1} f \in \text{PrimRec}' ; \text{pr-conv-2-to-1} g \in \text{PrimRec}' ; \text{pr-conv-2-to-1} h \in \text{PrimRec}' \] \implies \text{pr-conv-2-to-1} (\lambda x y. f (g x y) (h x y)) \in \text{PrimRec}'

proof

assume \( \text{pr-conv-2-to-1} f \in \text{PrimRec}' \)
and \( \text{pr-conv-2-to-1} g \in \text{PrimRec}' \) (is \(?g1 \in \text{PrimRec}'\))
and \( \text{pr-conv-2-to-1} h \in \text{PrimRec}' \) (is \(?h1 \in \text{PrimRec}'\))
then have \( (\lambda x. f \ (?g1 x) \ (?h1 x)) \in \text{PrimRec}' \) by (rule pr-comp2-1')
then show \(?\text{thesis}\) by (simp add: pr-conv-2-to-1-def)
qed

lemma \( \text{pr-comp2-3}' \): \([ \text{pr-conv-2-to-1} f \in \text{PrimRec}' ; \text{pr-conv-3-to-1} g \in \text{PrimRec}' ; \text{pr-conv-3-to-1} h \in \text{PrimRec}' \] \implies \text{pr-conv-3-to-1} (\lambda x y z. f (g x y z) (h x y z)) \in \text{PrimRec}'

proof

assume \( \text{pr-conv-2-to-1} f \in \text{PrimRec}' \)
and \( \text{pr-conv-3-to-1} g \in \text{PrimRec}' \) (is \(?g1 \in \text{PrimRec}'\))
and \( \text{pr-conv-3-to-1} h \in \text{PrimRec}' \) (is \(?h1 \in \text{PrimRec}'\))
then have \( (\lambda x. f \ (?g1 x) \ (?h1 x)) \in \text{PrimRec}' \) by (rule pr-comp2-1')
then show \(?\text{thesis}\) by (simp add: pr-conv-3-to-1-def)
qed

lemma \( \text{pr-comp3-2}' \): \([ \text{pr-conv-3-to-1} f \in \text{PrimRec}' ; \text{pr-conv-2-to-1} g \in \text{PrimRec}' ; \text{pr-conv-2-to-1} h \in \text{PrimRec}' ; \text{pr-conv-2-to-1} k \in \text{PrimRec}' \] \implies \text{pr-conv-2-to-1} (\lambda x y z. f (g x y z) (h x y z) (k x y z)) \in \text{PrimRec}'

proof

assume \( \text{pr-conv-3-to-1} f \in \text{PrimRec}' \)
and \( \text{pr-conv-2-to-1} g \in \text{PrimRec}' \) (is \(?g1 \in \text{PrimRec}'\))
and \( \text{pr-conv-2-to-1} h \in \text{PrimRec}' \) (is \(?h1 \in \text{PrimRec}'\))
and \( \text{pr-conv-2-to-1} k \in \text{PrimRec}' \) (is \(?k1 \in \text{PrimRec}'\))
then have \( (\lambda x. f \ (?g1 x) \ (?h1 x) \ (?k1 x)) \in \text{PrimRec}' \) by (rule pr-comp3-1')

qed
then show \( \text{thesis} \) by (simp add: pr-conv-2-to-1-def)

qed

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

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

lemma \( \text{ln}' \): \( (f1 \in \text{PrimRec1} \implies f1 \in \text{PrimRec1}) \land (g1 \in \text{PrimRec2} \implies \text{pr-conv-2-to-1 } g1 \in \text{PrimRec1}') \land (h1 \in \text{PrimRec3} \implies \text{pr-conv-3-to-1 } h1 \in \text{PrimRec1}')

proof (induct rule: PrimRec1-PrimRec2-PrimRec3_induct)
  case zero show \( \text{thesis} \) by (rule PrimRec1'.zero)
next case suc show \( \text{thesis} \) by (rule PrimRec1'.suc)
next case id1-1 show \( \text{thesis} \) by (rule pr-id1-1')
next case id2-1 show \( \text{thesis} \) by (rule pr-id2-1')
next case id2-2 show \( \text{thesis} \) by (rule pr-id2-2')
next case id3-1 show \( \text{thesis} \) by (rule pr-id3-1')
next case id3-2 show \( \text{thesis} \) by (rule pr-id3-2')
next case id3-3 show \( \text{thesis} \) by (rule pr-id3-3')
next case comp1-1 from comp1-1 show \( \text{thesis} \) by (simp add: PrimRec1'.comp)
next case comp1-2 from comp1-2 show \( \text{thesis} \) by (simp add: pr-comp1-2')
next case comp1-3 from comp1-3 show \( \text{thesis} \) by (simp add: pr-comp1-3')
next case comp2-1 from comp2-1 show \( \text{thesis} \) by (simp add: pr-comp2-1')
next case comp2-2 from comp2-2 show \( \text{thesis} \) by (simp add: pr-comp2-2')
next case comp2-3 from comp2-3 show \( \text{thesis} \) by (simp add: pr-comp2-3')
next case comp3-1 from comp3-1 show \( \text{thesis} \) by (simp add: pr-comp3-1')
next case comp3-2 from comp3-2 show \( \text{thesis} \) by (simp add: pr-comp3-2')
next case comp3-3 from comp3-3 show \( \text{thesis} \) by (simp add: pr-comp3-3')
next case prim-rec
  fix \( g \) \( h \) assume \( A1: g \in \text{PrimRec1}' \) and \( \text{pr-conv-3-to-1 } h \in \text{PrimRec1}' \)
  then have \( \text{UnaryRecOp } g \text{ (pr-conv-3-to-1 } h \text{ ) } \in \text{PrimRec1}' \) by (rule PrimRec1'.un-rec)
  moreover have \( \text{UnaryRecOp } g \text{ (pr-conv-3-to-1 } h \text{ ) } = \text{pr-conv-2-to-1 } (\text{PrimRecOp } g \text{ } h \text{ )} \) by (simp add: UnaryRecOp-def)
  ultimately show \( \text{pr-conv-2-to-1 } (\text{PrimRecOp } g \text{ } h \text{ ) } \in \text{PrimRec1}' \) by simp
qed

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

proof -
  have \( S1: \forall f. f \in \text{PrimRec1 } 
  \implies f \in \text{PrimRec1}' \) by (simp add: \( \text{lm}' \))
  have \( S2: \forall f. f \in \text{PrimRec1}' 
  \implies f \in \text{PrimRec1} \) by (simp add: \( \text{primrec}'-into-primrec \))
  from \( S1 \) \( S2 \) show \( \text{thesis} \) by blast
qed
4.2 The scheme datatype

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

primrec

sch-to-pr :: PrimScheme ⇒ (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'.intros)

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 S1 S2 have (λ x. f1 (f2 x)) = sch-to-pr (Comp-op sch1 sch2) by simp
then show ∃ sch. (λx. f1 (f2 x)) = 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 S1 S2 have c-f-pair f1 f2 = sch-to-pr (Pair-op sch1 sch2) by simp
then show \( \exists \textit{sch}. \textit{c-f-pair} f1 f2 = \textit{sch-to-pr} \textit{sch} \) by (rule exI)

next

fix \( f1 f2 \) assume \( B1: \exists \textit{sch}. \textit{f1} = \textit{sch-to-pr} \textit{sch} \) and \( B2: \exists \textit{sch}. \textit{f2} = \textit{sch-to-pr} \textit{sch} \)

from \( B1 \) obtain \( \textit{sch1} \) where \( S1: \textit{f1} = \textit{sch-to-pr} \textit{sch1} \) ..

from \( B2 \) obtain \( \textit{sch2} \) where \( S2: \textit{f2} = \textit{sch-to-pr} \textit{sch2} \) ..

from \( S1 \ S2 \) have \( \textit{UnaryRecOp} \textit{f1} \textit{f2} = \textit{sch-to-pr} (\textit{Rec-op} \textit{sch1} \textit{sch2}) \) by simp

then show \( \exists \textit{sch}. \textit{UnaryRecOp} \textit{f1} \textit{f2} = \textit{sch-to-pr} \textit{sch} \) by (rule exI)

qed

definition

\[
\text{loc-f} :: \text{n} \Rightarrow \text{PrimScheme} \Rightarrow \text{PrimScheme} \Rightarrow \text{PrimScheme} \text{ where } \\
\text{loc-f } n \text{ sch1 sch2 } = \\
\begin{align*}
\text{if } n & = 0 \text{ then Base-zero else} \\
\text{if } n & = 1 \text{ then Base-suc else} \\
\text{if } n & = 2 \text{ then Base-fst else} \\
\text{if } n & = 3 \text{ then Base-snd else} \\
\text{if } n & = 4 \text{ then (Comp-op sch1 sch2) else} \\
\text{if } n & = 5 \text{ then (Pair-op sch1 sch2) else} \\
\text{if } n & = 6 \text{ then (Rec-op sch1 sch2) else} \\
\text{Base-zero}
\end{align*}
\]

definition

\[
\text{mod7} :: \text{n} \Rightarrow \text{n} \text{ where } \\
\text{mod7 } = (\lambda x. x \text{ mod } 7)
\]

lemma \text{c-snd-snd-lt [termination-simp]}: \text{c-snd} (\text{c-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 \( \text{c-snd } ?y < ?y \) by (rule \text{c-snd-less-arg})

moreover have \( \text{c-snd} (\text{c-snd } ?y) \leq \text{c-snd } ?y \) by (rule \text{c-snd-le-arg})

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

qed

lemma \text{c-fst-snd-lt [termination-simp]}: \text{c-fst} (\text{c-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 \( \text{c-snd } ?y < ?y \) by (rule \text{c-snd-less-arg})

moreover have \( \text{c-fst} (\text{c-snd } ?y) \leq \text{c-snd } ?y \) by (rule \text{c-fst-le-arg})

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

qed

fun \text{nat-to-sch} :: \text{n} \Rightarrow \text{PrimScheme} \text{ where } \\
\text{nat-to-sch } 0 = \text{Base-zero} \\
| \text{nat-to-sch} (\text{Suc } 0) = \text{Base-zero}
nat-to-sch \( x \) = (let \( u = \text{mod7} \ (\text{c-fst} \ x) \); \( v = \text{c-snd} \ x \); \( v_1 = \text{c-fst} \ v \); \( v_2 = \text{c-snd} \ v \); \( \text{sch1} = \text{nat-to-sch} \ v_1 \); \( \text{sch2} = \text{nat-to-sch} \ v_2 \) in loc-f \( u \) \( \text{sch1} \) \( \text{sch2} \))

primrec \( \text{sch-to-nat} :: \text{PrimScheme} \Rightarrow \text{nat} \) where

\( \text{sch-to-nat Base-zero} = 0 \)
\( \text{sch-to-nat Base-suc} = \text{c-pair} \ 1 \ 0 \)
\( \text{sch-to-nat Base-fst} = \text{c-pair} \ 2 \ 0 \)
\( \text{sch-to-nat Base-snd} = \text{c-pair} \ 3 \ 0 \)
\( \text{sch-to-nat (Comp-op \( t_1 \) \( t_2 \))} = \text{c-pair} \ 4 \ (\text{c-pair} \ (\text{sch-to-nat} \ t_1) \ (\text{sch-to-nat} \ t_2)) \)
\( \text{sch-to-nat (Pair-op \( t_1 \) \( t_2 \))} = \text{c-pair} \ 5 \ (\text{c-pair} \ (\text{sch-to-nat} \ t_1) \ (\text{sch-to-nat} \ t_2)) \)
\( \text{sch-to-nat (Rec-op \( t_1 \) \( t_2 \))} = \text{c-pair} \ 6 \ (\text{c-pair} \ (\text{sch-to-nat} \ t_1) \ (\text{sch-to-nat} \ t_2)) \)

lemma \( \text{loc-srj-lm-1} : \text{nat-to-sch} \ (\text{Suc} \ (\text{Suc} \ x)) = (\text{let} \ u = \text{mod7} \ (\text{c-fst} \ (\text{Suc} \ (\text{Suc} \ x)))\); \( v = \text{c-snd} \ (\text{Suc} \ (\text{Suc} \ x))\); \( v_1 = \text{c-fst} \ v \); \( v_2 = \text{c-snd} \ v \); \( \text{sch1} = \text{nat-to-sch} \ v_1 \); \( \text{sch2} = \text{nat-to-sch} \ v_2 \) in loc-f \( u \) \( \text{sch1} \) \( \text{sch2} \)) \) by simp

lemma \( \text{loc-srj-lm-2} ; x > 1 \Rightarrow \text{nat-to-sch} \ x = (\text{let} \ u = \text{mod7} \ (\text{c-fst} \ x)\); \( v = \text{c-snd} \ x\); \( v_1 = \text{c-fst} \ v \); \( v_2 = \text{c-snd} \ v \); \( \text{sch1} = \text{nat-to-sch} \ v_1 \); \( \text{sch2} = \text{nat-to-sch} \ v_2 \) in loc-f \( u \) \( \text{sch1} \) \( \text{sch2} \)) \)

proof –

assume \( A1 : x > 1 \)

let \(?y = x - (2::nat)\)

from \( A1 \) have \( S1 : x = \text{Suc} \ (\text{Suc} \ ?y) \) by arith

have \( S2 : \text{nat-to-sch} \ (\text{Suc} \ (\text{Suc} \ ?y)) = (\text{let} \ u = \text{mod7} \ (\text{c-fst} \ (\text{Suc} \ (\text{Suc} \ ?y)))\); \( v = \text{c-snd} \ (\text{Suc} \ (\text{Suc} \ ?y))\); \( v_1 = \text{c-fst} \ v \); \( v_2 = \text{c-snd} \ v \); \( \text{sch1} = \text{nat-to-sch} \ v_1 \); \( \text{sch2} = \text{nat-to-sch} \ v_2 \) in loc-f \( u \) \( \text{sch1} \) \( \text{sch2} \)) \) by \( \text{rule loc-srj-lm-1} \)

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

qed

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

proof –

let \(?y = \text{c-pair} \ 1 \ 0\)

have \( S1 : ?x = 2 \) by \( \text{simpl add: c-pair-def sf-def} \)

then have \( S2 : ?x = \text{Suc} \ (\text{Suc} \ 0) \) by simp

let \(?y = \text{Suc} \ (\text{Suc} \ 0)\)

have \( S3 : \text{nat-to-sch} \ ?y = (\text{let} \ u = \text{mod7} \ (\text{c-fst} \ ?y)\); \( v = \text{c-snd} \ ?y\); \( v_1 = \text{c-fst} \ v \); \( v_2 = \text{c-snd} \ v \); \( \text{sch1} = \text{nat-to-sch} \ v_1 \); \( \text{sch2} = \text{nat-to-sch} \ v_2 \) in loc-f \( u \) \( \text{sch1} \) \( \text{sch2} \)) \) (is - = \(?R) \)

by \( \text{rule loc-srj-lm-1} \)

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

proof –

from \( S2 \) have \( \text{c-fst} \ ?y = \text{c-fst} \ ?x \) by simp

then show \(?thesis \) 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 \(?thesis \) by simp

qed

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

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

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

lemma loc-srj-2: nat-to-sch (c-pair 3 0) = Base-snd
proof –
  let ?x = c-pair 3 0
  have 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 \( S_4 : \text{c-snd } ?x = 0 \) by simp
from \( S_3 \) have \( S_6 : \text{mod7 } (\text{c-fst } ?x) = 3 \) by (simp add: \text{mod7-def})
from \( S_3 \) \( S_4 \) \( S_6 \) have \( S_7 : ?R = \text{loc-f } 3 \text{ Base-zero Base-zero} \) by (simp add: \text{Let-def} \text{ c-fst-at-0 c-snd-at-0})
then have \( S_8 : ?R = \text{Base-snd} \) by (simp add: \text{loc-f-def})
with \( S_2 \) have \( S_{10} : \text{nat-to-sch } ?x = \text{Base-snd} \) by simp
from \( S_2 \) this show \( \text{?thesis} \) by simp
qed

lemma \text{loc-srj-3} : \[[ \text{nat-to-sch } (\text{sch-to-nat sch1}) = \text{sch1}; \text{nat-to-sch } (\text{sch-to-nat sch2}) = \text{sch2} ] \implies \text{nat-to-sch } (\text{c-pair } 4 \text{ (c-pair } (\text{sch-to-nat sch1}) (\text{sch-to-nat sch2}))) = \text{Comp-op sch1 sch2}\]
proof -
assume \( A_1 : \text{nat-to-sch } (\text{sch-to-nat sch1}) = \text{sch1} \)
assume \( A_2 : \text{nat-to-sch } (\text{sch-to-nat sch2}) = \text{sch2} \)
let \( ?x = \text{c-pair } 4 \text{ (c-pair } \text{sch-to-nat sch1} \text{) (sch-to-nat sch2)} \)
have \( S_1 : ?x > 1 \) by (simp add: \text{c-pair-def sf-def})
from \( S_1 \) have \( S_2 : \text{nat-to-sch } ?x = (\text{let } u = \text{mod7 } (\text{c-fst } ?x); v = \text{c-snd } ?x; v_1 = \text{c-fst} v; v_2 = \text{c-snd } v; \text{sch1=} \text{nat-to-sch } v_1; \text{sch2=} \text{nat-to-sch } v_2 \text{ in loc-f } u \text{ sch1 sch2} \) (is - = ?R) by (rule \text{loc-srj-lm-2})
have \( S_3 : \text{c-fst } ?x = 4 \) 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 \( S_5 : \text{mod7 } (\text{c-fst } ?x) = 4 \) by (simp add: \text{mod7-def})
from \( A_1 \) \( A_2 \) \( S_4 \) \( S_5 \) have \( ?R = \text{Comp-op sch1 sch2} \) by (simp add: \text{Let-def} \text{ c-fst-at-0 c-snd-at-0 loc-f-def})
with \( S_2 \) show \( \text{?thesis} \) by simp
qed

lemma \text{loc-srj-3-1} : \[ \text{nat-to-sch } (\text{c-pair } 4 \text{ (c-pair } n_1 n_2)) = \text{Comp-op } (\text{nat-to-sch } n_1) (\text{nat-to-sch } n_2) \]
proof -
let \( ?x = \text{c-pair } 4 \text{ (c-pair } n_1 n_2) \)
have \( S_1 : \text{c-fst } ?x = 4 \) by simp
from \( S_1 \) have \( S_2 : \text{nat-to-sch } ?x = \text{let } u = \text{mod7 } (\text{c-fst } ?x); v = \text{c-snd } ?x; v_1 = \text{c-fst} v; v_2 = \text{c-snd } v; \text{sch1=} \text{nat-to-sch } v_1; \text{sch2=} \text{nat-to-sch } v_2 \text{ in loc-f } u \text{ sch1 sch2} \) (is - = ?R) by (rule \text{loc-srj-lm-2})
have \( S_3 : \text{c-fst } ?x = 4 \) by simp
have \( S_4 : \text{c-snd } ?x = \text{c-pair } n_1 n_2 \) by simp
from \( S_3 \) have \( S_5 : \text{mod7 } (\text{c-fst } ?x) = 4 \) by (simp add: \text{mod7-def})
from \( S_1 \) \( S_4 \) \( S_5 \) have \( ?R = \text{Comp-op } (\text{nat-to-sch } n_1) (\text{nat-to-sch } n_2) \) by (simp add: \text{Let-def} \text{ c-fst-at-0 c-snd-at-0 loc-f-def})
with \( S_2 \) show \( \text{?thesis} \) by simp
qed

lemma \text{loc-srj-4} : \[ \text{nat-to-sch } (\text{sch-to-nat sch1}) = \text{sch1}; \text{nat-to-sch } (\text{sch-to-nat sch2}) = \text{sch2} \]
implies \( \text{nat-to-sch } (\text{c-pair } 5 \text{ (c-pair } (\text{sch-to-nat sch1}) (\text{sch-to-nat sch2}))) = \text{Pair-op sch1 sch2} \]
proof –
assume $A1$: nat-to-sch $(\text{sch-to-nat sch1}) = \text{sch1}$
assume $A2$: nat-to-sch $(\text{sch-to-nat sch2}) = \text{sch2}$
let $\exists x = \text{c-pair} 5$ $(\text{c-pair} (\text{sch-to-nat sch1}) (\text{sch-to-nat sch2}))$
have $S1$: $\exists x > 1$ by (simp add: c-pair-def sf-def)
from $S1$ have $S2$: nat-to-sch $?x = (\text{let } u = \text{mod7} (c-fst ?x); v = c-snd ?x; v1 = c-fst v; v2 = c-snd v; sch1 = \text{nat-to-sch} v1; \text{sch2} = \text{nat-to-sch} v2 \text{ in } \text{loc-f u sch1 sch2})$ (is - = $?R$) by (rule loc-srj-lm-2)
have $S3$: c-fst $?x = 5$ by simp
have $S4$: c-snd $?x = \text{c-pair} (\text{sch-to-nat sch1}) (\text{sch-to-nat sch2})$ by simp
from $S3$ have $S5$: mod7 (c-fst $?x) = 5$ by (simp add: mod7-def)
from $A1 A2 S4 S5$ have $?R = \text{Pair-op sch1 sch2}$ 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 $(\text{c-pair} 5 (\text{c-pair n1 n2})) = \text{Pair-op (nat-to-sch n1) (nat-to-sch n2)}$
proof –
let $\exists x = \text{c-pair} 5$ $(\text{c-pair n1 n2})$
have $S1$: $\exists x > 1$ by (simp add: c-pair-def sf-def)
from $S1$ have $S2$: nat-to-sch $?x = (\text{let } u = \text{mod7} (c-fst ?x); v = c-snd ?x; v1 = c-fst v; v2 = c-snd v; sch1 = \text{nat-to-sch} v1; \text{sch2} = \text{nat-to-sch} v2 \text{ in } \text{loc-f u sch1 sch2})$ (is - = $?R$) by (rule loc-srj-lm-2)
have $S3$: c-fst $?x = 5$ by simp
have $S4$: c-snd $?x = \text{c-pair} n1 n2$ by simp
from $S3$ have $S5$: mod7 (c-fst $?x) = 5$ by (simp add: mod7-def)
from $S4 S5$ have $?R = \text{Pair-op (nat-to-sch n1) (nat-to-sch n2)}$ by (simp add: Let-def c-fst-at-0 c-snd-at-0 loc-f-def)
with $S2$ show ?thesis by simp
qed

lemma loc-srj-5: $[(\text{nat-to-sch (sch-to-nat sch1)}) = \text{sch1}; (\text{nat-to-sch (sch-to-nat sch2)}) = \text{sch2}]$
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 $(\text{sch-to-nat sch1}) = \text{sch1}$
assume $A2$: nat-to-sch $(\text{sch-to-nat sch2}) = \text{sch2}$
let $\exists x = \text{c-pair} 6$ $(\text{c-pair (sch-to-nat sch1) (sch-to-nat sch2)})$
have $S1$: $\exists x > 1$ by (simp add: c-pair-def sf-def)
from $S1$ have $S2$: nat-to-sch $?x = (\text{let } u = \text{mod7} (c-fst ?x); v = c-snd ?x; v1 = c-fst v; v2 = c-snd v; sch1 = \text{nat-to-sch} v1; \text{sch2} = \text{nat-to-sch} v2 \text{ in } \text{loc-f u sch1 sch2})$ (is - = $?R$) by (rule loc-srj-lm-2)
have $S3$: c-fst $?x = 6$ by simp
have $S4$: c-snd $?x = \text{c-pair (sch-to-nat sch1) (sch-to-nat sch2)}$ by simp
from $S3$ have $S5$: mod7 (c-fst $?x) = 6$ by (simp add: mod7-def)
from $A1 A2 S4 S5$ have $?R = \text{Rec-op sch1 sch2}$ 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-5-1: nat-to-sch (c-pair 6 (c-pair n1 n2)) = Rec-op (nat-to-sch n1) (nat-to-sch n2)
proof
  let ?x = c-pair 6 (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 = 6 by simp
  have S4: c-snd ?x = c-pair n1 n2 by simp
  from S3 S4 have S5: mod7 (c-fst ?x) = 6 by (simp add: mod7-def)
  from S4 S5 have ?R = Rec-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

theorem nat-to-sch-srj: nat-to-sch (sch-to-nat sch) = 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 nat-to-pr :: nat ⇒ (nat ⇒ nat) where
  nat-to-pr = (λ x. sch-to-pr (nat-to-sch x))

theorem nat-to-pr-into-pr: nat-to-pr n ∈ PrimRec1 by (simp add: nat-to-pr-def sch-to-pr-into-pr)

lemma nat-to-pr-srj: f ∈ PrimRec1 ⇒ (∃ n. f = nat-to-pr n)
proof
  assume f ∈ PrimRec1
  then have S1: (∃ t. f = sch-to-pr t) by (rule sch-to-pr-srj)
  from S1 obtain t where S2: f = sch-to-pr t ..
  let ?n = sch-to-nat t
  have S3: nat-to-pr ?n = sch-to-pr (nat-to-sch ?n) by (simp add: nat-to-pr-def)
  have S4: nat-to-sch ?n = t by (rule nat-to-sch-srj)
  from S3 S4 have S5: nat-to-pr ?n = sch-to-pr t by simp
  from S2 S5 have nat-to-pr ?n = f by simp
  then have f = nat-to-pr ?n by simp
  then show ?thesis ..
qed
lemma nat-to-pr-at-0: nat-to-pr 0 = (λ x. 0) by (simp add: nat-to-pr-def)

definition
index-of-pr :: (nat ⇒ nat) ⇒ nat where
index-of-pr f = (SOME n. f = nat-to-pr n)

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

definition
comp-by-index :: nat ⇒ nat ⇒ nat where
comp-by-index = (λ n1 n2. c-pair 4 (c-pair n1 n2))

definition
pair-by-index :: nat ⇒ nat ⇒ nat where
pair-by-index = (λ n1 n2. c-pair 5 (c-pair n1 n2))

definition
rec-by-index :: nat ⇒ nat ⇒ nat where
rec-by-index = (λ n1 n2. c-pair 6 (c-pair n1 n2))

lemma comp-by-index-is-pr: comp-by-index ∈ PrimRec2
unfolding comp-by-index-def
using const-is-pr-2 [of 4] by prec

lemma comp-by-index-inj: comp-by-index x1 y1 = comp-by-index x2 y2 ⇒ x1=x2 ∧ y1=y2
proof –
assume comp-by-index x1 y1 = comp-by-index x2 y2
hence c-pair 4 (c-pair x1 y1) = c-pair 4 (c-pair x2 y2) by (unfold comp-by-index-def)
hence c-pair x1 y1 = c-pair x2 y2 by (rule c-pair-inj2)
thus ?thesis by (rule c-pair-inj)
qed

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

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

lemma comp-by-index-main: nat-to-pr (comp-by-index n1 n2) = (λ x. (nat-to-pr n1) ((nat-to-pr n2) x))
by (unfold comp-by-index-def, unfold nat-to-pr-def, simp add: loc-srj-3-1)
lemma pair-by-index-is-pr: pair-by-index ∈ PrimRec2 by (unfold pair-by-index-def, insert const-is-pr-2 [where ?n=(5::nat)], prec)

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

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

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

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

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

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

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

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

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

lemma 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 ⇒ nat where
  index-of-const = PrimRecOp1 0 (λ x y. c-pair 4 (c-pair 2 y))

lemma index-of-const-is-pr: index-of-const ∈ PrimRec1
proof –
  have (λ x y. c-pair (4 :: nat) (c-pair (2 :: nat) y)) ∈ 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) = (λ 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 ⇒ 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

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

definition index-of-c-pair-n :: nat \Rightarrow nat where
index-of-c-pair-n = (\lambda n. pair-by-index (index-of-const n) index-of-id)

lemma index-of-c-pair-n-is-pr: index-of-c-pair-n \in PrimRec1
proof -
  have (\lambda x. index-of-id) \in PrimRec1 by (rule const-is-pr)
  with pair-by-index-is-pr index-of-const-is-pr have (\lambda n. pair-by-index (index-of-const n) index-of-id) \in 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) = (\lambda 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 \dots = 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 \dots = c-f-pair (\lambda x. n) (\lambda 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 \Rightarrow 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-inj1)
  thus ?thesis by (rule index-of-const-inj)
qed

definition
s1-1 :: nat ⇒ nat ⇒ nat where
s1-1 = (λ n x. comp-by-index n (index-of-c-pair-n x))

lemma s1-1-is-pr: s1-1 ∈ PrimRec2 by (unfold s1-1-def, insert comp-by-index-is-pr index-of-c-pair-n-is-pr, prec)

theorem s1-1-th: (λ y. (nat-to-pr n) (c-pair x y)) = nat-to-pr (s1-1 n x)
proof –
have nat-to-pr (s1-1 n x) = nat-to-pr (comp-by-index n (index-of-c-pair-n x))
  by (simp add: s1-1-def)
also have ... = (λ z. (nat-to-pr n) ((nat-to-pr (index-of-c-pair-n x)) z)) by
  (simp add: comp-by-index-main)
also have ... = (λ z. (nat-to-pr n) ((λ u. c-pair x u) z)) by
  (simp add: index-of-c-pair-n-main)
finally show ?thesis by simp
qed

lemma s1-1-inj: s1-1 x1 y1 = s1-1 x2 y2 =⇒ x1=x2 ∧ y1=y2
proof –
assume s1-1 x1 y1 = s1-1 x2 y2
then have comp-by-index x1 (index-of-c-pair-n y1) = comp-by-index x2 (index-of-c-pair-n y2)
  by (simp add: s1-1-def)
then have S1: x1=x2 ∧ index-of-c-pair-n y1 = index-of-c-pair-n y2 by
  (rule comp-by-index-inj)
then have S2: x1=x2 ..
from S1 have index-of-c-pair-n y1 = index-of-c-pair-n y2 ..
then have y1 = y2 by
  (rule index-of-c-pair-n-inj)
with S2 show ?thesis ..
qed

lemma s1-1-inj1: s1-1 x1 y1 = s1-1 x2 y2 =⇒ x1=x2 by (frule s1-1-inj, drule conjunct1)

lemma s1-1-inj2: s1-1 x1 y1 = s1-1 x2 y2 =⇒ y1=y2 by (frule s1-1-inj, drule conjunct2)

primrec
pr-index-enumerator :: nat ⇒ nat ⇒ 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

71
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
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 (rule c-pair-strict-mono2)
by (simp add: comp-by-index-def)
  then have comp-by-index index-of-id ?a < c-pair 4 (c-pair index-of-id ?b) by (simp add: comp-by-index-def)
  then have comp-by-index index-of-id ?a < comp-by-index index-of-id ?b by (simp add: c-pair-strict-mono2)
  with S1 S2 show ?thesis by auto
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 (simp add: comp-by-index-def)
have S3: 4 + c-pair index-of-id ?a ≤ c-pair 4 (c-pair index-of-id ?a) by (rule 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 ≤ 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 f-inc-mono: \( (\forall (x::\text{nat}). (f::\text{nat} \Rightarrow \text{nat}) \ x < f (x+1)) \implies (\forall (y::\text{nat}). (y::\text{nat})). (x < y \implies f x < f y) \)
proof (rule allI, rule allI)
fix x y assume A: \( (\forall (x::\text{nat}). (f::\text{nat} \Rightarrow \text{nat}) \ x < f (x+1)) \implies (\forall (y::\text{nat}). (y::\text{nat})). (x < y \implies f x < f y) \)
proof
assume \( A1: x < y \)
have \( L1: \forall u v. f u < f (u + (v+1)) \)
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 \( S1: f (u + (n + 1)) < f (u + (\text{Suc} n + 1)) \) by auto
from A2 S1 show \( f u < f (u + (\text{Suc} n + 1)) \) by (rule less-trans)
qed
qed
let \( ?v = (y - x) - 1 \)
from A1 have \( S2: y = x + (?v + 1) \) by auto
have \( f x < f (x + (?v + 1)) \) by (rule L1)
with S2 show \( f x < f y \) by auto
qed
qed

lemma pr-index-enumerator-mono1: \( n1 < n2 \implies \text{pr-index-enumerator} n1 m < \text{pr-index-enumerator} n2 m \)
proof –
assume A: \( n1 < n2 \)
define f where \( f x = \text{pr-index-enumerator} x m \) for \( x \)
have f-inc: \( \forall x. f x < f (x+1) \)
proof
fix x show \( f x < f (x+1) \) by (unfold f-def, rule pr-index-enumerator-increase1)
qed
from f-inc have \( \forall x y. (x < y \implies f x < f y) \) by (rule f-inc-mono)
with A f-def show \( \text{thesis} \) by auto
qed

lemma pr-index-enumerator-mono2: \( m1 < m2 \implies \text{pr-index-enumerator} n m1 < \text{pr-index-enumerator} n m2 \)
proof –
assume A: \( m1 < m2 \)
define f where \( f x = \text{pr-index-enumerator} n x \) for \( x \)
have f-inc: \( \forall x. f x < f (x+1) \)
proof
fix x show \( f x < f (x+1) \) by (unfold f-def, rule pr-index-enumerator-increase2)
qed
from f-inc have \( \forall x y. (x < y \implies f x < f y) \) 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 m1 ⇒ n1 = n2
proof −
  assume A: pr-index-enumerator n1 m = pr-index-enumerator n2 m1
  define f where f x = pr-index-enumerator 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 = pr-index-enumerator n x 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-mono2)
  qed
  from f-mono have ∀ x y. (f x = f y → x = y) by (rule f-mono-inj)
  with A f-def show ?thesis by auto
  qed

74
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: \( \text{pr-index-enumerator} \ n \ (\text{Suc} \ na) = \text{comp-by-index} \ \text{index-of-id} \ ?a \) by simp
    - Have: \( \text{nat-to-pr} \ (\text{comp-by-index} \ \text{index-of-id} \ ?a) = (\lambda \ x. \ (\text{nat-to-pr} \ \text{index-of-id}) \ (\text{nat-to-pr} \ ?a \ x)) \) by (rule \( \text{comp-by-index-main} \))
    - With \( \text{index-of-id-main} \) have: \( \text{nat-to-pr} \ (\text{comp-by-index} \ \text{index-of-id} ?a) = \text{nat-to-pr} \ ?a \) 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: \( \text{c-in :: nat} \Rightarrow \text{nat} \Rightarrow \text{nat where} \)

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

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

Proof:

- From: \( \text{mod-is-pr power-is-pr div-is-pr} \) have: \( (\lambda \ x \ u. \ (u \ \text{div} \ (2 ^ x)) \ \text{mod} \ 2) \in \text{PrimRec2} \) by prec
  - With \( \text{c-in-def} \) show: \( ?\text{thesis} \) by auto

Qed

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

\( \text{nat-to-set} \ u \equiv \{ x. \ 2 ^ x \leq u \land \text{c-in} \ x \ u = 1 \} \)

Lemma: \( \text{c-in-upper-bound: } \text{c-in} \ x \ u = 1 \implies 2 ^ x \leq u \)

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

lemma nat-to-set-upper-bound: $x \in \text{nat-to-set } u \Rightarrow 2^\sim x \leq u$ by (simp add: nat-to-set-def)

lemma x-lt-2-x: $x < 2^\sim x$
  by (rule less-exp)

lemma nat-to-set-upper-bound1: $x \in \text{nat-to-set } u \Rightarrow x < u$
proof
  assume $x \in \text{nat-to-set } u$
  then have $S1: 2^\sim x \leq u$ by (simp add: nat-to-set-def)
  have $S2: x < 2^\sim x$ by (rule x-lt-2-x)
  from $S2$ $S1$ show ?thesis
    by (rule less-le-trans)
qed

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

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

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
\[ \log_2 : \text{nat} \to \text{nat} \]
\[ \log_2 = (\lambda x. \text{Least}(\%z. x < 2^z)) \]

**Lemma log2-at-0:** \( \log_2 0 = 0 \)

**Proof** -
- let \(?v = \log_2 0\) by auto
- 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 \(S3: (0::\text{nat}) < 2^{(0+1)}\) 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
- thus \(?v = 0\) by auto

**Qed**

**Lemma log2-at-1:** \( \log_2 1 = 0 \)

**Proof** -
- let \(?v = \log_2 1\) by auto
- 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 \(S3: (1::\text{nat}) < 2^{(0+1)}\) 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:** \( x > 0 \implies 2^\log_2 x \leq x \)

**Proof** -
- assume \(A: x > 0\)
- show \(?thesis\)
  - proof (cases)
    - assume \(A1: \log_2 x = 0\)
      - with \(A\) show \(?thesis\) by auto
    - next
      - assume \(A1: \log_2 x \neq 0\)
        - then have \(S1: \log_2 x > 0\) by auto
        - define \(y\) where \(y = \log_2 x - 1\)
        - from \(S1\) have \(S2: \log_2 x = y + 1\) by auto
        - then have \(S3: y < \log_2 x\) by auto
        - have \(2^{y+1} \leq x\)
          - proof (rule ccontr)
            - assume \(A2: \neg 2^{y+1} \leq x\) 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
    - qed

**Qed**
lemma $\log_2 \text{-} gt: x < 2 \wedge (\log_2 x + 1)$
proof
  have $x < 2^x$ by (rule $x$-lt-2-x)
  then have $S1: x < 2^{(x+1)}$
    by (simp add: numeral-2-eq-2)
define $y$ where $y = x$
from $S1$ y-def have $S2: x < 2^{y+1}$ by auto
let $?P = \lambda z. x < 2^{z+1}$
from $S2$ have $S3: ?P y$ by auto
then have $S4: ?P (\text{Least } ?P)$ by (rule LeastI)
from log2-def have $S5: \log_2 x = \text{Least } ?P$ by (unfold log2-def, auto)
from $S4$ $S5$ show $?\text{thesis}$ by auto
qed

lemma $x$-$\text{-} \div$-$x: x > 0 \implies (x::nat) \div x = 1$ by auto
lemma div-ge: $(k::nat) \leq m \div n \implies n*k \leq m$
proof
  assume $A: k \leq m \div n$
  have $S1: n * (m \div n) + m \mod n = m$ by (rule $\text{mult-div-mod-eq}$)
  have $S2: 0 \leq m \mod n$ by auto
  from $S1$ $S2$ have $S3: n * (m \div n) \leq m$ by arith
  from $A$ have $S4: n * k \leq n * (m \div n)$ by auto
  from $S4$ $S3$ show $?\text{thesis}$ by (rule order-trans)
qed

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

lemma $\log_2$-$\text{lm1}: u > 0 \implies u \div 2 \wedge (\log_2 u) = 1$
proof
  assume $A: u > 0$
  then have $S1: 2^{\wedge (\log_2 u)} \leq u$ by (rule log2-le)
  have $S2: u < 2^{\wedge (\log_2 u+1)}$ by (rule log2-gt)
  then have $S3: u < (2^{\wedge \log_2 u})*2$ by simp
  have $(2::nat) > 0$ by simp
  then have $(2::nat)^{\wedge}\log_2 u > 0$ by simp
  then have $S4: (2::nat)^{\wedge}\log_2 u \div 2^{\wedge}\log_2 u = 1$ by auto
  from $S1$ have $S5: (2::nat)^{\wedge}\log_2 u \div 2^{\wedge}\log_2 u \leq u \div 2^{\wedge}\log_2 u$ by (rule div-le-mono)
  with $S4$ have $S6: 1 \leq u \div 2^{\wedge}\log_2 u$ by auto
\textbf{lemma} \texttt{log2-lm2}: \(u > 0 \implies \text{c-in} (\log_2 u) u = 1\)

\textbf{proof} –
\begin{itemize}
  \item \textbf{assume} \(A: u > 0\)
  \item \textbf{then have} \(S1: u \text{ div } 2 \wedge (\log_2 u) = 1\) \textbf{by} \texttt{rule log2-lm1}
  \item \textbf{have} \(\text{c-in} (\log_2 u) u = (u \text{ div } 2 \wedge (\log_2 u)) \mod 2\) \textbf{by} \texttt{simp add: c-in-def}
  \item \textbf{also from} \(S1\) \textbf{have} \ldots \textbf{=} \text{1 mod 2} \textbf{by} \texttt{simp}
  \item \textbf{also have} \ldots \textbf{=} \text{1} \textbf{by} \texttt{auto}
\end{itemize}
\textbf{finally show} \?thesis \textbf{by} \texttt{auto}

\textbf{qed}

\textbf{lemma} \texttt{log2-lm3}: \(\log_2 u < x \implies \text{c-in} x u = 0\)

\textbf{proof} –
\begin{itemize}
  \item \textbf{assume} \(A: \log_2 u < x\)
  \item \textbf{then have} \(S1: (\log_2 u)+1 \leq x\) \textbf{by} \texttt{auto}
  \item \textbf{have} \(S2: 1 \leq (2::nat)\) \textbf{by} \texttt{auto}
  \item \textbf{from} \(S1\) \textbf{and} \(S2\) \textbf{have} \(S3: (2::nat)^{\wedge} ((\log_2 u)+1) \leq 2^x\) \textbf{by} \texttt{rule power-increasing}
  \item \textbf{have} \(S4: u < (2::nat)^{\wedge} ((\log_2 u)+1)\) \textbf{by} \texttt{rule log2-gl}
  \item \textbf{from} \(S3\) \textbf{and} \(S4\) \textbf{have} \(S5: u < 2^x\) \textbf{by} \texttt{auto}
  \item \textbf{then have} \(S6: u \text{ div } 2^x = 0\) \textbf{by} \texttt{rule div-less}
  \item \textbf{have} \(\text{c-in} x u = (u \text{ div } 2^x) \mod 2\) \textbf{by} \texttt{simp add: c-in-def}
  \item \textbf{also from} \(S6\) \textbf{have} \ldots \textbf{=} \text{0 mod 2} \textbf{by} \texttt{simp}
  \item \textbf{also have} \ldots \textbf{=} \text{0} \textbf{by} \texttt{auto}
  \item \textbf{finally have} \?thesis \textbf{by} \texttt{auto}
  \item \textbf{thus} \?thesis \textbf{by} \texttt{auto}
\end{itemize}

\textbf{qed}

\textbf{lemma} \texttt{log2-lm4}: \(\text{c-in} x u = 1 \implies x \leq \log_2 u\)

\textbf{proof} –
\begin{itemize}
  \item \textbf{assume} \(A: \text{c-in} x u = 1\)
  \item \textbf{show} \?thesis \textbf{proof} \texttt{(rule ccontr)}
    \item \textbf{assume} \(\neg x \leq \log_2 u\)
    \item \textbf{then have} \(S1: \log_2 u < x\) \textbf{by} \texttt{auto}
    \item \textbf{then have} \(S2: \text{c-in} x u = 0\) \textbf{by} \texttt{rule log2-lm3}
    \item \textbf{with} \(A\) \textbf{show} \texttt{False} \textbf{by} \texttt{auto}
\end{itemize}

\textbf{qed}

\textbf{qed}

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

\textbf{proof} –
\begin{itemize}
  \item \textbf{assume} \(x \in \text{nat-to-set} u\)
  \item \textbf{then have} \(S1: \text{c-in} x u = 1\) \textbf{by} \texttt{simp add: x-in-u-eq}
  \item \textbf{then show} \?thesis \textbf{by} \texttt{rule log2-lm4}
\end{itemize}

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

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

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

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

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

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

80
**qed**

lemma \textit{div-mod-lm}: \( y < x \implies ((u + (2::nat) \cdot x) \div (2::nat) \cdot y) \mod 2 = (u \div (2::nat) \cdot y) \mod 2 \)

**proof** –
  assume \( y < x \)
  let \(?n = (2::nat) \cdot y\)
  have \( n \cdot pos \cdot 0 < ?n \) \textit{by auto}
  let \(?s = x - y\)
  from \( y \cdot lt \cdot x \) have \( s \cdot pos \cdot 0 < ?s \) \textit{by auto}
  from \( y \cdot lt \cdot x \) have \( S3: x = y + ?s \) \textit{by auto}
  from \( S3 \) have \((2::nat) \cdot x = (2::nat) \cdot (y + ?s) \) \textit{by auto}
  moreover have \((2::nat) \cdot (y + ?s) = (2::nat) \cdot y \ast 2^?s \) \textit{by (rule power-add)}
  ultimately have \((2::nat) \cdot x = 2^?y \ast 2^?s \) \textit{by auto}
  then have \( S4: u + (2::nat) \cdot x = u + (2::nat) \cdot y \ast 2^?s \) \textit{by auto}
  from \( n \cdot pos \) have \( S5: (u + (2::nat) \cdot y \ast 2^?s) \div 2^?y = 2^?s + (u \div 2^y) \) \textit{by simp}
  from \( S4 \) \( S5 \) have \( S6: (u + (2::nat) \cdot x) \div 2^?y = 2^?s + (u \div 2^y) \) \textit{by auto}
  from \( s \cdot pos \) have \( S8: ?s = (?s - 1) + 1 \) \textit{by auto}
  have \((2::nat) \cdot (?s - (1::nat)) + (1::nat)) = (2::nat) \cdot (?s - (1::nat)) \ast 2^?s \ast 2^1 \)
  \textit{by (rule power-add)}
  with \( S8 \) have \( S9: (2::nat) \cdot (?s - (1::nat)) \ast 2 \) \textit{by auto}
  then have \( S10: 2^?s + (u \div 2^y) = (u \div 2^y) + (2::nat) \cdot (\?s - (1::nat)) \)
  \ast 2 \textit{by auto}
  have \( S11: ((u \div 2^y) + (2::nat) \cdot (?s - (1::nat)) \ast 2) \mod 2 = (u \div 2^y) \mod 2 \) \textit{by (rule mod-mul-self1)}
  from \( S6 \) \( S10 \) \( S11 \) \textit{show ?thesis by auto}
\textbf{qed}

lemma \textit{add-power}: \( u < 2^x \implies \text{nat-to-set} (u + 2^x) = \text{nat-to-set} u \cup \{ x \} \)

**proof** –
  assume \( A: u < 2^x \)
  have \( \log2-is-x: \log2 (u + 2^x) = x \)
  **proof** (unfold \log2-def, rule \textit{Least-equality})
    from \( A \) \textit{show u + 2^x < 2^1(x+1) by auto}
  next
  fix \( z \)
  assume \( A1: u + 2^x < 2^1(z+1) \)
  show \( x \leq z \)
  **proof** (rule \textit{ccontr})
    assume \(?x \leq z\)
    then have \( z < x \) \textit{by auto}
    then have \( L1: z+1 \leq x \) \textit{by auto}
    have \( L2: z+1 \leq (2::nat) \) \textit{by auto}
    from \( L1 \) \( L2 \) have \( L3: (2::nat) \cdot (z+1) \leq (2::nat) \cdot x \) \textit{by (rule power-increasing)}
    with \( A1 \) \textit{show False by auto}
\textbf{qed}
\textbf{qed}
  show ?thesis
proof (rule subset-antisym)
  show nat-to-set (u + 2 ^ x) ⊆ nat-to-set u ∪ {x}
proof fix y
  assume A1: y ∈ nat-to-set (u + 2 ^ x)
  show y ∈ nat-to-set u ∪ {x}
proof
  assume y ∉ {x} then have S1: y ≠ x by auto
  from A1 have y ≤ log2 (u + 2 ^ x) by (rule nat-to-set-lub)
  with log2-is-x have y ≤ x by auto
  with S1 have y-lt-x: y < x by auto
  from A1 have c-in y (u + 2 ^ x) = 1 by (simp add: x-in-u-eq)
  then have S2: ((u + 2 ^ x) 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 div 2 ^ y) mod 2 = 1 by auto
  then have c-in y u = 1 by (simp add: c-in-def)
  then show ?thesis by (simp add: x-in-u-eq)
qed
next
show nat-to-set u ∪ {x} ⊆ nat-to-set (u + 2 ^ x)
proof fix y
  assume A1: y ∈ nat-to-set u ∪ {x}
  show y ∈ nat-to-set (u + 2 ^ x)
proof cases
  assume y ∈ {x}
  then have y=x by auto
  then have y = log2 (u + 2 ^ x) by (simp add: log2-is-x)
  then show ?thesis by (simp add: log2-lm5)
next
assume y-notin: y ∉ {x}
then have y-ne-x: y ≠ x by auto
from A1 y-notin have y-in: y ∈ nat-to-set u by auto
have y-lt-x: y < x
proof (rule ccontr)
  assume ¬ y < x
  with y-ne-x have y-gt-x: x < y by auto
  have 1 < (2::nat) by auto
  from y-gt-x this have L1: (2::nat) ^ x < 2 ^ y by (rule power-strict-increasing)
  from y-in have L2: 2 ^ y ≤ u by (rule nat-to-set-upper-bound)
  from L1 L2 have (2::nat) ^ x < u by arith
  with A show False by auto
qed
from y-in have c-in y u = 1 by (simp add: x-in-u-eq)
then have S2: (u div 2 ^ 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)
\textbf{then show} \( y \in \text{nat-to-set} \ (u + (2::\text{nat}) \sim x) \) \by (simp add: x-in-u-eq) 
qed

\textbf{qed}

\textbf{qed}

\textbf{qed}

\textbf{theorem} \text{nat-to-set-inj}: \text{nat-to-set} \ u = \text{nat-to-set} \ v \implies u = v

\textbf{proof} –

\textbf{assume} \( A: \text{nat-to-set} \ u = \text{nat-to-set} \ v \)

\textbf{let} \( ?P = \lambda (n::\text{nat}). (\forall (D::\text{nat set}). \text{finite} \ D \land \text{card} \ D \leq n \implies (\forall u v. \text{nat-to-set} \ u = D \land \text{nat-to-set} \ v = D \implies u = v)) \)

\textbf{have} \( P\text{-at-0}: \ ?P \ 0 \)

\textbf{proof} \( D \text{ show} \ \text{finite} \ D \land \text{card} \ D \leq 0 \implies (\forall u v. \text{nat-to-set} \ u = D \land \text{nat-to-set} \ v = D \implies u = v) \)

\textbf{proof} (rule \text{impI})

\textbf{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

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

\textbf{proof} (rule \text{allI}, rule \text{allI}) \( \text{fix} \ u v \text{ show} \ \text{nat-to-set} \ u = D \land \text{nat-to-set} \ v = D \implies u = v \)

\textbf{proof}

\textbf{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 \( \text{nat-to-set} \ u = \{\} \) \by auto

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

from \( L2 \ S3 \) have \( \text{nat-to-set} \ v = \{\} \) \by auto

then have \( v\text{-z}: v = 0 \) \by (rule \text{empty-is-zero})

from \( u\text{-z} \ v\text{-z} \) show \( u=v \) \by auto

\textbf{qed}

\textbf{qed}

\textbf{qed}

\textbf{have} \( P\text{-at-Suc}: \ \land \ n. \ ?P \ n \implies ?P \ (\text{Suc} \ n) \)

\textbf{proof} – \( \text{fix} \ n \)

\textbf{assume} \( A\text{-n}: ?P \ n \)

\textbf{show} \( ?P \ (\text{Suc} \ n) \)

\textbf{proof} \( D \text{ show} \ \text{finite} \ D \land \text{card} \ D \leq \text{Suc} \ n \implies (\forall u v. \text{nat-to-set} \ u = D \land \text{nat-to-set} \ v = D \implies u = v) \)

\textbf{proof} (rule \text{impI})

\textbf{assume} \( A1: \text{finite} \ D \land \text{card} \ D \leq \text{Suc} \ n \)

from \( A1 \) have \( S1: \text{finite} \ D \) \by auto

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

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

\textbf{proof} (rule \text{allI}, rule \text{allI}, rule \text{impl})

\textbf{fix} \( u v \)

\textbf{assume} \( A2: \text{nat-to-set} \ u = D \land \text{nat-to-set} \ v = D \)

83
from A2 have d-u-d: nat-to-set u = D by auto
from A2 have d-v-d: 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 ≠ {}
  from A3 d-u-d have nat-to-set u ≠ {} by auto
  then have u-pos: u > 0 by (rule ne-imp-pos)
  from A3 d-v-d have nat-to-set v ≠ {} by auto
  then have v-pos: v > 0 by (rule ne-imp-pos)
  define m where m = Max D
  from S1 m-def A3 have m-in: m ∈ D by auto
  from d-u-d m-def have m-u: m = Max (nat-to-set u) by auto
  from d-v-d m-def have m-v: m = Max (nat-to-set v) by auto
  from u-pos m-u log2-is-max have m-log-w: m = log2 u by auto
  from v-pos m-v log2-is-max have m-log-v: m = 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: card D1 ≤ n proof
    from D1-def S1 m-in have card D1 = (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 ^ log2 u ≤ u by (rule log2-le)
    with m-log-u have L2: 2 ^ m ≤ u by auto
    with u1-def show ?thesis by auto
  qed
  have u1-d1: 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: nat-to-set u = nat-to-set u1 ∪ {m} by (simp add: add-power)
    have m-notin: m ∉ nat-to-set u1
    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
  qed

84
from L1 m-notin have nat-to-set u1 = nat-to-set u - {m} by auto
with d-u-d have nat-to-set u1 = D - {m} by auto
with D1-def show ?thesis by auto
qed
have v-v1: v = v1 + 2^m
proof –
from v-pos have L1: 2 ^ log2 v ≤ v by (rule log2-le)
with m-log-v have L2: 2 ^ m ≤ v by auto
with v1-def show ?thesis by auto
qed
have v1-d1: nat-to-set v1 = D1
proof –
from m-log-v log2-gt have v < 2^(m+1) by auto
with v-v1 have v1-lt-2-m: v1 < 2^m by auto
with v1-d1 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 L1 m-notin have nat-to-set v1 = nat-to-set v - {m} by auto
with d-v-d have nat-to-set v1 = D - {m} by auto
with D1-def show ?thesis by auto
qed
from S1 D1-def have P1: finite D1 by auto
with card-D1 have P2: finite D1 ∧ card D1 ≤ n by auto
from A-n P2 have (∀ u 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
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
qed
definition
set-to-nat :: nat set => nat where
set-to-nat = (\n x. 2 ^ x \ D)

lemma two-power-sum: sum (\x. (2 :: nat) ^ x) \ i. i < Suc m = (2 ^ Suc m) - 1
proof (induct m)
  show sum (\x. (2 :: nat) ^ x) \ i. i < Suc 0 = (2 ^ Suc 0) - 1 by auto
next
  fix n
  assume  A: sum (\x. (2 :: nat) ^ x) \ i. i < Suc n = (2 ^ Suc n) - 1
  show sum (\x. (2 :: nat) ^ x) \ i. i < Suc (Suc n) = (2 ^ Suc (Suc n)) - 1
    proof -
      let \f = \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 \union \ Suc n \ by auto
      from \S1 \S2 have \S3: \ i. i < Suc (Suc n) = \ i. i < Suc n \union \ Suc n \ by auto
      have \S4: \ i. i < Suc n = \ (\ 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 \notin \ i. i < Suc n \ by auto
      from \S5 \S6 sum.insert have \S7: sum ?f \ i. i < Suc n \union \ Suc n = 2 ^ Suc
      n + sum ?f \ i. i < Suc n \ by auto
      from \S3 have sum ?f \ i. i < Suc (Suc n) = sum ?f \ i. i < Suc n \union \ Suc n \ by auto
      also from \S7 have \dots = 2 ^ Suc n + sum ?f \ i. i < Suc n \ by auto
      also from \A have \dots = 2 ^ Suc n + ((2 :: nat) ^ Suc n - Suc (Suc (Suc n)) \by auto
      also have \dots = (2 ^ Suc (Suc (Suc n)) - 1 \ by auto
      finally show \thesis by auto
    qed
  qed

lemma finite-interval: finite \ i. \ i :: nat < m \}
proof -
  have \ i. i < m = \ (\ 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 \} = \ by (unfold set-to-nat-def, rule sum.empty)

lemma set-to-nat-of-interval: set-to-nat \ i. \ i :: nat < m \ = 2 ^ m - 1
proof (induct m)
  show set-to-nat \ i. i < 0 \ = 2 ^ 0 - 1
    proof -
      have \S1: \ i. \ i :: nat < 0 \ = \ by auto
      with set-to-nat-at-empty have set-to-nat \ i. i < 0 \ = 0 by auto
      thus \thesis by auto
    qed
  qed
next
fix n show set-to-nat \{ i. i < Suc n \} = 2 \sim Suc n - 1 by (unfold set-to-nat-def, rule two-power-sum)
qed

lemma set-to-nat-mono: \[ \text{finite } B; A \subseteq B \] \implies set-to-nat A \leq set-to-nat B
proof –
assume b-finite: finite B
assume a-le-b: A \subseteq B
let \( \forall x. 2 \cdot x \in B - A \implies 0 \leq 2 \cdot x \) by auto
from b-finite a-le-b have sum \( \forall A \leq B \) by (rule sum-mono2)
with S1 S2 show \?thesis by auto
qed

theorem nat-to-set-srj: finite (D::nat set) \implies nat-to-set (set-to-nat D) = D
proof –
assume A: finite D
let \( \forall n. \exists D. \text{finite } D \land \text{card } D = n \implies \text{nat-to-set (set-to-nat D) = D} \)
have P-at-0: \?P 0
proof (rule allI)
fix D
show finite D \land \text{card } D = 0 \implies \text{nat-to-set (set-to-nat D) = D}
proof
assume A1: finite D \land \text{card } D = 0
from A1 have S1: finite D by auto
from A1 have S2: \text{card } D = 0 by auto
from S1 S2 have S3: D = \{\} by auto
with set-to-nat-def have set-to-nat D = \text{sum (} \lambda x. 2 \cdot x \text{) } D \text{ 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: \\land n. \?P n \implies \?P (Suc n)
proof – fix n
assume A-n: \?P n
show \?P (Suc n)
proof
fix D show finite D \land \text{card } D = Suc n \implies \text{nat-to-set (set-to-nat D) = D}
proof
assume A1: finite D \land \text{card } D = Suc n
from A1 have S1: \text{finite } D \text{ by auto}
from A1 have S2: \text{card } D = Suc n by auto
define m where m = Max D
from S2 have D-ne: D \neq \{\} by auto
with S1 m-def have m-in: m \in D by auto

87
define $D_1$ where $D_1 = D - \{m\}$
from $S_1$ $D_1$-def have $d_1$-finite: finite $D_1$ by auto

from $D_1$-def $m$-in $S_1$ have $d_1$-finite: finite $D_1$ by auto

with $S_2$ have card-$D_1$: card $D_1 = n$ by auto
from $d_1$-finite card-$D_1$ have finite $D_1$ ∧ card $D_1 = n$ by auto

define $u$ where $u = \text{set-to-nat} D$
define $u_1$ where $u_1 = \text{set-to-nat} D_1$

from $S_1$ $m$-in have \(\sum (\lambda x. (2 :: \text{nat})^x) D = 2^m + \sum (\lambda x. (2 :: \text{nat})^x) (D - \{m\})\) by (rule sum.remove)

with $\text{set-to-nat-def}$ have \(\text{set-to-nat} \ D = 2^m + \text{set-to-nat} \ (D - \{m\})\) by auto

with $u$-def $u_1$-def $D_1$-def have $u$-$u_1$: $u = u_1 + 2^m$ by auto
from $S_3$ $u$-def have $d_1$-$u_1$: nat-to-set $u_1 = D_1$ by auto

have $u$-$l$-$l$: $u_1 < 2^m$
proof

have $L1$: $D_1 \subseteq \{i. i < m\}$
proof fix $x$
assume $A1$: $x \in D_1$
show $x \in \{i. i < m\}$
proof
from $A1$ $D_1$-def have $L1$-$1$: $x \in D$ by auto
from $S_1$ $D$-$ne$ $L1$-$1$ $m$-def have $L1$-$2$: $x \leq m$ by auto
with $A1$ $L1$-$1$ $D_1$-def have $x \neq m$ 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 $D_1 \leq \text{set-to-nat} \ \{i. i < m\}$ by (rule set-to-nat-mono)

with $u$-def have $u$-$l$-$l$: $u_1 \leq \text{set-to-nat} \ \{i. i < m\}$ by auto
with set-to-nat-of-interval have $L3$: $u_1 \leq 2^m - 1$ by auto

then have \((2 :: \text{nat})^m - 1 < (2 :: \text{nat})^m\) by auto
with $L3$ show ?thesis by arith
qed

from $u$-def have nat-to-set ($\text{set-to-nat} \ D$) \(= \text{nat-to-set} \ u\) by auto
also from $u$-$u_1$ have \(\ldots = \text{nat-to-set} \ (u_1 + 2^m)\) by auto
also from $u$-$l$-$l$ have \(\ldots = \text{nat-to-set} \ u_1 \cup \{m\}\) by (rule add-power)
also from $d_1$-$u_1$ have \(\ldots = D_1 \cup \{m\}\) by auto
also from $D_1$-def $m$-in have \(\ldots \ = D\) by auto
finally show nat-to-set ($\text{set-to-nat} \ D$) \(= D\) by auto
qed
qed

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

88
theorem nat-to-set-srj1: finite (D :: nat set) \implies \exists u. nat-to-set u = D
proof
  assume A: finite D
  show \exists u. nat-to-set u = D
    proof
      from A show nat-to-set (set-to-nat D) = D by (rule nat-to-set-srj)
    qed
  qed

lemma sum-of-pr-is-pr: \( g \in \text{PrimRec1} \implies (\lambda n. \text{sum} g \{i. i < n\}) \in \text{PrimRec1} \)
proof
  assume g-is-pr: g \in \text{PrimRec1}
  define f where f n = \text{sum} (\lambda x. g x) \{i. i < n\} for n
  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 (Suc y) = h y (f y)
    proof
      fix y show f (Suc y) = h y (f y)
        proof
          from f-def have S1: f (Suc y) = \text{sum} g \{i. i < Suc y\} by auto
          have S2: \{i. i < Suc y\} = \{i. i < y\} \cup \{y\} by auto
          have S3: finite \{i. i < y\} by (rule finite-interval)
          have S4: y \notin \{i. i < y\} by auto
          from S1 S2 have f (Suc y) = \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 \ldots 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: \( p \in \text{PrimRec2} \implies (\lambda n m. \text{sum} (\lambda x. p x) \{i. i < n\}) \in \text{PrimRec2} \)
proof
  assume p-is-pr: p \in \text{PrimRec2}
  define f where f n m = \text{sum} (\lambda x. p x) \{i. i < n\} for n m
  define g :: nat \Rightarrow 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])
  have f-at-0: \forall x. f 0 x = g x
    proof
      fix x from f-def g-def show f 0 x = g x by auto
    qed
  define h where h a b c = p a c + b for a b c
qed
from p-is-pr have h-is-pr: h ∈ PrimRec3 unfolding h-def by prec
have f-at-Suc: ∀ x y. f (Suc y) x = h y (f y x) x
proof (rule allI, rule allI)
  fix x y show f (Suc y) x = h y (f y x) x
  proof –
  from f-def have S1: f (Suc y) x = sum (λ z z x) {i. i < Suc y} by auto
  have S2: {i. i < Suc y} = {i. i < y} ∪ \{y\} by auto
  have S3: finite \{i. i < y\} by (rule finite-interval)
  have S4: y ∉ \{i. i < y\} by auto
  define g1 where g1 z = p z x for z
  from S1 S2 g1-def have f (Suc y) x = sum g1 (\{i. (i::nat) < y\} ∪ \{y\}) by auto
  also from S3 S4 sum.insert have ... = g1 y + sum g1 \{i. i<y\} by auto
  also from f-def g1-def have ... = g1 y + f y x by auto
  also from h-def g1-def have ... = h y (f y x) x by auto
  finally show ?thesis by auto
qed

from g-is-pr h-is-pr f-at-0 f-at-Suc have f-is-pr: f ∈ PrimRec2 by (rule pr-rec-scheme)
with f-def [abs-def] show ?thesis by auto
qed

lemma sum-is-pr: g ∈ PrimRec1 ⇒ (λ u. sum g (nat-to-set u)) ∈ PrimRec1
proof –
  assume g-is-pr: g ∈ PrimRec1
  define f where f u = sum (λ x. g1 x u) \{i. (i::nat) < u\} for u
  define f1 where f1 u v = sum (λ x. g1 x v) \{i. (i::nat) < u\} for u v
  from g1-is-pr have (λ (u::nat) v. sum (λ x. g1 x v) \{i. (i::nat) < u\}) ∈ PrimRec2
  by (rule sum-of-pr-is-pr2)
  with f1-def [abs-def] have f1-is-pr: f1 ∈ PrimRec2 by auto
  from f-def f1-def have ff1: f = (λ u. f1 u u) by auto
  from f1-is-pr have (λ u. f1 u u) ∈ PrimRec1 by prec
  with ff1 have f-is-pr: f ∈ PrimRec1 by auto
  have f-is-result: f = (λ u. sum g (nat-to-set u))
  proof –
    fix u show f u = sum g (nat-to-set u)
    proof –
      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 $U$-def $A$-def $B$-def have $A$-$B$: $A \cap B = \{ \}$ by auto
from $B$-def $g$-def have $B$-z: $\sum (\lambda x . g x u) B = 0$ by auto

have $u$-in-$U$: 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 $g$-def have $A$-$res$: $\sum (\lambda x . g x u) A = \sum g (\text{nat-to-set} u)$ by auto
from $f$-def have $f u = \sum (\lambda x . g x u) \{ i. (i::nat) < u \}$ by auto
also from $U$-def have $\ldots = \sum (\lambda x . g x u) U$ by auto
also from $A$-$B$ have $\ldots = \sum (\lambda x . g x u) (A \cup B)$ by auto
also from $A$-finite $B$-finite $A$-$B$ have $\ldots = \sum (\lambda x . g x u) A + \sum (\lambda x . g x u) B$ by (rule sum.union-disjoint)
also from $B$-z have $\ldots = \sum (\lambda x . g x u) A$ by auto
also from $A$-$res$ have $\ldots = \sum g (\text{nat-to-set} u)$ by auto
finally show $?thesis$ by auto
qed
definition $c$-$card :: \text{nat} \Rightarrow \text{nat}$ where
$c$-$card = (\lambda u. \text{card (nat-to-set} u))$

theorem $c$-$card$-$is$-$pr$: $c$-$card \in \text{PrimRec1}$
proof
  define $g :: \text{nat} \Rightarrow \text{nat}$ where $g x = 1$ for $x$
  have $g$-$is$-$pr$: $g \in \text{PrimRec1}$ by (unfold $g$-def, rule const-is-pr)
  have $c$-$card = (\lambda u. \text{sum} g (\text{nat-to-set} u))$
  proof
    fix $u$ show $c$-$card u = \text{sum} g (\text{nat-to-set} u)$ by (unfold $c$-$card$-$def$, unfold $g$-def, rule card-$eq$-$sum$)
  qed
moreover from $g$-$is$-$pr$ have $(\lambda u. \text{sum} g (\text{nat-to-set} u)) \in \text{PrimRec1}$ by (rule sum-$is$-$pr$)
ultimately show $?thesis$ by auto
qed
definition $c$-$insert :: \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat}$ where
$c$-$insert = (\lambda x u. \text{if c-in} x u = 1 \text{ then} u \text{ else} u + 2^x)$

lemma $c$-$insert$-$is$-$pr$: $c$-$insert \in \text{PrimRec2}$
proof (unfold $c$-$insert$-$def$, rule if-$eq$-$is$-$pr$2)
show \( c\text{-in} \in \text{PrimRec2} \) by (rule c-in-is-pr)

next

show \((\lambda x \ y \ . \ 1) \in \text{PrimRec2}\) by (rule const-is-pr-2)

next

show \((\lambda x \ y \ . \ y) \in \text{PrimRec2}\) by (rule pr-id2-2)

next

from power-is-pr show \((\lambda x \ y \ . \ y + 2 ^ x) \in \text{PrimRec2}\) by prec

qed

lemma [simp]: set-to-nat (nat-to-set \( u \)) = \( u \)

def

define \( D \) where \( D = \text{nat-to-set} \( u \) \)

from A D-def have \( S1: x \not\in D \) by auto

let \( \lambda f = \lambda (x::nat). \ (2::nat)^x \)

from set-to-nat-def have \( \text{set-to-nat} \ (D \cup \{x\}) = \text{sum} \ ?f \ (D \cup \{x\}) \) by simp

also from \( D\text{-finite} \) have \( \ldots = ?f \ x + \text{sum} \ ?f \ D \) by simp

also from set-to-nat-def have \( \ldots = 2 ^ x + \text{set-to-nat} \ D \) by auto

finally have \( \text{set-to-nat} \ (D \cup \{x\}) = \text{set-to-nat} \ D + 2 ^ x \) by auto

with \( D\text{-def} \) show ?thesis by auto

qed

lemma c-insert-df: c-insert = \( (\lambda x \ u \ . \ \text{set-to-nat} \ ((\text{nat-to-set} \ u) \cup \{x\})) \)

fix \( x \ u \) show c-insert \( x \ u = \text{set-to-nat} \ (\text{nat-to-set} \ u \cup \{x\}) \)

proof (cases)

assume A: \( x \in \text{nat-to-set} \ u \)

then have \( \text{set-to-nat} \ u \cup \{x\} = \text{nat-to-set} \ u \) by auto

from A have \( \text{c-in} \ x \ u = 1 \) 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: \( x \not\in \text{nat-to-set} \ u \)

then have \( S1: \text{c-in} \ x \ u \neq 1 \) by (simp add: x-in-u-eq)

then have \( S2: \text{c-insert} \ x \ u = u + 2 ^ x \) 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

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 (λx y. 0) ∈ PrimRec2 by (rule const-is-pr-2)
next
show (λx y. y) ∈ PrimRec2 by (rule pr-id2-2)
next
from power-is-pr show (λx y. y − 2 ^ x) ∈ PrimRec2 by prec
qed

lemma 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

93
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
qed
definition

\textit{c-union :: nat ⇒ nat ⇒ nat where}
c-union = \( \lambda \, u \, v. \text{set-to-nat} (\text{nat-to-set} u \cup \text{nat-to-set} v) \)

theorem c-union-is-pr: c-union ∈ PrimRec2

proof -
define \( \text{f} \) where \( f \, y \, x = \text{set-to-nat} ((\text{nat-to-set} (\text{c-fst} \, x)) \cup \{ \, z \in \text{nat-to-set} (\text{c-snd} \, x) \, \} \) for \( y \) \( x \)
have f-is-pr: \( f \in \text{PrimRec2} \)
proof -
define \( \text{g} \) where \( g = \text{c-fst} \)
from c-fst-is-pr g-def have g-is-pr: \( g \in \text{PrimRec1} \) by auto
define \( \text{h} \) where \( h \, a \, b \, c = (\text{if} \, \text{c-in} \, a \, (\text{c-snd} \, c) = 1 \, \text{then} \, \text{c-insert} \, a \, b \, \text{else} \, \text{b}) \) for \( a \, b \, c \)
from c-in-is-pr c-insert-is-pr have h-is-pr: \( h \in \text{PrimRec3} \) unfolding h-def by prec
have f-at-0: \( \forall \, x. \, f \, 0 \, x = y \, x \)
proof
fix x show f 0 x = y x by (unfold f-def, unfold g-def, simp)
qed
have f-at-Suc: \( \forall \, x \, y. \, f \, (\text{Suc} \, y) \, x = h \, y \, (f \, y \, x) \)
proof (rule allI, rule allII)
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: \( \{ \, z \in \text{nat-to-set} (\text{c-snd} \, x) \, . \, z < \text{Suc} \, y \} = \{ \, z \in \text{nat-to-set} (\text{c-snd} \, x) \, . \, z < \text{Suc} \, y \} \cup \{ y \} \) by auto
from nat-to-set-is-finite have S4: finite ((nat-to-set (c-fst x)) \cup \{ \, z \in \text{nat-to-set} (\text{c-snd} \, x) \, . \, z < \text{Suc} \, y \}) by auto
with nat-to-set-srj f-def have S5: nat-to-set (f y x) = (nat-to-set (c-fst x)) \cup \{ \, z \in \text{nat-to-set} (\text{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)) \cup \{ \, z \in \text{nat-to-set} (\text{c-snd} \, x) \, . \, z < \text{Suc} \, y \}) by simp
also from S3 have ... = set-to-nat ((nat-to-set (c-fst x)) \cup \{ \, z \in \text{nat-to-set} (\text{c-snd} \, x) \, . \, z < y \} \cup \{ y \}) by auto
also from S5 have ... = c-insert y (f y x) \cup \{ y \} by auto
also have ... = c-insert y (f y x) by (simp add: c-insert-df)
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)
  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
    have {z ∈ nat-to-set (c-snd x). z < Suc y} = {z ∈ nat-to-set (c-snd x). z < y} ∪ {z ∈ nat-to-set (c-snd x). z = y}
    by auto
    with S1 show ?thesis by auto
  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 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\})) by auto
  also have ... = set-to-nat (nat-to-set (f y x)) by auto
  finally show ?thesis 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))
theorem \texttt{c-diff-is-pr}: \texttt{c-diff} \in \texttt{PrimRec2}
proof
  define \texttt{f} where \texttt{f y x} = \texttt{set-to-nat ((nat-to-set (c-fst x)) \{-\{z \in \texttt{nat-to-set (c-snd x)}, z < y\}\})}
  for \texttt{y x}
  have \texttt{f-is-pr}: \texttt{f} \in \texttt{PrimRec2}
proof
  define \texttt{g} where \texttt{g} = \texttt{c-fst}
  from \texttt{c-fst-is-pr g-def} have \texttt{g-is-pr}: \texttt{g} \in \texttt{PrimRec1} by auto
  define \texttt{h} where \texttt{h a b c} = (if \texttt{c-in a (c-snd c)} = \texttt{1} then \texttt{c-remove a b else b})
  for \texttt{a b c}
  from \texttt{c-in-is-pr c-remove-is-pr} have \texttt{h-is-pr}: \texttt{h} \in \texttt{PrimRec3} unfolding \texttt{h-def}
by prec
  have \texttt{f-at-0}: \forall \texttt{x}. \texttt{f 0 x} = \texttt{g x}
proof
  fix \texttt{x} show \texttt{f 0 x} = \texttt{g x} by (unfold \texttt{f-def}, unfold \texttt{g-def}, simp)
qed
have \texttt{f-at-Suc}: \forall \texttt{x y}. \texttt{f (Suc y) x} = \texttt{h y (f y x) x}
proof (rule allI, rule allI)
  fix \texttt{x} \texttt{y} show \texttt{f (Suc y) x} = \texttt{h y (f y x) x}
proof (cases)
  assume \texttt{A}: \texttt{c-in y (c-snd x) = 1}
  then have \texttt{S1}: \texttt{y} \in (\texttt{nat-to-set (c-snd x))} by (simp add: \texttt{x-in-u-eq})
  from \texttt{A} \texttt{h-def} have \texttt{S2}: \texttt{h y (f y x) x} = \texttt{c-remove y (f y x) x} by auto
  have (\texttt{nat-to-set (c-fst x))} - (\{\texttt{z} \in \texttt{nat-to-set (c-snd x)}, \texttt{z < y}\} \cup \{\texttt{y}\}) =
    (\{\texttt{z} \in \texttt{nat-to-set (c-snd x)}, \texttt{z < y}\}) - \{\texttt{y}\} by auto
  from \texttt{S1} have \texttt{S3}: \{\texttt{z} \in \texttt{nat-to-set (c-snd x)}, \texttt{z < Suc y}\} = \{\texttt{z} \in \texttt{nat-to-set (c-snd x)}, \texttt{z < y}\} \cup \{\texttt{y}\} by auto
  from \texttt{nat-to-set-is-finite} have \texttt{S4}: \texttt{finite ((nat-to-set (c-fst x))} - \{\texttt{z} \in \texttt{nat-to-set (c-snd x)}, \texttt{z < Suc y}\}) by auto
  with \texttt{nat-to-set-srj f-def} have \texttt{S5}: \texttt{nat-to-set (f y x)} = \texttt{(nat-to-set (c-fst x))}
    - \{\texttt{z} \in \texttt{nat-to-set (c-snd x)}, \texttt{z < y}\} by auto
  from \texttt{f-def} have \texttt{S6}: \texttt{f (Suc y) x} = \texttt{set-to-nat ((nat-to-set (c-fst x))} - \{\texttt{z} \in \texttt{nat-to-set (c-snd x)}, \texttt{z < Suc y}\}) by simp
  also from \texttt{S3} have \ldots: = \texttt{set-to-nat ((nat-to-set (c-fst x))} - \{\texttt{z} \in \texttt{nat-to-set (c-snd x)}, \texttt{z < y}\} \cup \{\texttt{y}\}) by auto
  also have \ldots: = \texttt{set-to-nat ((nat-to-set (c-fst x))} - \{\texttt{z} \in \texttt{nat-to-set (c-snd x)}, \texttt{z < y}\} by (rule \texttt{lm1})
  also from \texttt{S5} have \ldots: = \texttt{set-to-nat (f y x)} - \{\texttt{y}\} by auto
  also have \ldots: = \texttt{c-remove y (f y x)} by (simp add: \texttt{c-remove-df})
  finally show \texttt{?thesis} by (simp add: \texttt{S2})
next
  assume \texttt{A}: \texttt{\neg c-in y (c-snd x) = 1}
  then have \texttt{S1}: \texttt{y} \notin (\texttt{nat-to-set (c-snd x))} by (simp add: \texttt{x-in-u-eq})

96
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} by auto
proof
  have {z ∈ nat-to-set (c-snd x). z < Suc y} = {z ∈ nat-to-set (c-snd x). z = y} by auto
  with S1 show ?thesis by auto
qed
from nat-to-set-is-finite have S4: finite ((nat-to-set (c-fst x)) − {z ∈ nat-to-set (c-snd x). z < Suc y}) by auto
with nat-to-set-srj f-def have S5: nat-to-set (c-fst x) = (nat-to-set (c-snd 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}) by auto
also from S5 have ... = set-to-nat ((nat-to-set (c-fst x) − {z ∈ nat-to-set (c-fst x). z < Suc y}) − {z ∈ nat-to-set (c-fst x). z < Suc y}) by simp
finally show ?thesis by simp add: S2
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
    from nat-to-set-upper-bound1 have {z ∈ nat-to-set v. z < v} = nat-to-set v by auto
    with diff-def f-def show ?thesis by auto
  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))
theorem c-intersect-is-pr: c-intersect ∈ PrimRec2
proof
  define f where f u v = c-diff (c-union u v) (c-union (c-diff u v) (c-diff v u)) for u v
from c-diff-is-pr c-union-is-pr have f-is-pr: f ∈ PrimRec2 unfolding f-def by
prec
have \( u \land v \cdot f u v = c\text{-intersect } u v \)
proof -
  fix \( u \land v \) show \( f u v = c\text{-intersect } u v \)
proof -
  let \(?A = nat\text{-to-set } u\)
  let \(?B = nat\text{-to-set } v\)
  have A-fin: finite \(?A\) by (rule nat-to-set-is-finite)
  have B-fin: finite \(?B\) by (rule nat-to-set-is-finite)
  have S1: c-union \( u v = set\text{-to-nat } (?A \cup ?B) \) by (simp add: c-union-def)
  have S2: c-diff \( u v = set\text{-to-nat } (?A - ?B) \) by (simp add: c-diff-def)
  have S3: c-diff \( v u = set\text{-to-nat } (?B - ?A) \) by (simp add: c-diff-def)
  from S2 A-fin B-fin have S4: nat-to-set (c-diff \( u v \)) = \(?A - ?B\) by (simp add: nat-to-set-srj)
  from S3 A-fin B-fin have S5: nat-to-set (c-diff \( v u \)) = \(?B - ?A\) by (simp add: nat-to-set-srj)
  from S4 S5 have S6: c-union (c-diff \( u v \)) (c-diff \( v u \)) = (?A - ?B) \( \cup (?B - ?A) \) by (simp add: c-union-def)
  from S1 A-fin B-fin have S7: nat-to-set (c-union \( u v \)) = \(?A \cup ?B\) by (simp add: nat-to-set-srj)
  from S6 A-fin B-fin have S8: nat-to-set (c-union (c-diff \( u v \)) (c-diff \( v u \))) = (?A - ?B) \( \cup (?B - ?A) \) by (simp add: nat-to-set-srj)
  from S7 S8 have S9: \( u v = set\text{-to-nat } (((?A \cup ?B) - ((?A - ?B) \cup (?B - ?A))) \) by auto
  with S9 have S10: \( ?A \cap ?B = ((?A \cup ?B) - ((?A - ?B) \cup (?B - ?A)) \) by auto
  have c-intersect \( u v = set\text{-to-nat } (?A \cap ?B) \) by (simp add: c-intersect-def)
  with S10 show \( \text{?thesis by auto} \)
qed
qed
then have \( f = c\text{-intersect} \) by (simp add: ext)
with f-is-pr show \( \text{?thesis by auto} \)
qed

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

theory PRecUnGr
imports PRecFun2 PRecList
begin

We introduce a particular function which is universal for primitive recursive functions of one variable.
definition
g-comp :: nat ⇒ nat ⇒ nat where
g-comp c-ls key = 
  let n = c-fst key; x = c-snd key; m = c-snd n;
  m1 = c-fst m; m2 = c-snd m in
  -- We have key = < n, x>; n = <?, m>; m = <m1, m2>.
  if c-assoc-have-key c-ls (c-pair m2 x) = 0 then
    (let y = c-assoc-value c-ls (c-pair m2 x) in
     if c-assoc-have-key c-ls (c-pair m1 y) = 0 then
       (let z = c-assoc-value c-ls (c-pair m1 y) in
        c-cons (c-pair key z) c-ls)
     else c-ls)
  else c-ls
)

definition
g-pair :: nat ⇒ nat ⇒ nat where
g-pair c-ls key = ( 
  let n = c-fst key; x = c-snd key; m = c-snd n;
  m1 = c-fst m; m2 = c-snd m in
  -- We have key = < n, x>; n = <?, m>; m = <m1, m2>.
  if c-assoc-have-key c-ls (c-pair m1 x) = 0 then
    (let y1 = c-assoc-value c-ls (c-pair m1 x) in
     if c-assoc-have-key c-ls (c-pair m2 x) = 0 then
       (let y2 = c-assoc-value c-ls (c-pair m2 x) in
        c-cons (c-pair key (c-pair y1 y2)) c-ls)
     else c-ls)
  else c-ls
)

definition
g-rec :: nat ⇒ nat ⇒ nat where
g-rec c-ls key = ( 
  let n = c-fst key; x = c-snd key; m = c-snd n;
  m1 = c-fst m; m2 = c-snd m; y1 = c-fst x; x1 = c-snd x in
  -- We have key = < n, x>; n = <?, m>; m = <m1, m2>; x = <y1, x1>.
  if y1 = 0 then
    (if c-assoc-have-key c-ls (c-pair m1 x1) = 0 then
      c-cons (c-pair key (c-assoc-value c-ls (c-pair m1 x1))) c-ls
    else c-ls)
  else
    (let y2 = y1-(1::nat) in
     if c-assoc-have-key c-ls (c-pair n (c-pair y2 x1)) = 0 then
       (let t1 = c-assoc-value c-ls (c-pair n (c-pair y2 x1)); t2 = c-pair (c-pair y2 t1) x1 in
        ...)
if c-assoc-have-key c-ls (c-pair m2 t2) = 0 then
c-cons (c-pair key (c-assoc-value c-ls (c-pair m2 t2))) c-ls
else c-ls
)
else c-ls
)

definition

g-step :: nat ⇒ nat ⇒ nat where

g-step c-ls key = ( let n = c-fst key; x = c-snd key; n1 = (c-fst n) mod 7 in
if n1 = 0 then c-cons (c-pair key 0) c-ls else
if n1 = 1 then c-cons (c-pair key (Suc x)) c-ls else
if n1 = 2 then c-cons (c-pair key (c-fst x)) c-ls else
if n1 = 3 then c-cons (c-pair key (c-snd x)) c-ls else
if n1 = 4 then g-comp c-ls key else
if n1 = 5 then g-pair c-ls key else
if n1 = 6 then g-rec c-ls key else
)

definition

pr-gr :: nat ⇒ nat where

pr-gr-def: pr-gr = PrimRecOp1 0 (λ a b. g-step b (c-fst a))

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

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

definition

univ-for-pr :: nat ⇒ nat where

univ-for-pr = pr-cone-2-to-1 nat-to-pr

theorem univ-is-not-pr: univ-for-pr ∉ PrimRec1

proof (rule ccontr)
  assume ¬ univ-for-pr ∉ PrimRec1 then have A1: univ-for-pr ∈ PrimRec1 by simp
  let ?f = λ n. univ-for-pr (c-pair n n) + 1
  let ?n0 = index-of-pr ?f
  from A1 have S1: ?f ∈ PrimRec1 by prec
  then have S2: nat-to-pr ?n0 = ?f by (rule index-of-pr-is-real)
  then have S3: nat-to-pr ?n0 ?n0 = ?f ?n0 by simp
  have S4: ?f ?n0 = univ-for-pr (c-pair ?n0 ?n0) + 1 by simp
  from S3 S4 show False by (simp add: univ-for-pr-def pr-cone-2-to-1-def)

qed

definition

100
\[
\text{c-is-sub-fun} :: \text{nat} \Rightarrow (\text{nat} \Rightarrow \text{nat}) \Rightarrow \text{bool}
\]

where
\[
c\text{-is-sub-fun } \text{ls } f \iff (\forall x. \text{c-assoc-have-key } \text{ls } x = 0 \rightarrow \text{c-assoc-value } \text{ls } x = f x)
\]

**lemma** c-is-sub-fun-lm-1: \[
\begin{array}{c}
\text{ls } f \\
\text{c-assoc-have-key } \text{ls } x = 0
\end{array} \Rightarrow \text{c-assoc-value } \text{ls } x = f x
\]

**apply** (unfold c-is-sub-fun-def)

**apply** (auto)

**done**

**lemma** c-is-sub-fun-lm-2: \[
c\text{-is-sub-fun } \text{ls } f = \Rightarrow c\text{-is-sub-fun } (\text{c-cons } (\text{c-pair } x (f x)) \text{ls}) f
\]

**proof** –

**assume** A1: c-is-sub-fun ls f

**show** ?thesis

**proof** (unfold c-is-sub-fun-def, rule allI, rule impI)

**fix** xa **assume** A2: c-assoc-have-key (c-cons (c-pair x (f x)) ls) xa = 0 **show** c-assoc-value (c-cons (c-pair x (f x)) ls) xa = f xa

**proof** cases

**assume** C1: xa = x

then **show** c-assoc-value (c-cons (c-pair x (f x)) ls) xa = f xa **by** (simp add: PRecList.c-assoc-lm-2)

next

**assume** C2: \(\neg xa = x\)

then **have** S1: c-assoc-have-key (c-cons (c-pair x (f x)) ls) xa = c-assoc-have-key ls xa by (rule c-assoc-lm-3)

from C2 have S2: c-assoc-value (c-cons (c-pair x (f x)) ls) xa = c-assoc-value ls xa by (rule c-assoc-lm-4)

from A2 S1 have S3: c-assoc-have-key ls xa = 0 by simp

from A1 S3 have c-assoc-value ls xa = f xa by (rule c-is-sub-fun-lm-1)

with S2 show ?thesis **by** simp

qed

qed

**lemma** mod7-lm: \((n::\text{nat}) \mod 7 = 0 \lor
\begin{array}{c}
(n::\text{nat}) \mod 7 = 1 \\
(n::\text{nat}) \mod 7 = 2 \\
(n::\text{nat}) \mod 7 = 3 \\
(n::\text{nat}) \mod 7 = 4 \\
(n::\text{nat}) \mod 7 = 5 \\
(n::\text{nat}) \mod 7 = 6
\end{array} \text{ by arith}
\]

**lemma** nat-to-sch-at-pos: \(x > 0 \Rightarrow \text{nat-to-sch } x = (\text{let } u=(c\text{-fst } x) \mod 7; v=c\text{-snd } x; v1=c\text{-fst } v; v2 = c\text{-snd } v; sch1=\text{nat-to-sch } v1; sch2=\text{nat-to-sch } v2 \\
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-fst x = 0
  proof
    have 1 = c-pair 0 1 by (simp add: c-pair-def sf-def)
    then have c-fst 1 = c-fst (c-pair 0 1) by simp
    then have c-fst 1 = 0 by simp
    with A1 show ?thesis by simp
  qed
  from A1 have S2: nat-to-sch x = Base-zero by simp
  from S1 S2 show nat-to-sch x = (let u = 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)
     apply(insert S1 S2)
     apply(simp add: Let-def loc-f-def)
     done
next
  assume ¬ x = 1
  from A this have A2: x > 1 by simp
  from this have nat-to-sch x = (let u = 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) by (rule loc-srj-lm-2)
  from this show nat-to-sch x = (let u = 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) by (simp add: mod7-def)
  qed
qed

lemma nat-to-sch-0: c-fst n mod 7 = 0 ⇒ nat-to-sch n = Base-zero
proof
  assume A: c-fst n mod 7 = 0
  show ?thesis
  proof cases
    assume n = 0
    then show nat-to-sch n = Base-zero by simp
  next
    assume ¬ n = 0 then have n > 0 by simp
    then have nat-to-sch n = (let u = c-fst n; 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 (rule nat-to-sch-at-pos)
    with A show nat-to-sch n = Base-zero by (simp add: Let-def loc-f-def)
  qed
qed

lemma loc-lm-1: c-fst n mod 7 ≠ 0 ⇒ n > 0
proof
  assume A: c-fst n mod 7 ≠ 0
  have n = 0 ⇒ False
  proof
    assume n = 0
then have \( c\text{-fst } n \mod 7 \neq 0 \) 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: \( c\text{-fst } n \mod 7 \neq 0 \implies \text{nat-to-sch } n = (\text{let } u = (c\text{-fst } n) \mod 7; v = c\text{-snd } n; v_1 = c\text{-fst } v; v_2 = c\text{-snd } v; \text{sch1} = \text{nat-to-sch } v_1; \text{sch2} = \text{nat-to-sch } v_2 \text{ in } \text{loc-f } u \text{ sch1 sch2}) \)
proof –
  assume c-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: \( c\text{-fst } n \mod 7 = 1 \implies \text{nat-to-sch } n = \text{Base-suc} \)
proof –
  assume A1: \( c\text{-fst } n \mod 7 = 1 \)
  then have nat-to-sch n = (let u = (c-fst n) mod 7; v = c-snd n; v_1 = c-fst v; v_2 = c-snd v; sch1 = nat-to-sch v_1; sch2 = nat-to-sch v_2 in loc-f u sch1 sch2) by (simp add: loc-lm-2)
  with A1 show nat-to-sch n = Base-suc by (simp add: Let-def loc-f-def)
qed

lemma nat-to-sch-2: \( c\text{-fst } n \mod 7 = 2 \implies \text{nat-to-sch } n = \text{Base-fst} \)
proof –
  assume A1: \( c\text{-fst } n \mod 7 = 2 \)
  then have nat-to-sch n = (let u = (c-fst n) mod 7; v = c-snd n; v_1 = c-fst v; v_2 = c-snd v; sch1 = nat-to-sch v_1; sch2 = nat-to-sch v_2 in loc-f u sch1 sch2) by (simp add: loc-lm-2)
  with A1 show nat-to-sch n = Base-fst by (simp add: Let-def loc-f-def)
qed

lemma nat-to-sch-3: \( c\text{-fst } n \mod 7 = 3 \implies \text{nat-to-sch } n = \text{Base-snd} \)
proof –
  assume A1: \( c\text{-fst } n \mod 7 = 3 \)
  then have nat-to-sch n = (let u = (c-fst n) mod 7; v = c-snd n; v_1 = c-fst v; v_2 = c-snd v; sch1 = nat-to-sch v_1; sch2 = nat-to-sch v_2 in loc-f u sch1 sch2) by (simp add: loc-lm-2)
  with A1 show nat-to-sch n = Base-snd by (simp add: Let-def loc-f-def)
qed

lemma nat-to-sch-4: \( c\text{-fst } n \mod 7 = 4 \implies \text{nat-to-sch } n = \text{Comp-op } (\text{nat-to-sch } (c\text{-fst } (c\text{-snd } n))) (\text{nat-to-sch } (c\text{-snd } (c\text{-snd } n))) \)
proof –
  assume A1: \( c\text{-fst } n \mod 7 = 4 \)
  then have nat-to-sch n = (let u = (c-fst n) mod 7; v = c-snd n; v_1 = c-fst v; v_2 = c-snd v; sch1 = nat-to-sch v_1; sch2 = nat-to-sch v_2 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)
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 S1: nat-to-pr n x = sch-to-pr (nat-to-sch n) x by (simp add: nat-to-pr-def)
  from A have S2: nat-to-sch n = Base-fst by (rule nat-to-sch-2)
  from S1 S2 show ?thesis by simp
qed

lemma nat-to-pr-lm-3: c-fst n mod 7 = 3 \implies nat-to-pr n x = c-snd x
proof -
  assume A: c-fst n mod 7 = 3
  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-snd by (rule nat-to-sch-3)
  from S1 S2 show ?thesis by simp
qed

lemma nat-to-pr-lm-4: c-fst n mod 7 = 4 \implies nat-to-pr n x = (nat-to-pr (c-fst (c-snd n)) (nat-to-pr (c-snd (c-snd n)))) x
proof -
  assume A: c-fst n mod 7 = 4
  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 = Comp-op (nat-to-sch (c-fst (c-snd n))) (nat-to-sch (c-snd (c-snd n))) by (rule nat-to-sch-4)
  from S1 S2 have S3: nat-to-pr n x = sch-to-pr (Comp-op (nat-to-sch (c-fst (c-snd n))) (nat-to-sch (c-snd (c-snd n)))) x by simp
  from S3 have S4: nat-to-pr n x = (sch-to-pr (nat-to-sch (c-fst (c-snd n)))) ((sch-to-pr (nat-to-sch (c-snd (c-snd n)))) x) by simp
  from S4 show ?thesis by (simp add: nat-to-pr-def)
qed

lemma nat-to-pr-lm-5: c-fst n mod 7 = 5 \implies nat-to-pr n x = (c-f-pair (nat-to-pr (c-fst (c-snd n))) (nat-to-pr (c-snd (c-snd n)))) x
proof -
  assume A: c-fst n mod 7 = 5
  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 = Pair-op (nat-to-sch (c-fst (c-snd n))) (nat-to-sch (c-snd (c-snd n))) by (rule nat-to-sch-5)
  from S1 S2 have S3: nat-to-pr n x = sch-to-pr (Pair-op (nat-to-sch (c-fst (c-snd n))) (nat-to-sch (c-snd (c-snd n)))) x by simp
  from S3 show ?thesis by (simp add: nat-to-pr-def)
qed

lemma nat-to-pr-lm-6: c-fst n mod 7 = 6 \implies 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 mod 7 = 6
  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 = 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 S1 S2 have S3: nat-to-pr n x = sch-to-pr (Rec-op (nat-to-sch (c-fst (c-snd n))) (nat-to-sch (c-snd (c-snd n)))) x by simp
  from S3 show ?thesis by (simp add: nat-to-pr-def)
qed

lemma univ-for-pr-lm-0: c-fst (c-fst key) mod 7 = 0 \implies univ-for-pr key = 0
proof
- assume $A$: $c\text{-fst} (c\text{-fst key}) \mod 7 = 0$
  have $S1$: $\text{univ-for-pr} = \text{nat-to-pr} (c\text{-fst key}) (c\text{-snd key})$ by (simp add: 
  $\text{univ-for-pr-def} \ \text{pr-conv-2-to-1-def}$)
  with $A$ show ?thesis by (simp add: nat-to-pr-lm-0)
qed

lemma $\text{univ-for-pr-lm-1}$: $c\text{-fst} (c\text{-fst key}) \mod 7 = 1 \implies \text{univ-for-pr} = \text{Suc} (c\text{-snd key})$
proof
- assume $A$: $c\text{-fst} (c\text{-fst key}) \mod 7 = 1$
  have $S1$: $\text{univ-for-pr} = \text{nat-to-pr} (c\text{-fst key}) (c\text{-snd key})$ by (simp add: 
  $\text{univ-for-pr-def} \ \text{pr-conv-2-to-1-def}$)
  with $A$ show ?thesis by (simp add: nat-to-pr-lm-1)
qed

lemma $\text{univ-for-pr-lm-2}$: $c\text{-fst} (c\text{-fst key}) \mod 7 = 2 \implies \text{univ-for-pr} = c\text{-fst} (c\text{-snd key})$
proof
- assume $A$: $c\text{-fst} (c\text{-fst key}) \mod 7 = 2$
  have $S1$: $\text{univ-for-pr} = \text{nat-to-pr} (c\text{-fst key}) (c\text{-snd key})$ by (simp add: 
  $\text{univ-for-pr-def} \ \text{pr-conv-2-to-1-def}$)
  with $A$ show ?thesis by (simp add: nat-to-pr-lm-2)
qed

lemma $\text{univ-for-pr-lm-3}$: $c\text{-fst} (c\text{-fst key}) \mod 7 = 3 \implies \text{univ-for-pr} = c\text{-snd} (c\text{-snd key})$
proof
- assume $A$: $c\text{-fst} (c\text{-fst key}) \mod 7 = 3$
  have $S1$: $\text{univ-for-pr} = \text{nat-to-pr} (c\text{-fst key}) (c\text{-snd key})$ by (simp add: 
  $\text{univ-for-pr-def} \ \text{pr-conv-2-to-1-def}$)
  with $A$ show ?thesis by (simp add: nat-to-pr-lm-3)
qed

lemma $\text{univ-for-pr-lm-4}$: $c\text{-fst} (c\text{-fst key}) \mod 7 = 4 \implies \text{univ-for-pr} = (\text{nat-to-pr} \ (c\text{-fst} (c\text{-snd} (c\text{-fst key})))) (\text{nat-to-pr} (c\text{-snd} (c\text{-snd} (c\text{-fst key})))) (c\text{-snd key}))$
proof
- assume $A$: $c\text{-fst} (c\text{-fst key}) \mod 7 = 4$
  have $S1$: $\text{univ-for-pr} = \text{nat-to-pr} (c\text{-fst key}) (c\text{-snd key})$ by (simp add: 
  $\text{univ-for-pr-def} \ \text{pr-conv-2-to-1-def}$)
  with $A$ show ?thesis by (simp add: nat-to-pr-lm-4)
qed

lemma $\text{univ-for-pr-lm-4-1}$: $c\text{-fst} (c\text{-fst key}) \mod 7 = 4 \implies \text{univ-for-pr} = \text{univ-for-pr} (c\text{-pair} (c\text{-fst} (c\text{-snd} (c\text{-fst key})))) (\text{univ-for-pr} (c\text{-pair} (c\text{-snd} (c\text{-snd} (c\text{-fst key}))))) (c\text{-snd key}))$
proof
- assume $A$: $c\text{-fst} (c\text{-fst key}) \mod 7 = 4$
have $S1$: $\text{univ-for-pr key} = \text{nat-to-pr (c-fst key) (c-snd key)}$ by (simp add: univ-for-pr-def pr-conv-2-to-1-def)

with A show ?thesis by (simp add: nat-to-pr-lm-4 univ-for-pr-def pr-conv-2-to-1-def)

qed

lemma univ-for-pr-lm-5: $\text{c-fst (c-fst key) mod 7 = 5 \Rightarrow \text{univ-for-pr key} = \text{c-pair (univ-for-pr (c-pair (c-fst (c-snd (c-fst key))) (c-snd key))) (univ-for-pr (c-pair (c-snd (c-snd (c-fst key)))) (c-snd key)))}$

proof –
  assume A: $\text{c-fst (c-fst key) mod 7 = 5}$
  have $S1$: $\text{univ-for-pr key} = \text{nat-to-pr (c-fst key) (c-snd key)}$ by (simp add: univ-for-pr-def pr-conv-2-to-1-def)

  with A show ?thesis by (simp add: nat-to-pr-lm-5 c-f-pair-def univ-for-pr-def pr-conv-2-to-1-def)

qed

lemma univ-for-pr-lm-6-1: $[ [\text{c-fst (c-fst key) mod 7 = 6;} \text{c-fst (c-snd key)} = 0] \Rightarrow \text{univ-for-pr key} = \text{univ-for-pr (c-pair (c-fst (c-snd (c-fst key))) (c-snd (c-snd key)))} ]$

proof –
  assume A1: $\text{c-fst (c-fst key) mod 7 = 6}$
  assume A2: $\text{c-fst (c-snd key)} = 0$
  have $S1$: $\text{univ-for-pr key} = \text{nat-to-pr (c-fst key) (c-snd key)}$ by (simp add: univ-for-pr-def pr-conv-2-to-1-def)

  with A1 A2 show ?thesis by (simp add: nat-to-pr-lm-6 UnaryRecOp-def univ-for-pr-def pr-conv-2-to-1-def)

qed

lemma univ-for-pr-lm-6-2: $[ [\text{c-fst (c-fst key) mod 7 = 6;} \text{c-fst (c-snd key)} = \text{Suc u}] \Rightarrow \text{univ-for-pr key} = \text{univ-for-pr (c-pair (c-snd (c-snd (c-fst key)))) (c-pair (c-pair (c-fst (c-snd key)) (c-pair u (c-snd (c-snd key)))) (c-snd (c-snd key)))} ]$

proof –
  assume A1: $\text{c-fst (c-fst key) mod 7 = 6}$
  assume A2: $\text{c-fst (c-snd key)} = \text{Suc u}$
  have $S1$: $\text{univ-for-pr key} = \text{nat-to-pr (c-fst key) (c-snd key)}$ by (simp add: univ-for-pr-def pr-conv-2-to-1-def)

  with A1 A2 show ?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: $[ [\text{c-fst (c-fst key) mod 7 = 6;} \text{c-fst (c-snd key)} \neq 0] \Rightarrow \text{univ-for-pr key} = \text{univ-for-pr (c-pair (c-snd (c-snd (c-fst key)))) (c-pair (c-pair (c-fst (c-snd key)) (c-pair (c-snd (c-snd key)) (c-pair u (c-snd (c-snd key)))) (c-snd (c-snd key)))))} ]$

107
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-fst key)))
    (c-pair (c-pair ?u (univ-for-pr (c-pair (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 ?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 Let-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)
qed

lemma g-comp-lm-1: [ c-fst (c-fst key) mod 7 = 4; c-is-sub-fun ls univ-for-pr]
\[ \vdash \text{c-is-sub-fun} \ (g\text{-comp} \ \text{ls key}) \ \text{univ-for-pr} \]

**proof**
- assume \( A1: \ c\text{-fst} \ (c\text{-fst} \ \text{key}) \ \text{mod} \ 7 = 4 \)
- assume \( A2: \ c\text{-is-sub-fun} \ \text{ls univ-for-pr} \)
- show \( \text{thesis} \)
  **proof**
  cases
  - assume \( g\text{-comp} \ \text{ls key} = \ \text{ls} \)
    with \( A2 \) show \( c\text{-is-sub-fun} \ (g\text{-comp} \ \text{ls key}) \ \text{univ-for-pr} \) by simp
  next
  - assume \( g\text{-comp} \ \text{ls key} \neq \ \text{ls} \)
    from \( A1 \ A2 \) this have \( S1: \ g\text{-comp} \ \text{ls key} = \ \text{c-cons} \ (\text{c-pair} \ \text{key} (\text{univ-for-pr} \ \text{key})) \) \( \text{ls} \) by \( \text{rule g-comp-lm-0} \)
    with \( A2 \) show \( c\text{-is-sub-fun} \ (g\text{-comp} \ \text{ls key}) \ \text{univ-for-pr} \) by \( \text{simp add: c-is-sub-fun-lm-2} \)
  qed

**qed**

**lemma** \( g\text{-pair-lm-0}; \ [ \ [ c\text{-fst} \ (c\text{-fst} \ \text{key}) \ \text{mod} \ 7 = 5; \ c\text{-is-sub-fun} \ \text{ls univ-for-pr}; \ g\text{-pair} \ \text{ls key} \neq \ \text{ls} ] \] \[ \Rightarrow \ g\text{-pair} \ \text{ls key} = \ \text{c-cons} \ (\text{c-pair} \ \text{key} (\text{univ-for-pr} \ \text{key})) \) \( \text{ls} \)

**proof**
- assume \( A1: \ c\text{-fst} \ (c\text{-fst} \ \text{key}) \ \text{mod} \ 7 = 5 \)
- assume \( A2: \ c\text{-is-sub-fun} \ \text{ls univ-for-pr} \)
- assume \( A3: \ g\text{-pair} \ \text{ls key} \neq \ \text{ls} \)
  let \( ?n = c\text{-fst} \ \text{key} \)
  let \( ?x = c\text{-snd} \ \text{key} \)
  let \( ?m = c\text{-snd} \ ?n \)
  let \( ?m1 = c\text{-fst} \ ?m \)
  let \( ?m2 = c\text{-snd} \ ?m \)
  let \( ?k1 = c\text{-pair} \ ?m1 \ ?x \)
  have \( S1: \ c\text{-assoc-have-key} \ \text{ls} \ ?k1 = 0 \)
  **proof** \( \text{rule ccontr} \)
    assume \( A1-1: \ c\text{-assoc-have-key} \ \text{ls} \ ?k1 \neq 0 \)
    then have \( g\text{-pair} \ \text{ls key} = \ \text{ls} \) by \( \text{simp add: g-pair-def} \)
    with \( A3 \) show \( \text{False} \) by simp
  qed

  let \( ?y1 = c\text{-assoc-value} \ \text{ls} \ ?k1 \)
  from \( A2 \ S1 \) have \( S2: \ ?y1 = \ \text{univ-for-pr} \ ?k1 \) by \( \text{rule c-is-sub-fun-lm-1} \)
  let \( ?k2 = c\text{-pair} \ ?m2 \ ?x \)
  have \( S3: c\text{-assoc-have-key} \ \text{ls} \ ?k2 = 0 \)
  **proof** \( \text{rule ccontr} \)
    assume \( A3-1: \ c\text{-assoc-have-key} \ \text{ls} \ ?k2 \neq 0 \)
    then have \( g\text{-pair} \ \text{ls key} = \ \text{ls} \) by \( \text{simp add: g-pair-def Let-def} \)
    with \( A3 \) show \( \text{False} \) by simp
  qed

  let \( ?y2 = c\text{-assoc-value} \ \text{ls} \ ?k2 \)
  from \( A2 \ S3 \) have \( S4: \ ?y2 = \ \text{univ-for-pr} \ ?k2 \) by \( \text{rule c-is-sub-fun-lm-1} \)
  let \( ?z = c\text{-pair} \ ?y1 \ ?y2 \)
  from \( S2 \ S4 \) have \( S5: \ ?z = c\text{-pair} \ (\text{univ-for-pr} \ ?k1) \ (\text{univ-for-pr} \ ?k2) \) by simp
  from \( A1 \ S5 \) have \( S6: \ ?z = \ \text{univ-for-pr} \ \text{key} \) by \( \text{simp add: univ-for-pr-lm-5} \)
  from \( S1 \ S3 \ S6 \) show \( \text{thesis} \) by \( \text{simp add: g-pair-def Let-def} \)
lemma \textit{g-pair-lm-1}[: \[ c-fst (c-fst \textit{key}) \mod 7 = 5; \textit{c-is-sub-fun \textit{ls} univ-for-pr} \] \implies\textit{c-is-sub-fun} (\textit{g-pair \textit{ls} \textit{key}) univ-for-pr}]

\textbf{proof} – 
\begin{itemize}
  \item assume \textit{A1}: \( c-fst (c-fst \textit{key}) \mod 7 = 5 \)
  \item assume \textit{A2}: \textit{c-is-sub-fun \textit{ls} unie-for-pr} 
  \end{itemize}
\textit{show} \( \?thesis \)
\textbf{proof} \cases 
\begin{itemize}
  \item assume \( g-pair \textit{ls} \textit{key} = \textit{ls} \)
    \begin{itemize}
      \item with \textit{A2} \textit{show} \textit{c-is-sub-fun (g-pair \textit{ls} \textit{key}) univ-for-pr} \textbf{by} simp 
    \end{itemize}
  \item from \textit{A1} \textit{A2} \textit{this} \( S1: g-pair \textit{ls} \textit{key} = c-cons (c-pair \textit{key} (univ-for-pr \textit{key})) \)
    \begin{itemize}
      \item \textit{ls} \textbf{by} (rule \textit{g-pair-lm-0}) 
      \item with \textit{A2} \textit{show} \textit{c-is-sub-fun (g-pair \textit{ls} \textit{key}) univ-for-pr} \textbf{by} (simp add: \textit{c-is-sub-fun-lm-2}) 
    \end{itemize}
  \end{itemize}
\textbf{qed}

\textbf{qed}

\textbf{lemma} \textit{g-rec-lm-0}[: \[ c-fst (c-fst \textit{key}) \mod 7 = 6; \textit{c-is-sub-fun \textit{ls} univ-for-pr}; \textit{g-rec \textit{ls} \textit{key} \neq \textit{ls}} \] \implies\textit{g-rec \textit{ls} \textit{key} = c-cons (c-pair \textit{key} (univ-for-pr \textit{key})) \textit{ls}}]

\textbf{proof} – 
\begin{itemize}
  \item assume \textit{A1}: \( c-fst (c-fst \textit{key}) \mod 7 = 6 \)
  \item assume \textit{A2}: \textit{c-is-sub-fun \textit{ls} unie-for-pr} 
  \item assume \textit{A3}: \textit{g-rec \textit{ls} \textit{key} \neq \textit{ls}} 
  \end{itemize}
\textit{let} \( ?n = c-fst \textit{key} \)
\textit{let} \( ?x = c-snd \textit{key} \)
\textit{let} \( ?m = c-snd ?n \)
\textit{let} \( ?m1 = c-fst ?m \)
\textit{let} \( ?m2 = c-snd ?m \)
\textit{let} \( ?x1 = c-fst ?x \)
\textit{let} \( ?x = c-snd ?x \)
\textit{show} \( \?thesis \)
\textbf{proof} \cases 
\begin{itemize}
  \item assume \textit{A1-1}: \( ?y1 = 0 \)
    \begin{itemize}
      \item let \( ?k1 = c-pair \textit{m1} \textit{x1} \)
      \item \textit{have} \( S1-1: c-assoc-have-key \textit{ls} \textit{k1} = 0 \)
      \item \textbf{proof} (rule \textit{ccontr}) 
        \begin{itemize}
          \item assume \( c-assoc-have-key \textit{ls} \textit{k1} \neq 0 \)
          \item \textit{with} \textit{A1-1} \textit{have} \textit{g-rec \textit{ls} \textit{key} = \textit{ls}} \textbf{by}(simp add: \textit{g-rec-def}) 
          \item \textbf{with} \textit{A3} \textit{show} \textit{False} \textbf{by} simp 
        \end{itemize}
    \end{itemize}
  \item assume \textit{A2-1}: \( ?y1 \neq 0 \) \textit{then} \textit{have} \textit{A2-2: ?y1 > \textit{0} by simp} 
    \begin{itemize}
      \item let \( ?y2 = ?y1 - (1::\textit{nat}) \)
    \end{itemize}
\end{itemize}
\textbf{qed}
let \( ?k2 = \text{c-pair} \ ?n \ (\text{c-pair} \ ?y2 \ ?x1) \)

have S2-1: \( \text{c-assoc-have-key} \ \text{ls} \ ?k2 = 0 \)

proof (rule ccontr)
  assume \( \text{c-assoc-have-key} \ \text{ls} \ ?k2 \neq 0 \)
  with A2-1 have \( \text{g-rec} \ \text{ls} \ \text{key} = \text{ls} \) by (simp add: g-rec-def Let-def)
  with A3 show False by simp

qed

let \( ?t1 = \text{c-assoc-value} \ \text{ls} \ ?k2 \)

from A2 S2-1 have S2-2: \( ?t1 = \text{univ-for-pr} \ ?k2 \)
  by (rule c-is-sub-fun-lm-1)

let \( ?k3 = \text{c-pair} \ ?m2 \ ?t2 \)

have S2-3: \( \text{c-assoc-have-key} \ \text{ls} \ ?k3 = 0 \)

proof (rule ccontr)
  assume \( \text{c-assoc-have-key} \ \text{ls} \ ?k3 \neq 0 \)
  with A2-1 have \( \text{g-rec} \ \text{ls} \ \text{key} = \text{ls} \)
    by (simp add: g-rec-def Let-def)
  with A3 show False by simp

qed

let \( ?u = \text{c-assoc-value} \ \text{ls} \ ?k3 \)

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

proof –
  assume A1: \( \text{c-fst} \ (\text{c-fst} \ \text{key}) \mod 7 = 6 \)
  assume A2: \( \text{c-is-sub-fun} \ \text{ls} \ \text{univ-for-pr} \)

  show ?thesis
  proof cases
    assume \( \text{g-rec} \ \text{ls} \ \text{key} = \text{ls} \)
    with A2 show \( \text{c-is-sub-fun} \ (\text{g-rec} \ \text{ls} \ \text{key}) \ \text{univ-for-pr} \) by simp
  next
    assume \( \text{g-rec} \ \text{ls} \ \text{key} \neq \text{ls} \)
    from A1 A2 this have S1: \( \text{g-rec} \ \text{ls} \ \text{key} = \text{c-cons} \ (\text{c-pair} \ \text{key} \ \text{univ-for-pr} \ \text{key}) \)

    \( \text{ls} \) by (rule g-rec-lm-0)
    with A2 show \( \text{c-is-sub-fun} \ (\text{g-rec} \ \text{ls} \ \text{key}) \ \text{univ-for-pr} \) by (simp add: c-is-sub-fun-lm-2)

  qed

  qed

lemma g-step-lm-0: \( \text{c-fst} \ (\text{c-fst} \ \text{key}) \mod 7 = 0 \implies \text{g-step} \ \text{ls} \ \text{key} = \text{c-cons} \ (\text{c-pair} \ \text{key} \ 0) \)

lemma g-step-lm-1: \( \text{c-fst} \ (\text{c-fst} \ \text{key}) \mod 7 = 1 \implies \text{g-step} \ \text{ls} \ \text{key} = \text{c-cons} \ (\text{c-pair} \ \text{key} \ \text{Suc} \ (\text{c-snd} \ \text{key})) \)

qed
lemma  \textit{g-step-lm-2}: c-fst (c-fst key) \text{ mod } 7 = 2 \implies g-step \text{ ls key} = c-cons (c-pair \text{ key} (c-fst (c-snd key))) \text{ ls by (simp add: g-step-def Let-def)}

lemma  \textit{g-step-lm-3}: c-fst (c-fst key) \text{ mod } 7 = 3 \implies g-step \text{ ls key} = c-cons (c-pair \text{ key} (c-snd (c-snd key))) \text{ ls by (simp add: g-step-def Let-def)}

lemma  \textit{g-step-lm-4}: c-fst (c-fst key) \text{ mod } 7 = 4 \implies g-step \text{ ls key} = g-comp \text{ ls key} \text{ by (simp add: g-step-def)}

lemma  \textit{g-step-lm-5}: c-fst (c-fst key) \text{ mod } 7 = 5 \implies g-step \text{ ls key} = g-pair \text{ ls key} \text{ by (simp add: g-step-def)}

lemma  \textit{g-step-lm-6}: c-fst (c-fst key) \text{ mod } 7 = 6 \implies g-step \text{ ls key} = g-rec \text{ ls key} \text{ by (simp add: g-step-def)}

lemma  \textit{g-step-lm-7}: c-is-sub-fun \text{ ls univ-for-pr} \implies c-is-sub-fun (g-step \text{ ls key}) \text{ univ-for-pr}

proof –
\begin{itemize}
    \item assume \textit{A1}: c-is-sub-fun \text{ ls univ-for-pr}\n    \item let \textit{?n} = c-fst \text{ key}\n    \item let \textit{?x} = c-snd \text{ key}\n    \item let \textit{?n1} = (c-fst \text{ ?n}) \text{ mod } 7
    \item have \textit{S1}: \textit{?n1} = 0 \implies \text{ ?thesis}
        \begin{itemize}
            \item assume \textit{A}: \textit{?n1} = 0
            \item then have \textit{S1-1}: g-step \text{ ls key} = c-cons (c-pair \text{ key} 0) \text{ ls by (rule g-step-lm-0)}
                \begin{itemize}
                    \item from \textit{A} have \textit{S1-2}: \text{ univ-for-pr key} = 0 \text{ by (rule univ-for-pr-lm-0)}
                        \begin{itemize}
                            \item from \textit{A1} have \textit{S1-3}: c-is-sub-fun (c-cons (c-pair key (univ-for-pr key))) \text{ ls} \text{ univ-for-pr by (rule c-is-sub-fun-lm-2)}
                                \begin{itemize}
                                    \item from \textit{S1-3} \textit{S1-1} \textit{S1-2} show \text{ ?thesis by simp}
                                \end{itemize}
                        \end{itemize}
                \end{itemize}
        \end{itemize}
    \item qed
    \begin{itemize}
        \item have \textit{S2}: \textit{?n1} = 1 \implies \text{ ?thesis}
            \begin{itemize}
                \item assume \textit{A}: \textit{?n1} = 1
                \item then have \textit{S2-1}: g-step \text{ ls key} = c-cons (c-pair key (Suc (c-snd key))) \text{ ls by (rule g-step-lm-1)}
                    \begin{itemize}
                        \item from \textit{A} have \textit{S2-2}: univ-for-pr key = Suc (c-snd key) \text{ by (rule univ-for-pr-lm-1)}
                            \begin{itemize}
                                \item from \textit{A1} have \textit{S2-3}: c-is-sub-fun (c-cons (c-pair key (univ-for-pr key))) \text{ ls} \text{ univ-for-pr by (rule c-is-sub-fun-lm-2)}
                                        \begin{itemize}
                                            \item from \textit{S2-3} \textit{S2-1} \textit{S2-2} show \text{ ?thesis by simp}
                                        \end{itemize}
                            \end{itemize}
                    \end{itemize}
            \end{itemize}
    \item qed
    \begin{itemize}
        \item have \textit{S3}: \textit{?n1} = 2 \implies \text{ ?thesis}
            \begin{itemize}
                \item assume \textit{A}: \textit{?n1} = 2
                \item then have \textit{S2-1}: g-step \text{ ls key} = c-cons (c-pair key (c-fst (c-snd key))) \text{ ls by (rule g-step-lm-2)}
                    \begin{itemize}
                        \item from \textit{A} have \textit{S2-2}: univ-for-pr key = c-fst (c-snd key) \text{ by (rule univ-for-pr-lm-2)}
                            \begin{itemize}
                                \item from \textit{A1} have \textit{S2-3}: c-is-sub-fun (c-cons (c-pair key (univ-for-pr key))) \text{ ls} \text{ univ-for-pr by (rule c-is-sub-fun-lm-2)}
                                        \begin{itemize}
                                            \item from \textit{A1} have \textit{S2-3}: c-is-sub-fun (c-cons (c-pair key (univ-for-pr key))) \text{ ls} \text{ univ-for-pr by (rule c-is-sub-fun-lm-2)}
                                        \end{itemize}
                            \end{itemize}
                    \end{itemize}
            \end{itemize}
    \item qed
\end{itemize}
\end{itemize}
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)
  with A have S2-2: univ-for-pr key = c-snd (c-snd key) by (rule univ-for-pr-lm-3)
  from A 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-rec ls key = ls \lor c-tl (g-rec ls key) = ls by(simp add: g-rec-def Let-def)
lemma step-next: g-step ls key = ls ∨ c-tl (g-step ls key) = ls
apply(simp add: g-step-def comp-next pair-next rec-next Let-def)
done

lemma lm1: pr-gr (Suc x) = pr-gr x ∨ c-tl (pr-gr (Suc x)) = pr-gr x by(simp add: pr-gr-at-Suc step-next)

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

lemma lm2: c-assoc-have-key (c-tl ls) key = 0 ⇒ c-assoc-have-key ls key = 0
proof
  assume A1: c-assoc-have-key (c-tl ls) key = 0
  from A1 have S1: c-tl ls > 0 by (rule c-assoc-have-key-pos)
  have S2: c-tl ls ≤ ls by (rule c-tl-le)
  from S1 S2 have S3: ls ≠ 0 by auto
  from A1 S3 show ?thesis by (auto simp add: c-assoc-have-key-lm-1)
qed

lemma lm3: c-assoc-have-key (pr-gr x) key = 0 ⇒ c-assoc-have-key (pr-gr (Suc x)) key = 0
proof
  assume A1: c-assoc-have-key (pr-gr x) key = 0
  have S1: pr-gr (Suc x) = pr-gr x ∨ c-tl (pr-gr (Suc x)) = pr-gr x by (rule lm1)
  from A1 have S2: pr-gr (Suc x) = pr-gr x ⇒ ?thesis by auto
  have S3: c-tl (pr-gr (Suc x)) = pr-gr x ⇒ ?thesis
  proof
    assume c-tl (pr-gr (Suc x)) = pr-gr x (is c-tl ?ls = -)
    with A1 have c-assoc-have-key (c-tl ?ls) key = 0 by auto
    then show c-assoc-have-key ?ls key = 0 by (rule lm2)
  qed
  from S1 S2 S3 show ?thesis by auto
qed

lemma lm4: c-assoc-have-key (pr-gr (x+y)) key = 0
apply (induct-tac y)
apply (auto)
apply (simp add: lm3)
done

lemma lm5: \[
\{ \text{c-assoc-have-key (pr-gr x) key = 0; } x \leq y \} \implies \text{c-assoc-have-key (pr-gr y) key = 0}
\]
proof –
  assume A1: c-assoc-have-key (pr-gr x) key = 0
  assume A2: x \leq y
  let \(?z = y - x\)
  from A2 have S1: 0 \leq \(?z by auto
  from A2 have S2: y = x + \(?z by auto
  from A1 S1 have S3: c-assoc-have-key (pr-gr (x+\(?z)) key = 0 by (rule lm4)
  from S2 S3 show \(?thesis by auto
qed

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

lemma loc-upb-lm-2: (c-fst n) mod 7 > 1 \implies c-snd n < n
proof –
  assume A1: c-fst n mod 7 > 1
  from A1 have S1: 1 < c-fst n by simp
  have S2: c-fst n \leq n by (rule c-fst-le-arg)
  from S1 S2 have S3: 1 < n by simp
  from S3 have S4: n > 1 by simp
  from S4 show \(?thesis by (rule c-snd-less-arg)
qed

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

lemma loc-upb-lm-2-2: (c-fst n) mod 7 = 4 \implies c-snd (c-snd n) < n
proof
  assume A1: c-fst n mod 7 = 4
  then have S0: c-fst n mod 7 > 1 by auto
  then have S1: c-snd n < n by (rule loc-upb-lm-2)
  have S2: c-snd (c-snd n) \leq c-snd n by (rule c-snd-le-arg)
  from S1 S2 show c-snd (c-snd n) < n by auto
qed
lemma loc-upb-lm-2-3: \((c\text{-}\text{fst } n) \mod 7 = 5 \rightarrow c\text{-}\text{fst } (c\text{-}\text{snd } n) < n\)
proof
assume A1: \(c\text{-}\text{fst } n \mod 7 = 5\)
then have S0: \(c\text{-}\text{fst } n \mod 7 > 1\) by auto
then have S1: \(c\text{-}\text{snd } n < n\) by (rule loc-upb-lm-2)
have S2: \(c\text{-}\text{fst } (c\text{-}\text{snd } n) \leq c\text{-}\text{snd } n\) by (rule c-fst-le-arg)
from S1 S2 show \(c\text{-}\text{fst } (c\text{-}\text{snd } n) < n\) by auto
qed

lemma loc-upb-lm-2-4: \((c\text{-}\text{fst } n) \mod 7 = 5 \rightarrow c\text{-}\text{snd } (c\text{-}\text{snd } n) < n\)
proof
assume A1: \(c\text{-}\text{fst } n \mod 7 = 5\)
then have S0: \(c\text{-}\text{fst } n \mod 7 > 1\) by auto
then have S1: \(c\text{-}\text{snd } n < n\) by (rule loc-upb-lm-2)
have S2: \(c\text{-}\text{snd } (c\text{-}\text{snd } n) \leq c\text{-}\text{snd } n\) by (rule c-snd-le-arg)
from S1 S2 show \(c\text{-}\text{snd } (c\text{-}\text{snd } n) < n\) by auto
qed

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

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

lemma loc-upb-lm-2-7: \([y_2 = y_1 - (1\text{-}::\text{nat}); 0 < y_1; x_1 = c\text{-}\text{snd } x; y_1 = c\text{-}\text{fst } x]\) \(\Rightarrow c\text{-}\text{pair } y_2 x_1 < x\)
proof
assume A1: \(y_2 = y_1 - (1\text{-}::\text{nat})\) and A2: \(0 < y_1\) and A3: \(x_1 = c\text{-}\text{snd } x\) and A4: \(y_1 = c\text{-}\text{fst } x\)
from A1 A2 have S1: \(y_2 < y_1\) by auto
from S1 have S2: \(c\text{-}\text{pair } y_2 x_1 < c\text{-}\text{pair } y_1 x_1\) by (rule c-pair-strict-monotone)
from A3 A4 have S3: \(c\text{-}\text{pair } y_1 x_1 = x\) by auto
from S2 S3 show \(c\text{-}\text{pair } y_2 x_1 < x\) by auto
qed

function loc-upb :: \(\text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat}\)
where
aa: \(\text{loc-upb } n x = \{\)
let n1 = (c-fst n) mod 7 in
  if n1 = 0 then (c-pair (c-pair n x) 0) + 1 else
  if n1 = 1 then (c-pair (c-pair n x) 0) + 1 else
  if n1 = 2 then (c-pair (c-pair n x) 0) + 1 else
  if n1 = 3 then (c-pair (c-pair n x) 0) + 1 else
  if n1 = 4 then (let m = c-snd n; m1 = c-fst m; m2 = c-snd m;
    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
  ) else
  if n1 = 5 then (let m = c-snd n; m1 = c-fst m; m2 = c-snd m in
    (c-pair (c-pair n x) (loc-upb m1 x + loc-upb m2 x)) + 1
  ) else
  if n1 = 6 then (let m = c-snd n; m1 = c-fst m; m2 = c-snd m; y1 = c-snd x; x1 = c-snd x in
    if y1 = 0 then (c-pair (c-pair n x) (loc-upb m1 x1)) + 1
  ) else (let y2 = y1 - (1::nat);
    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
  )
else 0
by auto

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

definition
lex-p :: ((nat × nat) × nat × nat) set where
lex-p = ((measure (\lam m. m)) <\lex*>) (measure (\lam n. n))

lemma wf-lex-p: wf(lex-p)
apply(simp add: lex-p-def)
apply(auto)
done

lemma lex-p-eq: ((n',x'), (n,x)) ∈ lex-p = (n'<n ∨ n'=n ∧ x'<x)
apply(simp add: lex-p-def)
done

117
lemma loc-upb-lex-0: \( c\text{-fst} \, n \mod 7 = 0 \implies \text{c-assoc-have-key} \, (\text{pr-gr} \, (\text{loc-upb} \, n \, x)) \, (c\text{-pair} \, n \, x) = 0 \)

proof –
assume \( A1: c\text{-fst} \, n \mod 7 = 0 \)
let \(?\text{key} = c\text{-pair} \, n \, x \)
let \(?s = c\text{-pair} \, ?\text{key} \, 0 \)
let \(?ls = \text{pr-gr} \, ?s \)

from \( A1 \) have \( \text{loc-upb} \, n \, x = ?s + 1 \) by simp

then have \( S1: \text{pr-gr} \, (\text{loc-upb} \, n \, x) = g\text{-step} \, (\text{pr-gr} \, ?s) \, (c\text{-fst} \, ?s) \) by (simp add: pr-gr-at-Suc)

from \( A1 \) have \( S2: g\text{-step} \, ?ls \, ?\text{key} \, = c\text{-cons} \, (c\text{-pair} \, ?\text{key} \, 0) \, ?ls \) by (simp add: g-step-def)
from \( S1 \, S2 \) have \( \text{pr-gr} \, (\text{loc-upb} \, n \, x) = c\text{-cons} \, (c\text{-pair} \, ?\text{key} \, 0) \, ?ls \) by auto
thus \(?\text{thesis} \) by (simp add: c-assoc-lm-1)
qed

lemma loc-upb-lex-1: \( c\text{-fst} \, n \mod 7 = 1 \implies \text{c-assoc-have-key} \, (\text{pr-gr} \, (\text{loc-upb} \, n \, x)) \, (c\text{-pair} \, n \, x) = 0 \)

proof –
assume \( A1: c\text{-fst} \, n \mod 7 = 1 \)
let \(?\text{key} = c\text{-pair} \, n \, x \)
let \(?s = c\text{-pair} \, ?\text{key} \, 0 \)
let \(?ls = \text{pr-gr} \, ?s \)

from \( A1 \) have \( \text{loc-upb} \, n \, x = ?s + 1 \) by simp

then have \( S1: \text{pr-gr} \, (\text{loc-upb} \, n \, x) = g\text{-step} \, (\text{pr-gr} \, ?s) \, (c\text{-fst} \, ?s) \) by (simp add: pr-gr-at-Suc)

from \( A1 \) have \( S2: g\text{-step} \, ?ls \, ?\text{key} \, = c\text{-cons} \, (c\text{-pair} \, ?\text{key} \, (\text{Suc} \, x)) \, ?ls \) by (simp add: g-step-def)
from \( S1 \, S2 \) have \( \text{pr-gr} \, (\text{loc-upb} \, n \, x) = c\text{-cons} \, (c\text{-pair} \, ?\text{key} \, (\text{Suc} \, x)) \, ?ls \) by auto
thus \(?\text{thesis} \) by (simp add: c-assoc-lm-1)
qed

lemma loc-upb-lex-2: \( c\text{-fst} \, n \mod 7 = 2 \implies \text{c-assoc-have-key} \, (\text{pr-gr} \, (\text{loc-upb} \, n \, x)) \, (c\text{-pair} \, n \, x) = 0 \)

proof –
assume \( A1: c\text{-fst} \, n \mod 7 = 2 \)
let \(?\text{key} = c\text{-pair} \, n \, x \)
let \(?s = c\text{-pair} \, ?\text{key} \, 0 \)
let \(?ls = \text{pr-gr} \, ?s \)

from \( A1 \) have \( \text{loc-upb} \, n \, x = ?s + 1 \) by simp

then have \( S1: \text{pr-gr} \, (\text{loc-upb} \, n \, x) = g\text{-step} \, (\text{pr-gr} \, ?s) \, (c\text{-fst} \, ?s) \) by (simp add: pr-gr-at-Suc)

from \( A1 \) have \( S2: g\text{-step} \, ?ls \, ?\text{key} \, = c\text{-cons} \, (c\text{-pair} \, ?\text{key} \, (c\text{-fst} \, x)) \, ?ls \) by (simp add: g-step-def)
from \( S1 \, S2 \) have \( \text{pr-gr} \, (\text{loc-upb} \, n \, x) = c\text{-cons} \, (c\text{-pair} \, ?\text{key} \, (c\text{-fst} \, x)) \, ?ls \) by auto
thus \(?\text{thesis} \) by (simp add: c-assoc-lm-1)
qed

lemma loc-upb-lex-3: \( c\text{-fst} \, n \mod 7 = 3 \implies \text{c-assoc-have-key} \, (\text{pr-gr} \, (\text{loc-upb} \, n \, x)) \, (c\text{-pair} \, n \, x) = 0 \)

proof –
assume \( A1: c\text{-fst} \, n \mod 7 = 3 \)
let \(?\text{key} = c\text{-pair} \, n \, x \)
let \(?s = c\text{-pair} \, ?\text{key} \, 0 \)
let \(?ls = \text{pr-gr} \, ?s \)

from \( A1 \) have \( \text{loc-upb} \, n \, x = ?s + 1 \) by simp

then have \( S1: \text{pr-gr} \, (\text{loc-upb} \, n \, x) = g\text{-step} \, (\text{pr-gr} \, ?s) \, (c\text{-fst} \, ?s) \) by (simp add: pr-gr-at-Suc)

from \( A1 \) have \( S2: g\text{-step} \, ?ls \, ?\text{key} \, = c\text{-cons} \, (c\text{-pair} \, ?\text{key} \, (c\text{-fst} \, x)) \, ?ls \) by (simp add: g-step-def)
from \( S1 \, S2 \) have \( \text{pr-gr} \, (\text{loc-upb} \, n \, x) = c\text{-cons} \, (c\text{-pair} \, ?\text{key} \, (c\text{-fst} \, x)) \, ?ls \) by auto
thus \(?\text{thesis} \) by (simp add: c-assoc-lm-1)
qed
\[(c\text{-pair } n \ x) = 0\]

**proof**

- **assume** \(A1\): \(\text{c-fst } n \mod 7 = 3\)
- \(\text{let } ?\text{key} = c\text{-pair } n \ x\)
- \(\text{let } ?s = c\text{-pair } ?\text{key} 0\)
- \(\text{let } ?ls = pr\text{-gr } ?s\)

  from \(A1\) have \(loc\text{-upb } n \ x = ?s + 1\) by simp
  then have \(S1\): \(pr\text{-gr } (loc\text{-upb } n \ x) = g\text{-step } (pr\text{-gr } ?s)\) (c-fst \(?s\)) by (simp add: pr-gr-at-Suc)

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

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

  thus \(?\text{thesis}\) by (simp add: c-assoc-lm-1)

**qed**

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

\[c\text{-fst } n \mod 7 = 4 \implies c\text{-assoc-have-key } (pr\text{-gr } (loc\text{-upb } n \ x)) (c\text{-pair } n \ x) = 0\]

**proof**

- **assume** \(A1\): \(\bigwedge n' x'. ((n',x'), (n,x)) \in lex-p \implies c\text{-assoc-have-key } (pr\text{-gr } (loc\text{-upb } n' x')) (c\text{-pair } n' x') = 0;\)
- **assume** \(A2\): \(c\text{-fst } n \mod 7 = 4\)
- \(\text{let } ?\text{key} = c\text{-pair } n \ x\)
- \(\text{let } ?m1 = c\text{-fst } (\text{c-snd } n)\)
- \(\text{let } ?m2 = c\text{-snd } (\text{c-snd } n)\)

  define \(upb1\) where \(upb1 = loc\text{-upb } ?m2 \ x\)
  from \(A2\) have \(m2\text{-lt-n}: ?m2 < n\) by (simp add: loc-upb-lm-2-2)
  then have \(M2\): \((?(m2, x), (n,x)) \in lex-p\) by (simp add: lex-p-eq)
  with \(A1\) \(upb1\text{-def}\) have \(S1\): \(c\text{-assoc-have-key } (pr\text{-gr } upb1) (c\text{-pair } ?m2 \ x) = 0\) by auto
  from \(M2\) have \(M2': ((?m2, x), n, x) \in measure (\lambda m. m) < lex*> measure (\lambda n. n)\) by (simp add: lex-p-def)
  have \(T1\): \(\text{c-is-sub-fun } (pr\text{-gr } upb1)\) \(\text{univ-for-pr}\) by (rule pr-gr-1)
  from \(T1 \ S1\) have \(T2\): \(c\text{-assoc-value } (pr\text{-gr } upb1) (c\text{-pair } ?m2 \ x) = \text{univ-for-pr}\)
  \((c\text{-pair } ?m2 \ x)\) by (rule c-is-sub-fun-lm-1)
  define \(y\) where \(y = c\text{-assoc-value } (pr\text{-gr } upb1) (c\text{-pair } ?m2 \ x)\)
  from \(T2 \ y\text{-def}\) have \(T3\): \(y = \text{univ-for-pr } (c\text{-pair } ?m2 \ x)\) by auto

  define \(upb2\) where \(upb2 = loc\text{-upb } ?m1 \ y\)
  from \(A2\) have \(?m1 < n\) by (simp add: loc-upb-lm-2-0)
  then have \(M1\): \((?(m1, y), (n,x)) \in lex-p\) by (simp add: lex-p-eq)
  with \(A1\) have \(S2\): \(c\text{-assoc-have-key } (pr\text{-gr } (loc\text{-upb } ?m1 \ y)) (c\text{-pair } ?m1 \ y) = 0\) by auto
  from \(M1\) have \(M1': ((?m1, y), n, x) \in measure (\lambda m. m) < lex*> measure (\lambda n. n)\) by (simp add: lex-p-def)
  from \(S1 \ upb1\text{-def}\) have \(S3\): \(c\text{-assoc-have-key } (pr\text{-gr } upb1) (c\text{-pair } ?m2 \ x) = 0\) by auto

119
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 ?ls = pr-gr ?s
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) <λn. n> measure (λm. m)
by (simp add: lex-p-def)
from A2 M2 M1 have S11: loc-upb n x = (?s + 1) (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 uniq-for-pr (rule pr-gr-1)
from T-ls S19 have T-ls2: c-assoc-value ?ls (c-pair ?m2 x) = uniq-for-pr (c-pair ?m2 x) (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
  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: \( \forall n' x'. ((n',x'), (n,x)) \in \text{lex-p} \Rightarrow \text{c-assoc-have-key} (\text{pr-gr} (\text{loc-upb} n' x')) (\text{c-pair} n' x') = 0; \)
\( \text{c-fst} \ n \mod 7 = 5 \Rightarrow \text{c-assoc-have-key} (\text{pr-gr} (\text{loc-upb} n x)) (\text{c-pair} n x) = 0 \)
proof –
assume A1: \( \forall n' x'. ((n',x'), (n,x)) \in \text{lex-p} \Rightarrow \text{c-assoc-have-key} (\text{pr-gr} (\text{loc-upb} n' x')) (\text{c-pair} n' x') = 0 \)
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} by (simp add: lex-p-eq)
with A1 have S1: c-assoc-have-key (pr-gr (loc-upb ?m1 x)) (c-pair ?m1 x) = 0
by auto
from A2 have ?m2 < n by (simp add: loc-upb-lm-2-4)
then have ((?m2, x), (n,x)) \in \text{lex-p} by (simp add: lex-p-eq)
with A1 have S2: c-assoc-have-key (pr-gr (loc-upb ?m2 x)) (c-pair ?m2 x) = 0
by auto
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 (pr-gr upb1) (c-pair ?m1 x) = 0
by auto
from upb2-def S2 have S4: c-assoc-have-key (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 \leq 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: \[ c-fst n \text{ mod } 7 = 6; c-fst x = 0 \] \implies
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: \[ c-fst n \text{ mod } 7 = 6; c-fst x \neq 0 \] \implies 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: \[ \bigwedge n' x', ((n',x'), (n,x)) \in \text{lex-p} \implies c-assoc-have-key (pr-gr
(loc-upb n' x')) (c-pair n' x') = 0; \]
c-fst n \text{ mod } 7 = 6 \implies
\[ c-assoc-have-key (pr-gr (loc-upb n x)) (c-pair n x) = 0 \]
proof
assume A1: \[ \bigwedge n' x', ((n',x'), (n,x)) \in \text{lex-p} \implies c-assoc-have-key (pr-gr
(loc-upb n' x')) (c-pair n' x') = 0; \]
assume A2: c-fst n \text{ mod } 7 = 6
let ?key = c-pair n x
let ?m1 = c-fst (c-snd n)
let ?m2 = c-snd (c-snd n)
let ?y1 = c-fst x
let ?x1 = c-snd x
define upb where upb = loc-upb n x

122
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: c-assoc-have-key ?ls (c-pair ?m1 ?x1) = 0
  proof
    from A2 have ?m1 < n by (simp add: loc-upb-lm-2-5)
    then have (Suc ?m1, ?x1) ∈ lex-p by (simp add: lex-p-eq)
    with A1 upb1-def have c-assoc-have-key (pr-gr upb1) (c-pair ?m1 ?x1) = 0
    by auto
    also have upb1 ≤ ?s by (rule arg2-le-c-pair)
    ultimately show ?thesis by (rule lm5)
  qed
  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 ≠ 0 then have y1-pos: c-fst x > 0 by auto
  let ?y2 = ?y1 - 1
  from A2 A have loc-upb n x = (let m = c-snd n; m1 = c-fst m; m2 = c-snd m; y1 = c-fst x; x1 = c-snd x;
    y2 = y1 - 1;
    t1 = c-assoc-value (pr-gr (loc-upb n (c-pair y2 x1))) (c-pair n (c-pair y2 x1));
    t2 = c-pair (c-pair y2 t1) x1 in
    c-pair (c-pair n x) (loc-upb n (c-pair y2 x1) + (loc-upb m2 t2)) + 1) by (rule loc-upb-6)
    then have S1: loc-upb n x = (let t1 = c-assoc-value (pr-gr (loc-upb n (c-pair ?y2 ?x1))) (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 del: loc-upb.simps add: Let-def)
    let ?t1 = unify-for-pr (c-pair n (c-pair ?y2 ?x1))
    let ?t2 = c-pair (c-pair ?y2 ?t1) ?x1
    have S1-1: c-assoc-have-key (pr-gr (loc-upb n (c-pair ?y2 ?x1))) (c-pair n (c-pair ?y2 ?x1)) = 0

123
proof
  from A have ?y2 < ?y1 by auto
  then have c-pair ?y2 ?x1 < c-pair ?y1 ?x1 by (rule c-pair-strict-mono1)
  then have ((n, c-pair ?y2 ?x1), n, x) ∈ lex-p by (simp add: lex-p-eq)
  with A1 show ?thesis by auto
qed
have S2: c-assoc-value (pr-gr (loc-upb n (c-pair ?y2 ?x1))) (c-pair n (c-pair ?y2 ?x1)) = univ-for-pr (c-pair n (c-pair ?y2 ?x1))
proof
  have c-is-sub-fun (pr-gr (loc-upb n (c-pair ?y2 ?x1))) univ-for-pr by (rule pr-gr-1)
  with S1-1 show ?thesis by (simp add: c-is-sub-fun-lm-1)
qed
from S1 S2 have S3: loc-upb n x = c-pair (c-pair n x) (loc-upb n (c-pair ?y2 ?x1) + loc-upb ?m2 ?t2) + 1 by (simp del: loc-upb.simps add: Let-def)
let ?s = c-pair (c-pair n x) (loc-upb n (c-pair ?y2 ?x1) + loc-upb ?m2 ?t2)
from S3 have S4: pr-gr (loc-upb n x) = pr-gr (Suc ?s) by (simp del: loc-upb.simps)
  have pr-gr (Suc ?s) = g-step (pr-gr ?s) (c-fst ?s) by (rule pr-gr-at-Suc)
  with S4 have S5: pr-gr (loc-upb n x) = g-step (pr-gr ?s) ?key by (simp del: loc-upb.simps)
let ?ls = pr-gr ?s
from A2 have g-step ?ls ?key = g-rec ?ls ?key by (simp add: g-step-def)
with S5 have S6: pr-gr (loc-upb n x) = g-rec ?ls ?key by (simp del: loc-upb.simps)
have S7: c-assoc-have-key ?ls (c-pair n (c-pair ?y2 ?x1)) = 0
proof
  have loc-upb n (c-pair ?y2 ?x1) ≤ loc-upb n (c-pair ?y2 ?x1) + loc-upb ?m2 ?t2 by (auto simp del: loc-upb.simps)
  also have loc-upb n (c-pair ?y2 ?x1) + loc-upb ?m2 ?t2 ≤ ?s by (rule arg2-le-c-pair)
  ultimately have S7-1: loc-upb n (c-pair ?y2 ?x1) ≤ ?s by (auto simp del: loc-upb.simps)
  from S1-1 S7-1 show ?thesis by (rule lm5)
qed
have S8: c-assoc-value ?ls (c-pair n (c-pair ?y2 ?x1)) = ?t1
proof
  have c-is-sub-fun ?ls univ-for-pr by (rule pr-gr-1)
  with S7 show ?thesis by (simp add: c-is-sub-fun-lm-1)
qed
have S9: c-assoc-have-key ?ls (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) ∈ lex-p by (simp add: lex-p-eq)
  with A1 have c-assoc-have-key (pr-gr (loc-upb ?m2 ?t2)) (c-pair ?m2 ?t2) = 0 by auto
  also have loc-upb ?m2 ?t2 ≤ ?s by (auto simp del: loc-upb.simps)
  ultimately have S7-1: loc-upb n (c-pair ?y2 ?x1) + loc-upb ?m2 ?t2 ≤ ?s by (rule pr-gr-1)
  with S8 show ?thesis by (simp add: c-is-sub-fun-lm-1)
qed

ultimately show \(?thesis by\) (auto simp del: loc-upb.simps)

ultimately show \(?thesis by\) (rule lm5)

from A S7 S8 S9 have g-rec ?ls ?key = c-cons (c-pair ?key (c-assoc-value ?ls (c-pair ?m2 ?t2))) ?ls by (simp del: loc-upb.simps add: g-rec-def Let-def)

with S6 show \(?thesis by\) (simp add: c-assoc-lm-1)

qed

lemma wf-upb-step-0:

\[ \forall n' x'. ((n',x'), (n,x)) \in \text{lex-p} \Longrightarrow c\text{-assoc-have-key} (\text{pr-gr} (\text{loc-upb} n' x')) \]

\((c\text{-pair} n' x') = 0\]

\(c\text{-assoc-have-key} (\text{pr-gr} (\text{loc-upb} \ n \ x)) (c\text{-pair} \ n \ x) = 0\]

proof –

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

let \(?n1 = (\text{c-fst} \ n) \mod 7\)

have S1: \(?n1 = 0 \Longrightarrow \?thesis\)

proof –

assume A: \(?n1 = 0\)

thus \(?thesis by\) (rule loc-upb-lex-0)

qed

have S2: \(?n1 = 1 \Longrightarrow \?thesis\)

proof –

assume A: \(?n1 = 1\)

thus \(?thesis by\) (rule loc-upb-lex-1)

qed

have S3: \(?n1 = 2 \Longrightarrow \?thesis\)

proof –

assume A: \(?n1 = 2\)

thus \(?thesis by\) (rule loc-upb-lex-2)

qed

have S4: \(?n1 = 3 \Longrightarrow \?thesis\)

proof –

assume A: \(?n1 = 3\)

thus \(?thesis by\) (rule loc-upb-lex-3)

qed

have S5: \(?n1 = 4 \Longrightarrow \?thesis\)

proof –

assume A: \(?n1 = 4\)

from A1 A show \(?thesis by\) (rule loc-upb-lex-4)

qed

have S6: \(?n1 = 5 \Longrightarrow \?thesis\)

proof –

assume A: \(?n1 = 5\)

from A1 A show \(?thesis by\) (rule loc-upb-lex-5)

qed
have \( S7 : \neg n1 = 6 \implies \neg \text{thesis} \)  
proof  
  assume \( A : \neg n1 = 6 \)  
  from \( A \) show \( \neg \text{thesis} \) by (rule \( \text{loc-upb-lex-6} \))  
qed  

have \( S8 : \neg n1 = 0 \lor \neg n1 = 1 \lor \neg n1 = 2 \lor \neg n1 = 3 \lor \neg n1 = 4 \lor \neg n1 = 5 \lor \neg n1 = 6 \)  
by (rule \( \text{mod7-lm} \))  
from \( S1 S2 S3 S4 S5 S6 S7 S8 \) show \( \neg \text{thesis} \) by \( \text{fast} \)  
qed  

lemma \( \text{wf-upb-step} \):  
assumes \( A1 : \bigwedge p2. (p2, (p1)) \in \text{lex-p} \implies \)  
c-assoc-have-key \((\text{pr-gr} (\text{loc-upb} (\text{fst} p2) (\text{snd} p2))) (\text{c-pair} (\text{fst} p2) (\text{snd} p2))) = 0 \)  
shows \( \neg \text{c-assoc-have-key} (\text{pr-gr} (\text{loc-upb} (\text{fst} p1) (\text{snd} p1))) (\text{c-pair} (\text{fst} p1) (\text{snd} p1)) = 0 \)  
proof  
  let \( ?n = \text{fst} p1 \)  
  let \( ?x = \text{snd} p1 \)  
  from \( A1 \) have \( S1 : \bigwedge p2. ((?n, ?x), (p2)) \in \text{lex-p} \implies \)  
c-assoc-have-key \((\text{pr-gr} (\text{loc-upb} (?n' x')) (\text{c-pair} (?n' x')) = 0 \) \( \implies \)  
c-assoc-have-key \((\text{pr-gr} (\text{loc-upb} (\text{fst} p1) (\text{snd} p1))) (\text{c-pair} (\text{fst} p1) (\text{snd} p1)) = 0 \)  
  by (rule \( \text{wf-upb-step-0} \))  
  then have \( S2 : \bigwedge n' x'. (\neg \text{c-assoc-have-key} (\text{pr-gr} (\text{loc-upb} (?n' x')) (\text{c-pair} (?n' x')) = 0) \implies \)  
c-assoc-have-key \((\text{pr-gr} (\text{loc-upb} (\text{fst} p1) (\text{snd} p1))) (\text{c-pair} (\text{fst} p1) (\text{snd} p1)) = 0 \) by auto  
  have \( S3 : \bigwedge n' x'. (\neg \text{c-assoc-have-key} (\text{pr-gr} (\text{loc-upb} (?n' x')) (\text{c-pair} (?n' x')) = 0) \implies \)  
c-assoc-have-key \((\text{pr-gr} (\text{loc-upb} (?n' x')) (\text{c-pair} (?n' x')) = 0) \) by auto  
  then show \( \text{c-assoc-have-key} (\text{pr-gr} (\text{loc-upb} (n' x')) (\text{c-pair} n' x') = 0 \) by auto  
qed  

from \( S4 S3 \) show \( \neg \text{thesis} \) by \( \text{auto} \)  
qed  

theorem \( \text{loc-upb-main} \):  
c-assoc-have-key \((\text{pr-gr} (\text{loc-upb} n x)) (\text{c-pair} n x) = 0 \)  
proof  

\( \text{end of proof} \)
have \( \text{loc-upb-lm}: \forall p. \ c-\text{assoc-have-key} (\pr-gr (\text{loc-upb} (\fst p) (\snd p))) (\c-pair (\fst p) (\snd p)) = 0 \)

proof –

fix \( p \) show \( \c-\text{assoc-have-key} (\pr-gr (\text{loc-upb} (\fst p) (\snd p))) (\c-pair (\fst p) (\snd p)) = 0 \)

proof –

have \( \text{S1: wf lex-p by (auto simp add: lex-p-def)} \)

from \( \text{S1 wf-upb-step show ?thesis by (rule wf-induct-rule)} \)

qed

let \( ?p = (n, x) \)

have \( \c-\text{assoc-have-key} (\pr-gr (\text{loc-upb} (\fst ?p) (\snd ?p))) (\c-pair (\fst ?p) (\snd ?p)) = 0 \) by (rule \text{loc-upb-lm})

thus ?thesis by simp

qed

theorem \( \pr-gr-value: \ c-\text{assoc-value} (\pr-gr (\text{loc-upb} n x)) (\c-pair n x) = \text{univ-for-pr (c-pair n x)} \)

by (simp del: \text{loc-upb}.simps add: \text{loc-upb-main pr-gr-1 c-is-sub-fun-lm-1})

theorem \( \g-comp-is-pr: \ g-comp \in \text{PrimRec2} \)

proof –

from \( \c-\text{assoc-have-key-is-pr c-assoc-value-is-pr c-cons-is-pr} \) have \( (\lambda x y. g-comp x y) \in \text{PrimRec2} \)

unfolding \( g-comp-def \ Let-def \) by prec

thus ?thesis by auto

qed

theorem \( \g-pair-is-pr: \ g-pair \in \text{PrimRec2} \)

proof –

from \( \c-\text{assoc-have-key-is-pr c-assoc-value-is-pr c-cons-is-pr} \) have \( (\lambda x y. g-pair x y) \in \text{PrimRec2} \)

unfolding \( g-pair-def \ Let-def \) by prec

thus ?thesis by auto

qed

theorem \( \g-rec-is-pr: \ g-rec \in \text{PrimRec2} \)

proof –

from \( \c-\text{assoc-have-key-is-pr c-assoc-value-is-pr c-cons-is-pr} \) have \( (\lambda x y. g-rec x y) \in \text{PrimRec2} \)

unfolding \( g-rec-def \ Let-def \) by prec

thus ?thesis by auto

qed

theorem \( \g-step-is-pr: \ g-step \in \text{PrimRec2} \)

proof –

from \( g-comp-is-pr g-pair-is-pr g-rec-is-pr \) mod-is-pr \( \c-\text{assoc-have-key-is-pr c-assoc-value-is-pr c-cons-is-pr} \) have

\( (\lambda ls \text{ key}. g-step \ ls \text{ key}) \in \text{PrimRec2} \) unfolding \( g-step-def \ Let-def \) by prec

thus ?thesis by auto

qed
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 prec
    qed
  from S1 S2 show ?thesis by auto
qed

end

7 Computably enumerable sets of natural numbers

theory RecEnSet
imports PRecList PRecFun2 PRecFinSet PRecUnGr
begin

7.1 Basic definitions

definition fn-to-set :: (nat ⇒ nat ⇒ nat) ⇒ nat set where
  fn-to-set f = { x. ∃ y. f x y = 0 }

definition ce-sets :: (nat set) set where
  ce-sets = { (fn-to-set p) | p. p ∈ PrimRec2 }

7.2 Basic properties of computably enumerable sets

lemma ce-set-lm-1: p ∈ PrimRec2 ⇒ fn-to-set p ∈ ce-sets by (auto simp add: ce-sets-def)

lemma ce-set-lm-2: [ p ∈ PrimRec2; ∀ x. (x ∈ A) = (∃ y. p x y = 0) ]⇒ A ∈ ce-sets
  proof –
    assume p-is-pr: p ∈ PrimRec2
    assume ∀ x. (x ∈ A) = (∃ y. p x y = 0)
    then have A = fn-to-set p by (unfold fn-to-set-def, auto)
    with p-is-pr show A ∈ ce-sets by (simp add: ce-set-lm-1)
  qed
lemma ce-set-lm-3: \( A \in \text{ce-sets} \implies \exists\ p \in \text{PrimRec2}.\ A = \text{fn-to-set} \ p \)

proof –

assumes \( A \in \text{ce-sets} \)

then have \( A \in \{\ \text{fn-to-set} \ p \mid p.\ p \in \text{PrimRec2} \} \) by (simp add: ce-sets-def)

thus \(?\text{thesis}\) by auto

qed

lemma ce-set-lm-4: \( A \in \text{ce-sets} \implies \exists\ p \in \text{PrimRec2}.\ \forall\ x.\ (x \in A) = (\exists\ y.\ p\ x\ y = 0) \)

proof –

assumes \( A \in \text{ce-sets} \)

then have \( \exists\ p \in \text{PrimRec2}.\ A = \text{fn-to-set} \ p \) by (rule ce-set-lm-3)

then obtain \( p \) where \( \text{p-is-pr} \): \( p \in \text{PrimRec2} \) and \( \text{S1} \): \( A = \text{fn-to-set} \ p \)..

from \( \text{p-is-pr} \) \( \text{S1} \) show \(?\text{thesis}\) by (unfold fn-to-set-def, auto)

qed

lemma ce-set-lm-5: \( [\ A \in \text{ce-sets};\ p \in \text{PrimRec1} \] \implies \{\ x.\ p\ x\in\ A\} \in \text{ce-sets} \)

proof –

assumes \( A1: A \in \text{ce-sets} \)

assumes \( A2: p \in \text{PrimRec1} \)

from \( \text{A1} \) have \( \exists\ pA \in \text{PrimRec2}.\ A = \text{fn-to-set} \ pA \) by (rule ce-set-lm-3)

then obtain \( pA \) where \( pA\text{-is-pr} \): \( pA \in \text{PrimRec2} \) and \( \text{S1} \): \( A = \text{fn-to-set} \ pA \)..

from \( \text{S1} \) have \( \text{S2} \): \( A = \{\ x.\ (\exists\ y.\ pA\ x\ y = 0) \} \) by (simp add: fn-to-set-def)

define \( q \) where \( q\ x\ y = pA\ (p\ x)\ y \) for \( x\ y \)

from \( \text{pA-is-pr} \) \( \text{A2} \) have \( q\text{-is-pr} \): \( q \in \text{PrimRec2} \) unfolding \( q\text{-def} \) 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

assumes \( A: p\ x\in\ A \)

with \( \text{S2} \) obtain \( y \) where \( \text{L1} \): \( pA\ (p\ x)\ y = 0 \) by auto

then have \( q\ x\ y = 0 \) by (simp add: \( q\text{-def} \))

thus \( \exists\ y.\ q\ x\ y = 0 \) ..

next

assumes \( A: (\exists\ y.\ q\ x\ y = 0) \)

then obtain \( y \) where \( \text{L1} \): \( q\ x\ y = 0 \) ..

then have \( pA\ (p\ x)\ y = 0 \) by (simp add: \( q\text{-def} \))

with \( \text{S2} \) show \( p\ x\in\ A \) by auto

qed

qed

then have \( \{\ x.\ p\ x\in\ A\} = \{\ x.\ (\exists\ y.\ q\ x\ y = 0) \} \) by auto

then have \( \{\ x.\ p\ x\in\ A\} = \text{fn-to-set} \ q \) by (simp add: fn-to-set-def)

moreover from \( \text{q-is-pr} \) have \( \text{fn-to-set} \ q \in \text{ce-sets} \) by (rule ce-set-lm-1)

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

qed

lemma ce-set-lm-6: \( [\ A \in \text{ce-sets};\ A \neq \{\} \] \implies \exists\ q \in \text{PrimRec1}.\ A = \{\ q\ x \mid x.\ x \in \text{UNIV} \} \)

proof –
assume $A_1$: $A \in \text{ce-sets}$
assume $A_2$: $A \neq \{ \}$
from $A_1$ have $\exists \ pA \in \text{PrimRec2}. \ A = \text{fn-to-set} \ pA$ by (rule $\text{ce-set-lm-3}$)
then obtain $pA$ where $pA\text{-is-pr}: pA \in \text{PrimRec2}$ and $S1$: $A = \text{fn-to-set} \ pA$.
from $S1$ have $S2$: $A = \{ x. \ \exists \ y. \ pA \ x \ y = 0 \}$ by (simp add: $\text{fn-to-set-def}$)
from $A_2$ obtain $a$ where $\text{a-in} : a \in A$ by auto
define $q$ where $q z = (\text{if} \ pA (c\text{-fst} z) (c\text{-snd} z) = 0 \ \text{then} \ c\text{-fst} z \ \text{else} \ a)$ for $z$
from $pA\text{-is-pr}$ have $q\text{-is-pr}: q \in \text{PrimRec1}$ unfolding $q\text{-def}$ by prec
have $S3$: $\forall \ z. \ q z \in A$ proof
fix $z$ show $q z \in A$
proof
\begin{proof}
\begin{cases}
\text{assume A: } pA (c\text{-fst} z) (c\text{-snd} z) = 0 \\
with S2 have c\text{-fst} z \in A by auto
moreover from A $q\text{-def}$ have $q z = c\text{-fst} z$ by simp
ultimately show $q z \in A$ by auto
\end{cases}
\end{proof}
\end{proof}
\begin{proof}
\begin{cases}
\text{assume A: } pA (c\text{-fst} z) (c\text{-snd} z) \neq 0 \\
with q\text{-def}$ have $q z = a$ by simp
with a-in show $q z \in A$ by auto
\end{cases}
\end{proof}
qed
\end{proof}
then have $S4$: $\{ q x | x. \ x \in \text{UNIV} \} \subseteq A$ by auto
have $S5$: $A \subseteq \{ q x | x. \ x \in \text{UNIV} \}$
proof
fix $x$ assume $A$: $x \in A$ show $x \in \{ q x | x. \ x \in \text{UNIV} \}$
proof
\begin{proof}
\begin{cases}
from A $S2$ obtain $y$ where $L1$: $pA \ x \ y = 0$ by auto
let $?z = c\text{-pair} x \ y$
from $L1$ have $q \ ?z = x$ by (simp add: $q\text{-def}$)
then have $\exists \ u. \ q u = x$ by blast
then show $\exists \ u. \ x = q u \land u \in \text{UNIV}$ by auto
\end{cases}
\end{proof}
\end{proof}
\begin{proof}
\begin{cases}
from $S4 \ S5$ have $S6$: $A = \{ q x | x. \ x \in \text{UNIV} \}$ by auto
with $q\text{-is-pr}$ show $\text{thesis}$ by blast
\end{cases}
\end{proof}
\end{proof}

lemma $\text{ce-set-lm-7}$: $[ A \in \text{ce-sets}; \ p \in \text{PrimRec1}] \Longrightarrow \{ p x | x. \ x \in A \} \in \text{ce-sets}$
proof
\begin{proof}
\begin{cases}
assume $A1$: $A \in \text{ce-sets}$
assume $A2$: $p \in \text{PrimRec1}$
let $?B = \{ p \ x | x. \ x \in A \}$
fix $y$ have $S1$: $(y \in ?B) = (\exists \ x. \ x \in A \land (y = p \ x))$ by auto
from $A1$ have $\exists \ pA \in \text{PrimRec2}. \ A = \text{fn-to-set} \ pA$ by (rule $\text{ce-set-lm-3}$)
then obtain $pA$ where $pA\text{-is-pr}: pA \in \text{PrimRec2}$ and $S2$: $A = \text{fn-to-set} \ pA$.
from $S2$ have $S3$: $A = \{ x. \ \exists \ y. \ pA \ x \ y = 0 \}$ by (simp add: $\text{fn-to-set-def}$)
define $q$ where $q y t = (\text{if} \ y = p (c\text{-snd} t) \ \text{then} \ pA (c\text{-snd} t) (c\text{-fst} t) \ \text{else} \ I)$ for $y t$
\end{cases}
\end{proof}
\end{proof}
from pA-is-pr A2 have q-is-pr: q ∈ PrimRec2 unfolding q-def by prec
have L1: \( \bigwedge y. (y \in ?B) = (\exists z. q y z = 0) \)
proof - fix y show (y \in ?B) = (\exists z. q y z = 0)
proof
assume AA1: y \in ?B
then obtain x0 where LL-2: x0 \in A and LL-3: y = p x0 by auto
from S3 have LL-4: (x0 \in A) = (\exists z. pA x0 z = 0) by auto
from LL-2 LL-4 obtain z0 where LL-5: pA x0 z0 = 0 by auto
define t where t = c-pair z0 x0
from t-def q-def LL-3 LL-5 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-snd z0)
proof (rule ccontr)
assume y \#= p (c-snd z0)
with q-def LL1 have q y z0 = 1 by auto
with LL1 show False by auto
qed
from LL2 LL-1 q-def have LL3: pA (c-snd z0) (c-fst z0) = 0 by auto
with S3 have LL4: c-snd z0 \in A by auto
with LL2 show y \in {p x | x \in A} by auto
qed
qed
then have L2: \(?B = \{ y | y. \exists z. q y z = 0 \} \) by auto
with fn-to-set-def have \(?B = fn-to-set q \) by auto
with q-is-pr ce-set-lm-1 show ?thesis by auto
qed

theorem ce-empty: \{\} \in ce-sets
proof -
let \(?f = (\lambda x a. (1::nat))\)
have S1: \(?f \in PrimRec2 \) by (rule const-is-pr-2)
then have \( \forall x a. \?f x a \neq 0 \) by simp
then have \{x. \exists a. \?f x a = 0 \}={\} by auto
also have \( fn-to-set \ ?f = \ldots \) by (simp add: fn-to-set-def)
with S1 show ?thesis by (auto simp add: ce-sets-def)
qed

theorem ce-univ: UNIV \in ce-sets
proof -
let \(?f = (\lambda x a. (0::nat))\)
have S1: \(?f \in PrimRec2 \) by (rule const-is-pr-2)
then have \( \forall x a. \?f x a = 0 \) by simp
then have \{x. \exists a. \?f x a = 0 \}=UNIV by auto
also have \( fn-to-set \ ?f = \ldots \) by (simp add: fn-to-set-def)
with S1 show ?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: \( \lambda f. \in \text{PrimRec2 using const-is-pr-2 [where } \text{n=a} \by \text{prec} \)
  then have \( \forall x y. (\lambda f. x y) = (x=a \land y=0) \) by (simp add: abs-of-diff-eq)
  then have S2: \( \{ x. \exists y. \lambda f. x y = 0 \} = \{ a \} \) by auto
  have fn-to-set \( \lambda f. \exists \{ x. \exists y. \lambda f. x y = 0 \} \) by (simp add: fn-to-set-def)
  with S2 have fn-to-set \( \lambda f. \{ a \} \) by simp
  with S1 show \( \text{thesis} \) by (auto simp add: ce-sets-def)
qed

theorem ce-intersect: \[ A \in \text{ce-sets}; B \in \text{ce-sets} \] \( \implies A \cap B \in \text{ce-sets} \)
proof
  assume A1: \( A \in \text{ce-sets} \)
  then obtain p-a where S2: \( \in A \text{ PrimRec2 and } S3: A = \text{fn-to-set p-a} \)
    by (auto simp add: ce-sets-def)
  assume A2: \( B \in \text{ce-sets} \)
  then obtain p-b where S5: \( \in B \text{ PrimRec2 and } S6: B = \text{fn-to-set p-b} \)
    by (auto simp add: ce-sets-def)
  let \( \lambda p. (\lambda x y. (p-a x y) * (p-b x y)) \)
  from S2 S5 have S7: \( \lambda f. \in \text{PrimRec2 by prec} \)
  have S8: \( \forall x y. (\lambda p. x y) = ((p-a x y = 0) \lor (p-b x y = 0)) \) by simp
  let \( \lambda f. \in \text{set-def} \)
  have S9: \( \lambda C. \exists x y. \lambda p x y = 0 \) by (simp add: fn-to-set-def)
  from S3 have S10: \( A = \{ x. \exists y. p-a x y = 0 \} \) by (simp add: fn-to-set-def)
  from S6 have S11: \( B = \{ x. \exists y. p-b x y = 0 \} \) by (simp add: fn-to-set-def)
  from S10 S11 S9 S8 have S12: \( \lambda C. A \cup B \) by auto
  from S7 have \( \lambda C. \in \text{ce-sets by (auto simp add: ce-sets-def)} \)
  with S12 show \( \text{thesis by simp} \)
qed
proof
  assume \((\exists \ z. \ p\ a \ x \ z = 0) \land (\exists \ z. \ p\ b \ x \ z = 0)\)
  then obtain \(z_1 \ z_2\) where \(s\-23: \ p\ a \ x \ z_1 = 0\) and \(s\-24: \ p\ b \ x \ z_2 = 0\) by auto
  let \(?y_1\ = \ c\-pair \ z_1 \ z_2\)
  from \(s\-23\) have \(s\-25: \ p\ a \ x \ (c\-fst \ ?y_1) = 0\) by simp
  from \(s\-24\) have \(s\-27: \ p\ b \ x \ (c\-snd \ ?y_1) + p\ b \ x \ (c\-snd \ ?y_1) = 0\) by simp
  then show \(?thesis\ ..\)
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 \(?thesis\) by auto
qed

7.3 Enumeration of computably enumerable sets

definition
  \(nat\-to\-ce\-set :: \ nat \Rightarrow (nat \ set)\ where\)
  \(nat\-to\-ce\-set = (\lambda \ n. fn\-to\-set (pr\-conv\-1\-to\-2 (nat\-to\-pr \ n)))\)

lemma \(nat\-to\-ce\-set\-lm\-1: \ nat\-to\-ce\-set \ n = \{ \ x . \ \exists \ y. (nat\-to\-pr \ n) (c\-pair \ x \ y) \}
\ = \emptyset\)
proof
  have \(S1: \ nat\-to\-ce\-set \ n = fn\-to\-set (pr\-conv\-1\-to\-2 (nat\-to\-pr \ n))\) by (simp add: fn\-to\-set-def)
  then have \(S2: \ nat\-to\-ce\-set \ n = \{ \ x . \ \exists \ y. (pr\-conv\-1\-to\-2 (nat\-to\-pr \ n)) \ x \ y \ = \emptyset\\}
\) by (simp add: fn\-to\-set-def)
  have \(S3: \ \bigwedge \ x \ y. (pr\-conv\-1\-to\-2 (nat\-to\-pr \ n)) \ x \ y = (nat\-to\-pr \ n) (c\-pair \ x \ y)\)
\) by (simp add: pr\-conv\-1\-to\-2-def)
  from \(S2\ S3\) show \(?thesis\) by auto
qed

lemma \(nat\-to\-ce\-set\-into\-ce: \ nat\-to\-ce\-set \ n \in ce\-sets\)
proof
  have \(S1: \ nat\-to\-ce\-set \ n = fn\-to\-set (pr\-conv\-1\-to\-2 (nat\-to\-pr \ n))\) by (simp add: fn\-to\-set-def)
  have \((nat\-to\-pr \ n) \in PrimRec1\) by (rule nat\-to\-pr\-into\-pr)
  then have \(S2: (pr\-conv\-1\-to\-2 (nat\-to\-pr \ n)) \in PrimRec2\) 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: $A \in \text{ce-sets} \implies \exists \ n. \ A = \text{nat-to-ce-set} \ n$
proof
  assume $A \in \text{ce-sets}$
  then have $\exists \ p \in \text{PrimRec2}. \ A = \text{fn-to-set} \ p$ by (rule ce-set-lm-3)
  then obtain $p$ where $p \in \text{PrimRec2} \ \text{and} \ S1: \ A = \text{fn-to-set} \ p$ ..
define $q$ where $q = \text{pr-conv-2-to-1} \ p$
from $p-is-pr$ have $q-is-pr: \ q \in \text{PrimRec1}$ by (unfold $q$-def, rule pr-conv-2-to-1-lm)
let $?n = \text{index-of-pr} \ q$
from $q-is-pr$ have $\text{nat-to-pr} \ ?n = q$ by (rule index-of-pr-is-real)
then have $\text{nat-to-ce-set} \ ?n = A$ by auto
then have $A = \text{nat-to-ce-set} \ ?n$ by (simp add: nat-to-ce-set-def)
thus $\text{thesis}$ ..
qed

7.4 Characteristic functions

definition
  chf :: \text{nat set} \Rightarrow (\text{nat} \Rightarrow \text{nat}) — Characteristic function where
  chf = ($\lambda \ A \ x. \ \text{if} \ x \in \ A \ \text{then} \ 0 \ \text{else} \ 1$)
definition
  zero-set :: (\text{nat} \Rightarrow \text{nat}) \Rightarrow \text{nat set} where
  zero-set = ($\lambda \ f. \ \{ \ x. \ f \ x = 0 \}$)
lemma chf-lm-1 [simp]: zero-set ($\text{chf} \ A$) = $A$ by (unfold chf-def, unfold zero-set-def, simp)
lemma chf-lm-2: ($x \in \ A$) = ($\text{chf} \ A \ x = 0$) by (unfold chf-def, simp)
lemma chf-lm-3: ($x \notin \ A$) = ($\text{chf} \ A \ x = 1$) by (unfold chf-def, simp)
lemma chf-lm-4: $\text{chf} \ A \in \text{PrimRec1} \implies A \in \text{ce-sets}$
proof
  assume $A: \text{chf} \ A \in \text{PrimRec1}$
define $p$ where $p = \text{chf} \ A$
from $A \ p-def$ have $p-is-pr: \ p \in \text{PrimRec1}$ by auto
define $q$ where $q x y = p x$ for $x y :: \text{nat}$
from $p-is-pr$ have $q-is-pr: \ q \in \text{PrimRec2}$ unfolding $q$-def by prec
have $S1: \ A = \{ \ x. \ p(x) = 0 \}$
proof
  have zero-set $p = A$ by (unfold $p$-def, simp)
  thus $\text{thesis}$ by (simp add: zero-set-def)
qed

have $S2: \text{fn-to-set} \ q = \{ \ x. \ \exists \ y. \ q x y = 0 \}$ by (simp add: fn-to-set-def)
have $S3: \ \forall \ x. \ (p x = 0) = (\exists \ y. \ q x y = 0)$ by (unfold $q$-def, auto)
then have $S4: \{ \ x. \ p x = 0 \} = \{ \ x. \ \exists \ y. \ q x y = 0 \}$ by auto
with $S_1 S_2$ have $S_5$: fn-to-set $q = A$ by auto
from $q$-is-pr have fn-to-set $q \in \text{ce-sets}$ by (rule ce-set-lm-1)
with $S_5$ show ?thesis by auto
qed

lemma chf-lm-5: finite $A \implies \text{chf} A \in \text{PrimRec1}$
proof –
assume $A$: finite $A$
define $u$ where $u = \text{set-to-nat} A$
from $A$ have $S_1$: nat-to-set $u = A$ by (unfold $u$-def, rule nat-to-set-srj)
have chf $A = (\lambda x. \text{sgn2} (\text{c-in} x u))$
proof
fix $x$ show chf $A x = \text{sgn2} (\text{c-in} x u)$
proof cases
assume $A$: $x \in A$
then have $S_1$-1: chf $A x = 0$ by (simp add: chf-lm-2)
from $A S_1$ have $x \in \text{nat-to-set} u$ by auto
then have $\text{c-in} x u = 1$ by (simp add: $x$-in-$u$-eq)
with $S_1$-1 show ?thesis by simp
next
assume $A$: $x \notin A$
then have $S_1$-1: chf $A x = 1$ by (simp add: chf-def)
from $A S_1$ have $x \notin \text{nat-to-set} u$ by auto
then have $\text{c-in} x u = 0$ by (simp add: $x$-in-$u$-eq c-in-def)
with $S_1$-1 show ?thesis by simp
qed
qed
moreover from c-in-is-pr have $(\lambda x. \text{sgn2} (\text{c-in} x u)) \in \text{PrimRec1}$ by prec
ultimately show ?thesis by auto
qed

theorem ce-finite: finite $A \implies A \in \text{ce-sets}$
proof –
assume $A$: finite $A$
then have 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 :: nat set $\Rightarrow$ (nat * nat) set where
ce-set-to-rel = ($\lambda$ A. { (c-fst $x$, c-snd $x$) | $x \in A$})
definition ce-rel-to-set :: (nat * nat) set $\Rightarrow$ nat set where
ce-rel-to-set = ($\lambda$ R. { c-pair $x$ $y$ | $x$ $y$. ($x,y$) $\in$ R})
definition
ce-rels :: ((nat * nat) set) set where
cce = { R | R. ce-rel-to-set R ∈ ce-sets }

lemma ce-rel-lm-1 [simp]: ce-set-to-rel (ce-rel-to-set r) = r
proof
show ce-set-to-rel (ce-rel-to-set r) ⊆ r
proof fix z
  assume A: z ∈ ce-set-to-rel (ce-rel-to-set r)
  then obtain u where L1: u ∈ (ce-rel-to-set r) and L2: z = (c-fst u, c-snd u)
    unfolding ce-set-to-rel-def by auto
from L1 obtain x y where L3: (x,y) ∈ r and L4: u = c-pair x y
  unfolding ce-rel-to-set-def by auto
from L4 have L5: c-fst u = x by simp
from L4 have L6: c-snd u = y by simp
from L5 L6 L2 have z = (x,y) by simp
with L3 show z ∈ r by auto
qed
next
show r ⊆ ce-set-to-rel (ce-rel-to-set r)
proof fix z show z ∈ r ⇒ z ∈ ce-set-to-rel (ce-rel-to-set r)
proof --
  assume A: z ∈ r
  define x where x = fst z
  define y where y = snd z
  from x-def y-def have L1: z = (x,y) by simp
  define u where u = c-pair x y
  from A L1 u-def have L2: u ∈ ce-rel-to-set r by (unfold ce-rel-to-set-def, auto)
  from L2 L1 u-def have L3: z = (c-fst u, c-snd u) by simp
  from L2 L3 show z ∈ ce-set-to-rel (ce-rel-to-set r) by (unfold ce-set-to-rel-def, auto)
  qed
qed

lemma ce-rel-lm-2 [simp]: ce-rel-to-set (ce-set-to-rel A) = A
proof
show ce-set-to-rel (ce-set-to-rel A) ⊆ A
proof fix z show z ∈ ce-set-to-rel (ce-set-to-rel A) ⇒ z ∈ A
proof --
  assume A: z ∈ ce-set-to-rel (ce-set-to-rel A)
  then obtain x y where L1: z = c-pair x y and L2: (x,y) ∈ ce-set-to-rel A
    unfolding ce-set-to-rel-def by auto
  from L2 obtain u where L3: (x,y) = (c-fst u, c-snd u) and L4: u ∈ A
    unfolding ce-set-to-rel-def by auto
  from L3 L1 have L5: z = u by simp
  with L4 show z ∈ A by auto
  qed
  qed

136
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 \( L1 : (c-fst z, c-snd z) \in \text{ce-set-to-rel} \ A \) by (unfold \text{ce-set-to-rel}-def, auto)
define \( x \) where \( x = c-fst z \)
define \( y \) where \( y = c-snd z \)
from \( L1 \) \( x \)-def \( y \)-def have \( L2 : (x, y) \in \text{ce-rel-to-set} (\text{ce-set-to-rel} \ A) \) by (unfold \text{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 \( \text{ce-rels-def1} : \text{ce-rels} = \{ \text{ce-set-to-rel} \ A | A. A \in \text{ce-sets} \} \)
proof
show \( \text{ce-rels} \subseteq \{ \text{ce-set-to-rel} \ A | A. A \in \text{ce-sets} \} \)
proof fix \( r \) show \( r \in \text{ce-rels} \implies r \in \{ \text{ce-set-to-rel} \ A | A. A \in \text{ce-sets} \} \)
proof
assume \( A : r \in \text{ce-rels} \)
then have \( L1 : \text{ce-rel-to-set} \ r \in \text{ce-sets} \) by (unfold \( \text{ce-rels-def} \), 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 \( r \in \{ \text{ce-set-to-rel} \ A | A. A \in \text{ce-sets} \} \) by auto
qed
qed
qed

next
show \( \{ \text{ce-set-to-rel} \ A | A. A \in \text{ce-sets} \} \subseteq \text{ce-rels} \)
proof fix \( r \) show \( r \in \{ \text{ce-set-to-rel} \ A | A. A \in \text{ce-sets} \} \implies r \in \text{ce-rels} \)
proof
assume \( A : r \in \{ \text{ce-set-to-rel} \ A | A. A \in \text{ce-sets} \} \)
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 \( \text{ce-rels-def} \) by auto
qed
qed
qed

lemma \( \text{ce-rel-to-set-inj} : \text{inj} \text{ ce-rel-to-set} \)
proof (rule inj-on-inverse1)
fix \( x \) assume \( A : (x : (\text{nat} \times \text{nat}) \text{ set}) \in \text{UNIV} \) show \( \text{ce-set-to-rel} (\text{ce-rel-to-set} \ x) = x \) by (rule \text{ce-rel-lm}-1)
qed

lemma \( \text{ce-rel-to-set-srj} : \text{surj} \text{ ce-rel-to-set} \)
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-surj)
qed

lemma ce-set-to-rel-inj: inj ce-set-to-rel
proof (rule inj-on-inverseI)
  fix x assume A: (x::nat set) ∈ UNIV show ce-rel-to-set (ce-set-to-rel x) = x by (rule ce-rel-lm-2)
qed

lemma ce-set-to-rel-surj: surj ce-set-to-rel
proof (rule surjI [where ?f = ce-set-to-rel])
  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-surj)
qed

lemma ce-rel-lm-3: A ∈ ce-sets =⇒ ce-set-to-rel A ∈ ce-rels
proof –
  assume A: A ∈ ce-sets
  from A ce-rels-def1 show ?thesis by auto
qed

lemma ce-rel-lm-4: ce-set-to-rel A ∈ ce-rels =⇒ A ∈ ce-sets
proof –
  assume A: ce-set-to-rel A ∈ ce-rels
  from A show ?thesis by (unfold ce-rels-def, auto)
qed

lemma ce-rel-lm-5: (A ∈ ce-sets) = (ce-set-to-rel A ∈ ce-rels)
proof
  assume A ∈ ce-sets then show ce-set-to-rel A ∈ ce-rels by (rule ce-rel-lm-3)
next
  assume ce-set-to-rel A ∈ ce-rels then show A ∈ ce-sets by (rule ce-rel-lm-4)
qed

lemma ce-rel-lm-6: r ∈ ce-rels =⇒ ce-rel-to-set r ∈ ce-sets
proof
  \begin{itemize}
  \item assume \( A : r \in \text{ce-rels} \)
  \item then show \(?thesis by (unfold \text{ce-rels-def, auto}) \)
  \end{itemize}
qed

lemma \text{ce-rel-lm-7: ce-rel-to-set r \in ce-sets \implies r \in ce-rels} 
proof
  \begin{itemize}
  \item assume \( \text{ce-rel-to-set r} \in \text{ce-sets} \)
  \item then show \(?thesis by (unfold \text{ce-rels-def, auto}) \)
  \end{itemize}
qed

lemma \text{ce-rel-lm-8: (r \in \text{ce-rels}) = (ce-rel-to-set r \in \text{ce-sets}) by (unfold \text{ce-rels-def, auto})}

lemma \text{ce-rel-lm-9: \( (x,y) \in r \implies \text{c-pair x y} \in \text{ce-rel-to-set r} \) by (unfold \text{ce-rels-def, auto})}

lemma \text{ce-rel-lm-10: x \in A \implies (c-fst x, c-snd x) \in \text{ce-set-to-rel A} by (unfold \text{ce-set-to-rel-def, auto})}

lemma \text{ce-rel-lm-11: c-pair x y \in \text{ce-rel-to-set r} \implies (x, y) \in r \) by simp}

lemma \text{ce-rel-lm-12: (c-pair x y \in \text{ce-rel-to-set r}) = ((x,y) \in r) by simp}

lemma \text{ce-rel-lm-13: (x,y) \in \text{ce-set-to-rel A} \implies c-pair x y \in A \) by (unfold \text{ce-rels-def, auto})}

lemma \text{ce-rel-lm-14: c-pair x y \in A \implies (x,y) \in \text{ce-set-to-rel A} \) by simp}

qed
lemma ce-rel-lm-15: \((x, y) \in \text{ce-set-to-rel } A\) = \((\text{c-pair } x \; y \; \in \; A)\)
proof
  assume \((x, y) \in \text{ce-set-to-rel } A\) then show \(\text{c-pair } x \; y \; \in \; A\) by (rule ce-rel-lm-13)
next
  assume \(\text{c-pair } x \; y \; \in \; A\) then show \((x, y) \in \text{ce-set-to-rel } A\) by (rule ce-rel-lm-14)
qed

lemma ce-rel-lm-16: \(x \in \text{ce-rel-to-set } r\) = \(((\text{c-fst } x, \; \text{c-snd } x) \in r)\)
proof
  assume \(x \in \text{ce-rel-to-set } r\)
  then have \((\text{c-fst } x, \; \text{c-snd } x) \in \text{ce-set-to-rel } (\text{ce-rel-to-set } r)\) by (rule ce-rel-lm-10)
  then show ?thesis by simp
qed

lemma ce-rel-lm-17: \((\text{c-fst } x, \; \text{c-snd } x) \in \text{ce-set-to-rel } A\) = \((x \in A)\)
proof
  assume \((\text{c-fst } x, \; \text{c-snd } x) \in \text{ce-set-to-rel } A\)
  then have \(\text{c-pair } (\text{c-fst } x) \; (\text{c-snd } x) \in A\) by (rule ce-rel-lm-13)
  then show ?thesis by simp
qed

lemma ce-rel-lm-18: (((\text{c-fst } x, \; \text{c-snd } x) \in \text{ce-set-to-rel } A\)) = \((x \in A)\)
proof
  assume \(((\text{c-fst } x, \; \text{c-snd } x) \in \text{ce-set-to-rel } A\)
  then show \(x \in A\) by (rule ce-rel-lm-17)
next
  assume \(x \in A\) then show \(((\text{c-fst } x, \; \text{c-snd } x) \in \text{ce-set-to-rel } A\) by (rule ce-rel-lm-10)
qed

lemma ce-rel-lm-19: \((\text{c-fst } x, \; \text{c-snd } x) \in r\) = \((x \in \text{ce-rel-to-set } r)\)
proof
  assume \((\text{c-fst } x, \; \text{c-snd } x) \in r\)
  then have \((\text{c-fst } x, \; \text{c-snd } x) \in \text{ce-set-to-rel } (\text{ce-rel-to-set } r)\) by simp
  then show ?thesis by (rule ce-rel-lm-17)
qed

lemma ce-rel-lm-20: \(((\text{c-fst } x, \; \text{c-snd } x) \in r)\) = \((x \in \text{ce-rel-to-set } r)\)
proof
  assume \(((\text{c-fst } x, \; \text{c-snd } x) \in r)\) then show \(x \in \text{ce-rel-to-set } r\) by (rule ce-rel-lm-19)
next
  assume \(x \in \text{ce-rel-to-set } r\) then show \(((\text{c-fst } x, \; \text{c-snd } x) \in r)\) by (rule ce-rel-lm-16)
qed

lemma ce-rel-lm-21: \(r \in \text{ce-rels} \implies \exists \; p \in \text{PrimRec3.} \; \forall \; x \; y. \; \((x, y) \in r\) = \((\exists \; u. \; p \; x \; y \; u \; = \; 0)\)\)
proof
define \(A \) where \(A = \text{ce-rel-to-set } r\)
from r-ce have A-ce: \(A \in \text{ce-sets}\) by (unfold A-def, rule ce-rel-lm-6)
then have \(\exists \; p \in \text{PrimRec2.} \; A = \text{fin-to-set } p\) by (rule ce-set-lm-3)
then obtain $q$ where $q$-is-pr: $q \in \text{PrimRec2}$ and $A$-def1: $A = \text{fn-to-set} \ q ..$

from $A$-def1 have $A$-def2: $A = \{ x. \exists y. q \ x \ y = 0 \}$ by (unfold \ fn-to-set-def)

define $p$ where $p \ x \ y = q \ (c \text{-pair} \ x \ y) \ u \ for \ x \ y \ u$

from $q$-is-pr have $p$-is-pr: $p \in \text{PrimRec3}$ unfolding $p$-def by prec

have $\exists \ x \ y. (x,y) \in r = (\exists \ u. p \ x \ y \ u = 0)$

proof - fix $x \ y$ show $(x,y) \in r = (\exists \ u. p \ x \ y \ u = 0)$

proof
assume $A$: $(x,y) \in r$

define $z$ where $z = c \text{-pair} \ x \ y$

with $A$-def $A$ have $z$-in-$A$: $z \in A$ by (unfold ce-rel-to-set-def, auto)

with $A$-def2 have $z \in \{ x. \exists y. q \ x \ y = 0 \}$ by auto

then obtain $u$ where $q \ z \ u = 0$ 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 $z = c \text{-pair} \ x \ y$

from $A$ obtain $u$ where $p \ x \ y \ u = 0$ by auto

then have $q$-z: $q \ z \ u = 0$ by (simp add: $z$-def $p$-def)

with $A$-def2 have $z$-in-$A$: $z \in A$ by auto

then have $c \text{-pair} \ x \ y \in A$ by (unfold $z$-def)

then have $c \text{-pair} \ x \ y \in \text{ce-rel-to-set} \ r$ by (unfold $A$-def)

then show $(x,y) \in r$ by (rule ce-rel-lm-11)

qed

qed with $p$-is-pr show $\exists$thesis by auto

qed

lemma ce-rel-lm-22: $r \in \text{ce-rels} \Rightarrow \exists \ p \in \text{PrimRec3}. \ r = \{ (x,y). \exists \ u. p \ x \ y \ u = 0 \}$

proof -
assume $r$-ce: $r \in \text{ce-rels}$

then have $\exists \ p \in \text{PrimRec3}. \ \forall \ x \ y. ((x,y) \in r) = (\exists \ u. p \ x \ y \ u = 0)$ by (rule ce-rel-lm-21)

then obtain $p$ where $p$-is-pr: $p \in \text{PrimRec3 \ and \ L1}: \forall \ x \ y. ((x,y) \in r) = (\exists \ u. p \ x \ y \ u = 0)$ by auto

from $p$-is-pr L1 show $\exists$thesis by blast

qed

lemma ce-rel-lm-23: $[ p \in \text{PrimRec3}; \ \forall \ x \ y. ((x,y) \in r) = (\exists \ u. p \ x \ y \ u = 0) ]$

$\Rightarrow \ r \in \text{ce-rels}$

proof -
assume $p$-is-pr: $p \in \text{PrimRec3}$
assume $A$: $\forall \ x \ y. ((x,y) \in r) = (\exists \ u. p \ x \ y \ u = 0)$

define $q$ where $q \ z \ u = p \ (c\text{-fst} \ z) \ (c\text{-snd} \ z) \ u \ for \ z \ u$

from $p$-is-pr have $q$-is-pr: $q \in \text{PrimRec2}$ unfolding $q$-def by prec

define $A$ where $A = \{ x. \exists y. q \ x \ y = 0 \}$

then have $A$-def1: $A = \text{fn-to-set} \ q \$ by (unfold fn-to-set-def, auto)

from $q$-is-pr $A$-def1 have $A$-ce: $A \in \text{ce-sets}$ by (simp add: ce-set-lm-1)

141
have main: \( A = \text{ce-rel-to-set } r \)
proof
  show \( A \subseteq \text{ce-rel-to-set } r \)
  proof
    fix \( z \) assume \( z \in A \)
    show \( z \in \text{ce-rel-to-set } r \)
    proof
      define \( x \) where \( x = \text{c-fst } z \)
      define \( y \) where \( y = \text{c-snd } z \)
      from \( z \in A \) A-def obtain \( u \) where \( q z u = 0 \) by auto
      with \( x \text{-def } y \text{-def } q \text{-def } u \text{-def } \)
      have \( \exists u. \ p x y u = 0 \) by auto
      with \( A \) have \( (x,y) \in r \) by auto
      then have \( \exists u. \ p x y u = 0 \) by auto
      with \( A \) have \( c-pair x y \in \text{ce-rel-to-set } r \) by \( \text{rule ce-rel-lm-9} \)
      with \( x \text{-def } y \text{-def } \) show ?thesis by simp
    qed
  qed
next
  show \( \text{ce-rel-to-set } r \subseteq A \)
  proof
    fix \( z \) assume \( z \in \text{ce-rel-to-set } r \)
    show \( z \in A \)
    proof
      define \( x \) where \( x = \text{c-fst } z \)
      define \( y \) where \( y = \text{c-snd } z \)
      from \( z \in r \) have \( (\text{c-fst } z, \text{c-snd } z) \in r \) by \( \text{rule ce-rel-lm-16} \)
      with \( x \text{-def } y \text{-def } \)
      have \( (x,y) \in r \) by simp
      with \( A \) obtain \( u \) where \( L1: \ p x y u = 0 \) by auto
      with \( x \text{-def } y \text{-def } q \text{-def } u \text{-def } \)
      have \( q z u = 0 \) by simp
      with \( A \text{-def } \) show \( z \in A \) by auto
    qed
  qed
  qed
with \( A \text{-ce } \)
have \( \text{ce-rel-to-set } r \in \text{ce-sets } \) by auto
then show \( r \in \text{ce-rels } \) by \( \text{rule ce-rel-lm-7} \)
qed

lemma ce-rel-lm-24: \[ \forall r \in \text{ce-rels}; \ s \in \text{ce-rels } \implies s \ O \ r \in \text{ce-rels} \]
proof
  assume \( r \text{-ce } r \in \text{ce-rels } \)
  assume \( s \text{-ce } s \in \text{ce-rels } \)
  from \( r \text{-ce } \) have \( \exists p \in \text{PrimRec3. } \forall x y. ((x,y) \in r) = (\exists u. \ p x y u = 0) \) by \( \text{rule ce-rel-lm-21} \)
  then obtain \( p-r \) where \( p-r \text{-is-pr: } p-r \in \text{PrimRec3 and } R1: \forall x y. ((x,y) \in r) = (\exists u. \ p-r x y u = 0) \) by auto
  from \( s \text{-ce } \) have \( \exists p \in \text{PrimRec3. } \forall x y. ((x,y) \in s) = (\exists u. \ p x y u = 0) \) by \( \text{rule ce-rel-lm-21} \)
  then obtain \( p-s \) where \( p-s \text{-is-pr: } p-s \in \text{PrimRec3 and } S1: \forall x y. ((x,y) \in \)}
\[(\exists \ u. \ p \cdot s \ x \ y \ u = 0)\]

by auto

define \( p \) where \( p \cdot x \cdot z \cdot u = (p \cdot s \ x \ (c \cdot \text{fst} \ u) \ (c \cdot \text{fst} \ (c \cdot \text{snd} \ u))) + (p \cdot r \ (c \cdot \text{fst} \ u) \ z \ (c \cdot \text{snd} \ (c \cdot \text{snd} \ u)))\)

for \( x \cdot z \cdot u \)

from \( p \cdot r \cdot \text{is-pr} \) \( p \cdot s \cdot \text{is-pr} \) have \( p \cdot \text{is-pr} \)

proof (rule allI, rule allI)

fix \( x \cdot z \)

show \((x, z) \in s \cdot O \cdot r\) = \((\exists u. \ p \cdot x \cdot z \cdot u = 0)\)

proof

assume \( A: (x, z) \in s \cdot O \cdot r\)

show \((\exists u. \ p \cdot x \cdot z \cdot u = 0)\)

proof

from \( A \) \( s \cdot O \cdot r \)-def obtain \( y \) where \( L1: (x, y) \in s \) and \( L2: (y, z) \in r \) by auto

from \( L1 \) \( S1 \)-obtain \( u \cdot s \) where \( L3: p \cdot s \cdot x \cdot y \cdot u \cdot s = 0 \) by auto

from \( L2 \) \( R1 \)-obtain \( u \cdot r \) where \( L4: p \cdot r \cdot y \cdot z \cdot u \cdot r = 0 \) by auto

define \( u \) where \( u = c \cdot \text{pair} \ y \ (c \cdot \text{pair} \ u \cdot s \cdot u \cdot r)\)

from \( L3 \) \( L4 \) have \( p \cdot x \cdot z \cdot u = 0 \) by (unfold \( p \)-def, unfold \( u \)-def, simp)

then show ?thesis by auto

qed

next

assume \( A: \exists u. \ p \cdot x \cdot z \cdot u = 0\)

show \((x, z) \in s \cdot O \cdot r\)

proof

from \( A \) obtain \( u \) where \( L1: p \cdot x \cdot z \cdot u = 0 \) by auto

then have \( L2: (p \cdot s \cdot x \cdot (c \cdot \text{fst} \ u) \ (c \cdot \text{fst} \ (c \cdot \text{snd} \ u))) + (p \cdot r \ (c \cdot \text{fst} \ u) \ z \ (c \cdot \text{snd} \ (c \cdot \text{snd} \ u))) = 0 \) by (unfold \( p \)-def)

from \( L2 \) have \( L3: p \cdot s \cdot x \cdot (c \cdot \text{fst} \ u) \ (c \cdot \text{fst} \ (c \cdot \text{snd} \ u)) = 0 \) by auto

from \( L2 \) have \( L4: p \cdot r \ (c \cdot \text{fst} \ u) \ z \ (c \cdot \text{snd} \ (c \cdot \text{snd} \ u)) = 0 \) by auto

from \( L3 \) \( S1 \)-have \( L5: (x, (c \cdot \text{fst} \ u)) \in s \) by auto

from \( L4 \) \( R1 \)-have \( L6: ((c \cdot \text{fst} \ u), z) \in r \) by auto

from \( L5 \) \( L6 \) have \((x, z) \in s \cdot O \cdot r\) by auto

with \( s \cdot O \cdot r \)-def show ?thesis by auto

qed

qed

qed

from \( p \cdot \text{is-pr} \) \( \text{main} \) have \( s \in \text{ce-rels} \) by (rule \( \text{ce-rel-lm-23} \))

then show ?thesis by (unfold \( s \)-def)

qed

lemma \( \text{ce-rel-lm-25} \): \( r \in \text{ce-rels} \implies r^{\sim} \cdot 1 \in \text{ce-rels} \)

proof

assume \( r \cdot \text{ce}: r \in \text{ce-rels} \)

have \( r^{\sim} \cdot 1 = \{(y, x). \ (x, y) \in r\} \) by auto

then have \( L1: \forall x \cdot y. \ ((x, y) \in r) = ((y, x) \in r^{\sim} \cdot 1) \) by auto

from \( r \cdot \text{ce} \) have \( \exists p \in \text{PrimRec3}. \ \forall x \cdot y. \ ((x, y) \in r) = (\exists u. \ p \cdot x \cdot y \cdot u = 0) \) by (rule \( \text{ce-rel-lm-21} \))

143
then obtain \( p \) where \( p \text{-is-pr}: p \in \text{PrimRec3} \) and \( R1: \forall \ x \ y. ((x,y) \in r) = (\exists u. p x y u = 0) \) by auto

define \( q \) where \( q x y u = p y x u \) for \( x y u \)

from \( p \text{-is-pr} \) have \( q \text{-is-pr}: q \in \text{PrimRec3} \) unfolding \( q \text{-def} \) by prec

from \( L1 \) \( R1 \) have \( L2: \forall \ x y. ((x,y) \in r^-1) = (\exists u. p x y u = 0) \) by auto

with \( q \text{-def} \) have \( L3: \forall \ x y. ((x,y) \in r^-1) = (\exists u. q x y u = 0) \) by auto

with \( q \text{-is-pr} \) show \( \text{thesis} \) by (rule \( \text{ce-rel-lm-23} \))

qed

\textbf{lemma} \( \text{ce-rel-lm-26}: r \in \text{ce-rels} \implies \text{Domain } r \in \text{ce-sets} \)

\textbf{proof} –

assume \( r \text{-ce}: r \in \text{ce-rels} \)

have \( L1: \forall \ x. (x \in \text{Domain } r) = (\exists y. (x,y) \in r) \) by auto

define \( A \) where \( A = \text{ce-rel-to-set } r \)

from \( r \text{-ce} \) have \( \text{ce-rel-to-set } r \in \text{ce-sets} \) by (rule \( \text{ce-rel-lm-6} \))

then have \( A \text{-ce}: A \in \text{ce-sets} \) by (unfold \( A \text{-def} \))

have \( \forall x y. ((x,y) \in r) = (\text{c-pair } x y \in \text{ce-rel-to-set } r) \) by (simp add: \( \text{ce-rel-lm-12} \))

then have \( L2: \forall \ x y. ((x,y) \in r) = (\text{c-pair } x y \in A) \) by (unfold \( A \text{-def} \))

from \( A \text{-ce } c\text{-fst-is-pr} \) have \( L3: \{ \text{c-fst } z \mid z \in A \} \in \text{ce-sets} \) by (rule \( \text{ce-set-lm-7} \))

have \( L4: \forall x. (x \in \{ \text{c-fst } z \mid z \in A \}) = (\exists y. \text{c-pair } x y \in A) \)

proof fix \( x \) show \( (x \in \{ \text{c-fst } z \mid z \in A \}) = (\exists y. \text{c-pair } x y \in A) \)

proof

assume \( A: x \in \{ \text{c-fst } z \mid z \in A \} \)

then obtain \( z \) where \( z \text{-in- } A: z \in A \) and \( x\text{-z: } x = \text{c-fst } z \) by auto

from \( x\text{-z} \) have \( z = \text{c-pair } x (\text{c-snd } z) \) by simp

with \( z \text{-in- } A \) have \( \text{c-pair } x (\text{c-snd } z) \in A \) by auto

then show \( \exists y. \text{c-pair } x y \in A \) by auto

next

assume \( A: \exists y. \text{c-pair } x y \in A \)

then obtain \( y \) where \( y\text{-1: } \text{c-pair } x y \in A \) by auto

define \( z \) where \( z = \text{c-pair } x y \)

from \( y\text{-1} \) have \( z\text{-in- } A: z \in A \) by (unfold \( z\text{-def} \))

from \( z\text{-def} \) have \( x\text{-z: } x = \text{c-fst } z \) by (unfold \( z\text{-def} \), simp)

from \( z\text{-in- } A \) \( x\text{-z} \) show \( x \in \{ \text{c-fst } z \mid z \in A \} \) by auto

qed

qed

from \( L1 \) \( L2 \) have \( L5: \forall x. (x \in \text{Domain } r) = (\exists y. \text{c-pair } x y \in A) \) by auto

from \( L4 \) \( L5 \) have \( L6: \forall x. (x \in \text{Domain } r) = (x \in \{ \text{c-fst } z \mid z \in A \}) \) by auto

then have \( \text{Domain } r = \{ \text{c-fst } z \mid z \in A \} \) by auto

with \( L3 \) show \( \text{Domain } r \in \text{ce-sets} \) by auto

qed

\textbf{lemma} \( \text{ce-rel-lm-27}: r \in \text{ce-rels} \implies \text{Range } r \in \text{ce-sets} \)

\textbf{proof} –

assume \( r \text{-ce}: r \in \text{ce-rels} \)

then have \( r^-1 \in \text{ce-rels} \) by (rule \( \text{ce-rel-lm-25} \))

then have \( \text{Domain } (r^-1) \in \text{ce-sets} \) by (rule \( \text{ce-rel-lm-26} \))

then show \( \text{thesis} \) by (unfold \( \text{Domain-converse} \) \{\text{symmetric}\})

144
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: Domain \( r \in \text{ce-sets} \) by (rule ce-rel-lm-26)
  from r-ce have L2: Range \( r \in \text{ce-sets} \) by (rule ce-rel-lm-27)
  from L1 L2 have L3: 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,0::nat) \mid x \in A \}
  define r-b where r-b = \{ (0::nat),z \mid z \in B \}
  have L1: r-a O r-b = A \times B by (unfold r-a-def, unfold r-b-def, auto)
  have r-a-ce: r-a \in \text{ce-rels}
  proof -
    have loc1: ce-rel-to-set r-a = \{ c-pair x 0 \mid x \in A \} by (unfold r-a-def, unfold ce-rel-to-set-def, auto)
    define p where p x = c-pair x 0 for x
    have p-is-pr: p \in PrimRec1 unfolding p-def by prec
    from A-ce p-is-pr have \{ c-pair x 0 \mid x \in A \} \in \text{ce-sets}
    unfolding p-def by (simp add: ce-set-lm-7)
    with loc1 have 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: ce-rel-to-set r-b = \{ c-pair 0 z \mid z \in B \}
    by (unfold r-b-def, unfold ce-rel-to-set-def, auto)
    define p where p z = c-pair 0 z for z
    have p-is-pr: p \in PrimRec1 unfolding p-def by prec
    from B-ce p-is-pr have \{ c-pair 0 z \mid z \in B \} \in \text{ce-sets}
    unfolding p-def by (simp add: ce-set-lm-7)
    with loc1 have 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 O r-b \in \text{ce-rels} by (rule ce-rel-lm-24)
  with L1 show ?thesis by auto
qed

lemma ce-rel-lm-30: {} \in \text{ce-rels}
proof -
  have ce-rel-to-set {} = {} by (unfold ce-rel-to-set-def, auto)
  with ce-empty have ce-rel-to-set {} \in \text{ce-sets} by auto
  then show ?thesis by (rule ce-rel-lm-7)
qed

lemma \textit{ce-rel-lm-31}: UNIV $\in$ ce-rels
proof -
  from ce-univ ce-univ have UNIV $\times$ UNIV $\in$ ce-rels by (rule ce-rel-lm-29)
  then show ?thesis by auto
qed

lemma \textit{ce-rel-lm-32}: \(ce-rel-to-set\ (r \cup s) = (ce-rel-to-set r) \cup (ce-rel-to-set s)\) by
  (unfold ce-rel-to-set-def, auto)

lemma \textit{ce-rel-lm-33}: \(\{r \in ce-rels; s \in ce-rels\} \implies r \cup s \in ce-rels\)
proof -
  assume \(r \in ce-rels\)
  then have \(r-ce\): \(ce-rel-to-set r \in ce-sets\) by (rule ce-rel-lm-6)
  assume \(s \in ce-rels\)
  then have \(s-ce\): \(ce-rel-to-set s \in ce-sets\) by (rule ce-rel-lm-6)
  have \(ce-rel-to-set\ (r \cup s) = (ce-rel-to-set r) \cup (ce-rel-to-set s)\) by (unfold ce-rel-to-set-def, auto)
    moreover from \(r-ce\ s-ce\) have \((ce-rel-to-set r) \cup (ce-rel-to-set s) \in ce-sets\) by (rule ce-union)
    ultimately have \(ce-rel-to-set\ (r \cup s) \in ce-sets\) by auto
  then show ?thesis by (rule ce-rel-lm-7)
qed

lemma \textit{ce-rel-lm-34}: \(ce-rel-to-set\ (r \cap s) = (ce-rel-to-set r) \cap (ce-rel-to-set s)\)
proof -
  show \(ce-rel-to-set\ (r \cap s) \subseteq ce-rel-to-set r \cap ce-rel-to-set s\) by (unfold ce-rel-to-set-def, auto)
next
  show \(ce-rel-to-set\ (r \cap s) \subseteq ce-rel-to-set (r \cap s)\)
proof fix \(x\) assume \(A\): \(x \in ce-rels \cap ce-rels\)
    from \(A\) have \(L1\): \(x \in ce-rel-to-set r \cap ce-rel-to-set s\) by auto
    from \(A\) have \(L2\): \(x \in ce-rel-to-set s\) by auto
    from \(L1\) obtain \(u v\) where \(L3\): \((u,v) \in r\) and \(L4\): \(x = c\-pair\ u v\)
      unfolding ce-rel-to-set-def by auto
from \(L2\) obtain \(u1 v1\) where \(L5\): \((u1,v1) \in s\) and \(L6\): \(x = c\-pair\ u1\ v1\)
    unfolding ce-rel-to-set-def by auto
from \(L4\ L6\) have \(L7\): \(c\-pair\ u1 v1 = c\-pair\ u v\) by auto
    then have \(u1 = u\) by (rule c-pair-inj1)
moreover from \(L7\) have \(v1 = v\) by (rule c-pair-inj2)
    ultimately have \((u,v) = (u1,v1)\) by auto
    with \(L3\ L5\) have \((u,v) \in r \cap s\) by auto
    with \(L4\) show \(x \in ce-rel-to-set (r \cap s)\) by (unfold ce-rel-to-set-def, auto)
  qed
  qed

lemma \textit{ce-rel-lm-35}: \(\{r \in ce-rels; s \in ce-rels\} \implies r \cap s \in ce-rels\)
proof -

assume $r \in \text{ce-rels}$
then have $r$-ce: $\text{ce-rel-to-set} \ r \in \text{ce-sets}$ by (rule ce-rel-lm-6)
assume $s \in \text{ce-rels}$
then have $s$-ce: $\text{ce-rel-to-set} \ s \in \text{ce-sets}$ by (rule ce-rel-lm-6)

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 $r$-ce $s$-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 ?thesis by (rule ce-rel-lm-7)
qed

lemma ce-rel-lm-36: $\text{ce-set-to-rel} \ (A \cup B) = (\text{ce-set-to-rel} \ A) \cup (\text{ce-set-to-rel} \ B)$
by (unfold ce-set-to-rel-def, auto)

lemma ce-rel-lm-37: $\text{ce-set-to-rel} \ (A \cap B) = (\text{ce-set-to-rel} \ A) \cap (\text{ce-set-to-rel} \ B)$
proof –
  define $f$ where $f \ x = (\text{c-fst} \ x, \text{c-snd} \ x)$ for $x$
  have $f$-inj: inj $f$
  proof (unfold $f$-def, rule inj-on-inverseI [where $g = \lambda (u,v). \text{c-pair} \ u \ v$])
    fix $x :: \text{nat}$
    assume $x \in \text{UNIV}$
    show case-prod $\text{c-pair} \ (\text{c-fst} \ x, \text{c-snd} \ x) = x$ by simp
  qed
  from $f$-inj have $f' \ (A \cap B) = f' \ A \cap f' \ B$ by (rule image-Int)
  then show ?thesis by (unfold $f$-def, unfold ce-set-to-rel-def, auto)
qed

lemma ce-rel-lm-38: $\ [ r \in \text{ce-rels}; \ A \in \text{ce-sets} \ ] \implies r''A \in \text{ce-sets}$
proof –
  assume $r$-ce: $r \in \text{ce-rels}$
  assume $A$-ce: $A \in \text{ce-sets}$
  have $L1$: $r''A = \text{Range} \ (r \cap A \times \text{UNIV})$ by blast
  have $L2$: $\text{Range} \ (r \cap A \times \text{UNIV}) \in \text{ce-sets}$
  proof (rule ce-rel-lm-27)
    show $r \cap A \times \text{UNIV} \in \text{ce-rels}$
    proof (rule ce-rel-lm-35)
      show $r \in \text{ce-rels}$ by (rule $r$-ce)
    next
    show $A \times \text{UNIV} \in \text{ce-rels}$
    proof (rule ce-rel-lm-29)
      show $A \in \text{ce-sets}$ by (rule $A$-ce)
    next
    show $\text{UNIV} \in \text{ce-sets}$ by (rule ce-univ)
    qed
    qed
  qed
  from $L1 \ L2$ show ?thesis by auto
qed
7.6 Total computable functions

definition
graph :: (nat ⇒ nat) ⇒ (nat × nat) set where
graph = (λ f. { (x, f x) | x. x ∈ UNIV})

lemma graph-lm-1: (x,y) ∈ graph f ⇒ y = f x by (unfold graph-def, auto)

lemma graph-lm-2: y = f x ⇒ (x,y) ∈ graph f by (unfold graph-def, auto)

lemma graph-lm-3: ((x,y) ∈ graph f) = (y = f x) by (unfold graph-def, auto)

lemma graph-lm-4: graph (f o g) = graph g O graph f by (unfold graph-def, auto)

definition
c-graph :: (nat ⇒ nat) ⇒ nat set where
c-graph = (λ f. { c-pair x (f x) | x. x ∈ UNIV})

lemma c-graph-lm-1: c-pair x y ∈ c-graph f ⇒ y = f x
proof
  assume A: c-pair x y ∈ c-graph f
  have S1: c-graph f = { c-pair x (f x) | x. x ∈ UNIV} by (simp add: c-graph-def)
  from A S1 obtain z where S2: c-pair x y = c-pair z (f z) by auto
  moreover from S2 have y = f z by (rule c-pair-inj1)
  ultimately show ?thesis by auto
qed

lemma c-graph-lm-2: y = f x ⇒ c-pair x y ∈ c-graph f by (unfold c-graph-def, auto)

lemma c-graph-lm-3: (c-pair x y ∈ c-graph f) = (y = f x)
proof
  assume c-pair x y ∈ c-graph f then show y = f x by (rule c-graph-lm-1)
next
  assume y = f x then show c-pair x y ∈ c-graph f by (rule c-graph-lm-2)
qed

lemma c-graph-lm-4: c-graph f = ce-rel-to-set (graph f) by (unfold c-graph-def ce-rel-to-set-def graph-def, auto)

lemma c-graph-lm-5: graph f = ce-set-to-rel (c-graph f) by (simp add: c-graph-lm-4)

definition
total-recursive :: (nat ⇒ nat) ⇒ bool where
total-recursive = (λ f. graph f ∈ ce-rels)

lemma total-recursive-def1: total-recursive = (λ f. c-graph f ∈ ce-sets)
proof (rule ext) fix f show total-recursive f = (c-graph f ∈ ce-sets)
proof
  assume A: total-recursive f
  then have graph f ∈ ce-rels by (unfold total-recursive-def)
  then have ce-rel-to-set (graph f) ∈ ce-sets by (rule ce-rel-lm-6)
  then show c-graph f ∈ ce-sets by (simp add: c-graph-lm-4)

next
  assume c-graph f ∈ ce-sets
  then have ce-rel-to-set (graph f) ∈ ce-sets by (simp add: c-graph-lm-4)
  then have graph f ∈ ce-rels by (rule ce-rel-lm-7)
  then show total-recursive f by (unfold total-recursive-def)
qed

theorem pr-is-total-rec: f ∈ PrimRec1 ⇒ total-recursive f
proof
  assume A: f ∈ PrimRec1
  define p where p x = c-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 ce-univ p-is-pr have U-ce: ?U ∈ ce-sets by (rule ce-set-lm-7)
  have U-1: ?U = { c-pair x (f x) | x ∈ UNIV } by (simp add: p-def)
  with U-ce have S1: { c-pair x (f x) | x ∈ UNIV } ∈ ce-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 graph f ∈ ce-rels by (rule ce-rel-lm-24)
  then have graph (f o g) ∈ ce-rels by (simp add: graph-lm-4)
  then show ?thesis by (unfold total-recursive-def)
qed

lemma univ-for-pr-tot-rec-lm: c-graph univ-for-pr ∈ ce-sets
proof
  define A where A = c-graph univ-for-pr
  from A-def have S1: A = { c-pair x (univ-for-pr x) | x ∈ UNIV } (simp add: c-graph-def)
  from S1 have S2: A = { z . ∃ x. z = c-pair x (univ-for-pr x) } by auto
  have S3: c-snd z)
  proof
    fix z show (exists x. (z = c-pair x (univ-for-pr x))) = (univ-for-pr (c-fst z) = c-snd z)
  qed
proof
assume A: \( \exists x . \, z = \text{c-pair} \, x \) (\text{univ-for-pr} \, x)
then obtain x where S3-1: \( z = \text{c-pair} \, x \) (\text{univ-for-pr} \, x) ..
then show \( \text{univ-for-pr} \, (\text{c-fst} \, z) = \text{c-snd} \, z \) by simp
next
assume A: \( \text{univ-for-pr} \, (\text{c-fst} \, z) = \text{c-snd} \, z \)
from A have \( z = \text{c-pair} \, (\text{c-fst} \, z) \) (\text{univ-for-pr} \, (\text{c-fst} \, z)) by simp
thus \( \exists x . \, z = \text{c-pair} \, x \) (\text{univ-for-pr} \, x) ..
qed

with S2 have S4: A = \{ \, z . \, \text{univ-for-pr} \, (\text{c-fst} \, z) = \text{c-snd} \, z \} \} by auto
define p where p \( x \, y \) =
(if \( \text{c-assoc-have-key} \, (\text{pr-gr} \, y) \, (\text{c-fst} \, x) \) = 0 then
(if \( \text{c-assoc-value} \, (\text{pr-gr} \, y) \, (\text{c-fst} \, x) \) = \text{c-snd} \, x \) then (0::\text{nat}) else 1)
else 1) for x y
from \( \text{c-assoc-have-key-is-pr} \, \text{c-assoc-value-is-pr} \, \text{pr-gr-is-pr} \) have p-is-pr: p \( \in \text{PrimRec2} \)

unfolding p-def by prec
have S5: \( \forall z . \, (\text{univ-for-pr} \, (\text{c-fst} \, z) = \text{c-snd} \, z) = (\exists y . \, p \, z \, y = 0) \)

proof
fix z show \( (\text{univ-for-pr} \, (\text{c-fst} \, z) = \text{c-snd} \, z) = (\exists y . \, p \, z \, y = 0) \)

proof
assume A: \( \text{univ-for-pr} \, (\text{c-fst} \, z) = \text{c-snd} \, z \)
let \( ?n = \text{c-fst} \, (\text{c-fst} \, z) \)
let \( ?x = \text{c-snd} \, (\text{c-fst} \, z) \)
let \( ?y = \text{loc-upb} \, ?n \, ?x \)

have S5-1: \( \text{c-assoc-have-key} \, (\text{pr-gr} \, y) \, (\text{c-pair} \, ?n \, ?x) \) = 0 by (rule loc-upb-main)

have S5-2: \( \text{c-assoc-value} \, (\text{pr-gr} \, y) \, (\text{c-pair} \, ?n \, ?x) \) = \( \text{univ-for-pr} \, (\text{c-pair} \, ?n \, ?x) \)

from S5-1 have S5-3: \( \text{c-assoc-have-key} \, (\text{pr-gr} \, y) \, (\text{c-fst} \, z) \) = 0 by simp

from S5-2 A have S5-4: \( \text{c-assoc-value} \, (\text{pr-gr} \, y) \, (\text{c-fst} \, z) = \text{c-snd} \, z \) by simp

from S5-3 S5-4 have p \( z \, ?y = 0 \) by (simp add: p-def)

thus \( \exists y . \, p \, z \, y = 0 \) ..

next
assume A: \( \exists y . \, p \, z \, y = 0 \)
then obtain y where S5-1: p \( z \, y = 0 \) ..

have S5-2: \( \text{c-assoc-have-key} \, (\text{pr-gr} \, y) \, (\text{c-fst} \, z) \) = 0
proof (rule ccontr)
assume A-1: \( \text{c-assoc-have-key} \, (\text{pr-gr} \, y) \, (\text{c-fst} \, z) \) \( \neq 0 \)
then have p \( z \, y = 1 \) by (simp add: p-def)

with S5-1 show False by auto

qed

then have S5-3: p \( z \, y = (\text{if} \, \text{c-assoc-value} \, (\text{pr-gr} \, y) \, (\text{c-fst} \, z) = \text{c-snd} \, z \text{then} 
(0::\text{nat}) \text{ else } 1) \) by (simp add: p-def)

have S5-4: \( \text{c-assoc-value} \, (\text{pr-gr} \, y) \, (\text{c-fst} \, z) = \text{c-snd} \, z \)
proof (rule ccontr)
assume A-2: \( \text{c-assoc-value} \, (\text{pr-gr} \, y) \, (\text{c-fst} \, z) \) \( \neq \text{c-snd} \, z \)
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-def, auto) 
qed 

7.7 Computable sets, Post’s theorem 

definition 
computable :: nat set ⇒ bool where 
computable = (λ A. A ∈ ce-sets ∧ − A ∈ ce-sets) 

lemma computable-complement-1: computable A =⇒ computable (− A) 
proof – 

assume computable A 
then show ?thesis by (unfold computable-def, auto) 
qed 

lemma computable-complement-2: computable (− A) =⇒ computable A 
proof – 

assume computable (− A) 
then show ?thesis by (unfold computable-def, auto) 
qed 

lemma computable-complement-3: (computable A) = (computable (− A)) by (unfold computable-def, auto) 

theorem comp-impl-tot-rec: computable A =⇒ total-recursive (chf A) 
proof – 

assume A: computable A 
from A have A1: A ∈ ce-sets by (unfold computable-def, simp) 
from A have A2: − A ∈ ce-sets by (unfold computable-def, simp) 
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 ∈ PrimRec1 unfolding p-def by prec
from q-def have q-is-pr: q ∈ PrimRec1 unfolding q-def by prec
define U0 where U0 = { p x | x. x ∈ A}
define U1 where U1 = { q x | x. x ∈ A}

from A1 p-is-pr have U0-ce: U0 ∈ ce-sets by (unfold U0-def, rule ce-set-lm-7)
from A2 q-is-pr have U1-ce: U1 ∈ ce-sets by (unfold U1-def, rule ce-set-lm-7)
define U where U = U0 ∪ U1

from U0-ce U1-ce have U-ce: U ∈ 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 ∈ UNIV} by (simp add: V-def c-graph-def)
from U0-def p-def have U0-1: U0 = { c-pair x y | x y. x ∈ A ∧ y=0} by auto
from U1-def q-def have U1-1: U1 = { c-pair x y | x y. x ∉ A ∧ y=1} by auto
from U0-1 U1-1 U-def have U-1: U = { c-pair x y | x y. (x ∈ A ∧ y=0) ∨ (x ∉ A ∧ y=1)} by auto

from V-1 have V-2: V = { c-pair x y | x y. y = chf A x} by auto
have L1: ∀ x y. ((x ∈ A ∧ y=0) ∨ (x ∉ A ∧ y=1)) = (y = chf A x)
proof -
  fix x y
  show ((x ∈ A ∧ y=0) ∨ (x ∉ A ∧ y=1)) = (y = chf A x) by (unfold chf-def, auto)
qed

from V-2 U-1 L1 have U=V by simp
with U-ce have V-ce: V ∈ ce-sets by auto
with V-def have c-graph (chf A) ∈ ce-sets by auto
then show ?thesis by (unfold total-recursive-def1)

theorem tot-rec-impl-comp: total-recursive (chf A) ⟷ computable A
proof -
  assume A: total-recursive (chf A)
  then have A1: c-graph (chf A) ∈ ce-sets by (unfold total-recursive-def1)
  let ?U = c-graph (chf A)
  have L1: ∀U = { c-pair x (chf A x) | x. x ∈ UNIV} by (simp add: c-graph-def)
  have L2: ∀ x y. ((x ∈ A ∧ y=0) ∨ (x ∉ A ∧ y=1)) = (y = chf A x)
  proof -
  fix x y show ((x ∈ A ∧ y=0) ∨ (x ∉ A ∧ y=1)) = (y = chf A x)
  by (unfold chf-def, auto)
  qed

from L1 L2 have L3: ?U = { c-pair x y | x y. (x ∈ A ∧ y=0) ∨ (x ∉ A ∧ y=1)} by auto
  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: − A = { x. q x ∈ V}
proof -
  have ⋀ x. (q x ∈ V) = (x ∉ A)
  proof - fix x show (q x ∈ V) = (x ∉ A)
    proof
      assume A: q x ∈ V
      then have c-pair x 1 ∈ V by (unfold q-def)
      with V-def obtain x1 y1 where L5-2: c-pair x 1 = c-pair x1 y1
        and L5-3: ((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-1: 1=y1 by (rule c-pair-inj2)
      from L5-3 X-eq-X1 Y1-eq-1 show x ∉ A 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 ?thesis by auto
  qed
from V-ce q-is-pr have { x. q x ∈ V} ∈ ce-sets by (rule ce-set-lm-5)
with CA-eq have CA-ce: − A ∈ ce-sets by simp
from A-ce CA-ce show ?thesis by (simp add: computable-def)
  qed

theorem post-th-0: (computable A) = (total-recursive (chf A))

153
proof
  assume computable A then show total-recursive (chf A) by (rule comp-impl-tot-rec)
next
  assume total-recursive (chf A) then show computable A by (rule tot-rec-impl-comp)
qed

7.8 Universal computably enumerable set

definition
univ-ce :: nat set where
  univ-ce = { c-pair n x | n x ∈ nat-to-ce-set n }

lemma univ-for-pr-lm: univ-for-pr (c-pair n x) = (nat-to-pr n) x
  by (simp add: univ-for-pr-def pr-conv-2-to-1-def)

theorem univ-is-ce: univ-ce ∈ ce-sets
proof
  define A where A = c-graph univ-for-pr
  then have A ∈ ce-sets by (simp add: univ-for-pr-tot-rec-lm)
  then have ∃ pA ∈ PrimRec2. A = fn-to-set pA by (rule ce-set-lm-3)
  then obtain pA where pA-is-pr: pA ∈ PrimRec2
    and S1: A = fn-to-set pA
  by auto
  from S1 have S2: A = { x. ∃ y. pA x y = 0 } by (simp add: fn-to-set-def)
  define p where p z y = pA (c-pair (c-pair (c-fst z) (c-snd z) (c-fst y)))
  for z y
  from pA-is-pr have p-is-pr: p ∈ PrimRec2 unfolding p-def by prec
  have ∨ z. (∃ n x. z = c-pair n x ∧ x ∈ nat-to-ce-set n) = (c-snd z ∈ nat-to-ce-set (c-fst z))
  proof
    assume A: ∃ n x. z = c-pair n x ∧ x ∈ nat-to-ce-set n
    then obtain n x where L1: z = c-pair n x ∧ x ∈ nat-to-ce-set n by auto
    from L1 have L2: z = c-pair n x by auto
    from L1 have L3: x ∈ nat-to-ce-set n by auto
    from L1 have L4: c-fst z = n by simp
    from L1 have L5: c-snd z = x by simp
    from L3 L4 L5 show c-snd z ∈ nat-to-ce-set (c-fst z) by auto
  next
    assume A: c-snd z ∈ nat-to-ce-set (c-fst z)
    let ?n = c-fst z
    let ?x = c-snd z
    have L1: z = c-pair ?n ?x by simp
    from L1 A have z = c-pair ?n ?x ∧ ?x ∈ nat-to-ce-set ?n by auto
    thus ∃ n x. z = c-pair n x ∧ x ∈ nat-to-ce-set n by blast
  qed
  qed
then have \( \{ \text{c-pair } n \ x \mid n \ x \in \text{nat-to-ce-set } n \} = \{ \text{z. c-snd } z \in \text{nat-to-ce-set} \text{ (c-fst } z) \} \) by auto
then have \( S3: \text{univ-ce} = \{ \text{z. c-snd } z \in \text{nat-to-ce-set (c-fst } z) \} \) by (simp add: univ-ce-def)
have \( S4: \bigwedge z. (\text{c-snd } z \in \text{nat-to-ce-set } (\text{c-fst } z)) = (\exists y. p \ z \ y = 0) \)
proof
  fix \( z \) show \((\text{c-snd } z \in \text{nat-to-ce-set } (\text{c-fst } z)) = (\exists y. p \ z \ y = 0) \)
  proof
    assume \( A: \text{c-snd } z \in \text{nat-to-ce-set } (\text{c-fst } z) \)
    have \( \text{nat-to-ce-set } (\text{c-fst } z) = \{ x. \exists y. (\text{nat-to-pr } (\text{c-fst } z)) \text{ (c-pair } x \ y) = 0 \} \) by (simp add: nat-to-ce-set-lm-1)
    with \( A \) obtain \( u \) where \( S4-1: (\text{nat-to-pr } (\text{c-fst } z)) \text{ (c-pair } (\text{c-snd } z) \ u) = 0 \)
    by auto
    then have \( S4-2: \text{univ-for-pr } (\text{c-pair } (\text{c-fst } z)) \text{ (c-pair } (\text{c-snd } z) \ u) = 0 \)
    by (simp add: univ-for-pr-lm)
    from \( A \)-def have \( S4-3: A = \{ \text{c-pair } x \ (\text{univ-for-pr } x) \mid x \in \text{UNIV } \} \)
    by (simp add: c-graph-def)
    then have \( S4-4: \bigwedge x. \text{c-pair } x \ (\text{univ-for-pr } x) \in A \) by auto
    then have \( \text{c-pair } (\text{c-pair } (\text{c-fst } z) \ (\text{c-pair } (\text{c-snd } z) \ u)) \text{ (univ-for-pr } (\text{c-pair } (\text{c-fst } z) (\text{c-pair } (\text{c-snd } z) \ u))) \in A \) by auto
    with \( S4-2 \) have \( S4-5: \text{c-pair } (\text{c-pair } (\text{c-fst } z) (\text{c-pair } (\text{c-snd } z) \ u)) = 0 \)
    by auto
    with \( S4-4 \)
    with S2 obtain \( v \) where \( S4-6: pA \ (\text{c-pair } (\text{c-pair } (\text{c-fst } z) (\text{c-pair } (\text{c-snd } z) \ u)) = 0 \)
    by auto
    define \( y \) where \( y = \text{c-pair } u \ v \)
    from \( y \)-def have \( S4-7: u = \text{c-fst } y \)
    by simp
    from \( y \)-def have \( S4-8: v = \text{c-snd } y \)
    by simp
    from \( S4-6 \) \( S4-7 \) \( S4-8 \)
    p-def have \( p \ z \ y = 0 \) by simp
    thus \( \exists y. p \ z \ y = 0 \) ..
next
  assume \( A: \exists y. p \ z \ y = 0 \)
  then obtain \( y \) where \( S4-1: p \ z \ y = 0 \) ..
  from \( S4-1 \)
  p-def have \( S4-2: pA \ (\text{c-pair } (\text{c-pair } (\text{c-fst } z) (\text{c-pair } (\text{c-snd } z) \ u)) = 0 \)
  by (simp add: c-graph-lm-1)
  with \( S4-1 \)
  have \( S4-3: \text{c-pair } (\text{c-pair } (\text{c-fst } z) (\text{c-pair } (\text{c-snd } z) \ (\text{c-fst } y))) = 0 \)
  by auto
  with \( A \)-def have \( \text{c-pair } (\text{c-pair } (\text{c-fst } z) (\text{c-pair } (\text{c-snd } z) \ (\text{c-fst } y))) = 0 \)
  by (rule c-graph-lm-1)
  then have \( S4-4: \text{univ-for-pr } (\text{c-pair } (\text{c-fst } z) (\text{c-pair } (\text{c-snd } z) \ (\text{c-fst } y))) = 0 \)
  by auto
  then have \( S4-5: (\text{nat-to-pr } (\text{c-fst } z)) \text{ (c-pair } (\text{c-snd } z) \ (\text{c-fst } y)) = 0 \)
  by (simp add: univ-for-pr-lm)
  then have \( S4-7: \exists y. (\text{nat-to-pr } (\text{c-fst } z)) \text{ (c-pair } (\text{c-snd } z) \ y) = 0 \) ..
  have \( S4-8: \text{nat-to-ce-set } (\text{c-fst } z) = \{ x. \exists y. (\text{nat-to-pr } (\text{c-fst } z)) \text{ (c-pair } x \ y) = 0 \} \)
  by (simp add: nat-to-ce-set-lm-1)
  from \( S4-7 \)
  have \( S4-9: \text{c-snd } z \in \{ x. \exists y. (\text{nat-to-pr } (\text{c-fst } z)) \text{ (c-pair } x \ y) = 0 \} \)
= 0 } by auto

with S4-8 show c-snd z ∈ nat-to-ce-set (c-fst z) by auto

qed

qed

with S3 have univ-ce = {z. ∃ y. p z y = 0} by auto

then have univ-ce = fn-to-set p by (simp add: fn-to-set-def)

moreover from p-is-pr have fn-to-set p ∈ ce-sets by (rule ce-set-lm-1)

ultimately show univ-ce ∈ ce-sets by auto

qed

lemma univ-ce-lm-1: (c-pair n x ∈ univ-ce) = (x ∈ nat-to-ce-set n)

proof

from univ-ce-def have S1: univ-ce = {z. ∃ n x. z = c-pair n x ∧ x ∈ nat-to-ce-set n} by auto

have S2: (∃ n1 x1. c-pair n x = c-pair n1 x1 ∧ x1 ∈ nat-to-ce-set n1) = (x ∈ nat-to-ce-set n)

proof

assume ∃ n1 x1. c-pair n x = c-pair n1 x1 ∧ x1 ∈ nat-to-ce-set n1

then obtain n1 x1 where L1: c-pair n x = c-pair n1 x1 and L2: x1 ∈ nat-to-ce-set n1 by auto

from L1 have L3: n = n1 by (rule c-pair-inj1)

from L1 have L4: x = x1 by (rule c-pair-inj2)

from L2 L3 L4 show x ∈ nat-to-ce-set n by auto

next

assume A: x ∈ nat-to-ce-set n

then have c-pair n x = c-pair n x ∧ x ∈ nat-to-ce-set n by auto

thus ∃ n1 x1. c-pair n x = c-pair n1 x1 ∧ x1 ∈ nat-to-ce-set n1 by blast

qed

with S1 show ?thesis by auto

qed

theorem univ-ce-is-not-comp1: ¬ univ-ce ∈ ce-sets

proof (rule ccontr)

assume ¬ univ-ce ∈ ce-sets

then have A: ¬ univ-ce ∈ ce-sets by auto

define p where p x = c-pair x x for x

have p-is-pr: p ∈ PrimRec1 unfolding p-def by prec

define A where A = { x. p x ∈ ¬ univ-ce }

from A p-is-pr have { x. p x ∈ ¬ univ-ce } ∈ ce-sets by (rule ce-set-lm-5)

with A-def have S1: A ∈ ce-sets by auto

then have ∃ n. A = nat-to-ce-set n by (rule nat-to-ce-set-srj)

then obtain n where S2: A = nat-to-ce-set n ..

from A-def have (n ∈ A) = (p n ∈ ¬ univ-ce) by auto

with p-def have (n ∈ A) = (c-pair n n ∉ univ-ce) by auto

with univ-ce-def univ-ce-lm-1 have (n ∈ A) = (n ∉ nat-to-ce-set n) by auto

with S2 have (n ∈ A) = (n ∉ A) by auto

thus False by auto

qed
theorem univ-ce-is-not-comp2: ¬ total-recursive (chf univ-ce)
proof
  assume total-recursive (chf univ-ce)
  then have computable univ-ce by (rule tot-rec-impl-comp)
  then have ¬ univ-ce ∈ ce-sets by (unfold computable-def, auto)
  with univ-ce-is-not-comp1 show False by auto
qed

theorem univ-ce-is-not-comp3: ¬ computable univ-ce
proof (rule ccontr)
  assume ¬¬ computable univ-ce
  then have computable univ-ce by auto
  then have total-recursive (chf univ-ce) by (rule comp-impl-tot-rec)
  with univ-ce-is-not-comp2 show False by auto
qed

7.9 s-1-1 theorem, one-one and many-one reducibilities

definition
  index-of-r-to-l :: nat where
  index-of-r-to-l =
  pair-by-index
  (pair-by-index index-of-c-fst (comp-by-index index-of-c-fst index-of-c-snd))
  (comp-by-index index-of-c-snd index-of-c-snd)

lemma index-of-r-to-l-lm: nat-to-pr index-of-r-to-l (c-pair x (c-pair y z)) = c-pair (c-pair x y) z
  apply (unfold index-of-r-to-l-def)
  apply (simp add: pair-by-index-main)
  apply (unfold c-f-pair-def)
  apply (simp add: index-of-c-fst-main)
  apply (simp add: comp-by-index-main)
  apply (simp add: index-of-c-fst-main)
  apply (simp add: index-of-c-snd-main)
done

definition
  s-ce :: nat ⇒ nat ⇒ nat where
  s-ce e x = (λ e x. s1-1 (comp-by-index e index-of-r-to-l) x)

lemma s-ce-is-pr: s-ce ∈ PrimRec2
  unfolding s-ce-def using comp-by-index-is-pr s1-1-is-pr by prec

lemma s-ce-inj: s-ce e1 x1 = s-ce e2 x2 ⇒ e1 = e2 ∧ x1 = x2
proof
  let ?n1 = index-of-r-to-l
  assume s-ce e1 x1 = s-ce e2 x2
  then have s1-1 (comp-by-index e1 ?n1) x1 = s1-1 (comp-by-index e2 ?n1) x2
    by (unfold s-ce-def)
then have \( L1: \) \( \text{comp-by-index } e1 \ ?n1 = \text{comp-by-index } e2 \ ?n1 \land x1=x2 \) by (rule \( s1-1-inj \))

from \( L1 \) have \( \text{comp-by-index } e1 \ ?n1 = \text{comp-by-index } e2 \ ?n1 \) ..
then have \( e1=e2 \) by (rule \( \text{comp-by-index-inj1} \))
moreover from \( L1 \) have \( x1=x2 \) by auto
ultimately show \( \text{thesis} \) by auto
qed

\[ \text{lemma } s-ce-inj1: s-ce \ e1 \ x = s-ce \ e2 \ x \implies e1=e2 \]  
proof –
assume \( s-ce \ e1 \ x = s-ce \ e2 \ x \)
then have \( e1=e2 \land x=x \) by (rule \( s-ce-inj \))
then show \( e1=e2 \) by auto
qed

\[ \text{lemma } s-ce-inj2: s-ce \ e \ x1 = s-ce \ e \ x2 \implies x1=x2 \]  
proof –
assume \( s-ce \ e \ x1 = s-ce \ e \ x2 \)
then have \( e=e \land x1=x2 \) by (rule \( s-ce-inj \))
then show \( x1=x2 \) by auto
qed

\[ \text{theorem } s1-1-th1: \forall \ n \ x \ y. \ ((\text{nat-to-pr } n) \ (\text{c-pair } x \ y)) = (\text{nat-to-pr } (s1-1 n \ x)) \ y \]  
proof (rule allI, rule allI, rule allI)
fix \( n \ x \ y \) show \( \text{nat-to-pr } n \ (\text{c-pair } x \ y) = \text{nat-to-pr } (s1-1 n \ x) \ y \)
proof –
have \( (\lambda y. \ (\text{nat-to-pr } n) \ (\text{c-pair } x \ y)) = \text{nat-to-pr } (s1-1 n \ x) \ y \) by (rule \( s1-1-th \))
then show \( \text{thesis} \) by (simp add: fun-eq-iff)
qed

\[ \text{lemma } s-lm: (\text{nat-to-pr } (s-ce \ e \ x)) \ (\text{c-pair } y \ z) = (\text{nat-to-pr } e) \ (\text{c-pair } (\text{c-pair } x \ y) \ z) \]  
proof –
let \( ?n1 = \text{index-of-r-to-l} \)
have \( (\text{nat-to-pr } (s-ce \ e \ x)) \ (\text{c-pair } y \ z) = \text{nat-to-pr } (s1-1 \ (\text{comp-by-index } e \ ?n1)) \ x \) \( (\text{c-pair } y \ z) \) by (unfold s-ce-def, simp)
also have \( \ldots = (\text{nat-to-pr } (\text{comp-by-index } e \ ?n1)) \ (\text{c-pair } x \ (\text{c-pair } y \ z)) \) by (simp add: s1-1-th1)
also have \( \ldots = (\text{nat-to-pr } e) \ ((\text{nat-to-pr } ?n1) \ (\text{c-pair } x \ (\text{c-pair } y \ z))) \) by (simp add: comp-by-index-main)
finally show \( \text{thesis} \) by (simp add: index-of-r-to-l-lm)
qed

\[ \text{theorem } s-ce-1-1-th: (\text{c-pair } x \ y \in \text{nat-to-ce-set } e) = (y \in \text{nat-to-ce-set } (s-ce \ e \ x)) \]  
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)

\[ 158 \]
have (nat-to-pr (s-ce e x)) (c-pair y z) = 0 by (simp add: s-lm L1)
with nat-to-ce-set-lm-1 show y ∈ nat-to-ce-set (s-ce e x) by auto
next
assume A: y ∈ nat-to-ce-set (s-ce e x)
then obtain z where L1: (nat-to-pr (s-ce e x)) (c-pair y z) = 0
  by (auto simp add: nat-to-ce-set-lm-1)
then have (nat-to-pr e) (c-pair (c-pair x y) z) = 0
  by (simp add: s-lm)
with nat-to-ce-set-lm-1 show c-pair x y ∈ nat-to-ce-set e by auto
qed

definition one-reducible-to-via :: (nat set → nat set) ⇒ (nat set → nat set) ⇒ (nat ⇒ nat) ⇒ bool where
  one-reducible-to-via = (λ A B f. total-recursive f ∧ inj f ∧ (∀ x. (x ∈ A) = (f x ∈ B)))

definition one-reducible-to :: (nat set ⇒ nat set) ⇒ bool where
  one-reducible-to = (λ A B. ∃ f. one-reducible-to-via A B f)

definition many-reducible-to-via :: (nat set ⇒ nat set) ⇒ (nat set ⇒ nat set) ⇒ (nat ⇒ nat) ⇒ bool where
  many-reducible-to-via = (λ A B f. total-recursive f ∧ (∀ x. (x ∈ A) = (f x ∈ B)))

definition many-reducible-to :: (nat set ⇒ nat set) ⇒ bool where
  many-reducible-to = (λ A B. ∃ f. many-reducible-to-via A B f)

lemma one-reducible-to-via-trans: [ one-reducible-to-via A B f; one-reducible-to-via B C g ] ⇒ one-reducible-to-via A C (g o f)
proof –
  assume A1: one-reducible-to-via A B f
  assume A2: one-reducible-to-via B C g
  from A1 have f-tr: total-recursive f by (unfold one-reducible-to-via-def, auto)
  from A1 have f-inj: inj f by (unfold one-reducible-to-via-def, auto)
  from A1 have L1: ∀ x. (x ∈ A) = (f x ∈ B) by (unfold one-reducible-to-via-def, auto)
  from A2 have g-tr: total-recursive g by (unfold one-reducible-to-via-def, auto)
  from A2 have g-inj: inj g by (unfold one-reducible-to-via-def, auto)
  from A2 have L2: ∀ x. (x ∈ B) = (g x ∈ C) by (unfold one-reducible-to-via-def, auto)
  from g-tr f-tr have fg-tr: total-recursive (g o f) by (rule comp-tot-rec)
  from g-inj f-inj have fg-inj: inj (g o f) by (rule inj-compose)
  from L1 L2 have L3: ∀ x. (x ∈ A) = ((g o f) x ∈ C)) by auto
  with fg-tr fg-inj show ?thesis by (unfold one-reducible-to-via-def, auto)
qed

lemma one-reducible-to-trans: [ one-reducible-to A B; one-reducible-to B C ] ⇒ one-reducible-to A C
proof
—

assume one-reducible-to A B
then obtain f where A1: one-reducible-to-via A B f unfolding one-reducible-to-def
by auto

assume one-reducible-to B C
then obtain g where A2: one-reducible-to-via B C g unfolding one-reducible-to-def
by auto
from A1 A2 have one-reducible-to-via A C (g o f) by (rule one-reducible-to-via-trans)
then show ?thesis unfolding one-reducible-to-def by auto
qed

lemma one-reducible-to-via-refl: one-reducible-to-via A A (λ x. x)
proof —
have is-pr: (λ x. x) ∈ PrimRec1 by (rule pr-id1-1)
then have is-tr: total-recursive (λ x. x) by (rule pr-is-total-rec)
have is-inj: inj (λ x. x) by simp
have L1: ∀ x. (x ∈ A) = (((λ x. x) x) ∈ A) by simp
with is-tr is-inj show ?thesis by (unfold one-reducible-to-via-def, auto)
qed

lemma one-reducible-to-refl: one-reducible-to A A
proof —
have one-reducible-to-via A A (λ x. x) by (rule one-reducible-to-via-refl)
then show ?thesis by (unfold one-reducible-to-def, auto)
qed

lemma many-reducible-to-via-trans: [ many-reducible-to-via A B f; many-reducible-to-via B C g ] ⇒ many-reducible-to-via A C (g o f)
proof —
assume A1: many-reducible-to-via A B f
assume A2: many-reducible-to-via B C g
from A1 have f-tr: total-recursive f by (unfold many-reducible-to-via-def, auto)
from A1 have L1: ∀ x. (x ∈ A) = (f x ∈ B) by (unfold many-reducible-to-via-def, auto)
from A2 have g-tr: total-recursive g by (unfold many-reducible-to-via-def, auto)
from A2 have L2: ∀ x. (x ∈ B) = (g x ∈ C) by (unfold many-reducible-to-via-def, auto)
from f-tr g-tr have fg-tr: total-recursive (g o f) by (rule comp-tot-rec)
from L1 L2 have L3: (∀ x. (x ∈ A) = ((g o f) x ∈ C)) by auto
with fg-tr show ?thesis by (unfold many-reducible-to-via-def, auto)
qed

lemma many-reducible-to-trans: [ many-reducible-to A B; many-reducible-to B C ] ⇒ many-reducible-to A C
proof —
assume many-reducible-to A B
then obtain f where A1: many-reducible-to-via A B f
unfolding many-reducible-to-def by auto
assume many-reducible-to B C
then obtain \$g\$ where \$A2: many-reducible-to-via B C g\$

unfolding \$many-reducible-to-def\$ by \$auto\$

from \$A1 A2\$ have \$many-reducible-to-via A C (g o f)\$ by (rule \$many-reducible-to-via-trans\$)

then show \$\textsf{thesis}\$ unfolding \$many-reducible-to-def\$ by \$auto\$

\textbf{qed}

\textbf{lemma} \texttt{one-reducibility-via-is-many: one-reducible-to-via A B f} \implies \texttt{many-reducible-to-via A B f} \\
\textbf{proof} –

\begin{tabular}{l}
\hspace{1em} assume \$A: one-reducible-to-via A B f\$\\
\hspace{1em} from \$A\$ have \$f-tr: total-recursive f\$ by (unfold \$one-reducible-to-via-def\$, \$auto\$)\\
\hspace{1em} from \$A\$ have \$\forall x. (x \in A) = (f x \in B)\$ by (unfold \$one-reducible-to-via-def\$, \$auto\$)\\
\hspace{1em} with \$f-tr\$ show \$\textsf{thesis}\$ by (unfold \$many-reducible-to-via-def\$, \$auto\$)
\end{tabular}

\textbf{qed}

\textbf{lemma} \texttt{one-reducibility-is-many: one-reducible-to A B} \implies \texttt{many-reducible-to A B} \\
\textbf{proof} –

\begin{tabular}{l}
\hspace{1em} assume \$\texttt{one-reducible-to A B}\$\\
\hspace{1em} then obtain \$f\$ where \$A: one-reducible-to-via A B f\$\\
\hspace{1em} unfolding \$\texttt{one-reducible-to-def}\$ by \$\texttt{auto}\$
\end{tabular}

then have \$\texttt{many-reducible-to-via A B f}\$ by (rule \$\texttt{one-reducibility-via-is-many}\$)

then show \$\textsf{thesis}\$ unfolding \$many-reducible-to-def\$ by \$auto\$

\textbf{qed}

\textbf{lemma} \texttt{many-reducible-to-via-refl: many-reducible-to-via A A (}\$\lambda x. x\$\texttt{)} \\
\textbf{proof} –

\begin{tabular}{l}
\hspace{1em} have \$\texttt{one-reducible-to-via A A (}\$\lambda x. x\$\texttt{)}\$ by (rule \$\texttt{one-reducibility-via-refl}\$)\\
\hspace{1em} then show \$\textsf{thesis}\$ by (rule \$\texttt{one-reducibility-via-is-many}\$)
\end{tabular}

\textbf{qed}

\textbf{lemma} \texttt{many-reducible-to-refl: many-reducible-to A A} \\
\textbf{proof} –

\begin{tabular}{l}
\hspace{1em} have \$\texttt{one-reducible-to A A}\$ by (rule \$\texttt{one-reducible-to-refl}\$)\\
\hspace{1em} then show \$\textsf{thesis}\$ by (rule \$\texttt{one-reducibility-is-many}\$)
\end{tabular}

\textbf{qed}

\textbf{theorem} \texttt{m-red-to-comp: [ many-reducible-to A B; computable B ] } \implies \texttt{computable A} \\
\textbf{proof} –

\begin{tabular}{l}
\hspace{1em} assume \$\texttt{many-reducible-to A B}\$\\
\hspace{1em} then obtain \$f\$ where \$A1: many-reducible-to-via A B f\$\\
\hspace{1em} unfolding \$\texttt{many-reducible-to-def}\$ by \$\texttt{auto}\$
\end{tabular}

from \$A1\$ have \$f-tr: total-recursive f\$ by (unfold \$many-reducible-to-via-def\$, \$auto\$) \\
from \$A1\$ have \$L1: \forall x. (x \in A) = (f x \in B)\$ by (unfold \$many-reducible-to-via-def\$, \$auto\$)

\begin{tabular}{l}
\hspace{1em} assume \$\texttt{computable B}\$\\
\hspace{1em} then have \$L2: total-recursive (chf B)\$ by (rule \$\texttt{comp-impl-tot-rec}\$)
\end{tabular}

\begin{tabular}{l}
\hspace{1em} have \$L3: chf A = (chf B) o f\$ \\
\hspace{1em} proof fix \$x\$
\end{tabular}
have \( \text{chf} A \, x = (\text{chf} B) \, (f \, x) \)

proof cases

assume \( A \) \( x \in A \)
then have \( L3-1: \text{chf} A \, x = 0 \) by (simp add: chf-lm-2)
from \( A \, L1 \) have \( f \, x \in B \) by auto
then have \( L3-2: (\text{chf} B) \, (f \, x) = 0 \) by (simp add: chf-lm-2)
from \( L3-1 \) \( L3-2 \) show \( \text{chf} A \, x = (\text{chf} B) \, (f \, x) \) by auto

next

assume \( A \) \( x \notin A \)
then have \( L3-1: \text{chf} A \, x = 1 \) by (simp add: chf-lm-3)
from \( A \, L1 \) have \( f \, x \notin B \) by auto
then have \( L3-2: (\text{chf} B) \, (f \, x) = 1 \) by (simp add: chf-lm-3)
from \( L3-1 \) \( L3-2 \) show \( \text{chf} A \, x = (\text{chf} B) \, (f \, x) \) by auto

qed

then show \( \text{chf} A \, x = (\text{chf} B \circ f) \, x \) by auto

qed

from \( L2 \) \( f-tr \) have \( \text{total-recursive} \, (\text{chf} B \circ f) \) by (rule comp-tot-rec)
with \( L3 \) have \( \text{total-recursive} \, (\text{chf} A) \) by auto
then show \( ?\text{thesis} \) by (rule tot-rec-impl-comp)

qed

lemma many-reducible-lm-1: many-reducible-to univ-ce \( A \) \( \implies \neg \) computable \( A \)

proof (rule ccontr)

assume \( A1: \) many-reducible-to univ-ce \( A \)
assume \( \neg \neg \) computable \( A \)
then have \( A2: \) computable \( A \) by auto
from \( A1 \) \( A2 \) have \( \text{computable univ-ce} \) by (rule m-red-to-comp)
with \( \text{univ-ce-is-not-comp3} \) show \( \text{False} \) by auto

qed

lemma one-reducible-lm-1: one-reducible-to univ-ce \( A \) \( \implies \neg \) computable \( A \)

proof

assume one-reducible-to univ-ce \( A \)
then have many-reducible-to univ-ce \( A \) by (rule one-reducibility-is-many)
then show \( ?\text{thesis} \) by (rule many-reducible-lm-1)

qed

lemma one-reducible-lm-2: one-reducible-to-via \( \text{nat-to-ce-set} \, n \) univ-ce \( \lambda x. \, c\text{-pair} \, n \, x \)

proof

define \( f \) where \( f \, x = c\text{-pair} \, n \, x \) for \( x \)
have \( f\text{-is-pr:} \, f \in \text{PrimRec1} \) unfolding \( f\text{-def} \) by prec
then have \( f\text{-tr:} \, \text{total-recursive} \, f \) by (rule pr-is-total-rec)
have \( f\text{-inj:} \, \text{inj} \, f \)
proof (rule injI)
fix \( x \) \( y \) assume \( A: \, f \, x = f \, y \)
then have \( c\text{-pair} \, n \, x = c\text{-pair} \, n \, y \) by (unfold \( f\text{-def} \))
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: 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 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 \( = (\lambda A. A \in \text{ce-sets} \land (\forall B. B \in \text{ce-sets} \implies \text{one-reducible-to } B A)) \)

theorem univ-is-complete: one-complete \( \text{univ-ce} \)

proof (unfold one-complete-def)

show \( \text{univ-ce} \in \text{ce-sets} \land (\forall B. B \in \text{ce-sets} \implies \text{one-reducible-to } B \text{univ-ce}) \)

proof

show \( \text{univ-ce} \in \text{ce-sets} \) by (rule univ-is-ce)

next

show \( \forall B. B \in \text{ce-sets} \implies \text{one-reducible-to } B \text{univ-ce} \)

proof (rule allI, rule impI)

fix \( B \) assume \( B \in \text{ce-sets} \) then show \( \text{one-reducible-to } B \text{univ-ce} \) by (rule one-reducible-lm-4)

qed

qed

7.11 Index sets, Rice’s theorem

definition index-set :: \( \text{nat set} \Rightarrow \text{bool} \) where

index-set \( = (\lambda A. \forall n m. n \in A \land (\text{nat-to-ce-set } n = \text{nat-to-ce-set } m) \implies m \in A) \)
lemma index-set-lm-1: \[ \text{index-set } A; n \in A; \text{nat-to-ce-set } n = \text{nat-to-ce-set } m \] \[\Rightarrow m \in A\]

proof

assume A1: index-set A
assume A2: n \in A
assume A3: nat-to-ce-set n = nat-to-ce-set m

from A2 A3 have L1: \( n \in A \land (\text{nat-to-ce-set } n = \text{nat-to-ce-set } m) \) by auto
from A1 have L2: \( \forall n \in A. n \in A \land (\text{nat-to-ce-set } n = \text{nat-to-ce-set } m) \) \(\Rightarrow m \in A \)

from L1 L2 show \(\text{thesis} \) by auto

qed

lemma index-set-lm-2: index-set A \(\Rightarrow \) index-set (\(-A\))

proof

assume A: index-set A

show index-set (\(-A\))

proof (unfold index-set-def)

show \(\forall n m. n \in -A \land \text{nat-to-ce-set } n = \text{nat-to-ce-set } m \) \(\Rightarrow m \in -A \)

proof (rule allI, rule allI, rule impI)

fix n m assume A1: \( n \in -A \land \text{nat-to-ce-set } n = \text{nat-to-ce-set } m \)
from A1 have A2: \( n \in -A \) by auto
from A1 have A3: \( \text{nat-to-ce-set } m = \text{nat-to-ce-set } n \) by auto

show \( m \in -A \)

proof

assume m \(\in A \)
from A this A3 have \( n \in A \) by (rule index-set-lm-1)
with A2 show \( \text{False} \) by auto

qed

qed

lemma Rice-lm-1: \[ \text{index-set } A; A \neq \{\}; A \neq \text{UNIV}; \exists n \in A. \text{nat-to-ce-set } n = \{\} \] \(\Rightarrow \) one-reducible-to univ-ce (\(-A\))

proof

assume A1: index-set A
assume A2: A \(\neq \{\}\)
assume A3: A \(\neq \text{UNIV}\)
assume \( \exists n \in A. \text{nat-to-ce-set } n = \{\} \)
then obtain e-0 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 e-1 where e-1-not-in-A: \( e-1 \in (\neg A) \) by auto
with e-0-in-A have e-0-neq-e-1: \( e-0 \neq e-1 \) by auto
have nat-to-ce-set e-0 \(\neq \) nat-to-ce-set e-1

proof

assume nat-to-ce-set e-0 = nat-to-ce-set e-1
with A1 e-0-in-A have e-1 \(\in A \) by (rule index-set-lm-1)
with e-1-not-in-A show \( \text{False} \) by auto

164
qed

with e-0-empty have e1-not-empty: nat-to-ce-set e-1 ≠ {} by auto
define we-1 where we-1 = nat-to-ce-set e-1
from e1-not-empty have we-1-not-empty: we-1 ≠ {} by (unfold we-1-def)
define r where r = univ-ce × we-1
have loc-lm-1: \( \forall x. x ∈ \text{univ-ce} \Rightarrow \forall y. (y \in \text{we-1}) = ((x,y) \in r) \) by (unfold r-def, auto)
have loc-lm-2: \( \forall x. x \notin \text{univ-ce} \Rightarrow \forall y. (y \in \{\}) = ((x,y) \in r) \) by (unfold r-def, auto)

proof (unfold r-def, rule ce-rel-lm-29)
show univ-ce ∈ ce-sets by (rule univ-is-ce)
show we-1 ∈ ce-sets by (unfold we-1-def, rule nat-to-ce-set-into-ce)
qed

define we-n where we-n = ce-rel-to-set r
from r-ce have we-n-ce: we-n ∈ ce-sets by (unfold 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 we-n-df1: we-n = nat-to-ce-set n by auto
define f where f = s-ce n x for x
from s-ce-is-pr 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 f x = f y
  then have s-ce n x = s-ce n y by (unfold f-def)
  then show x = y by (rule s-ce-inj2)

qed

have 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 f-def, unfold we-n-df1, simp add: s-ce-1-1-th)

qed

from A1 have loc-lm-4: index-set (¬ A) by (rule index-set-lm-2)

have loc-lm-5: \( \forall x. (x ∈ \text{univ-ce}) = (f x ∈ \neg A) \)

proof
  fix x show \( (x ∈ \text{univ-ce}) = (f x ∈ \neg A) \)
  proof
    assume A: \( x ∈ \text{univ-ce} \)
    then have S1: \( \forall y. (y \in \text{we-1}) = ((x,y) \in r) \) by (rule loc-lm-1)
    from ce-rel-lm-12 have \( \forall y. (\text{c-pair} x y \in \text{ce-rel-to-set} r) = ((x,y) \in r) \) by auto
    then have \( \forall y. ((x,y) \in r) = (\text{c-pair} x y \in \text{we-n}) \) by (unfold we-n-def, auto)
    with S1 have \( \forall y. (y \in \text{we-1}) = (\text{c-pair} x y \in \text{we-n}) \) by auto
    with loc-lm-3 have \( \forall y. (y \in \text{we-1}) = (y \in \text{nat-to-ce-set} (f x)) \) by auto
    then have S2: \( \text{we-1} = \text{nat-to-ce-set} (f x) \) by auto
    then have \( \text{nat-to-ce-set} e-1 = \text{nat-to-ce-set} (f x) \) by (unfold we-1-def)
    with loc-lm-4 e-1-not-in-A show \( f x ∈ \neg A \) by (rule index-set-lm-1)
  next
  show \( f x ∈ \neg A \Rightarrow x ∈ \text{univ-ce} \)

165
proof (rule ccontr)
  assume fx-in-A: f x ∈ − A
  assume x-not-in-univ: x /∈ univ-ce
  then have S1: ∀ y. (y ∈ {}) = ((x, y) ∈ r) by (rule loc-lm-2)
  from ce-rel-lm-12 have ∀ y. (c-pair x y ∈ ce-rel-to-set r) = ((x, y) ∈ r) by auto
  then have ∃ y. ((x, y) ∈ r) = (c-pair x y ∈ we-n) by (unfold we-n-def, auto)
  with S1 have ∀ y. ((x, y) ∈ r) = (c-pair x y ∈ we-n) by (unfold we-n-def, auto)
  then show False by auto
qed

lemma Rice-lm-2: [ 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 ≠ UNIV
  assume A4: n ∈ A
  assume A5: nat-to-ce-set n = {}
  from A4 A5 have S1: ∃ n ∈ A. nat-to-ce-set n = {} by auto
  from A1 A2 A3 S1 show ?thesis by (rule Rice-lm-1)
qed

theorem Rice-1: [ index-set A; A ≠ {}]; A ≠ UNIV ] ⇒ one-reducible-to univ-ce
A ∨ one-reducible-to univ-ce (¬ A)
proof –
  assume A1: index-set A
  assume A2: A ≠ {}
  assume A3: A ≠ UNIV
  from ce-empty have ∃ n. {} = nat-to-ce-set n by (rule nat-to-ce-set-srj)
  then obtain n where n-empty: nat-to-ce-set n = {} by auto
  show ?thesis
  proof cases
    assume A: n ∈ A
    from A1 A2 A3 A n-empty have one-reducible-to univ-ce (¬ A) by (rule Rice-lm-2)
    then show ?thesis by auto
next
assume \( n \notin A \) then have \( A \) \( n \notin A \) by auto
from \( A1 \) have \( S1 \): \( \text{index-set} \ (\neg A) \) by \( \text{rule index-set-lm-2} \)
from \( A3 \) have \( S2 \): \( \neg A \neq \{\} \) by auto
from \( A2 \) have \( S3 \): \( \neg A \neq \text{UNIV} \) by auto
from \( S1 \ S2 \ S3 \) \( A \ n\)-empty have \( \text{one-reducible-to univ-ce} \ (\neg \ (\neg A)) \) by \( \text{rule Rice-lm-2} \)
then have \( \text{one-reducible-to univ-ce} \ A \) by simp
then show \( \text{thesis} \) by auto
qed

**Theorem Rice-2**: \( [ \left[ \text{index-set} \ A; \ A \neq \{\}; \ A \neq \text{UNIV} \right] ] \implies \neg \text{computable} \ A \)

**Proof**
assume \( A1 \): \( \text{index-set} \ A \)
assume \( A2 \): \( A \neq \{\} \)
assume \( A3 \): \( A \neq \text{UNIV} \)
from \( A1 \ A2 \ A3 \) have \( \text{one-reducible-to univ-ce} \ A \lor \text{one-reducible-to univ-ce} \ (\neg A) \) by \( \text{rule Rice-1} \)
then have \( S1 \): \( \neg \text{one-reducible-to univ-ce} \ A \implies \text{one-reducible-to univ-ce} \ (\neg A) \) by auto
show \( \text{thesis} \)
proof cases
assume \( \text{one-reducible-to univ-ce} \ A \)
then show \( \neg \text{computable} \ A \) by \( \text{rule one-reducible-lm-1} \)
next
assume \( \neg \text{one-reducible-to univ-ce} \ A \)
with \( S1 \) have \( \text{one-reducible-to univ-ce} \ (\neg A) \) by auto
then have \( \neg \text{computable} \ (\neg A) \) by \( \text{rule one-reducible-lm-1} \)
with \( \text{computable-complement-3} \) show \( \neg \text{computable} \ A \) by auto
qed

**Theorem Rice-3**: \( [ \left[ C \subseteq \text{ce-sets}; \ \text{computable} \ \{ \ n. \ \text{nat-to-ce-set} \ n \in C \} \right] ] \implies C = \{\} \lor C = \text{ce-sets} \)

**Proof** \( \text{rule ccontr} \)
assume \( A1 \): \( C \subseteq \text{ce-sets} \)
assume \( A2 \): \( \text{computable} \ \{ \ n. \ \text{nat-to-ce-set} \ n \in C \} \)
assume \( A3 \): \( \neg (C = \{\} \lor C = \text{ce-sets}) \)
from \( A3 \) have \( A4 \): \( C \neq \{\} \) by auto
from \( A3 \) have \( A5 \): \( C \neq \text{ce-sets} \) by auto
define \( A \) where \( A = \{ \ n. \ \text{nat-to-ce-set} \ n \in C \} \)
have \( S1 \): \( \text{index-set} \ A \)
**Proof** \(\text{unfold index-set-def}\)
show \( \forall n \ m. \ n \in A \land \text{nat-to-ce-set} \ n = \text{nat-to-ce-set} \ m \implies m \in A \)
proof \( \text{rule allI}, \text{rule allI}, \text{rule impl} \)
fix \( n \ m \) assume \( A1-1 \): \( n \in A \land \text{nat-to-ce-set} \ n = \text{nat-to-ce-set} \ m \)
from \( A1-1 \) have \( n \in A \) by auto
then have \( S1-1 \): \( \text{nat-to-ce-set} \ n \in C \) by \( \text{unfold A-def, auto} \)
from \( A1-1 \) have \( \text{nat-to-ce-set} \ n = \text{nat-to-ce-set} \ m \) by auto

167
with S1-1 have nat-to-ce-set \( m \in C \) by auto
then show \( m \in A \) by (unfold A-def, auto)
qed
qed
have S2: \( A \neq \{\} \)
proof –
from A4 obtain B where S2-1: \( B \in C \) by auto
with A1 have \( B \in \text{ce-sets} \) by auto
then have \( \exists \ n. \ B = \text{nat-to-ce-set } n \) by (rule nat-to-ce-set-srj)
then obtain n where \( B = \text{nat-to-ce-set } n \) ..
with S2-1 have nat-to-ce-set \( n \in C \) by auto
then show \( \text{thesis} \) by (unfold A-def, auto)
qed
have S3: \( A \neq \text{UNIV} \)
proof –
from A1 A5 obtain B where S2-1: \( B \notin C \) and S2-2: \( B \in \text{ce-sets} \) by auto
from S2-2 have \( \exists \ n. \ B = \text{nat-to-ce-set } n \) by (rule nat-to-ce-set-srj)
then obtain n where \( B = \text{nat-to-ce-set } n \) ..
with S2-1 have nat-to-ce-set \( n \notin C \) by auto
then show \( \text{thesis} \) by (unfold A-def, auto)
qed
from S1 S2 S3 have \( \neg \text{computable } A \) by (rule Rice-2)
with A2 show False unfolding A-def by auto
qed
end

References