

Recursion Theory I

Michael Nedzelsky

April 10, 2026

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	128
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: $\bigwedge x\ y.\ x=y \implies x\ div\ 2 = y\ div\ 2$ **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** *?thesis* **by** (*simp add: sf-def*)
qed

lemma *arg-le-sf*: $x \leq sf\ x$

proof –

have $x + x \leq x*(x + 1)$ **by** *simp*

hence $(x + x) \text{ div } 2 \leq x*(x+1) \text{ div } 2$ **by** (*rule div-le-mono*)

hence $x \leq x*(x+1) \text{ div } 2$ **by** *simp*

thus *?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** $x*(x+1) \leq y*(y+1)$ **by** (*rule mult-le-mono*)

then have $x*(x+1) \text{ div } 2 \leq y*(y+1) \text{ div } 2$ **by** (*rule div-le-mono*)

thus *?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 *S1* *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** *?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 *?thesis* **by** *simp*

qed

lemma *arg-less-sf*: $x > 1 \implies x < sf(x)$

proof –

assume *A1*: $x > 1$

let *?y* = $x - (1::nat)$

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** *?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 *?thesis* **by** *simp*
qed

lemma *sf-le-sfD*: $sf\ x \leq sf\ y \implies x \leq y$
proof –
 assume $A1: sf\ x \leq sf\ y$
 have $S1: y < x \implies sf\ y < sf\ x$ **by** (*rule sf-strict-mono*)
 have $S2: y < x \vee x \leq y$ **by** (*auto*)
 from $A1\ S1\ S2$ **show** *?thesis* **by** (*auto*)
qed

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

lemma *sf-inj*: $sf\ x = sf\ y \implies x = y$
proof –
 assume $A1: sf\ x = sf\ y$
 have $S1: sf\ x \leq sf\ y \implies x \leq y$ **by** (*rule sf-le-sfD*)
 have $S2: sf\ y \leq sf\ x \implies y \leq x$ **by** (*rule sf-le-sfD*)
 from $A1$ **have** $S3: sf\ x \leq sf\ y \wedge sf\ y \leq sf\ x$ **by** (*auto*)
 from $S3\ S1\ S2$ **have** $S4: x \leq y \wedge y \leq x$ **by** (*auto*)
 from $S4$ **show** *?thesis* **by** (*auto*)
qed

Auxiliary lemmas

lemma *sf-aux1*: $x + y < z \implies sf(x+y) + x < sf(z)$
proof –
 assume $A1: x+y < z$
 from $A1$ **have** $S1: x+y+1 \leq z$ **by** (*auto*)
 from $S1$ **have** $S2: sf(x+y+1) \leq sf(z)$ **by** (*rule sf-mono*)
 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** *?thesis* **by** (*auto*)
qed

lemma *sf-aux2*: $sf(z) \leq sf(x+y) + x \implies z \leq x+y$
proof –
 assume $A1: sf(z) \leq sf(x+y) + x$
 from $A1$ **have** $S1: \neg(sf(x+y) + x < sf(z))$ **by** (*auto*)
 from $S1\ sf-aux1$ **have** $S2: \neg(x+y < z)$ **by** (*auto*)

from $S2$ show *?thesis* by (auto)
qed

lemma *sf-aux3*: $sf(z) + m < sf(z+1) \implies m \leq z$

proof –

assume $A1: sf(z) + m < sf(z+1)$

have $S1: sf(z+1) = sf(z) + z + 1$ by (rule *sf-at-Suc*)

from $A1$ $S1$ have $S2: sf(z) + m < sf(z) + z + 1$ by (auto)

from $S2$ have $S3: m < z + 1$ by (auto)

from $S3$ show *?thesis* by (auto)

qed

lemma *sf-aux4*: $(s::nat) < t \implies (sf\ s) + s < sf\ t$

proof –

assume $A1: (s::nat) < t$

have $s*(s + 1) + 2*(s+1) \leq t*(t+1)$

proof –

from $A1$ have $S1: (s::nat) + 1 \leq t$ by (auto)

from $A1$ have $(s::nat) + 2 \leq t+1$ by (auto)

with $S1$ have $((s::nat)+1)*(s+2) \leq t*(t+1)$ by (rule *mult-le-mono*)

thus *?thesis* by (auto)

qed

then have $S1: (s*(s+1) + 2*(s+1))\ div\ 2 \leq t*(t+1)\ div\ 2$ by (rule *div-le-mono*)

have $(0::nat) < 2$ by (auto)

then have $(s*(s+1) + 2*(s+1))\ div\ 2 = (s+1) + (s*(s+1))\ div\ 2$ by *simp*

with $S1$ have $(s*(s+1))\ div\ 2 + (s+1) \leq t*(t+1)\ div\ 2$ by (auto)

then have $(s*(s+1))\ div\ 2 + s < t*(t+1)\ div\ 2$ by (auto)

thus *?thesis* by (*simp add: sf-def*)

qed

Basic properties of *c_pair* function

lemma *sum-le-c-pair*: $x + y \leq c_pair\ x\ y$

proof –

have $x+y \leq sf(x+y)$ by (rule *arg-le-sf*)

thus *?thesis* by (*simp add: c-pair-def*)

qed

lemma *arg1-le-c-pair*: $x \leq c_pair\ x\ y$

proof –

have $(x::nat) \leq x + y$ by (*simp*)

moreover have $x + y \leq c_pair\ x\ y$ by (rule *sum-le-c-pair*)

ultimately show *?thesis* by (*simp*)

qed

lemma *arg2-le-c-pair*: $y \leq c_pair\ x\ y$

proof –

have $(y::nat) \leq x + y$ by (*simp*)

moreover have $x + y \leq c_pair\ x\ y$ by (rule *sum-le-c-pair*)

ultimately show *?thesis* by (*simp*)

qed

lemma *c-pair-sum-mono*: $(x1::nat) + y1 < x2 + y2 \implies c\text{-pair } x1 \ y1 < c\text{-pair } x2 \ y2$

proof –

assume $(x1::nat) + y1 < x2 + y2$

hence $sf \ (x1+y1) + (x1+y1) < sf \ (x2+y2)$ **by** (*rule sf-aux4*)

hence $sf \ (x1+y1) + x1 < sf \ (x2+y2) + x2$ **by** (*auto*)

thus *?thesis* **by** (*simp add: c-pair-def*)

qed

lemma *c-pair-sum-inj*: $c\text{-pair } x1 \ y1 = c\text{-pair } x2 \ y2 \implies x1 + y1 = x2 + y2$

proof –

assume *A1*: $c\text{-pair } x1 \ y1 = c\text{-pair } x2 \ y2$

have *S1*: $(x1::nat) + y1 < x2 + y2 \implies c\text{-pair } x1 \ y1 \neq c\text{-pair } x2 \ y2$ **by** (*rule less-not-refl3, rule c-pair-sum-mono, auto*)

have *S2*: $(x2::nat) + y2 < x1 + y1 \implies c\text{-pair } x1 \ y1 \neq c\text{-pair } x2 \ y2$ **by** (*rule less-not-refl2, rule c-pair-sum-mono, auto*)

from *S1 S2* **have** $(x1::nat) + y1 \neq x2 + y2 \implies c\text{-pair } x1 \ y1 \neq c\text{-pair } x2 \ y2$ **by** (*arith*)

with *A1* **show** *?thesis* **by** (*auto*)

qed

lemma *c-pair-inj*: $c\text{-pair } x1 \ y1 = c\text{-pair } x2 \ y2 \implies x1 = x2 \wedge y1 = y2$

proof –

assume *A1*: $c\text{-pair } x1 \ y1 = c\text{-pair } x2 \ y2$

from *A1* **have** *S1*: $x1 + y1 = x2 + y2$ **by** (*rule c-pair-sum-inj*)

from *A1* **have** *S2*: $sf \ (x1+y1) + x1 = sf \ (x2+y2) + x2$ **by** (*unfold c-pair-def*)

from *S1 S2* **have** *S3*: $x1 = x2$ **by** (*simp*)

from *S1 S3* **have** *S4*: $y1 = y2$ **by** (*simp*)

from *S3 S4* **show** *?thesis* **by** (*auto*)

qed

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

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

lemma *c-pair-strict-mono1*: $x1 < x2 \implies c\text{-pair } x1 \ y < c\text{-pair } x2 \ y$

proof –

assume $x1 < x2$

then **have** $x1 + y < x2 + y$ **by** *simp*

then **show** *?thesis* **by** (*rule c-pair-sum-mono*)

qed

lemma *c-pair-mono1*: $x1 \leq x2 \implies c\text{-pair } x1 \ y \leq c\text{-pair } x2 \ y$

proof –

assume *A1*: $x1 \leq x2$

```

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

```

```

lemma  $c\text{-pair-strict-mono2}$ :  $y1 < y2 \implies c\text{-pair } x\ y1 < c\text{-pair } x\ y2$ 
proof -
  assume  $A1$ :  $y1 < y2$ 
  from  $A1$  have  $S1$ :  $x + y1 < x + y2$  by simp
  then show ?thesis by (rule  $c\text{-pair-sum-mono}$ )
qed

```

```

lemma  $c\text{-pair-mono2}$ :  $y1 \leq y2 \implies c\text{-pair } x\ y1 \leq c\text{-pair } x\ y2$ 
proof -
  assume  $A1$ :  $y1 \leq y2$ 
  show ?thesis
  proof cases
    assume  $y1 < y2$ 
    then have  $c\text{-pair } x\ y1 < c\text{-pair } x\ y2$  by (rule  $c\text{-pair-strict-mono2}$ )
    then show ?thesis by simp
  next
    assume  $\neg y1 < y2$ 
    with  $A1$  have  $y1 = y2$  by simp
    then show ?thesis by simp
  qed
qed

```

1.2 Inverse mapping

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

definition

```

 $c\text{-sum} :: nat \Rightarrow nat$  where
 $c\text{-sum } u = (LEAST\ z.\ u < sf\ (z+1))$ 

```

definition

```

 $c\text{-fst} :: nat \Rightarrow nat$  where
 $c\text{-fst } u = u - sf\ (c\text{-sum } u)$ 

```

definition

```

 $c\text{-snd} :: nat \Rightarrow nat$  where
 $c\text{-snd } u = c\text{-sum } u - c\text{-fst } u$ 

```

lemma *arg-less-sf-at-Suc-of-c-sum*: $u < sf ((c-sum\ u) + 1)$

proof –

have $u+1 \leq sf(u+1)$ **by** (*rule arg-le-sf*)

hence $u < sf(u+1)$ **by** *simp*

thus *?thesis* **by** (*unfold c-sum-def, rule LeastI*)

qed

lemma *arg-less-sf-imp-c-sum-less-arg*: $u < sf(x) \implies c-sum\ u < x$

proof –

assume *A1*: $u < sf(x)$

then show *?thesis*

proof (*cases x*)

assume $x=0$

with *A1* **show** *?thesis* **by** (*simp add: sf-def*)

next

fix y

assume *A2*: $x = Suc\ y$

show *?thesis*

proof –

from *A1 A2* **have** $u < sf(y+1)$ **by** *simp*

hence (*Least (%z. u < sf (z+1))*) $\leq y$ **by** (*rule Least-le*)

hence $c-sum\ u \leq y$ **by** (*fold c-sum-def*)

with *A2* **show** *?thesis* **by** *simp*

qed

qed

qed

lemma *sf-c-sum-le-arg*: $u \geq sf (c-sum\ u)$

proof –

let $?z = c-sum\ u$

from *arg-less-sf-at-Suc-of-c-sum* **have** *S1*: $u < sf (?z+1)$ **by** (*auto*)

have *S2*: $\neg c-sum\ u < c-sum\ u$ **by** (*auto*)

from *arg-less-sf-imp-c-sum-less-arg S2* **have** *S3*: $\neg u < sf (c-sum\ u)$ **by** (*auto*)

from *S3* **show** *?thesis* **by** (*auto*)

qed

lemma *c-sum-le-arg*: $c-sum\ u \leq u$

proof –

have $c-sum\ u \leq sf (c-sum\ u)$ **by** (*rule arg-le-sf*)

moreover have $sf(c-sum\ u) \leq u$ **by** (*rule sf-c-sum-le-arg*)

ultimately show *?thesis* **by** *simp*

qed

lemma *c-sum-of-c-pair [simp]*: $c-sum (c-pair\ x\ y) = x + y$

proof –

let $?u = c-pair\ x\ y$

let $?z = c-sum\ ?u$

have *S1*: $?u < sf(?z+1)$ **by** (*rule arg-less-sf-at-Suc-of-c-sum*)

have *S2*: $sf(?z) \leq ?u$ **by** (*rule sf-c-sum-le-arg*)

from $S1$ **have** $S3: sf(x+y)+x < sf(?z+1)$ **by** (*simp add: c-pair-def*)
from $S2$ **have** $S4: sf(?z) \leq sf(x+y) + x$ **by** (*simp add: c-pair-def*)
from $S3$ **have** $S5: sf(x+y) < sf(?z+1)$ **by** (*auto*)
from $S5$ **have** $S6: x+y < ?z+1$ **by** (*rule sf-less-sfD*)
from $S6$ **have** $S7: x+y \leq ?z$ **by** (*auto*)
from $S4$ **have** $S8: ?z \leq x+y$ **by** (*rule sf-aux2*)
from $S7$ $S8$ **have** $S9: ?z = x+y$ **by** (*auto*)
from $S9$ **show** $?thesis$ **by** (*simp*)
qed

lemma *c-fst-of-c-pair[simp]*: $c\text{-fst } (c\text{-pair } x \ y) = x$
proof –

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

lemma *c-snd-of-c-pair[simp]*: $c\text{-snd } (c\text{-pair } x \ y) = y$
proof –

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

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

lemma *c-fst-at-0*: $c\text{-fst } 0 = 0$
proof –

have $c\text{-pair } 0 \ 0 = 0$ **by** (*rule c-pair-at-0*)
hence $c\text{-fst } 0 = c\text{-fst } (c\text{-pair } 0 \ 0)$ **by** *simp*
thus $?thesis$ **by** *simp*
qed

lemma *c-snd-at-0*: $c\text{-snd } 0 = 0$
proof –

have $c\text{-pair } 0 \ 0 = 0$ **by** (*rule c-pair-at-0*)
hence $c\text{-snd } 0 = c\text{-snd } (c\text{-pair } 0 \ 0)$ **by** *simp*
thus $?thesis$ **by** *simp*
qed

lemma *sf-c-sum-plus-c-fst*: $sf(c\text{-sum } u) + c\text{-fst } u = u$
proof –

have $S1: sf(c\text{-sum } u) \leq u$ **by** (*rule sf-c-sum-le-arg*)
have $S2: c\text{-fst } u = u - sf(c\text{-sum } u)$ **by** (*simp add: c-fst-def*)
from $S1$ $S2$ **show** $?thesis$ **by** (*auto*)
qed

lemma *c-fst-le-c-sum*: $c\text{-fst } u \leq c\text{-sum } u$

proof –

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

have *S2*: $u < \text{sf}((c\text{-sum } u) + 1)$ **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*: $c\text{-snd } u \leq c\text{-sum } u$ **by** (*simp add: c-snd-def*)

lemma *c-fst-le-arg*: $c\text{-fst } u \leq u$

proof –

have $c\text{-fst } u \leq c\text{-sum } u$ **by** (*rule c-fst-le-c-sum*)

moreover **have** $c\text{-sum } u \leq u$ **by** (*rule c-sum-le-arg*)

ultimately show *?thesis* **by** *simp*

qed

lemma *c-snd-le-arg*: $c\text{-snd } u \leq u$

proof –

have $c\text{-snd } u \leq c\text{-sum } u$ **by** (*rule c-snd-le-c-sum*)

moreover **have** $c\text{-sum } u \leq u$ **by** (*rule c-sum-le-arg*)

ultimately show *?thesis* **by** *simp*

qed

lemma *c-sum-is-sum*: $c\text{-sum } u = c\text{-fst } u + c\text{-snd } u$ **by** (*simp add: c-snd-def c-fst-le-c-sum*)

lemma *proj-eq-imp-arg-eq*: $\llbracket c\text{-fst } u = c\text{-fst } v; c\text{-snd } u = c\text{-snd } v \rrbracket \implies u = v$

proof –

assume *A1*: $c\text{-fst } u = c\text{-fst } v$

assume *A2*: $c\text{-snd } u = c\text{-snd } v$

from *A1 A2 c-sum-is-sum* **have** *S1*: $c\text{-sum } u = c\text{-sum } v$ **by** (*auto*)

have *S2*: $\text{sf}(c\text{-sum } u) + c\text{-fst } u = u$ **by** (*rule sf-c-sum-plus-c-fst*)

from *A1 S1 S2* **have** *S3*: $\text{sf}(c\text{-sum } v) + c\text{-fst } v = u$ **by** (*auto*)

from *S3 sf-c-sum-plus-c-fst* **show** *?thesis* **by** (*auto*)

qed

lemma *c-pair-of-c-fst-c-snd[simp]*: $c\text{-pair } (c\text{-fst } u) (c\text{-snd } u) = u$

proof –

let *?x* = $c\text{-fst } u$

let *?y* = $c\text{-snd } u$

have *S1*: $c\text{-pair } ?x ?y = \text{sf}(?x + ?y) + ?x$ **by** (*simp add: c-pair-def*)

have *S2*: $c\text{-sum } u = ?x + ?y$ **by** (*rule c-sum-is-sum*)

from *S1 S2* **have** $c\text{-pair } ?x ?y = \text{sf}(c\text{-sum } u) + c\text{-fst } u$ **by** (*auto*)

thus *?thesis* **by** (*simp add: sf-c-sum-plus-c-fst*)

qed

lemma *c-sum-eq-arg*: $c\text{-sum } x = x \implies x \leq 1$

proof –

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

lemma *c-sum-eq-arg-2*: $c\text{-sum } x = x \implies c\text{-fst } x = 0$

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

lemma *c-fst-eq-arg*: $c\text{-fst } x = x \implies x = 0$

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

lemma *c-fst-less-arg*: $x > 0 \implies c\text{-fst } x < x$

proof –
assume $A1: x > 0$
show *?thesis*
proof *cases*
assume $c\text{-fst } x < x$
then **show** *?thesis* **by** *simp*
next
assume $\neg c\text{-fst } x < x$
then **have** $S1: c\text{-fst } x \geq x$ **by** *simp*
have $c\text{-fst } x \leq x$ **by** (*rule c-fst-le-arg*)
with $S1$ **have** $c\text{-fst } x = x$ **by** *simp*
then **have** $x = 0$ **by** (*rule c-fst-eq-arg*)
with $A1$ **show** *?thesis* **by** *simp*
qed
qed

lemma *c-snd-eq-arg*: $c\text{-snd } x = x \implies x \leq 1$

proof –
assume $A1: c\text{-snd } x = x$
have $S1: c\text{-snd } x \leq c\text{-sum } x$ **by** (*rule c-snd-le-c-sum*)
have $S2: c\text{-sum } x \leq x$ **by** (*rule c-sum-le-arg*)

```

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

```

```

lemma c-snd-less-arg: x > 1  $\implies$  c-snd x < x

```

```

proof –

```

```

  assume A1: x > 1

```

```

  show ?thesis

```

```

  proof cases

```

```

    assume c-snd x < x

```

```

    then show ?thesis .

```

```

  next

```

```

    assume  $\neg$  c-snd x < x

```

```

    then have S1: c-snd x  $\geq$  x by auto

```

```

    have c-snd x  $\leq$  x by (rule c-snd-le-arg)

```

```

    with S1 have c-snd x = x by simp

```

```

    then have x  $\leq$  1 by (rule c-snd-eq-arg)

```

```

    with A1 show ?thesis by simp

```

```

  qed

```

```

qed

```

```

end

```

2 Primitive recursive functions

```

theory PRecFun imports CPair

```

```

begin

```

This theory contains definition of the primitive recursive functions.

2.1 Basic definitions

```

primrec

```

```

  PrimRecOp :: (nat  $\Rightarrow$  nat)  $\Rightarrow$  (nat  $\Rightarrow$  nat  $\Rightarrow$  nat  $\Rightarrow$  nat)  $\Rightarrow$  (nat  $\Rightarrow$  nat  $\Rightarrow$  nat)

```

```

where

```

```

  PrimRecOp g h 0 x = g x

```

```

| PrimRecOp g h (Suc y) x = h y (PrimRecOp g h y x) x

```

```

primrec

```

```

  PrimRecOp-last :: (nat  $\Rightarrow$  nat)  $\Rightarrow$  (nat  $\Rightarrow$  nat  $\Rightarrow$  nat  $\Rightarrow$  nat)  $\Rightarrow$  (nat  $\Rightarrow$  nat  $\Rightarrow$ 
nat)

```

```

where

```

```

  PrimRecOp-last g h x 0 = g x

```

```

| PrimRecOp-last g h x (Suc y) = h x (PrimRecOp-last g h x y) y

```

```

primrec

```

```

  PrimRecOp1 :: nat  $\Rightarrow$  (nat  $\Rightarrow$  nat  $\Rightarrow$  nat)  $\Rightarrow$  (nat  $\Rightarrow$  nat)

```

```

where

```

```

  PrimRecOp1 a h 0 = a

```

| $\text{PrimRecOp1 } a \ h \ (\text{Suc } y) = h \ y \ (\text{PrimRecOp1 } a \ h \ y)$

inductive-set

$\text{PrimRec1} :: (\text{nat} \Rightarrow \text{nat}) \text{ set and}$

$\text{PrimRec2} :: (\text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat}) \text{ set and}$

$\text{PrimRec3} :: (\text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat}) \text{ set}$

where

$\text{zero}: (\lambda x. 0) \in \text{PrimRec1}$

| $\text{suc}: \text{Suc} \in \text{PrimRec1}$

| $\text{id1-1}: (\lambda x. x) \in \text{PrimRec1}$

| $\text{id2-1}: (\lambda x \ y. x) \in \text{PrimRec2}$

| $\text{id2-2}: (\lambda x \ y. y) \in \text{PrimRec2}$

| $\text{id3-1}: (\lambda x \ y \ z. x) \in \text{PrimRec3}$

| $\text{id3-2}: (\lambda x \ y \ z. y) \in \text{PrimRec3}$

| $\text{id3-3}: (\lambda x \ y \ z. z) \in \text{PrimRec3}$

| $\text{comp1-1}: \llbracket f \in \text{PrimRec1}; g \in \text{PrimRec1} \rrbracket \Longrightarrow (\lambda x. f \ (g \ x)) \in \text{PrimRec1}$

| $\text{comp1-2}: \llbracket f \in \text{PrimRec1}; g \in \text{PrimRec2} \rrbracket \Longrightarrow (\lambda x \ y. f \ (g \ x \ y)) \in \text{PrimRec2}$

| $\text{comp1-3}: \llbracket f \in \text{PrimRec1}; g \in \text{PrimRec3} \rrbracket \Longrightarrow (\lambda x \ y \ z. f \ (g \ x \ y \ z)) \in \text{PrimRec3}$

| $\text{comp2-1}: \llbracket f \in \text{PrimRec2}; g \in \text{PrimRec1}; h \in \text{PrimRec1} \rrbracket \Longrightarrow (\lambda x. f \ (g \ x) \ (h \ x)) \in \text{PrimRec1}$

| $\text{comp3-1}: \llbracket f \in \text{PrimRec3}; g \in \text{PrimRec1}; h \in \text{PrimRec1}; k \in \text{PrimRec1} \rrbracket \Longrightarrow (\lambda x. f \ (g \ x) \ (h \ x) \ (k \ x)) \in \text{PrimRec1}$

| $\text{comp2-2}: \llbracket f \in \text{PrimRec2}; g \in \text{PrimRec2}; h \in \text{PrimRec2} \rrbracket \Longrightarrow (\lambda x \ y. f \ (g \ x \ y) \ (h \ x \ y)) \in \text{PrimRec2}$

| $\text{comp2-3}: \llbracket f \in \text{PrimRec2}; g \in \text{PrimRec3}; h \in \text{PrimRec3} \rrbracket \Longrightarrow (\lambda x \ y \ z. f \ (g \ x \ y \ z) \ (h \ x \ y \ z)) \in \text{PrimRec3}$

| $\text{comp3-2}: \llbracket f \in \text{PrimRec3}; g \in \text{PrimRec2}; h \in \text{PrimRec2}; k \in \text{PrimRec2} \rrbracket \Longrightarrow (\lambda x \ y. f \ (g \ x \ y) \ (h \ x \ y) \ (k \ x \ y)) \in \text{PrimRec2}$

| $\text{comp3-3}: \llbracket f \in \text{PrimRec3}; g \in \text{PrimRec3}; h \in \text{PrimRec3}; k \in \text{PrimRec3} \rrbracket \Longrightarrow (\lambda x \ y \ z. f \ (g \ x \ y \ z) \ (h \ x \ y \ z) \ (k \ x \ y \ z)) \in \text{PrimRec3}$

| $\text{prim-rec}: \llbracket g \in \text{PrimRec1}; h \in \text{PrimRec3} \rrbracket \Longrightarrow \text{PrimRecOp } g \ h \in \text{PrimRec2}$

lemmas $\text{pr-zero} = \text{PrimRec1-PrimRec2-PrimRec3.zero}$

lemmas $\text{pr-suc} = \text{PrimRec1-PrimRec2-PrimRec3.suc}$

lemmas $\text{pr-id1-1} = \text{PrimRec1-PrimRec2-PrimRec3.id1-1}$

lemmas $\text{pr-id2-1} = \text{PrimRec1-PrimRec2-PrimRec3.id2-1}$

lemmas $\text{pr-id2-2} = \text{PrimRec1-PrimRec2-PrimRec3.id2-2}$

lemmas $\text{pr-id3-1} = \text{PrimRec1-PrimRec2-PrimRec3.id3-1}$

lemmas $\text{pr-id3-2} = \text{PrimRec1-PrimRec2-PrimRec3.id3-2}$

lemmas $\text{pr-id3-3} = \text{PrimRec1-PrimRec2-PrimRec3.id3-3}$

lemmas $\text{pr-comp1-1} = \text{PrimRec1-PrimRec2-PrimRec3.comp1-1}$

lemmas $\text{pr-comp1-2} = \text{PrimRec1-PrimRec2-PrimRec3.comp1-2}$

lemmas $\text{pr-comp1-3} = \text{PrimRec1-PrimRec2-PrimRec3.comp1-3}$

lemmas $\text{pr-comp2-1} = \text{PrimRec1-PrimRec2-PrimRec3.comp2-1}$

lemmas $\text{pr-comp2-2} = \text{PrimRec1-PrimRec2-PrimRec3.comp2-2}$

lemmas $\text{pr-comp2-3} = \text{PrimRec1-PrimRec2-PrimRec3.comp2-3}$

lemmas $\text{pr-comp3-1} = \text{PrimRec1-PrimRec2-PrimRec3.comp3-1}$

lemmas $\text{pr-comp3-2} = \text{PrimRec1-PrimRec2-PrimRec3.comp3-2}$

lemmas $\text{pr-comp3-3} = \text{PrimRec1-PrimRec2-PrimRec3.comp3-3}$

lemmas *pr-rec* = *PrimRec1-PrimRec2-PrimRec3.prim-rec*

ML-file \langle *Utils.ML* \rangle

named-theorems *prec*

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

lemmas [*prec*] = *pr-zero pr-suc pr-id1-1 pr-id2-1 pr-id2-2 pr-id3-1 pr-id3-2 pr-id3-3*

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

theorem *pr-rec-scheme*: $\llbracket g \in \text{PrimRec1}; h \in \text{PrimRec3}; \forall x. f 0 x = g x; \forall x y. f (\text{Suc } y) x = h y (f y x) x \rrbracket \implies f \in \text{PrimRec2}$

proof –

assume *g-is-pr*: $g \in \text{PrimRec1}$

assume *h-is-pr*: $h \in \text{PrimRec3}$

assume *f-at-0*: $\forall x. f 0 x = g x$

assume *f-at-Suc*: $\forall x y. f (\text{Suc } y) x = h y (f y x) x$

from *f-at-0 f-at-Suc* **have** $\bigwedge x y. f y x = \text{PrimRecOp } g h y x$ **by** (*induct-tac y, simp-all*)

then have $f = \text{PrimRecOp } g h$ **by** (*simp add: ext*)

with *g-is-pr h-is-pr* **show** *?thesis* **by** (*simp add: pr-rec*)

qed

lemma *op-plus-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*: $\text{PrimRecOp } (\lambda x. x) (\lambda x y z. \text{Suc } y) \in \text{PrimRec2}$

proof (*rule pr-rec*)

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

next

show $(\lambda x y z. \text{Suc } y) \in \text{PrimRec3}$ **by** *prec0*

qed

have $(\lambda x y. y+x) = \text{PrimRecOp } (\lambda x. x) (\lambda x y z. \text{Suc } y)$ (**is** $- = ?f$)

proof –

have $\bigwedge x y. (?f y x = y + x)$ **by** (*induct-tac y, auto*)

thus *?thesis* **by** (*simp add: ext*)

qed

with *S1* **show** *?thesis* **by** *simp*

qed

qed

```

lemma op-mult-is-pr [prec]:  $(\lambda x y. x*y) \in PrimRec2$ 
proof (rule pr-swap)
  show  $(\lambda x y. y*x) \in PrimRec2$ 
  proof –
    have S1: PrimRecOp  $(\lambda x. 0)$   $(\lambda x y z. y+z) \in PrimRec2$ 
    proof (rule pr-rec)
      show  $(\lambda x. 0) \in PrimRec1$  by (rule pr-zero)
    next
      show  $(\lambda x y z. y+z) \in PrimRec3$  by prec0
    qed
    have  $(\lambda x y. y*x) = PrimRecOp$   $(\lambda x. 0)$   $(\lambda x y z. y+z)$  (is  $- = ?f$ )
    proof –
      have  $\bigwedge x y. (?f\ y\ x = y * x)$  by (induct-tac y, auto)
      thus ?thesis by (simp add: ext)
    qed
  with S1 show ?thesis by simp
qed
qed

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

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

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

theorem pr-rec-last:  $[[g \in PrimRec1; h \in PrimRec3]] \implies PrimRecOp\text{-last}\ g\ h \in PrimRec2$ 
proof –
  assume A1:  $g \in PrimRec1$ 
  assume A2:  $h \in PrimRec3$ 
  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:  $?h1 \in PrimRec3$  by (rule pr-comp3-3)

```

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

theorem pr-rec1: $h \in \text{PrimRec2} \implies \text{PrimRecOp1 } (a::\text{nat}) \ h \in \text{PrimRec1}$
proof –

assume $A1: h \in \text{PrimRec2}$
let $?g = (\lambda x. a)$
have $g\text{-is-pr}: ?g \in \text{PrimRec1}$ **by** (*rule const-is-pr*)
let $?h1 = (\lambda x y z. h \ x \ y)$
from $A1$ **have** $h1\text{-is-pr}: ?h1 \in \text{PrimRec3}$ **by** *prec0*
let $?f1 = \text{PrimRecOp } ?g \ ?h1$
from $g\text{-is-pr } h1\text{-is-pr}$ **have** $f1\text{-is-pr}: ?f1 \in \text{PrimRec2}$ **by** (*rule pr-rec*)
let $?f = (\lambda x. ?f1 \ x \ 0)$
from $f1\text{-is-pr } pr\text{-id1-1 } pr\text{-zero}$ **have** $f\text{-is-pr}: ?f \in \text{PrimRec1}$ **by** (*rule pr-comp2-1*)
have $\bigwedge y. ?f \ y = \text{PrimRecOp1 } a \ h \ y$ **by** (*induct-tac y, auto*)
then have $?f = \text{PrimRecOp1 } a \ h$ **by** (*simp add: ext*)
with $f\text{-is-pr}$ **show** $?thesis$ **by** (*auto*)
qed

theorem pr-rec1-scheme: $\llbracket h \in \text{PrimRec2}; f \ 0 = a; \forall y. f \ (\text{Suc } y) = h \ y \ (f \ y) \rrbracket$
 $\implies f \in \text{PrimRec1}$

proof –
assume $h\text{-is-pr}: h \in \text{PrimRec2}$
assume $f\text{-at-0}: f \ 0 = a$
assume $f\text{-at-Suc}: \forall y. f \ (\text{Suc } y) = h \ y \ (f \ y)$
from $f\text{-at-0 } f\text{-at-Suc}$ **have** $\bigwedge y. f \ y = \text{PrimRecOp1 } a \ h \ y$ **by** (*induct-tac y, simp-all*)
then have $f = \text{PrimRecOp1 } a \ h$ **by** (*simp add: ext*)
with $h\text{-is-pr}$ **show** $?thesis$ **by** (*simp add: pr-rec1*)
qed

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

proof –
have $S1: \text{PrimRecOp1 } 0 \ (\lambda x y. x) \in \text{PrimRec1}$
proof (*rule pr-rec1*)
show $(\lambda x y. x) \in \text{PrimRec2}$ **by** (*rule pr-id2-1*)
qed
have $(\lambda x. x - (1::\text{nat})) = \text{PrimRecOp1 } 0 \ (\lambda x y. x)$ (**is - = ?f**)
proof –
have $\bigwedge x. (?f \ x = x - (1::\text{nat}))$ **by** (*induct-tac x, auto*)
thus $?thesis$ **by** (*simp add: ext*)
qed
with $S1$ **show** $?thesis$ **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*: $\text{PrimRecOp } (\lambda x. x) (\lambda x y z. y - (1::\text{nat})) \in \text{PrimRec2}$

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** $(\lambda x y z. y - (1::\text{nat})) \in \text{PrimRec3}$ **by** (*rule pr-comp1-3*)

qed

have $(\lambda x y. y - x) = \text{PrimRecOp } (\lambda x. x) (\lambda x y z. y - (1::\text{nat}))$ (**is** - = ?*f*)

proof -

have $\bigwedge x y. (?f y x = x - y)$ **by** (*induct-tac y, auto*)

thus ?*thesis* **by** (*simp add: ext*)

qed

with *S1* **show** ?*thesis* **by** *simp*

qed

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* = (*case* *x* *of* 0 \Rightarrow 0 | *Suc* *y* \Rightarrow 1)

definition

sgn2 :: $\text{nat} \Rightarrow \text{nat}$ **where**

sgn2 *x* \equiv (*case* *x* *of* 0 \Rightarrow 1 | *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* (*Suc* *y*) = 1 **by** (*simp add: sgn1-def*)

lemma [*simp*]: *sgn2* 0 = 1 **by** (*simp add: sgn2-def*)

lemma [*simp*]: *sgn2* (*Suc* *y*) = 0 **by** (*simp add: sgn2-def*)

lemma [*simp*]: $x \neq 0 \Longrightarrow \text{sgn1 } x = 1$ **by** (*simp add: sgn1-def, cases x, auto*)

lemma [*simp*]: $x \neq 0 \Longrightarrow \text{sgn2 } x = 0$ **by** (*simp add: sgn2-def, cases x, auto*)

lemma *sgn1-nz-impl-arg-pos*: $\text{sgn1 } x \neq 0 \Longrightarrow x > 0$ **by** (*cases x*) *auto*

lemma *sgn1-zero-impl-arg-zero*: $\text{sgn1 } x = 0 \Longrightarrow x = 0$ **by** (*cases x*) *auto*

lemma *sgn2-nz-impl-arg-zero*: $\text{sgn2 } x \neq 0 \Longrightarrow x = 0$ **by** (*cases x*) *auto*

lemma *sgn2-zero-impl-arg-pos*: $\text{sgn2 } x = 0 \implies x > 0$ **by** (*cases x*) *auto*

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

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

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

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

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

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

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

proof (*rule ext*)

fix *x* **show** $\text{sgn2 } x = 1 - x$ **by** (*cases x*) *auto*

qed

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

proof

fix *x* **show** $\text{sgn1 } x = 1 - \text{sgn2 } x$

proof -

have $1 - \text{sgn2 } x = 1 - (1 - x)$ **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*]: $\text{sgn2} \in \text{PrimRec1}$

proof -

have $(\lambda x. 1 - x) \in \text{PrimRec1}$ **by** *prec0*

thus *?thesis* **by** (*simp add: sgn2-eq-1-sub-arg*)

qed

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

proof -

from *sgn2-is-pr* **have** $(\lambda x. 1 - (\text{sgn2 } x)) \in \text{PrimRec1}$ **by** *prec0*

thus *?thesis* **by** (*simp add: sgn1-eq-1-sub-sgn2*)

qed

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

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

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

proof -

have *S1*: $\text{PrimRecOp1 } 0 (\lambda x y. y + x + 1) \in \text{PrimRec1}$

proof (*rule pr-rec1*)

show $(\lambda x y. y + x + 1) \in \text{PrimRec2}$ **by** *prec0*

qed

have $(\lambda x. \text{sf } x) = \text{PrimRecOp1 } 0 (\lambda x y. y + x + 1)$ (**is** - = *?f*)

```

proof –
  have  $\bigwedge x. (?f\ x = sf\ x)$ 
  proof (induct-tac x)
    show  $?f\ 0 = sf\ 0$  by (simp add: sf-at-0)
  next
    fix x assume  $?f\ x = sf\ x$ 
    with sf-at-Suc show  $?f\ (Suc\ x) = sf\ (Suc\ x)$  by auto
  qed
  thus ?thesis by (simp add: ext)
qed
with S1 show ?thesis by simp
qed

```

```

lemma c-pair-is-pr [prec]:  $c\text{-pair} \in PrimRec2$ 
proof –
  have  $c\text{-pair} = (\lambda\ x\ y. sf\ (x+y) + x)$  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:  $\llbracket p \in PrimRec1; q1 \in PrimRec1; q2 \in PrimRec1 \rrbracket \implies (\lambda\ x. if\ (p\ x = 0)\ then\ (q1\ x)\ else\ (q2\ x)) \in PrimRec1$ 
proof –
  have if-as-pr:  $(\lambda\ x. if\ (p\ x = 0)\ then\ (q1\ x)\ else\ (q2\ x)) = (\lambda\ x. (sgn2\ (p\ x)) * (q1\ x) + (sgn1\ (p\ x)) * (q2\ x))$ 
  proof (rule ext)
    fix x show  $(if\ (p\ x = 0)\ then\ (q1\ x)\ else\ (q2\ x)) = (sgn2\ (p\ x)) * (q1\ x) + (sgn1\ (p\ x)) * (q2\ x)$  (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
  assume  $p \in PrimRec1$  and  $q1 \in PrimRec1$  and  $q2 \in PrimRec1$ 
  then have  $(\lambda\ x. (sgn2\ (p\ x)) * (q1\ x) + (sgn1\ (p\ x)) * (q2\ x)) \in PrimRec1$  by prec0
  with if-as-pr show ?thesis by simp
qed

```

```

lemma if-eq-is-pr [prec]:  $\llbracket p1 \in PrimRec1; p2 \in PrimRec1; q1 \in PrimRec1; q2 \in PrimRec1 \rrbracket \implies (\lambda\ x. if\ (p1\ x = p2\ x)\ then\ (q1\ x)\ else\ (q2\ x)) \in PrimRec1$ 
proof –
  have S1:  $(\lambda\ x. if\ (p1\ x = p2\ x)\ then\ (q1\ x)\ else\ (q2\ x)) = (\lambda\ x. if\ (abs\ of\ diff\ (p1$ 

```

$x) (p2\ x) = 0) \text{ then } (q1\ x) \text{ else } (q2\ x))$ (**is** $?L = ?R$) **by** (*simp add: abs-of-diff-eq*)
assume $A1: p1 \in PrimRec1$ **and** $A2: p2 \in PrimRec1$
with *abs-of-diff-is-pr* **have** $S2: (\lambda\ x. \text{abs-of-diff } (p1\ x) (p2\ x)) \in PrimRec1$ **by**
prec0
assume $q1 \in PrimRec1$ **and** $q2 \in PrimRec1$
with $S2$ **have** $?R \in PrimRec1$ **by** (*rule if-is-pr*)
with $S1$ **show** *?thesis* **by** *simp*
qed

lemma *if-is-pr2* [*prec*]: $\llbracket p \in PrimRec2; q1 \in PrimRec2; q2 \in PrimRec2 \rrbracket \implies (\lambda\ x\ y. \text{if } (p\ x\ y = 0) \text{ then } (q1\ x\ y) \text{ else } (q2\ x\ y)) \in PrimRec2$
proof –
have *if-as-pr*: $(\lambda\ x\ y. \text{if } (p\ x\ y = 0) \text{ then } (q1\ x\ y) \text{ else } (q2\ x\ y)) = (\lambda\ x\ y. (\text{sgn2 } (p\ x\ y)) * (q1\ x\ y) + (\text{sgn1 } (p\ x\ y)) * (q2\ x\ y))$
proof (*rule ext, rule ext*)
fix x **fix** y **show** $(\text{if } (p\ x\ y = 0) \text{ then } (q1\ x\ y) \text{ else } (q2\ x\ y)) = (\text{sgn2 } (p\ x\ y)) * (q1\ x\ y) + (\text{sgn1 } (p\ x\ y)) * (q2\ x\ y)$ (**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 $A2: p\ x\ y \neq 0$
then **have** $S3: p\ x\ y > 0$ **by** *simp*
then **show** *?thesis* **by** *simp*
qed
qed
assume $p \in PrimRec2$ **and** $q1 \in PrimRec2$ **and** $q2 \in PrimRec2$
then **have** $(\lambda\ x\ y. (\text{sgn2 } (p\ x\ y)) * (q1\ x\ y) + (\text{sgn1 } (p\ x\ y)) * (q2\ x\ y)) \in PrimRec2$ **by** *prec0*
with *if-as-pr* **show** *?thesis* **by** *simp*
qed

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

lemma *if-is-pr3* [*prec*]: $\llbracket p \in PrimRec3; q1 \in PrimRec3; q2 \in PrimRec3 \rrbracket \implies (\lambda$

$x\ y\ z.$ *if* ($p\ x\ y\ z = 0$) *then* ($q1\ x\ y\ z$) *else* ($q2\ x\ y\ z$) $\in PrimRec3$
proof –
have *if-as-pr*: $(\lambda\ x\ y\ z. \text{if } (p\ x\ y\ z = 0) \text{ then } (q1\ x\ y\ z) \text{ else } (q2\ x\ y\ z)) = (\lambda\ x\ y\ z. (sgn2\ (p\ x\ y\ z)) * (q1\ x\ y\ z) + (sgn1\ (p\ x\ y\ z)) * (q2\ x\ y\ z))$
proof (*rule ext*, *rule ext*, *rule ext*)
fix x **fix** y **fix** z **show** (*if* ($p\ x\ y\ z = 0$) *then* ($q1\ x\ y\ z$) *else* ($q2\ x\ y\ z$)) = $(sgn2\ (p\ x\ y\ z)) * (q1\ x\ y\ z) + (sgn1\ (p\ x\ y\ z)) * (q2\ x\ y\ z)$ (**is** *?left = ?right*)
proof *cases*
assume $A1$: $p\ x\ y\ z = 0$
then have $S1$: *?left = q1 x y z* **by** *simp*
from $A1$ **have** $S2$: *?right = q1 x y z* **by** *simp*
from $S1\ S2$ **show** *?thesis* **by** *simp*
next
assume $A2$: $p\ x\ y\ z \neq 0$
then have $S3$: $p\ x\ y\ z > 0$ **by** *simp*
then show *?thesis* **by** *simp*
qed
qed
assume $p \in PrimRec3$ **and** $q1 \in PrimRec3$ **and** $q2 \in PrimRec3$
then have $(\lambda\ x\ y\ z. (sgn2\ (p\ x\ y\ z)) * (q1\ x\ y\ z) + (sgn1\ (p\ x\ y\ z)) * (q2\ x\ y\ z)) \in PrimRec3$
by *prec0*
with *if-as-pr* **show** *?thesis* **by** *simp*
qed

lemma *if-eq-is-pr3*: $\llbracket p1 \in PrimRec3; p2 \in PrimRec3; q1 \in PrimRec3; q2 \in PrimRec3 \rrbracket \implies (\lambda\ x\ y\ z. \text{if } (p1\ x\ y\ z = p2\ x\ y\ z) \text{ then } (q1\ x\ y\ z) \text{ else } (q2\ x\ y\ z)) \in PrimRec3$
proof –
have $S1$: $(\lambda\ x\ y\ z. \text{if } (p1\ x\ y\ z = p2\ x\ y\ z) \text{ then } (q1\ x\ y\ z) \text{ else } (q2\ x\ y\ z)) = (\lambda\ x\ y\ z. \text{if } (abs\text{-of}\text{-diff } (p1\ x\ y\ z)\ (p2\ x\ y\ z) = 0) \text{ then } (q1\ x\ y\ z) \text{ else } (q2\ x\ y\ z))$ (**is** *?L = ?R*) **by** (*simp add: abs-of-diff-eq*)
assume $A1$: $p1 \in PrimRec3$ **and** $A2$: $p2 \in PrimRec3$
with *abs-of-diff-is-pr* **have** $S2$: $(\lambda\ x\ y\ z. abs\text{-of}\text{-diff } (p1\ x\ y\ z)\ (p2\ x\ y\ z)) \in PrimRec3$
by *prec0*
assume $q1 \in PrimRec3$ **and** $q2 \in PrimRec3$
with $S2$ **have** $?R \in PrimRec3$ **by** (*rule if-is-pr3*)
with $S1$ **show** *?thesis* **by** *simp*
qed

ML \langle

fun *get-if-by-index* 1 = $\@ \{thm\ if\text{-eq}\text{-is}\text{-pr}\}$
| *get-if-by-index* 2 = $\@ \{thm\ if\text{-eq}\text{-is}\text{-pr}2\}$
| *get-if-by-index* 3 = $\@ \{thm\ if\text{-eq}\text{-is}\text{-pr}3\}$
| *get-if-by-index* - = *raise BadArgument*

fun *if-comp-tac* *ctxt* = *SUBGOAL* (*fn* (t, i) =>
let

```

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

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

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

```

2.2 Bounded least operator

definition

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

definition

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

lemma $b\text{-least-aux1}$: $b\text{-least } f x = x \vee (b\text{-least } f x < x \wedge (f x (b\text{-least } f x)) \neq 0)$

proof –

let $?P = \%y. y = x \vee (y < x \wedge (f x y) \neq 0)$
 have $?P x$ by simp

then have $?P$ (*Least* $?P$) **by** (*rule LeastI*)
thus $?thesis$ **by** (*simp add: b-least-def*)
qed

lemma *b-least-le-arg*: $b\text{-least } f \ x \leq x$

proof –

have $b\text{-least } f \ x = x \vee (b\text{-least } f \ x < x \wedge (f \ x \ (b\text{-least } f \ x)) \neq 0)$ **by** (*rule b-least-aux1*)

from *this* **show** $?thesis$ **by** (*arith*)

qed

lemma *less-b-least-impl-zero*: $y < b\text{-least } f \ x \implies f \ x \ y = 0$

proof –

assume $A1$: $y < b\text{-least } f \ x$ (**is** $- < ?b$)

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

with $A1$ **have** $S1$: $y < x$ **by** *simp*

with $A1$ **have** $y < (\text{Least } (\%y. y = x \vee (y < x \wedge (f \ x \ y) \neq 0)))$ **by** (*simp add: b-least-def*)

then have $\neg (y = x \vee (y < x \wedge (f \ x \ y) \neq 0))$ **by** (*rule not-less-Least*)

with $S1$ **show** $?thesis$ **by** *simp*

qed

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

proof (*rule ccontr*)

assume $A1$: $f \ x \ y \neq 0$

assume $\neg b\text{-least } f \ x \leq y$

then have $y < b\text{-least } f \ x$ **by** *simp*

with $A1$ **show** *False* **by** (*simp add: less-b-least-impl-zero*)

qed

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

proof –

assume $A1$: $b\text{-least } f \ x < x$

have $b\text{-least } f \ x = x \vee (b\text{-least } f \ x < x \wedge (f \ x \ (b\text{-least } f \ x)) \neq 0)$ **by** (*rule b-least-aux1*)

from $A1$ *this* **show** $?thesis$ **by** *simp*

qed

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

proof –

assume $A1$: $b\text{-least } f \ x < x$ (**is** $?b < -$)

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

from $A1$ **have** $S1$: $f \ x \ ?b \neq 0$ **by** (*rule b-least-less-impl-nz*)

from $S1$ **have** $S2$: $?B \leq ?b$ **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** $?thesis$ **by** *simp*

qed

lemma *less-b-least-impl-zero2*: $\llbracket y < x; b\text{-least } f x = x \rrbracket \implies f x y = 0$ **by** (*simp add: less-b-least-impl-zero*)

lemma *nz-impl-b-least-less*: $\llbracket y < x; (f x y) \neq 0 \rrbracket \implies (b\text{-least } f x) < x$

proof –

assume *A1*: $y < x$

assume $f x y \neq 0$

then have $b\text{-least } f x \leq y$ **by** (*rule nz-impl-b-least-le*)

with *A1* **show** *?thesis* **by** *simp*

qed

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

proof –

assume *A1*: $y < x$ **and** *A2*: $f x y \neq 0$

from *A1 A2* **have** $b\text{-least } f x < x$ **by** (*rule nz-impl-b-least-less*)

thus *?thesis* **by** (*rule b-least-less-impl-eq*)

qed

lemma *b-least2-aux1*: $b\text{-least2 } f x y = y \vee (b\text{-least2 } f x y < y \wedge (f x (b\text{-least2 } f x y)) \neq 0)$

proof –

let $?P = \%z. z = y \vee (z < y \wedge (f x z) \neq 0)$

have $?P y$ **by** *simp*

then have $?P (\text{Least } ?P)$ **by** (*rule LeastI*)

thus *?thesis* **by** (*simp add: b-least2-def*)

qed

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

proof –

let $?B = b\text{-least2 } f x y$

have $?B = y \vee (?B < y \wedge (f x ?B) \neq 0)$ **by** (*rule b-least2-aux1*)

from *this* **show** *?thesis* **by** (*arith*)

qed

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

proof –

assume *A1*: $z < b\text{-least2 } f x y$ (**is** $- < ?b$)

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

with *A1* **have** $S1: z < y$ **by** *simp*

with *A1* **have** $z < (\text{Least } (\%z. z = y \vee (z < y \wedge (f x z) \neq 0)))$ **by** (*simp add: b-least2-def*)

then have $\neg (z = y \vee (z < y \wedge (f x z) \neq 0))$ **by** (*rule not-less-Least*)

with *S1* **show** *?thesis* **by** *simp*

qed

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

proof –

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

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

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

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

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

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

qed

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

proof –

let $?B = b\text{-least2 } f x y$

assume $A1: ?B < y$

from $A1$ have $S1: f x ?B \neq 0$ **by** (rule *b-least2-less-impl-nz*)

from $S1$ have $S2: b\text{-least2 } f x (\text{Suc } y) \leq ?B$ **by** (simp add: *nz-impl-b-least2-le*)

from $A1 S2$ have $S3: b\text{-least2 } f x (\text{Suc } y) < \text{Suc } y$ **by** (simp)

from $S3$ have $S4: f x (b\text{-least2 } f x (\text{Suc } y)) \neq 0$ **by** (rule *b-least2-less-impl-nz*)

from $S4$ have $S5: ?B \leq b\text{-least2 } f x (\text{Suc } y)$ **by** (rule *nz-impl-b-least2-le*)

from $S2 S5$ show *?thesis* **by** simp

qed

lemma *b-least2-aux3*: $\llbracket b\text{-least2 } f x y = y; f x y \neq 0 \rrbracket \implies b\text{-least2 } f x (\text{Suc } y) = y$

proof –

assume $A1: b\text{-least2 } f x y = y$

assume $A2: f x y \neq 0$

from $A2$ have $S1: b\text{-least2 } f x (\text{Suc } y) \leq y$ **by** (rule *nz-impl-b-least2-le*)

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

proof –

assume $A2\text{-1}: b\text{-least2 } f x (\text{Suc } y) < y$ (**is** $?z < -$)

from $A2\text{-1}$ have $S2\text{-1}: ?z < \text{Suc } y$ **by** simp

from $S2\text{-1}$ have $S2\text{-2}: f x ?z \neq 0$ **by** (rule *b-least2-less-impl-nz*)

from $A2\text{-1 } S2\text{-2}$ have $S2\text{-3}: b\text{-least2 } f x y < y$ **by** (rule *nz-b-least2-impl-less*)

from $S2\text{-3 } A1$ show *?thesis* **by** simp

qed

from $S2$ have $S3: \neg (b\text{-least2 } f x (\text{Suc } y) < y)$ **by** auto

from $S1 S3$ show *?thesis* **by** simp

qed

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

proof (rule *ccontr*)

assume $A1: y1 \leq y2$

let $?b1 = b\text{-least2 } f x y1$ and $?b2 = b\text{-least2 } f x y2$

assume $\neg ?b1 \leq ?b2$ then have $A2: ?b2 < ?b1$ **by** simp

have $S1: ?b1 \leq y1$ **by** (rule *b-least2-le-arg*)

have $S2: ?b2 \leq y2$ **by** (rule *b-least2-le-arg*)

from $A1 A2 S1 S2$ have $S3: ?b2 < y2$ **by** simp

then have $S4: f x ?b2 \neq 0$ **by** (rule *b-least2-less-impl-nz*)

from $A2$ have $S5: f x ?b2 = 0$ **by** (rule *less-b-least2-impl-zero*)

from $S4 S5$ show *False* **by** simp

qed

lemma *b-least2-aux4*: $\llbracket b\text{-least2 } f x y = y; f x y = 0 \rrbracket \implies b\text{-least2 } f x (\text{Suc } y) = \text{Suc } y$

proof –

assume $A1: b\text{-least2 } f x y = y$

assume $A2: f x y = 0$

```

have S1: b-least2 f x (Suc y) ≤ Suc y by (rule b-least2-le-arg)
have S2: y ≤ b-least2 f x (Suc y)
proof -
  have y ≤ Suc y by simp
  then have b-least2 f x y ≤ b-least2 f x (Suc y) by (rule b-least2-mono)
  with A1 show ?thesis by simp
qed
from S1 S2 have b-least2 f x (Suc y) = y ∨ b-least2 f x (Suc y) = Suc y by
arith
moreover
{
  assume A3: b-least2 f x (Suc y) = y
  have f x y ≠ 0
  proof -
    have y < Suc y by simp
    with A3 have b-least2 f x (Suc y) < Suc y by simp
    from this have f x (b-least2 f x (Suc y)) ≠ 0 by (simp add: b-least2-less-impl-nz)
    with A3 show f x y ≠ 0 by simp
  qed
  with A2 have ?thesis by simp
}
moreover
{
  assume b-least2 f x (Suc y) = Suc y
  then have ?thesis by simp
}
ultimately show ?thesis by blast
qed

```

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

```

proof -
  have S1: b-least2 f x 0 ≤ 0 by (rule b-least2-le-arg)
  from S1 show ?thesis by auto
qed

```

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

```

proof -
  define loc-Op1 where loc-Op1 = (λ (f::nat ⇒ nat ⇒ nat) x y z. (sgn1 (z -
y)) * y + (sgn2 (z - y)) * ((sgn1 (f x z)) * z + (sgn2 (f x z)) * (Suc z)))
  define loc-Op2 where loc-Op2 = (λ f. PrimRecOp-last (λ x. 0) (loc-Op1 f))
  have loc-op2-lm-1: ∧ f x y. loc-Op2 f x y < y ⟹ loc-Op2 f x (Suc y) = loc-Op2
f x y
  proof -
    fix f x y
    let ?b = loc-Op2 f x y
    have S1: loc-Op2 f x (Suc y) = (loc-Op1 f) x ?b y by (simp add: loc-Op2-def)
    assume ?b < y
    then have y - ?b > 0 by simp
    then have loc-Op1 f x ?b y = ?b by (simp add: loc-Op1-def)
  qed

```

with $S1$ **show** $loc\text{-}Op2\ f\ x\ y < y \implies loc\text{-}Op2\ f\ x\ (Suc\ y) = loc\text{-}Op2\ f\ x\ y$ **by**
simp
qed
have $loc\text{-}op2\text{-}lm\text{-}2: \bigwedge f\ x\ y. [\neg(loc\text{-}Op2\ f\ x\ y < y); f\ x\ y \neq 0] \implies loc\text{-}Op2\ f\ x\ (Suc\ y) = y$
proof $-$
fix $f\ x\ y$
let $?b = loc\text{-}Op2\ f\ x\ y$ **and** $?h = loc\text{-}Op1\ f$
have $S1: loc\text{-}Op2\ f\ x\ (Suc\ y) = ?h\ x\ ?b\ y$ **by** (*simp add: loc-Op2-def*)
assume $\neg(?b < y)$
then have $S2: y - ?b = 0$ **by** *simp*
assume $f\ x\ y \neq 0$
with $S2$ **have** $?h\ x\ ?b\ y = y$ **by** (*simp add: loc-Op1-def*)
with $S1$ **show** $loc\text{-}Op2\ f\ x\ (Suc\ y) = y$ **by** *simp*
qed
have $loc\text{-}op2\text{-}lm\text{-}3: \bigwedge f\ x\ y. [\neg(loc\text{-}Op2\ f\ x\ y < y); f\ x\ y = 0] \implies loc\text{-}Op2\ f\ x\ (Suc\ y) = Suc\ y$
proof $-$
fix $f\ x\ y$
let $?b = loc\text{-}Op2\ f\ x\ y$ **and** $?h = loc\text{-}Op1\ f$
have $S1: loc\text{-}Op2\ f\ x\ (Suc\ y) = ?h\ x\ ?b\ y$ **by** (*simp add: loc-Op2-def*)
assume $\neg(?b < y)$
then have $S2: y - ?b = 0$ **by** *simp*
assume $f\ x\ y = 0$
with $S2$ **have** $?h\ x\ ?b\ y = Suc\ y$ **by** (*simp add: loc-Op1-def*)
with $S1$ **show** $loc\text{-}Op2\ f\ x\ (Suc\ y) = Suc\ y$ **by** *simp*
qed
have $Op2\text{-}eq\text{-}b\text{-}least2\text{-}at\text{-}point: \bigwedge f\ x\ y. loc\text{-}Op2\ f\ x\ y = b\text{-}least2\ f\ x\ y$
proof $-$ **fix** $f\ x$ **show** $\bigwedge y. loc\text{-}Op2\ f\ x\ y = b\text{-}least2\ f\ x\ y$
proof (*induct-tac y*)
show $loc\text{-}Op2\ f\ x\ 0 = b\text{-}least2\ f\ x\ 0$ **by** (*simp add: loc-Op2-def b-least2-at-zero*)
next
fix y
assume $A1: loc\text{-}Op2\ f\ x\ y = b\text{-}least2\ f\ x\ y$
then show $loc\text{-}Op2\ f\ x\ (Suc\ y) = b\text{-}least2\ f\ x\ (Suc\ y)$
proof *cases*
assume $A2: loc\text{-}Op2\ f\ x\ y < y$
then have $S1: loc\text{-}Op2\ f\ x\ (Suc\ y) = loc\text{-}Op2\ f\ x\ y$ **by** (*rule loc-op2-lm-1*)
from $A1\ A2$ **have** $b\text{-}least2\ f\ x\ y < y$ **by** *simp*
then have $S2: b\text{-}least2\ f\ x\ (Suc\ y) = b\text{-}least2\ f\ x\ y$ **by** (*rule b-least2-aux2*)
from $A1\ S1\ S2$ **show** $?thesis$ **by** *simp*
next
assume $A3: \neg loc\text{-}Op2\ f\ x\ y < y$
have $A3': b\text{-}least2\ f\ x\ y = y$
proof $-$
have $b\text{-}least2\ f\ x\ y \leq y$ **by** (*rule b-least2-le-arg*)
from $A1\ A3$ **this show** $?thesis$ **by** *simp*
qed
then show $?thesis$

proof *cases*
 assume $A_4: f\ x\ y \neq 0$
 with A_3 have $S_3: \text{loc-Op2}\ f\ x\ (\text{Suc}\ y) = y$ **by** (*rule loc-op2-lm-2*)
 from $A_3' A_4$ have $S_4: \text{b-least2}\ f\ x\ (\text{Suc}\ y) = y$ **by** (*rule b-least2-aux3*)
 from $S_3 S_4$ show *?thesis* **by** *simp*
next
 assume $\neg f\ x\ y \neq 0$
 then have $A_5: f\ x\ y = 0$ **by** *simp*
 with A_3 have $S_5: \text{loc-Op2}\ f\ x\ (\text{Suc}\ y) = \text{Suc}\ y$ **by** (*rule loc-op2-lm-3*)
 from $A_3' A_5$ have $S_6: \text{b-least2}\ f\ x\ (\text{Suc}\ y) = \text{Suc}\ y$ **by** (*rule b-least2-aux4*)
 from $S_5 S_6$ show *?thesis* **by** *simp*
qed
qed
qed
qed
 have *Op2-eq-b-least2*: $\text{loc-Op2} = \text{b-least2}$ **by** (*simp add: Op2-eq-b-least2-at-point ext*)
 assume $A_1: f \in \text{PrimRec2}$
 have *pr-loc-Op2*: $\text{loc-Op2}\ f \in \text{PrimRec2}$
proof –
 from A_1 have $S_1: \text{loc-Op1}\ f \in \text{PrimRec3}$ **by** (*simp add: loc-Op1-def, prec*)
 from *pr-zero* S_1 have $S_2: \text{PrimRecOp-last}\ (\lambda\ x.\ 0)\ (\text{loc-Op1}\ f) \in \text{PrimRec2}$
by (*rule pr-rec-last*)
 from *this* show *?thesis* **by** (*simp add: loc-Op2-def*)
qed
 from *Op2-eq-b-least2 this* show $\text{b-least2}\ f \in \text{PrimRec2}$ **by** *simp*
qed

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

theorem *pr-b-least*: $f \in \text{PrimRec2} \implies \text{b-least}\ f \in \text{PrimRec1}$

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

2.3 Examples

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

proof (*rule ext*)

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

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 ext, rule ext*)

```

fix x y show (sgn1 (sf(y+1) - x) ≠ 0) = (x < sf(y+1))
proof -
have (sgn1 (sf(y+1) - x) ≠ 0) = (sf(y+1) - x > 0) by (rule sgn1-nz-eq-arg-pos)
  thus (sgn1 (sf(y+1) - x) ≠ 0) = (x < sf(y+1)) by auto
qed
qed
let ?f = λ u z. (sgn1 (sf(z+1) - u))
have S1: ?f u u ≠ 0
proof -
  have S1-1: u+1 ≤ sf(u+1) by (rule arg-le-sf)
  have S1-2: u < u+1 by simp
  from S1-1 S1-2 have S1-3: u < sf(u+1) by simp
  from S1-3 have S1-4: sf(u+1) - u > 0 by simp
  from S1-4 have S1-5: sgn1 (sf(u+1)-u) = 1 by simp
  from S1-5 show ?thesis by simp
qed
have S3: u < Suc u by simp
from S3 S1 have S4: b-least2 ?f u (Suc u) = (Least (%z. (?f u z) ≠ 0)) by
(rule b-least2-less-impl-eq2)
let ?P = λ u z. ?f u z ≠ 0
let ?Q = λ u z. u < sf(z+1)
from lm-1 have S6: ?P = ?Q by simp
from S6 have S7: (%z. ?P u z) = (%z. ?Q u z) by (rule fun-cong)
from S7 have S8: (Least (%z. ?P u z)) = (Least (%z. ?Q u z)) by auto
from S4 S8 have S9: b-least2 ?f u (Suc u) = (Least (%z. u < sf(z+1))) by
(rule trans)
thus ?thesis by (simp add: c-sum-def)
qed
qed

```

theorem c-sum-is-pr: c-sum ∈ PrimRec1

```

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

```

theorem c-fst-is-pr [prec]: c-fst ∈ PrimRec1

```

proof -
have S1: (λ u. c-fst u) = (λ u. (u - sf (c-sum u))) by (simp add: c-fst-def ext)
from c-sum-is-pr have (λ u. (u - sf (c-sum u))) ∈ PrimRec1 by prec
with S1 show ?thesis by simp
qed

```

theorem c-snd-is-pr [prec]: c-snd ∈ PrimRec1

proof –

have $S1: c\text{-snd} = (\lambda u. (c\text{-sum } u) - (c\text{-fst } u))$ **by** (*simp add: c-snd-def ext*)
from *c-sum-is-pr c-fst-is-pr* **have** $S2: (\lambda u. (c\text{-sum } u) - (c\text{-fst } u)) \in \text{PrimRec1}$
by *prec*
from $S1$ **this show** *?thesis* **by** *simp*
qed

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

lemma *pr-conv-3-to-1-lm*: $f \in \text{PrimRec3} \implies \text{pr-conv-3-to-1 } f \in \text{PrimRec1}$ **by** (*simp add: pr-conv-3-to-1-def, 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** (*rule pr-conv-3-to-1-lm*)

thus *?thesis* **by** (*simp add: pr-conv-3-to-2-def pr-conv-1-to-2-lm*)

qed

lemma *pr-conv-2-to-3-lm*: $f \in \text{PrimRec2} \implies \text{pr-conv-2-to-3 } f \in \text{PrimRec3}$

proof –

assume $f \in \text{PrimRec2}$

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

theorem *b-least2-scheme*: $\llbracket f \in \text{PrimRec2}; g \in \text{PrimRec1}; \forall x. h\ x < g\ x; \forall x. f\ x\ (h\ x) \neq 0; \forall z\ x. z < h\ x \longrightarrow f\ x\ z = 0 \rrbracket \Longrightarrow$
 $h \in \text{PrimRec1}$

proof –

assume *f-is-pr*: $f \in \text{PrimRec2}$
assume *g-is-pr*: $g \in \text{PrimRec1}$
assume *h-lt-g*: $\forall x. h\ x < g\ x$
assume *f-at-h-nz*: $\forall x. f\ x\ (h\ x) \neq 0$
assume *h-is-min*: $\forall z\ x. z < h\ x \longrightarrow f\ x\ z = 0$
have *h-def*: $h = (\lambda x. b\text{-least2}\ f\ x\ (g\ x))$

proof

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

proof –

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

from *h-lt-g* **have** $h\ x < g\ x$ **by** *auto*

with $S1$ **have** $b\text{-least2}\ f\ x\ (g\ x) < g\ x$ **by** *simp*

then have $S2: f\ x\ (b\text{-least2}\ f\ x\ (g\ x)) \neq 0$ **by** (*rule b-least2-less-impl-nz*)

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

proof (*rule ccontr*)

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

auto

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

with $S2$ **show** *False* **by** *auto*

qed

from $S1\ S3$ **show** *?thesis* **by** *auto*

qed

qed

define $f1$ **where** $f1 = b\text{-least2}\ f$

from *f-is-pr* *f1-def* **have** *f1-is-pr*: $f1 \in \text{PrimRec2}$ **by** (*simp add: pr-b-least2*)

with *g-is-pr* **have** $(\lambda x. f1\ x\ (g\ x)) \in \text{PrimRec1}$ **by** *prec*

with *h-def* *f1-def* **show** $h \in \text{PrimRec1}$ **by** *auto*

qed

theorem *b-least2-scheme2*: $\llbracket f \in \text{PrimRec3}; g \in \text{PrimRec2}; \forall x\ y. h\ x\ y < g\ x\ y;$
 $\forall x\ y. f\ x\ y\ (h\ x\ y) \neq 0;$
 $\forall z\ x\ y. z < h\ x\ y \longrightarrow f\ x\ y\ z = 0 \rrbracket \Longrightarrow$
 $h \in \text{PrimRec2}$

proof –

assume *f-is-pr*: $f \in \text{PrimRec3}$

assume *g-is-pr*: $g \in \text{PrimRec2}$

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** $f1 = \text{pr-conv-3-to-2}\ f$

define $g1$ **where** $g1 = \text{pr-conv-2-to-1}\ g$

```

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: ∀ x. f1 x (h1 x) ≠ 0 by (simp
add: pr-conv-2-to-1-def pr-conv-3-to-2-def pr-conv-3-to-1-def pr-conv-1-to-2-def)
from h-is-min f1-def h1-def have h1-is-min: ∀ z x. z < h1 x → f1 x z = 0 by
(simp add: pr-conv-2-to-1-def pr-conv-3-to-2-def pr-conv-3-to-1-def pr-conv-1-to-2-def)
from f1-is-pr g1-is-pr h1-lt-g1 f1-at-h1-nz h1-is-min have h1-is-pr: h1 ∈ Prim-
Rec1 by (rule b-least2-scheme)
from h1-def have h = pr-conv-1-to-2 h1 by simp
with h1-is-pr show h ∈ PrimRec2 by (simp add: pr-conv-1-to-2-lm)
qed

```

theorem div-is-pr: $(\lambda a b. a \text{ div } b) \in \text{PrimRec2}$

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 * (h a b + 1) - a) by simp

```

```

    then have ?thesis = (sgn1 (b * (h 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 S4-1: a mod b < b by (rule mod-less-divisor)
    also have S4-2: b * (a div b) + a mod b = a by (rule mult-div-mod-eq)
    from S4-1 have S4-3: b * (a div b) + a mod b < b * (a div b) + b by arith
    from S4-2 S4-3 show ?thesis by auto
  qed
  from S3 S4 show ?thesis by auto
  qed
  have h-is-min:  $\forall z a b. z < h a b \longrightarrow f a b z = 0$ 
  proof (rule allI, rule allI, rule allI, rule impI)
    fix a b z assume A: z < h a b show f a b z = 0
    proof -
      from A h-def have S1: z < a div b by simp
      then have S2: a div b > 0 by simp
      have S3: b  $\neq$  0
      proof (rule ccontr)
        assume  $\neg b \neq 0$  then have b = 0 by auto
        then have a div 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 * (z+1)  $\leq$  b * (a div b)) = (z+1  $\leq$  a div b) by (rule
nat-mult-le-cancel1)
      with S4 have S5: b*(z+1)  $\leq$  b*(a div b) by simp
      moreover have b*(a div b)  $\leq$  a
      proof -
        have b*(a div b) + (a mod b) = a by (rule mult-div-mod-eq)
        moreover have 0  $\leq$  a mod b by auto
        ultimately show ?thesis by arith
      qed
      ultimately have S6: b*(z+1)  $\leq$  a
        by (simp add: minus-mod-eq-mult-div [symmetric])
      then have b*(z+1) - a = 0 by auto
      with S3 f-def show ?thesis by simp
    qed
  qed
  from f-is-pr g-is-pr h-lt-g f-at-h-nz h-is-min have h-is-pr: h  $\in$  PrimRec2 by (rule
b-least2-scheme2)
  with h-def [abs-def] show ?thesis by simp
  qed

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

```

also from *div-is-pr* have $(\lambda a b. a - (a \text{ div } b) * b) \in \text{PrimRec2}$ by *prec*
ultimately show *?thesis* by *auto*
qed

theorem *pr-rec-last-scheme*: $\llbracket g \in \text{PrimRec1}; h \in \text{PrimRec3}; \forall x. f x 0 = g x; \forall x y. f x (\text{Suc } y) = h x (f x y) y \rrbracket \implies f \in \text{PrimRec2}$

proof –

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

theorem *power-is-pr*: $(\lambda (x::nat) (n::nat). x \wedge n) \in \text{PrimRec2}$

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*: $g \in \text{PrimRec1}$ unfolding *g-def* by *prec*
have *h-is-pr*: $h \in \text{PrimRec3}$ unfolding *h-def* by *prec*
let $?f = \lambda (x::nat) (n::nat). x \wedge n$
have *f-at-0*: $\forall x. ?f x 0 = g x$
proof
fix x show $x \wedge 0 = g x$ by (*simp add: g-def*)
qed
have *f-at-Suc*: $\forall x y. ?f x (\text{Suc } y) = h x (?f x y) y$
proof (*rule allI, rule allI*)
fix $x y$ show $?f x (\text{Suc } y) = h x (?f x y) y$ 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

$c\text{-len} :: nat \Rightarrow nat$ where

$c-len = (\lambda (u::nat). (sgn1\ u) * (c-fst(u-(1::nat))+1))$

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

lemma *c-len-is-pr*: $c-len \in PrimRec1$ **unfolding** *c-len-def* **by** *prec*

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

lemma *c-len-2*: $u \neq 0 \Rightarrow c-len\ u = c-fst(u-(1::nat))+1$ **by** (*simp* *add*: *c-len-def*)

lemma *c-len-3*: $u > 0 \Rightarrow c-len\ u > 0$ **by** (*simp* *add*: *c-len-2*)

lemma *c-len-4*: $c-len\ u = 0 \Rightarrow u = 0$

proof *cases*

assume *A1*: $u = 0$

thus *?thesis* **by** *simp*

next

assume *A1*: $c-len\ u = 0$ **and** *A2*: $u \neq 0$

from *A2* **have** $c-len\ u > 0$ **by** (*simp* *add*: *c-len-3*)

from *A1* **this** **show** $u=0$ **by** *simp*

qed

lemma *c-len-5*: $c-len\ u > 0 \Rightarrow u > 0$

proof *cases*

assume *A1*: $c-len\ u > 0$ **and** *A2*: $u=0$

from *A2* **have** $c-len\ u = 0$ **by** *simp*

from *A1* **this** **show** *?thesis* **by** *simp*

next

assume *A1*: $u \neq 0$

from *A1* **show** $u > 0$ **by** *simp*

qed

fun *c-fold* :: $nat\ list \Rightarrow nat$ **where**

c-fold [] = 0

 | *c-fold* [x] = x

 | *c-fold* (x#ls) = *c-pair* x (*c-fold* ls)

lemma *c-fold-0*: $ls \neq [] \Rightarrow c-fold\ (x\#ls) = c-pair\ x\ (c-fold\ ls)$

proof –

assume *A1*: $ls \neq []$

then **have** *S1*: $ls = (hd\ ls)\#(tl\ ls)$ **by** *simp*

then **have** *S2*: $x\#ls = x\#(hd\ ls)\#(tl\ ls)$ **by** *simp*

have *S3*: $c-fold\ (x\#(hd\ ls)\#(tl\ ls)) = c-pair\ x\ (c-fold\ ((hd\ ls)\#(tl\ ls)))$ **by** *simp*

from *S1* *S2* *S3* **show** *?thesis* **by** *simp*

qed

primrec

c-unfold :: $nat \Rightarrow nat \Rightarrow nat\ list$

where

$c\text{-unfold } 0 \ u = []$
 $| \ c\text{-unfold } (Suc \ k) \ u = (if \ k = 0 \ then \ [u] \ else \ ((c\text{-fst } u) \ \# \ (c\text{-unfold } k \ (c\text{-snd } u))))$

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

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

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

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

lemma *c-unfold-3*: $c\text{-unfold } (Suc \ 1) \ u = [c\text{-fst } u, \ c\text{-snd } u]$ **by** *simp*

lemma *c-unfold-4*: $k > 0 \implies c\text{-unfold } (Suc \ k) \ u = (c\text{-fst } u) \ \# \ (c\text{-unfold } k \ (c\text{-snd } u))$ **by** *simp*

lemma *c-unfold-4-1*: $k > 0 \implies c\text{-unfold } (Suc \ k) \ u \neq []$ **by** (*simp add: c-unfold-4*)

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

lemma *c-unfold-5*: $c\text{-unfold } 2 \ u = [c\text{-fst } u, \ c\text{-snd } u]$ **by** (*simp add: two*)

lemma *c-unfold-6*: $k > 0 \implies c\text{-unfold } k \ u \neq []$

proof –

assume *A1*: $k > 0$

let $?k1 = k - (1::nat)$

from *A1* **have** *S1*: $k = Suc \ ?k1$ **by** *simp*

have *S2*: $?k1 = 0 \implies ?thesis$

proof –

assume *A2-1*: $?k1 = 0$

from *A1* *A2-1* **have** *S2-1*: $k = 1$ **by** *simp*

from *S2-1* **show** *?thesis* **by** (*simp add: c-unfold-1*)

qed

have *S3*: $?k1 > 0 \implies ?thesis$

proof –

assume *A3-1*: $?k1 > 0$

from *A3-1* **have** *S3-1*: $c\text{-unfold } (Suc \ ?k1) \ u \neq []$ **by** (*rule c-unfold-4-1*)

from *S1* *S3-1* **show** *?thesis* **by** *simp*

qed

from *S2* *S3* **show** *?thesis* **by** *arith*

qed

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

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

proof

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

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

proof –

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

qed

theorem *th-1*: $\forall u. c\text{-fold } (c\text{-unfold } (Suc \ k) \ u) = u$

apply *(induct k)*
apply *(simp add: c-fold-2)*
apply *(rule th-lm-3)*
apply *(assumption)*
done

theorem *th-2*: $k > 0 \implies (\forall u. c\text{-fold } (c\text{-unfold } k \ u) = u)$

proof –

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

qed

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

theorem *c-unfold-len*: $ALL \ u. length \ (c\text{-unfold } k \ u) = k$

apply *(induct k)*
apply *(simp)*
apply *(subgoal-tac n=(0::nat) \vee $n > 0$)*

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

lemma *th-3-lm-0*: $\llbracket c\text{-unfold } (\text{length } ls) (c\text{-fold } ls) = ls; ls = a \# ls1; ls1 = aa \# list \rrbracket \implies c\text{-unfold } (\text{length } (x \# ls)) (c\text{-fold } (x \# ls)) = x \# ls$

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*: $c\text{-unfold } (\text{Suc } (\text{length } ls)) (c\text{-fold } (x \# ls)) = c\text{-fst } (c\text{-fold } (x \# ls)) \# (c\text{-unfold } (\text{length } ls) (c\text{-snd } (c\text{-fold } (x \# ls))))$ **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*: $c\text{-unfold } (\text{Suc } (\text{length } ls)) (c\text{-fold } (x \# ls)) = x \# (c\text{-unfold } (\text{length } ls) (c\text{-fold } ls))$ **by** *simp*
from *A1* **have** *S10*: $x \# (c\text{-unfold } (\text{length } ls) (c\text{-fold } ls)) = x \# ls$ **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*: $\llbracket c\text{-unfold } (\text{length } ls) (c\text{-fold } ls) = ls; ls = a \# ls1 \rrbracket \implies c\text{-unfold } (\text{length } (x \# ls)) (c\text{-fold } (x \# ls)) = x \# ls$

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* \Rightarrow *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* \Rightarrow *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 \implies *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 \implies ?*thesis* **by** (*simp add: list-to-nat-def nat-to-list-def*)

have *S2*: *u*>0 \implies ?*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* \neq [] **by** *simp*

from *S2-5* **have** *S2-6*: *list-to-nat* *ls* = *c-pair* ((*length* *ls*)-(1::*nat*)) (*c-fold* *ls*)+1
by (*simp add: list-to-nat-def*)

have *S2-7*: *c-fold* *ls* = *c-snd*(*u*-(1::*nat*))

proof -

from *S2-1* **have** *S2-7-1*: *c-fold* *ls* = *c-fold* (*c-unfold* (*c-len* *u*) (*c-snd* (*u*-(1::*nat*))))

by *simp*

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

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

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

theorem *c-len-th-1: c-len (list-to-nat ls) = length ls*

proof (*cases*)
assume *ls = []*

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

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

proof –

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

definition

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

definition

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

definition

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

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

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

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

proof –

assume A1: $c\text{-}len\ u = 1$
then have S1: $nat\text{-}to\text{-}list\ u = [c\text{-}snd\ (u-(1::nat))]$ 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

lemma *c-hd-aux2*: $c\text{-len } u > 1 \implies c\text{-hd } u = c\text{-fst } (c\text{-snd } (u-(1::nat)))$
proof –
 assume *A1*: $c\text{-len } u > 1$
 let *?k* = $(c\text{-len } u) - 1$
 from *A1* **have** *S1*: $c\text{-len } u = \text{Suc } ?k$ **by** *simp*
 from *A1* **have** *S2*: $c\text{-len } u > 0$ **by** *simp*
 from *S2* **have** *S3*: $u > 0$ **by** (rule *c-len-5*)
 from *S3* **have** *S4*: $c\text{-hd } u = \text{hd } (\text{nat-to-list } u)$ **by** (*simp add: c-hd-def*)
 from *S3* **have** *S5*: $\text{nat-to-list } u = c\text{-unfold } (c\text{-len } u) (c\text{-snd } (u-(1::nat)))$ **by**
 (*rule nat-to-list-of-pos*)
 from *S1 S5* **have** *S6*: $\text{nat-to-list } u = c\text{-unfold } (\text{Suc } ?k) (c\text{-snd } (u-(1::nat)))$ **by**
 simp
 from *A1* **have** *S7*: $?k > 0$ **by** *simp*
 from *S7* **have** *S8*: $c\text{-unfold } (\text{Suc } ?k) (c\text{-snd } (u-(1::nat))) = (c\text{-fst } (c\text{-snd } (u-(1::nat))))$
 $\# (c\text{-unfold } ?k (c\text{-snd } (c\text{-snd } (u-(1::nat)))))$ **by** (rule *c-unfold-4*)
 from *S6 S8* **have** *S9*: $\text{nat-to-list } u = (c\text{-fst } (c\text{-snd } (u-(1::nat)))) \# (c\text{-unfold } ?k$
 $(c\text{-snd } (c\text{-snd } (u-(1::nat)))))$ **by** *simp*
 from *S9* **have** *S10*: $\text{hd } (\text{nat-to-list } u) = c\text{-fst } (c\text{-snd } (u-(1::nat)))$ **by** *simp*
 from *S4 S10* **show** *?thesis* **by** *simp*
qed

lemma *c-hd-aux3*: $u > 0 \implies c\text{-hd } u = (\text{if } (c\text{-len } u) = 1 \text{ then } c\text{-snd } (u-(1::nat))$
else $c\text{-fst } (c\text{-snd } (u-(1::nat)))$)
proof –
 assume *A1*: $u > 0$
 from *A1* **have** $c\text{-len } u > 0$ **by** (rule *c-len-3*)
 then **have** *S1*: $c\text{-len } u = 1 \vee c\text{-len } u > 1$ **by** *arith*
 let *?tmp* = *if* $(c\text{-len } u) = 1$ *then* $c\text{-snd } (u-(1::nat))$ *else* $c\text{-fst } (c\text{-snd } (u-(1::nat)))$
 have *S2*: $c\text{-len } u = 1 \implies ?thesis$
 proof –
 assume *A2-1*: $c\text{-len } u = 1$
 then **have** *S2-1*: $c\text{-hd } u = c\text{-snd } (u-(1::nat))$ **by** (rule *c-hd-aux1*)
 from *A2-1* **have** *S2-2*: $?tmp = c\text{-snd } (u-(1::nat))$ **by** *simp*
 from *S2-1* **this** **show** *?thesis* **by** *simp*
 qed
 have *S3*: $c\text{-len } u > 1 \implies ?thesis$
 proof –
 assume *A3-1*: $c\text{-len } u > 1$
 from *A3-1* **have** *S3-1*: $c\text{-hd } u = c\text{-fst } (c\text{-snd } (u-(1::nat)))$ **by** (rule *c-hd-aux2*)
 from *A3-1* **have** *S3-2*: $?tmp = c\text{-fst } (c\text{-snd } (u-(1::nat)))$ **by** *simp*
 from *S3-1* **this** **show** *?thesis* **by** *simp*
 qed
 from *S1 S2 S3* **show** *?thesis* **by** *auto*
qed

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

assume $u=0$ **then show** $?thesis$ **by** *simp*
next
assume $u \neq 0$ **then have** $A1: u > 0$ **by** *simp*
then show $?thesis$ **by** (*simp add: c-hd-aux3*)
qed

lemma *c-hd-is-pr*: $c-hd \in PrimRec1$

proof –

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

moreover have $?R \in PrimRec1$

proof (*rule if-is-pr*)

show $(\lambda x. x) \in PrimRec1$ **by** (*rule pr-id1-1*)

next show $(\lambda x. 0) \in PrimRec1$ **by** (*rule pr-zero*)

next show $(\lambda x. if\ c-len\ x = 1\ then\ c-snd\ (x - 1)\ else\ c-fst\ (c-snd\ (x - 1)))$

$\in PrimRec1$

proof (*rule if-eq-is-pr*)

show $c-len \in PrimRec1$ **by** (*rule c-len-is-pr*)

next show $(\lambda x. 1) \in PrimRec1$ **by** (*rule const-is-pr*)

next show $(\lambda x. c-snd\ (x - 1)) \in PrimRec1$ **by** *prec*

next show $(\lambda x. c-fst\ (c-snd\ (x - 1))) \in PrimRec1$ **by** *prec*

qed

qed

ultimately show $?thesis$ **by** *simp*

qed

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

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

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

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

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

proof –

assume $A1: c-len\ u > 1$

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

from $A1$ **have** $S1: c-len\ u = Suc\ ?k$ **by** *simp*

from $A1$ **have** $S2: c-len\ u > 0$ **by** *simp*

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

from $S3$ **have** $S4: nat-to-list\ u = c-unfold\ (c-len\ u)\ (c-snd\ (u-(1::nat)))$ **by** (*rule nat-to-list-of-pos*)

from $A1$ **have** $S5: ?k > 0$ **by** *simp*

from $S5$ **have** $S6: c-unfold\ (Suc\ ?k)\ (c-snd\ (u-(1::nat))) = (c-fst\ (c-snd\ (u-(1::nat))))$

$\# (c-unfold\ ?k\ (c-snd\ (c-snd\ (u-(1::nat))))$ **by** (*rule c-unfold-4*)

from $S6$ **have** $S7: tl\ (c-unfold\ (Suc\ ?k)\ (c-snd\ (u-(1::nat)))) = c-unfold\ ?k$

(c-snd (c-snd (u-(1::nat)))) **by simp**
from *S2 S4 S7* **have** *S8*: $tl (nat\text{-to-list } u) = c\text{-unfold } ?k (c\text{-snd } (c\text{-snd } (u-(1::nat))))$
by simp
define *ls* **where** $ls = tl (nat\text{-to-list } u)$
from *ls-def S8* **have** *S9*: $length\ ls = ?k$ **by** (*simp add: c-unfold-len*)
from *ls-def* **have** *S10*: $c\text{-tl } u = list\text{-to-nat } ls$ **by** (*simp add: c-tl-def*)
from *S5 S9* **have** *S11*: $length\ ls > 0$ **by simp**
from *S11* **have** *S12*: $ls \neq []$ **by simp**
from *S12* **have** *S13*: $list\text{-to-nat } ls = (c\text{-pair } ((length\ ls) - 1) (c\text{-fold } ls)) + 1$ **by**
(*simp add: list-to-nat-def*)
from *S10 S13* **have** *S14*: $c\text{-tl } u = (c\text{-pair } ((length\ ls) - 1) (c\text{-fold } ls)) + 1$ **by**
simp
from *S9* **have** *S15*: $(length\ ls) - (1::nat) = ?k - (1::nat)$ **by simp**
from *A1* **have** *S16*: $?k - (1::nat) = c\text{-len } u - (2::nat)$ **by arith**
from *S15 S16* **have** *S17*: $(length\ ls) - (1::nat) = c\text{-len } u - (2::nat)$ **by simp**
from *ls-def S8* **have** *S18*: $ls = c\text{-unfold } ?k (c\text{-snd } (c\text{-snd } (u-(1::nat))))$ **by simp**
from *S5* **have** *S19*: $c\text{-fold } (c\text{-unfold } ?k (c\text{-snd } (c\text{-snd } (u-(1::nat)))))) = c\text{-snd}$
($c\text{-snd } (u-(1::nat))$) **by** (*simp add: th-2*)
from *S18 S19* **have** *S20*: $c\text{-fold } ls = c\text{-snd } (c\text{-snd } (u-(1::nat)))$ **by simp**
from *S14 S17 S20* **show** *?thesis* **by simp**
qed

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

proof -

have *S1*: $u = 0 \implies ?thesis$ **by simp**

have *S2*: $u > 0 \implies ?thesis$

proof -

assume *A1*: $u > 0$

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

have *S2-2*: $c\text{-len } u \neq 1 \implies ?thesis$

proof -

assume *A2-2-1*: $c\text{-len } u \neq 1$

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

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

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

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

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

from *S2-2-2* **have** *S2-2-6*: $c\text{-tl } u = (c\text{-pair } (c\text{-len } u - (2::nat)) (c\text{-snd } (c\text{-snd}$
($u-(1::nat)))) + 1$ **by** (*rule c-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*: $u > 0 \implies c\text{-tl } u < u$

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

lemma $c\text{-tl-le}: c\text{-tl } u \leq u$
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 $S2: c\text{-tl } u < u$ **by** (rule $c\text{-tl-less}$)
with $A1$ **show** $c\text{-tl } u \leq u$ **by** simp
qed

theorem $c\text{-tl-is-pr}: c\text{-tl} \in \text{PrimRec1}$
proof –
have $c\text{-tl} = (\lambda u. (\text{sgn1 } ((c\text{-len } u) - 1)) * ((c\text{-pair } (c\text{-len } u - (2::nat)) (c\text{-snd } (c\text{-snd } (u-(1::nat)))))) + 1))$ (is $= ?R$) **by** (simp add: $c\text{-tl-aux3 ext}$)
moreover from $c\text{-len-is-pr } c\text{-pair-is-pr}$ **have** $?R \in \text{PrimRec1}$ **by** prec
ultimately show $?thesis$ **by** simp
qed

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

apply(*unfold list-to-nat-def*)
apply(*simp*)
done

lemma *c-cons-aux2*: $u > 0 \implies c-cons\ x\ u = (c-pair\ (c-len\ u)\ (c-pair\ x\ (c-snd\ (u-(1::nat)))) + 1$

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** $ls = nat-to-list\ u$
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* **have** *S7*: $c-cons\ x\ u = list-to-nat\ (x \# ls)$ **by** (*simp add: c-cons-def*)
have *S8*: $list-to-nat\ (x \# ls) = (c-pair\ ((length\ (x\#ls))-(1::nat))\ (c-fold\ (x\#ls))) + 1$ **by** (*simp add: list-to-nat-def*)
have *S9*: $(length\ (x\#ls))-(1::nat) = length\ ls$ **by** *simp*
from *S9 S4 S8* **have** *S10*: $list-to-nat\ (x \# ls) = (c-pair\ (c-len\ u)\ (c-fold\ (x\#ls))) + 1$ **by** *simp*
have *S11*: $c-fold\ (x\#ls) = c-pair\ x\ (c-snd\ (u-(1::nat)))$
proof –
from *S6* **have** *S11-1*: $c-fold\ (x\#ls) = c-pair\ x\ (c-fold\ ls)$ **by** (*rule c-fold-0*)
from *S3* **have** *S11-2*: $c-fold\ ls = c-fold\ (c-unfold\ (c-len\ u)\ (c-snd\ (u-(1::nat))))$
by *simp*
from *S1 S11-2* **have** *S11-3*: $c-fold\ ls = c-snd\ (u-(1::nat))$ **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*: $c-cons = (\lambda\ x\ u.\ (sgn2\ u)*((c-pair\ 0\ x)+1) + (sgn1\ u)*((c-pair\ (c-len\ u)\ (c-pair\ x\ (c-snd\ (u-(1::nat)))) + 1))$

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

from $S1\ S2$ **have** $c-cons\ x\ u = ?R$ **by** *simp*
then show *?thesis* .
qed
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 $\neg u=0$ **then have** $u>0$ **by** *simp*
then show $c-cons\ x\ u > 0$ **by** (*simp add: c-cons-aux2*)
qed

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

definition
 $c-drop :: nat \Rightarrow nat \Rightarrow nat$ **where**
 $c-drop = PrimRecOp\ (\lambda\ x. x)\ (\lambda\ x\ y\ z. c-tl\ y)$

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 $c-drop-is-pr$: $c-drop \in PrimRec2$
proof –
have $(\lambda\ x. x) \in PrimRec1$ **by** (*rule pr-id1-1*)
moreover from $c-tl-is-pr$ **have** $(\lambda\ x\ y\ z. c-tl\ y) \in PrimRec3$ **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 $c-drop-df$: $\forall\ ls. drop\ n\ ls = nat-to-list\ (c-drop\ n\ (list-to-nat\ ls))$
proof (*induct n*)

show $\forall ls. \text{drop } 0 \text{ } ls = \text{nat-to-list } (c\text{-drop } 0 \text{ } (list\text{-to-nat } ls))$ **by** (*simp add: c-drop-def*)
next
fix n **assume** $A1: \forall ls. \text{drop } n \text{ } ls = \text{nat-to-list } (c\text{-drop } n \text{ } (list\text{-to-nat } ls))$
then show $\forall ls. \text{drop } (Suc \ n) \text{ } ls = \text{nat-to-list } (c\text{-drop } (Suc \ n) \text{ } (list\text{-to-nat } ls))$
proof –
{
fix $ls::\text{nat list}$
have $S1: \text{drop } (Suc \ n) \text{ } ls = \text{drop } n \text{ } (tl \text{ } ls)$ **by** (*rule drop-Suc*)
from $A1$ **have** $S2: \text{drop } n \text{ } (tl \text{ } ls) = \text{nat-to-list } (c\text{-drop } n \text{ } (list\text{-to-nat } (tl \text{ } ls)))$ **by**
simp
also have $\dots = \text{nat-to-list } (c\text{-drop } n \text{ } (c\text{-tl } (list\text{-to-nat } ls)))$ **by** (*simp add: c-tl-eq-tl*)
also have $\dots = \text{nat-to-list } (c\text{-drop } (Suc \ n) \text{ } (list\text{-to-nat } ls))$ **by** (*simp add: c-drop-at-Suc1*)
finally have $\text{drop } n \text{ } (tl \text{ } ls) = \text{nat-to-list } (c\text{-drop } (Suc \ n) \text{ } (list\text{-to-nat } ls))$ **by** *simp*
with $S1$ **have** $\text{drop } (Suc \ n) \text{ } ls = \text{nat-to-list } (c\text{-drop } (Suc \ n) \text{ } (list\text{-to-nat } ls))$ **by**
simp
}
then show *?thesis* **by** *blast*
qed
qed

definition

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

lemma *c-nth-is-pr*: $c\text{-nth} \in \text{PrimRec2}$

proof (*unfold c-nth-def*)

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

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

lemma *c-hd-c-cons [simp]*: $c\text{-hd } (c\text{-cons } x \ y) = x$

proof –

have $c\text{-cons } x \ y > 0$ **by** (*rule c-cons-pos*)

then show *?thesis* **by** (*simp add: c-hd-def c-cons-def*)

qed

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

definition

$c\text{-f-list} :: (\text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat}) \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat}$ **where**

$c\text{-f-list} = (\lambda f.$

$\text{let } g = (\%x. c\text{-cons } (f \ 0 \ x) \ 0); h = (\%a \ b \ c. c\text{-cons } (f \ (Suc \ a) \ c) \ b)$ *in PrimRecOp*
 $g \ h)$

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

Let-def)

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

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

proof –

assume *A1*: $f \in PrimRec2$

let *?g* = $(\%x. c\text{-cons } (f\ 0\ x)\ 0)$

from *A1* *c-cons-is-pr* **have** *S1*: $?g \in PrimRec1$ **by** *prec*

let *?h* = $(\%a\ b\ c. c\text{-cons } (f (Suc\ a)\ c)\ b)$

from *A1* *c-cons-is-pr* **have** *S2*: $?h \in PrimRec3$ **by** *prec*

from *S1 S2* **show** *?thesis* **by** *(simp add: pr-rec c-f-list-def Let-def)*

qed

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

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

apply *(rule ext, rule ext)*

apply *(rule c-f-list-to-f-0)*

done

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

proof –

assume *A1*: $c\text{-f-list } f \in PrimRec2$

have *S1*: $f = (\lambda\ y\ x. c\text{-hd } (c\text{-f-list } f\ y\ x))$ **by** *(rule c-f-list-to-f)*

from *A1* *c-hd-is-pr* **have** *S2*: $(\lambda\ y\ x. c\text{-hd } (c\text{-f-list } f\ y\ x)) \in PrimRec2$ **by** *prec*

with *S1* **show** *?thesis* **by** *simp*

qed

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

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

proof –

assume $z < Suc\ n$

then have $Suc\ n - z = Suc\ (n - z)$ **by** *arith*

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

also have $\dots = c\text{-nth } (c\text{-cons } (f (Suc\ n)\ x) (c\text{-f-list } f\ n\ x)) (Suc\ (n - z))$ **by** *(simp add: c-f-list-at-Suc)*

also have $\dots = c\text{-nth } (c\text{-f-list } f\ n\ x) (n - z)$ **by** *(simp add: c-f-list-lm-1)*

finally show *?thesis* **by** *simp*

qed

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

proof (*induct y*)

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

proof

assume $z \leq 0$ **then have** *A1*: $z=0$ **by** *simp*

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

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

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

also have $\dots = f \ 0 \ x$ **by** *simp*

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

qed

next

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

proof

assume *A3*: $z \leq \text{Suc } n$

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

proof cases

assume *AA1*: $z \leq n$

then have *AA2*: $z < \text{Suc } n$ **by** *simp*

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

from *AA2* **have** $c\text{-nth } (c\text{-f-list } f \ (\text{Suc } n) \ x) \ (\text{Suc } n - z) = c\text{-nth } (c\text{-f-list } f \ n \ x) \ (n - z)$ **by** (*rule c-f-list-lm-2*)

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

next

assume $\neg z \leq n$

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

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

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

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

also have $\dots = c\text{-hd } (c\text{-cons } (f \ (\text{Suc } n) \ x) \ (c\text{-f-list } f \ n \ x))$ **by** (*simp add: c-f-list-at-Suc*)

also have $\dots = f \ (\text{Suc } n) \ x$ **by** *simp*

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

qed

qed

qed

theorem *th-pr-rec*: $\llbracket g \in \text{PrimRec1}; h \in \text{PrimRec3}; (\forall x. (f \ 0 \ x) = (g \ x)); (\forall x \ y. (f \ (\text{Suc } y) \ x) = h \ y \ (f \ y \ x) \ x) \rrbracket \implies f \in \text{PrimRec2}$

proof –

assume *g-is-pr*: $g \in \text{PrimRec1}$

assume *h-is-pr*: $h \in \text{PrimRec3}$

assume *f-0*: $\forall x. f \ 0 \ x = g \ x$

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

let $?f = \text{PrimRecOp } g \ h$

```

from g-is-pr h-is-pr have S1:  $?f \in \text{PrimRec2}$  by (rule pr-rec)
have f-2:  $\forall x. ?f\ 0\ x = g\ x$  by simp
have f-3:  $\forall x\ y. (?f\ (\text{Suc}\ y)\ x) = h\ y\ (?f\ y\ x)\ x$  by simp
have S2:  $f = ?f$ 
proof -
  have  $\bigwedge x\ y. f\ y\ x = ?f\ y\ x$ 
  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:  $\llbracket g \in \text{PrimRec1}; \alpha \in \text{PrimRec2}; h \in \text{PrimRec3}; (\forall x\ y. \alpha\ y\ x \leq y); (\forall x. (f\ 0\ x) = (g\ x)); (\forall x\ y. (f\ (\text{Suc}\ y)\ x) = h\ y\ (f\ (\alpha\ y\ x)\ x)\ x) \rrbracket \implies f \in \text{PrimRec2}$ 
proof -
  assume g-is-pr:  $g \in \text{PrimRec1}$ 
  assume a-is-pr:  $\alpha \in \text{PrimRec2}$ 
  assume h-is-pr:  $h \in \text{PrimRec3}$ 
  assume a-le:  $(\forall x\ y. \alpha\ y\ x \leq y)$ 
  assume f-0:  $\forall x. f\ 0\ x = g\ x$ 
  assume f-1:  $\forall x\ y. (f\ (\text{Suc}\ y)\ x) = h\ y\ (f\ (\alpha\ y\ x)\ x)\ x$ 
  let ?g' =  $\lambda x. c\text{-cons}\ (g\ x)\ 0$ 
  let ?h' =  $\lambda a\ b\ c. c\text{-cons}\ (h\ a\ (c\text{-nth}\ b\ (a - (\alpha\ a\ c))))\ c)\ b$ 
  let ?r = c-f-list f
  from g-is-pr c-cons-is-pr have g'-is-pr:  $?g' \in \text{PrimRec1}$  by prec
  from h-is-pr c-cons-is-pr c-nth-is-pr a-is-pr have h'-is-pr:  $?h' \in \text{PrimRec3}$  by
prec
  have S1:  $\forall x. ?r\ 0\ x = ?g'\ x$ 
  proof
    fix x have  $?r\ 0\ x = c\text{-cons}\ (f\ 0\ x)\ 0$  by (rule c-f-list-at-0)
    with f-0 have  $?r\ 0\ x = c\text{-cons}\ (g\ x)\ 0$  by simp
    then show  $?r\ 0\ x = ?g'\ x$  by simp
  qed
  have S2:  $\forall x\ y. ?r\ (\text{Suc}\ y)\ x = ?h'\ y\ (?r\ y\ x)\ x$ 
  proof (rule allI, rule allI)
    fix x y show  $?r\ (\text{Suc}\ y)\ x = ?h'\ y\ (?r\ y\ x)\ x$ 
    proof -
      have S2-1:  $?r\ (\text{Suc}\ y)\ x = c\text{-cons}\ (f\ (\text{Suc}\ y)\ x)\ (?r\ y\ x)$  by (rule c-f-list-at-Suc)
      with f-1 have S2-2:  $f\ (\text{Suc}\ y)\ x = h\ y\ (f\ (\alpha\ y\ x)\ x)\ x$  by simp
      from a-le have S2-3:  $\alpha\ y\ x \leq y$  by simp
      then have S2-4:  $f\ (\alpha\ y\ x)\ x = c\text{-nth}\ (?r\ y\ x)\ (y - (\alpha\ y\ x))$  by (simp add: c-f-list-nth)
      from S2-1 S2-2 S2-4 show ?thesis by simp
    qed
  qed

```

```

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

declare  $c$ -tl-less [termination-simp]

fun  $c$ -assoc-have-key ::  $nat \Rightarrow nat \Rightarrow nat$  where
   $c$ -assoc-have-key-df [simp del]:  $c$ -assoc-have-key  $y$   $x = (if$   $y = 0$  then  $1$  else
    (if  $c$ -fst ( $c$ -hd  $y$ ) =  $x$  then  $0$  else  $c$ -assoc-have-key ( $c$ -tl  $y$ )  $x$ )

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

theorem  $c$ -assoc-have-key-is-pr:  $c$ -assoc-have-key  $\in PrimRec2$ 
proof –
  let  $?h = \lambda a b c.$  if  $c$ -fst ( $c$ -hd ( $Suc$   $a$ )) =  $c$  then  $0$  else  $b$ 
  let  $?a = \lambda y x.$   $c$ -tl ( $Suc$   $y$ )
  let  $?g = \lambda x.$  ( $1::nat$ )
  have  $g$ -is-pr:  $?g \in PrimRec1$  by (rule const-is-pr)
  from  $c$ -tl-is-pr have  $a$ -is-pr:  $?a \in PrimRec2$  by prec
  have  $h$ -is-pr:  $?h \in PrimRec3$ 
  proof (rule if-eq-is-pr3)
    from  $c$ -fst-is-pr  $c$ -hd-is-pr show ( $\lambda x y z.$   $c$ -fst ( $c$ -hd ( $Suc$   $x$ )))  $\in PrimRec3$  by
prec
  next
    show ( $\lambda x y z.$   $z$ )  $\in PrimRec3$  by (rule pr-id3-3)
  next
    show ( $\lambda x y z.$   $0$ )  $\in PrimRec3$  by prec
  next
    show ( $\lambda x y z.$   $y$ )  $\in PrimRec3$  by (rule pr-id3-2)
  qed
  have  $a$ -le:  $\forall x y.$   $?a y x \leq y$ 
  proof (rule allI, rule allI)
    fix  $x y$  show  $?a y x \leq y$ 
    proof –
      have  $Suc$   $y > 0$  by simp
      then have  $?a y x < Suc$   $y$  by (rule  $c$ -tl-less)
      then show  $?thesis$  by simp
    qed
  qed
  have  $f$ -0:  $\forall x.$   $c$ -assoc-have-key  $0$   $x = ?g$   $x$  by (simp add:  $c$ -assoc-have-key-df)
  have  $f$ -1:  $\forall x y.$   $c$ -assoc-have-key ( $Suc$   $y$ )  $x = ?h$   $y$  ( $c$ -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  $?thesis$  by (rule th-rec)
qed

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

```

lemma *c-assoc-value-lm-1*: $y \neq 0 \implies c\text{-assoc-value } y \ x = (\text{if } c\text{-fst } (c\text{-hd } y) = x \text{ then } c\text{-snd } (c\text{-hd } y) \text{ else } c\text{-assoc-value } (c\text{-tl } y) \ x)$ **by** (*simp add: c-assoc-value-df*)

theorem *c-assoc-value-is-pr*: $c\text{-assoc-value} \in \text{PrimRec2}$

proof –

let $?h = \lambda a \ b \ c. \text{if } c\text{-fst } (c\text{-hd } (\text{Suc } a)) = c \text{ then } c\text{-snd } (c\text{-hd } (\text{Suc } a)) \text{ else } b$
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 $(\lambda x \ y \ z. z) \in \text{PrimRec3}$ **by** (*rule pr-id3-3*)
next
from *c-snd-is-pr c-hd-is-pr* **show** $(\lambda x \ y \ z. c\text{-snd } (c\text{-hd } (\text{Suc } x))) \in \text{PrimRec3}$
by *prec*
next
show $(\lambda x \ y \ z. y) \in \text{PrimRec3}$ **by** (*rule pr-id3-2*)
qed
have *a-le*: $\forall x \ y. ?a \ y \ x \leq y$
proof (*rule allI, rule allI*)
fix $x \ y$ **show** $?a \ y \ x \leq y$
proof –
have $\text{Suc } y > 0$ **by** *simp*
then have $?a \ y \ x < \text{Suc } y$ **by** (*rule c-tl-less*)
then show *thesis* **by** *simp*
qed
have *f-0*: $\forall x. c\text{-assoc-value } 0 \ x = ?g \ x$ **by** (*simp add: c-assoc-value-df*)
have *f-1*: $\forall x \ y. c\text{-assoc-value } (\text{Suc } y) \ x = ?h \ y \ (c\text{-assoc-value } (?a \ y \ x) \ x)$ **by** (*simp add: c-assoc-value-df*)
from *g-is-pr a-is-pr h-is-pr a-le f-0 f-1* **show** *thesis* **by** (*rule th-rec*)
qed

lemma *c-assoc-lm-1*: $c\text{-assoc-have-key } (c\text{-cons } (c\text{-pair } x \ y) \ z) \ x = 0$

apply (*simp add: c-assoc-have-key-df*)

apply (*simp add: c-cons-pos*)

done

lemma *c-assoc-lm-2*: $c\text{-assoc-value } (c\text{-cons } (c\text{-pair } x \ y) \ z) \ x = y$

apply (*simp add: c-assoc-value-df*)

apply (*rule impI*)

apply (*insert c-cons-pos [where x=(c-pair x y) and u=z]*)

apply (*auto*)

done

lemma *c-assoc-lm-3*: $x1 \neq x \implies c\text{-assoc-have-key } (c\text{-cons } (c\text{-pair } x y) z) x1 = c\text{-assoc-have-key } z x1$

proof –

assume *A1*: $x1 \neq x$
 let *?ls* = $(c\text{-cons } (c\text{-pair } x y) z)$
 have *S1*: $?ls \neq 0$ **by** (*simp add: c-cons-pos*)
 then have *S2*: $c\text{-assoc-have-key } ?ls x1 = (\text{if } c\text{-fst } (c\text{-hd } ?ls) = x1 \text{ then } 0 \text{ else } c\text{-assoc-have-key } (c\text{-tl } ?ls) x1)$ (**is** $- = ?R$) **by** (*rule c-assoc-have-key-lm-1*)
 have *S3*: $c\text{-fst } (c\text{-hd } ?ls) = x$ **by** *simp*
 with *A1* **have** *S4*: $\neg (c\text{-fst } (c\text{-hd } ?ls) = x1)$ **by** *simp*
 from *S4* **have** *S5*: $?R = c\text{-assoc-have-key } (c\text{-tl } ?ls) x1$ **by** (*rule if-not-P*)
 from *S2 S5* **show** *?thesis* **by** *simp*

qed

lemma *c-assoc-lm-4*: $x1 \neq x \implies c\text{-assoc-value } (c\text{-cons } (c\text{-pair } x y) z) x1 = c\text{-assoc-value } z x1$

proof –

assume *A1*: $x1 \neq x$
 let *?ls* = $(c\text{-cons } (c\text{-pair } x y) z)$
 have *S1*: $?ls \neq 0$ **by** (*simp add: c-cons-pos*)
 then have *S2*: $c\text{-assoc-value } ?ls x1 = (\text{if } c\text{-fst } (c\text{-hd } ?ls) = x1 \text{ then } c\text{-snd } (c\text{-hd } ?ls) \text{ else } c\text{-assoc-value } (c\text{-tl } ?ls) x1)$ (**is** $- = ?R$) **by** (*rule c-assoc-value-lm-1*)
 have *S3*: $c\text{-fst } (c\text{-hd } ?ls) = x$ **by** *simp*
 with *A1* **have** *S4*: $\neg (c\text{-fst } (c\text{-hd } ?ls) = x1)$ **by** *simp*
 from *S4* **have** *S5*: $?R = c\text{-assoc-value } (c\text{-tl } ?ls) x1$ **by** (*rule if-not-P*)
 from *S2 S5* **show** *?thesis* **by** *simp*

qed

end

4 Primitive recursive functions of one variable

theory *PRecFun2*
imports *PRecFun*
begin

4.1 Alternative definition of primitive recursive functions of one variable

definition

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

lemma *unary-rec-into-pr*: $\llbracket g \in PrimRec1; h \in PrimRec1 \rrbracket \implies UnaryRecOp g h \in PrimRec1$ **by** (*simp add: UnaryRecOp-def pr-conv-1-to-3-lm pr-conv-2-to-1-lm pr-rec*)

definition

$c\text{-}f\text{-}pair :: (nat \Rightarrow nat) \Rightarrow (nat \Rightarrow nat) \Rightarrow (nat \Rightarrow nat)$ **where**
 $c\text{-}f\text{-}pair = (\lambda f g x. c\text{-}pair (f x) (g x))$

lemma $c\text{-}f\text{-}pair\text{-}to\text{-}pr$: $\llbracket f \in PrimRec1; g \in PrimRec1 \rrbracket \Longrightarrow c\text{-}f\text{-}pair f g \in PrimRec1$
unfolding $c\text{-}f\text{-}pair\text{-}def$ **by** $prec$

inductive-set $PrimRec1'$:: $(nat \Rightarrow nat)$ *set*
where

$zero$: $(\lambda x. 0) \in PrimRec1'$
 $| suc$: $Suc \in PrimRec1'$
 $| fst$: $c\text{-}fst \in PrimRec1'$
 $| snd$: $c\text{-}snd \in PrimRec1'$
 $| comp$: $\llbracket f \in PrimRec1'; g \in PrimRec1' \rrbracket \Longrightarrow (\lambda x. f (g x)) \in PrimRec1'$
 $| pair$: $\llbracket f \in PrimRec1'; g \in PrimRec1' \rrbracket \Longrightarrow c\text{-}f\text{-}pair f g \in PrimRec1'$
 $| un\text{-}rec$: $\llbracket f \in PrimRec1'; g \in PrimRec1' \rrbracket \Longrightarrow UnaryRecOp f g \in PrimRec1'$

lemma $primrec'\text{-}into\text{-}primrec$: $f \in PrimRec1' \Longrightarrow f \in PrimRec1$

proof ($induct f$ *rule*: $PrimRec1'.induct$)
case $zero$ **show** $?case$ **by** ($rule pr\text{-}zero$)
next
case suc **show** $?case$ **by** ($rule pr\text{-}suc$)
next
case fst **show** $?case$ **by** ($rule c\text{-}fst\text{-}is\text{-}pr$)
next
case snd **show** $?case$ **by** ($rule c\text{-}snd\text{-}is\text{-}pr$)
next
case $comp$ **from** $comp$ **show** $?case$ **by** ($simp add$: $pr\text{-}comp1\text{-}1$)
next
case $pair$ **from** $pair$ **show** $?case$ **by** ($simp add$: $c\text{-}f\text{-}pair\text{-}to\text{-}pr$)
next
case $un\text{-}rec$ **from** $un\text{-}rec$ **show** $?case$ **by** ($simp add$: $unary\text{-}rec\text{-}into\text{-}pr$)
qed

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

proof –
have $c\text{-}f\text{-}pair c\text{-}fst c\text{-}snd \in PrimRec1'$ **by** ($simp add$: $PrimRec1'.fst PrimRec1'.snd PrimRec1'.pair$)
moreover have $c\text{-}f\text{-}pair c\text{-}fst c\text{-}snd = (\lambda x. x)$ **by** ($simp add$: $c\text{-}f\text{-}pair\text{-}def$)
ultimately show $?thesis$ **by** $simp$
qed

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

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

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

proof –

have $pr\text{-}conv\text{-}3\text{-}to\text{-}1 (\lambda x y z. x) = (\lambda x. c\text{-}fst (c\text{-}fst x))$ **by** (*simp add: pr-conv-3-to-1-def*)
moreover from $PrimRec1'.fst PrimRec1'.fst$ **have** $(\lambda x. c\text{-}fst (c\text{-}fst x)) \in Prim\text{-}Rec1'$ **by** (*rule PrimRec1'.comp*)
ultimately show *?thesis* **by** *simp*
qed

lemma $pr\text{-}id3\text{-}2'$: $pr\text{-}conv\text{-}3\text{-}to\text{-}1 (\lambda x y z. y) \in PrimRec1'$
proof –
have $pr\text{-}conv\text{-}3\text{-}to\text{-}1 (\lambda x y z. y) = (\lambda x. c\text{-}snd (c\text{-}fst x))$ **by** (*simp add: pr-conv-3-to-1-def*)
moreover from $PrimRec1'.snd PrimRec1'.fst$ **have** $(\lambda x. c\text{-}snd (c\text{-}fst x)) \in Prim\text{-}Rec1'$ **by** (*rule PrimRec1'.comp*)
ultimately show *?thesis* **by** *simp*
qed

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

lemma $pr\text{-}comp2\text{-}1'$: $\llbracket pr\text{-}conv\text{-}2\text{-}to\text{-}1 f \in PrimRec1'; g \in PrimRec1'; h \in Prim\text{-}Rec1' \rrbracket \implies (\lambda x. f (g x) (h x)) \in PrimRec1'$
proof –
assume $A1$: $pr\text{-}conv\text{-}2\text{-}to\text{-}1 f \in PrimRec1'$
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** $S2$: $c\text{-}f\text{-}pair g h \in PrimRec1'$ **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 $pr\text{-}comp3\text{-}1'$: $\llbracket pr\text{-}conv\text{-}3\text{-}to\text{-}1 f \in PrimRec1'; g \in PrimRec1'; h \in Prim\text{-}Rec1'; k \in PrimRec1' \rrbracket \implies (\lambda x. f (g x) (h x) (k x)) \in PrimRec1'$
proof –
assume $A1$: $pr\text{-}conv\text{-}3\text{-}to\text{-}1 f \in PrimRec1'$
assume $A2$: $g \in PrimRec1'$
assume $A3$: $h \in PrimRec1'$
assume $A4$: $k \in PrimRec1'$
from $A2 A3$ **have** $c\text{-}f\text{-}pair g h \in PrimRec1'$ **by** (*rule PrimRec1'.pair*)
from *this* $A4$ **have** $c\text{-}f\text{-}pair (c\text{-}f\text{-}pair g h) k \in PrimRec1'$ **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 *pr-comp1-2'*: $\llbracket f \in \text{PrimRec1}' ; \text{pr-conv-2-to-1 } g \in \text{PrimRec1}' \rrbracket \implies \text{pr-conv-2-to-1 } (\lambda x y. f (g x y)) \in \text{PrimRec1}'$

proof –

assume $f \in \text{PrimRec1}'$

and $\text{pr-conv-2-to-1 } g \in \text{PrimRec1}'$ (**is** $?g1 \in \text{PrimRec1}'$)

then have $(\lambda x. f (?g1 x)) \in \text{PrimRec1}'$ **by** (rule *PrimRec1'.comp*)

then show *?thesis* **by** (*simp add: pr-conv-2-to-1-def*)

qed

lemma *pr-comp1-3'*: $\llbracket f \in \text{PrimRec1}' ; \text{pr-conv-3-to-1 } g \in \text{PrimRec1}' \rrbracket \implies \text{pr-conv-3-to-1 } (\lambda x y z. f (g x y z)) \in \text{PrimRec1}'$

proof –

assume $f \in \text{PrimRec1}'$

and $\text{pr-conv-3-to-1 } g \in \text{PrimRec1}'$ (**is** $?g1 \in \text{PrimRec1}'$)

then have $(\lambda x. f (?g1 x)) \in \text{PrimRec1}'$ **by** (rule *PrimRec1'.comp*)

then show *?thesis* **by** (*simp add: pr-conv-3-to-1-def*)

qed

lemma *pr-comp2-2'*: $\llbracket \text{pr-conv-2-to-1 } f \in \text{PrimRec1}' ; \text{pr-conv-2-to-1 } g \in \text{PrimRec1}' ; \text{pr-conv-2-to-1 } h \in \text{PrimRec1}' \rrbracket \implies \text{pr-conv-2-to-1 } (\lambda x y. f (g x y) (h x y)) \in \text{PrimRec1}'$

proof –

assume $\text{pr-conv-2-to-1 } f \in \text{PrimRec1}'$

and $\text{pr-conv-2-to-1 } g \in \text{PrimRec1}'$ (**is** $?g1 \in \text{PrimRec1}'$)

and $\text{pr-conv-2-to-1 } h \in \text{PrimRec1}'$ (**is** $?h1 \in \text{PrimRec1}'$)

then have $(\lambda x. f (?g1 x) (?h1 x)) \in \text{PrimRec1}'$ **by** (rule *pr-comp2-1'*)

then show *?thesis* **by** (*simp add: pr-conv-2-to-1-def*)

qed

lemma *pr-comp2-3'*: $\llbracket \text{pr-conv-2-to-1 } f \in \text{PrimRec1}' ; \text{pr-conv-3-to-1 } g \in \text{PrimRec1}' ; \text{pr-conv-3-to-1 } h \in \text{PrimRec1}' \rrbracket \implies \text{pr-conv-3-to-1 } (\lambda x y z. f (g x y z) (h x y z)) \in \text{PrimRec1}'$

proof –

assume $\text{pr-conv-2-to-1 } f \in \text{PrimRec1}'$

and $\text{pr-conv-3-to-1 } g \in \text{PrimRec1}'$ (**is** $?g1 \in \text{PrimRec1}'$)

and $\text{pr-conv-3-to-1 } h \in \text{PrimRec1}'$ (**is** $?h1 \in \text{PrimRec1}'$)

then have $(\lambda x. f (?g1 x) (?h1 x)) \in \text{PrimRec1}'$ **by** (rule *pr-comp2-1'*)

then show *?thesis* **by** (*simp add: pr-conv-3-to-1-def*)

qed

lemma *pr-comp3-2'*: $\llbracket \text{pr-conv-3-to-1 } f \in \text{PrimRec1}' ; \text{pr-conv-2-to-1 } g \in \text{PrimRec1}' ; \text{pr-conv-2-to-1 } h \in \text{PrimRec1}' ; \text{pr-conv-2-to-1 } k \in \text{PrimRec1}' \rrbracket \implies \text{pr-conv-2-to-1 } (\lambda x y. f (g x y) (h x y) (k x y)) \in \text{PrimRec1}'$

proof –

assume $\text{pr-conv-3-to-1 } f \in \text{PrimRec1}'$

and $\text{pr-conv-2-to-1 } g \in \text{PrimRec1}'$ (**is** $?g1 \in \text{PrimRec1}'$)

and $\text{pr-conv-2-to-1 } h \in \text{PrimRec1}'$ (**is** $?h1 \in \text{PrimRec1}'$)

and $\text{pr-conv-2-to-1 } k \in \text{PrimRec1}'$ (**is** $?k1 \in \text{PrimRec1}'$)

then have $(\lambda x. f (?g1 x) (?h1 x) (?k1 x)) \in \text{PrimRec1}'$ **by** (rule *pr-comp3-1'*)

then show *?thesis* by (simp add: pr-conv-2-to-1-def)
qed

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

proof –

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

qed

lemma *lm'*: $(f1 \in \text{PrimRec1} \longrightarrow f1 \in \text{PrimRec1}') \wedge (g1 \in \text{PrimRec2} \longrightarrow \text{pr-conv-2-to-1 } g1 \in \text{PrimRec1}') \wedge (h1 \in \text{PrimRec3} \longrightarrow \text{pr-conv-3-to-1 } h1 \in \text{PrimRec1}')$

proof (induct rule: *PrimRec1-PrimRec2-PrimRec3.induct*)

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

qed

theorem *pr-1-eq-1'*: $\text{PrimRec1} = \text{PrimRec1}'$

proof –

have *S1*: $\bigwedge f. f \in \text{PrimRec1} \longrightarrow f \in \text{PrimRec1}'$ by (simp add: *lm'*)
have *S2*: $\bigwedge f. f \in \text{PrimRec1}' \longrightarrow f \in \text{PrimRec1}$ by (simp add: *primrec'-into-primrec*)
from *S1 S2* show *?thesis* by blast

qed

4.2 The scheme datatype

```

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

```

primrec

```

  sch-to-pr :: PrimScheme  $\Rightarrow$  (nat  $\Rightarrow$  nat)

```

where

```

  sch-to-pr Base-zero = ( $\lambda$  x. 0)
| sch-to-pr Base-suc = Suc
| sch-to-pr Base-fst = c-fst
| sch-to-pr Base-snd = c-snd
| sch-to-pr (Comp-op t1 t2) = ( $\lambda$  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* \in *PrimRec1* **by** (*simp add: pr-1-eq-1'*, *induct sch, simp-all add: PrimRec1'.intros*)

lemma *sch-to-pr-srj*: $f \in$ *PrimRec1* \implies (\exists *sch*. $f =$ *sch-to-pr sch*)

proof –

assume $f \in$ *PrimRec1* **then have** $A1: f \in$ *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 \exists *sch*. (λ u. 0) = *sch-to-pr sch* **by** (*rule exI*)

next

have *Suc* = *sch-to-pr Base-suc* **by** *simp*

then show \exists *sch*. *Suc* = *sch-to-pr sch* **by** (*rule exI*)

next

have *c-fst* = *sch-to-pr Base-fst* **by** *simp*

then show \exists *sch*. *c-fst* = *sch-to-pr sch* **by** (*rule exI*)

next

have *c-snd* = *sch-to-pr Base-snd* **by** *simp*

then show \exists *sch*. *c-snd* = *sch-to-pr sch* **by** (*rule exI*)

next

fix $f1$ $f2$ **assume** $B1: \exists$ *sch*. $f1 =$ *sch-to-pr sch* **and** $B2: \exists$ *sch*. $f2 =$ *sch-to-pr sch*

from $B1$ **obtain** *sch1* **where** $S1: f1 =$ *sch-to-pr sch1* ..

from $B2$ **obtain** *sch2* **where** $S2: f2 =$ *sch-to-pr sch2* ..

from $S1$ $S2$ **have** (λ x. $f1$ ($f2$ x)) = *sch-to-pr (Comp-op sch1 sch2)* **by** *simp*

then show \exists *sch*. (λ x. $f1$ ($f2$ x)) = *sch-to-pr sch* **by** (*rule exI*)

next

fix $f1$ $f2$ **assume** $B1: \exists$ *sch*. $f1 =$ *sch-to-pr sch* **and** $B2: \exists$ *sch*. $f2 =$ *sch-to-pr sch*

from $B1$ **obtain** *sch1* **where** $S1: f1 =$ *sch-to-pr sch1* ..

from $B2$ **obtain** *sch2* **where** $S2: f2 =$ *sch-to-pr sch2* ..

from $S1$ $S2$ **have** *c-f-pair* $f1$ $f2 =$ *sch-to-pr (Pair-op sch1 sch2)* **by** *simp*

```

    then show  $\exists sch. c\text{-}f\text{-}pair\ f1\ f2 = sch\text{-}to\text{-}pr\ sch$  by (rule exI)
  next
    fix f1 f2 assume B1:  $\exists sch. f1 = sch\text{-}to\text{-}pr\ sch$  and B2:  $\exists sch. f2 = sch\text{-}to\text{-}pr\ sch$ 
    from B1 obtain sch1 where S1:  $f1 = sch\text{-}to\text{-}pr\ sch1$  ..
    from B2 obtain sch2 where S2:  $f2 = sch\text{-}to\text{-}pr\ sch2$  ..
    from S1 S2 have UnaryRecOp f1 f2 = sch-to-pr (Rec-op sch1 sch2) by simp
    then show  $\exists sch. UnaryRecOp\ f1\ f2 = sch\text{-}to\text{-}pr\ sch$  by (rule exI)
  qed
qed

```

definition

```

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

```

definition

```

mod7 :: nat  $\Rightarrow$  nat where
mod7 = ( $\lambda x. x\ mod\ 7$ )

```

lemma *c-snd-snd-lt* [termination-simp]: $c\text{-}snd\ (c\text{-}snd\ (Suc\ (Suc\ x))) < Suc\ (Suc\ x)$

proof –

```

let ?y = Suc (Suc x)
have ?y > 1 by simp
then have c-snd ?y < ?y by (rule c-snd-less-arg)
moreover have c-snd (c-snd ?y)  $\leq$  c-snd ?y by (rule c-snd-le-arg)
ultimately show ?thesis by simp

```

qed

lemma *c-fst-snd-lt* [termination-simp]: $c\text{-}fst\ (c\text{-}snd\ (Suc\ (Suc\ x))) < Suc\ (Suc\ x)$

proof –

```

let ?y = Suc (Suc x)
have ?y > 1 by simp
then have c-snd ?y < ?y by (rule c-snd-less-arg)
moreover have c-fst (c-snd ?y)  $\leq$  c-snd ?y by (rule c-fst-le-arg)
ultimately show ?thesis by simp

```

qed

fun *nat-to-sch* :: nat \Rightarrow PrimScheme where

```

  nat-to-sch 0 = Base-zero
| nat-to-sch (Suc 0) = Base-zero

```

| $\text{nat-to-sch } x = (\text{let } u=\text{mod}7 \text{ (c-fst } x); v=\text{c-snd } x; v1=\text{c-fst } v; v2 = \text{c-snd } v;$
 $\text{sch1}=\text{nat-to-sch } v1; \text{sch2}=\text{nat-to-sch } v2 \text{ in loc-f } u \text{ sch1 sch2})$

primrec $\text{sch-to-nat} :: \text{PrimScheme} \Rightarrow \text{nat}$ **where**

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

lemma loc-srj-lm-1 : $\text{nat-to-sch } (\text{Suc } (\text{Suc } x)) = (\text{let } u=\text{mod}7 \text{ (c-fst } (\text{Suc } (\text{Suc } x)));$
 $v=\text{c-snd } (\text{Suc } (\text{Suc } x)); v1=\text{c-fst } v; v2 = \text{c-snd } v; \text{sch1}=\text{nat-to-sch } v1; \text{sch2}=\text{nat-to-sch}$
 $v2 \text{ in loc-f } u \text{ sch1 sch2})$ **by** simp

lemma loc-srj-lm-2 : $x > 1 \implies \text{nat-to-sch } x = (\text{let } u=\text{mod}7 \text{ (c-fst } x); v=\text{c-snd } x;$
 $v1=\text{c-fst } v; v2 = \text{c-snd } v; \text{sch1}=\text{nat-to-sch } v1; \text{sch2}=\text{nat-to-sch } v2 \text{ in loc-f } u \text{ sch1}$
 $\text{sch2})$

proof –

assume $A1$: $x > 1$
let $?y = x - (2::\text{nat})$
from $A1$ **have** $S1$: $x = \text{Suc } (\text{Suc } ?y)$ **by** arith
have $S2$: $\text{nat-to-sch } (\text{Suc } (\text{Suc } ?y)) = (\text{let } u=\text{mod}7 \text{ (c-fst } (\text{Suc } (\text{Suc } ?y)));$
 $v=\text{c-snd } (\text{Suc } (\text{Suc } ?y)); v1=\text{c-fst } v; v2 = \text{c-snd } v; \text{sch1}=\text{nat-to-sch } v1; \text{sch2}=\text{nat-to-sch}$
 $v2 \text{ in loc-f } u \text{ sch1 sch2})$ **by** $(\text{rule } \text{loc-srj-lm-1})$
from $S1 \ S2$ **show** $?thesis$ **by** simp
qed

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

proof –

let $?x = \text{c-pair } 1 \ 0$
have $S1$: $?x = 2$ **by** $(\text{simp add: c-pair-def sf-def})$
then have $S2$: $?x = \text{Suc } (\text{Suc } 0)$ **by** simp
let $?y = \text{Suc } (\text{Suc } 0)$
have $S3$: $\text{nat-to-sch } ?y = (\text{let } u=\text{mod}7 \text{ (c-fst } ?y); v=\text{c-snd } ?y; v1=\text{c-fst } v; v2 =$
 $\text{c-snd } v; \text{sch1}=\text{nat-to-sch } v1; \text{sch2}=\text{nat-to-sch } v2 \text{ in loc-f } u \text{ sch1 sch2})$ **(is - = ?R)**
by $(\text{rule } \text{loc-srj-lm-1})$
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** $S6$: $\text{mod}7 \text{ (c-fst } ?y) = 1$ **by** $(\text{simp add: mod7-def})$

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

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

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

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

have $S4$: $c\text{-snd } ?x = 0$ **by** *simp*
from $S3$ **have** $S6$: $\text{mod}7 (c\text{-fst } ?x) = 3$ **by** (*simp add: mod7-def*)
from $S3 S4 S6$ **have** $S7$: $?R = \text{loc-f } 3$ *Base-zero Base-zero* **by** (*simp add: Let-def c-fst-at-0 c-snd-at-0*)
then have $S8$: $?R = \text{Base-snd}$ **by** (*simp add: loc-f-def*)
with $S2$ **have** $S10$: $\text{nat-to-sch } ?x = \text{Base-snd}$ **by** *simp*
from $S2$ **this show** $?thesis$ **by** *simp*
qed

lemma *loc-srj-3*: $\llbracket \text{nat-to-sch } (\text{sch-to-nat } sch1) = sch1; \text{nat-to-sch } (\text{sch-to-nat } sch2) = sch2 \rrbracket$
 $\implies \text{nat-to-sch } (c\text{-pair } 4 (c\text{-pair } (\text{sch-to-nat } sch1) (\text{sch-to-nat } sch2))) =$
Comp-op sch1 sch2

proof –
assume $A1$: $\text{nat-to-sch } (\text{sch-to-nat } sch1) = sch1$
assume $A2$: $\text{nat-to-sch } (\text{sch-to-nat } sch2) = sch2$
let $?x = c\text{-pair } 4 (c\text{-pair } (\text{sch-to-nat } sch1) (\text{sch-to-nat } sch2))$
have $S1$: $?x > 1$ **by** (*simp add: c-pair-def sf-def*)
from $S1$ **have** $S2$: $\text{nat-to-sch } ?x = (\text{let } u = \text{mod}7 (c\text{-fst } ?x); v = c\text{-snd } ?x; v1 = c\text{-fst } v; v2 = c\text{-snd } v; sch1 = \text{nat-to-sch } v1; sch2 = \text{nat-to-sch } v2 \text{ in } \text{loc-f } u \text{ sch1 sch2})$ (**is** $= ?R$) **by** (*rule loc-srj-lm-2*)
have $S3$: $c\text{-fst } ?x = 4$ **by** *simp*
have $S4$: $c\text{-snd } ?x = c\text{-pair } (\text{sch-to-nat } sch1) (\text{sch-to-nat } sch2)$ **by** *simp*
from $S3$ **have** $S5$: $\text{mod}7 (c\text{-fst } ?x) = 4$ **by** (*simp add: mod7-def*)
from $A1 A2 S4 S5$ **have** $?R = \text{Comp-op } sch1 \text{ sch2}$ **by** (*simp add: Let-def c-fst-at-0 c-snd-at-0 loc-f-def*)
with $S2$ **show** $?thesis$ **by** *simp*
qed

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

proof –
let $?x = c\text{-pair } 4 (c\text{-pair } n1 \ n2)$
have $S1$: $?x > 1$ **by** (*simp add: c-pair-def sf-def*)
from $S1$ **have** $S2$: $\text{nat-to-sch } ?x = (\text{let } u = \text{mod}7 (c\text{-fst } ?x); v = c\text{-snd } ?x; v1 = c\text{-fst } v; v2 = c\text{-snd } v; sch1 = \text{nat-to-sch } v1; sch2 = \text{nat-to-sch } v2 \text{ in } \text{loc-f } u \text{ sch1 sch2})$ (**is** $= ?R$) **by** (*rule loc-srj-lm-2*)
have $S3$: $c\text{-fst } ?x = 4$ **by** *simp*
have $S4$: $c\text{-snd } ?x = c\text{-pair } n1 \ n2$ **by** *simp*
from $S3$ **have** $S5$: $\text{mod}7 (c\text{-fst } ?x) = 4$ **by** (*simp add: mod7-def*)
from $S4 S5$ **have** $?R = \text{Comp-op } (\text{nat-to-sch } n1) (\text{nat-to-sch } n2)$ **by** (*simp add: Let-def c-fst-at-0 c-snd-at-0 loc-f-def*)
with $S2$ **show** $?thesis$ **by** *simp*
qed

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

proof –

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

proof –

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

lemma *loc-srj-5*: $\llbracket \text{nat-to-sch } (\text{sch-to-nat } \text{sch1}) = \text{sch1}; \text{nat-to-sch } (\text{sch-to-nat } \text{sch2}) = \text{sch2} \rrbracket$

$\implies \text{nat-to-sch } (\text{c-pair } 6 (\text{c-pair } (\text{sch-to-nat } \text{sch1}) (\text{sch-to-nat } \text{sch2}))) = \text{Rec-op } \text{sch1 } \text{sch2}$

proof –

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

proof –

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

have $S3$: $c\text{-fst} ?x = 6$ **by** *simp*

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

from $S3$ **have** $S5$: $mod7 (c\text{-fst} ?x) = 6$ **by** (*simp add: mod7-def*)

from $S4$ $S5$ **have** $?R = Rec\text{-op} (nat\text{-to}\text{-sch} n1) (nat\text{-to}\text{-sch} n2)$ **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*: $nat\text{-to}\text{-sch} (sch\text{-to}\text{-nat} sch) = sch$

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

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

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

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

lemma *nat-to-pr-srj*: $f \in PrimRec1 \implies (\exists n. f = nat\text{-to}\text{-pr} n)$

proof –

assume $f \in PrimRec1$

then have $S1$: $(\exists t. f = sch\text{-to}\text{-pr} t)$ **by** (*rule sch-to-pr-srj*)

from $S1$ **obtain** t **where** $S2$: $f = sch\text{-to}\text{-pr} t$..

let $?n = sch\text{-to}\text{-nat} t$

have $S3$: $nat\text{-to}\text{-pr} ?n = sch\text{-to}\text{-pr} (nat\text{-to}\text{-sch} ?n)$ **by** (*simp add: nat-to-pr-def*)

have $S4$: $nat\text{-to}\text{-sch} ?n = t$ **by** (*rule nat-to-sch-srj*)

from $S3$ $S4$ **have** $S5$: $nat\text{-to}\text{-pr} ?n = sch\text{-to}\text{-pr} t$ **by** *simp*

from $S2$ $S5$ **have** $nat\text{-to}\text{-pr} ?n = f$ **by** *simp*

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

then show $?thesis$..

qed

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

definition

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

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

proof –

assume $f \in \text{PrimRec1}$

hence $\exists n. f = \text{nat-to-pr } n$ **by** (*rule nat-to-pr-srj*)

hence $f = \text{nat-to-pr } (\text{SOME } n. f = \text{nat-to-pr } n)$ **by** (*rule someI-ex*)

thus *?thesis* **by** (*simp add: index-of-pr-def*)

qed

definition

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

definition

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

definition

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

lemma *comp-by-index-is-pr*: $\text{comp-by-index} \in \text{PrimRec2}$

unfolding *comp-by-index-def*

using *const-is-pr-2 [of 4]* **by** *prec*

lemma *comp-by-index-inj*: $\text{comp-by-index } x1 y1 = \text{comp-by-index } x2 y2 \implies x1 = x2 \wedge y1 = y2$

proof –

assume $\text{comp-by-index } x1 y1 = \text{comp-by-index } x2 y2$

hence $\text{c-pair } 4 (\text{c-pair } x1 y1) = \text{c-pair } 4 (\text{c-pair } x2 y2)$ **by** (*unfold comp-by-index-def*)

hence $\text{c-pair } x1 y1 = \text{c-pair } x2 y2$ **by** (*rule c-pair-inj2*)

thus *?thesis* **by** (*rule c-pair-inj*)

qed

lemma *comp-by-index-inj1*: $\text{comp-by-index } x1 y1 = \text{comp-by-index } x2 y2 \implies x1 = x2$ **by** (*frule comp-by-index-inj, drule conjunct1*)

lemma *comp-by-index-inj2*: $\text{comp-by-index } x1 y1 = \text{comp-by-index } x2 y2 \implies y1 = y2$ **by** (*frule comp-by-index-inj, drule conjunct2*)

lemma *comp-by-index-main*: $\text{nat-to-pr } (\text{comp-by-index } n1 n2) = (\lambda x. (\text{nat-to-pr } n1) ((\text{nat-to-pr } n2) x))$ **by** (*unfold comp-by-index-def, unfold nat-to-pr-def, simp add: loc-srj-3-1*)

lemma *pair-by-index-is-pr*: $\text{pair-by-index} \in \text{PrimRec2}$ **by** (*unfold pair-by-index-def*, *insert const-is-pr-2* [**where** $?n=(5::\text{nat})$], *prec*)

lemma *pair-by-index-inj*: $\text{pair-by-index } x1 \ y1 = \text{pair-by-index } x2 \ y2 \implies x1=x2 \wedge y1=y2$

proof –

assume $\text{pair-by-index } x1 \ y1 = \text{pair-by-index } x2 \ y2$

hence $c\text{-pair } 5 \ (c\text{-pair } x1 \ y1) = c\text{-pair } 5 \ (c\text{-pair } x2 \ y2)$ **by** (*unfold pair-by-index-def*)

hence $c\text{-pair } x1 \ y1 = c\text{-pair } x2 \ y2$ **by** (*rule c-pair-inj2*)

thus *?thesis* **by** (*rule c-pair-inj*)

qed

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

lemma *pair-by-index-inj2*: $\text{pair-by-index } x1 \ y1 = \text{pair-by-index } x2 \ y2 \implies y1 = y2$
by (*frule pair-by-index-inj*, *drule conjunct2*)

lemma *pair-by-index-main*: $\text{nat-to-pr } (\text{pair-by-index } n1 \ n2) = c\text{-f-pair } (\text{nat-to-pr } n1) \ (\text{nat-to-pr } n2)$ **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*]: $\text{nat-to-sch } (\text{pair-by-index } n1 \ n2) = \text{Pair-op } (\text{nat-to-sch } n1) \ (\text{nat-to-sch } n2)$

by (*simp add: pair-by-index-def loc-srj-4-1*)

lemma *rec-by-index-is-pr*: $\text{rec-by-index} \in \text{PrimRec2}$ **by** (*unfold rec-by-index-def*, *insert const-is-pr-2* [**where** $?n=(6::\text{nat})$], *prec*)

lemma *rec-by-index-inj*: $\text{rec-by-index } x1 \ y1 = \text{rec-by-index } x2 \ y2 \implies x1=x2 \wedge y1=y2$

proof –

assume $\text{rec-by-index } x1 \ y1 = \text{rec-by-index } x2 \ y2$

hence $c\text{-pair } 6 \ (c\text{-pair } x1 \ y1) = c\text{-pair } 6 \ (c\text{-pair } x2 \ y2)$ **by** (*unfold rec-by-index-def*)

hence $c\text{-pair } x1 \ y1 = c\text{-pair } x2 \ y2$ **by** (*rule c-pair-inj2*)

thus *?thesis* **by** (*rule c-pair-inj*)

qed

lemma *rec-by-index-inj1*: $\text{rec-by-index } x1 \ y1 = \text{rec-by-index } x2 \ y2 \implies x1 = x2$
by (*frule rec-by-index-inj*, *drule conjunct1*)

lemma *rec-by-index-inj2*: $\text{rec-by-index } x1 \ y1 = \text{rec-by-index } x2 \ y2 \implies y1 = y2$
by (*frule rec-by-index-inj*, *drule conjunct2*)

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

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

definition

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

lemma *index-of-const-is-pr*: *index-of-const* \in *PrimRec1*

proof –

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

then show ?*thesis* **by** (*simp add: index-of-const-def pr-rec1*)

qed

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

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

lemma *index-of-const-main*: *nat-to-pr* (*index-of-const* *n*) = (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* \implies *n1* = *n2*

proof –

assume *index-of-const* *n1* = *index-of-const* *n2*

then have (*nat-to-pr* (*index-of-const* *n1*)) 0 = (*nat-to-pr* (*index-of-const* *n2*)) 0 **by** *simp*

thus ?*thesis* **by** (*simp add: index-of-const-lm-1*)

qed

definition *index-of-zero* = *sch-to-nat* *Base-zero*

definition *index-of-suc* = *sch-to-nat* *Base-suc*

definition *index-of-c-fst* = *sch-to-nat* *Base-fst*

definition *index-of-c-snd* = *sch-to-nat* *Base-snd*

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

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

lemma *index-of-suc-main*: *nat-to-pr* *index-of-suc* = *Suc*

apply(*simp add: index-of-suc-def nat-to-pr-def*)

apply(*insert loc-srj-0*)
apply(*simp*)
done

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

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

lemma *index-of-c-snd-main*: *nat-to-pr index-of-c-snd = c-snd* **by** (*simp add: index-of-c-snd-def nat-to-pr-def loc-srj-2*)

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

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

definition

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

lemma *index-of-c-pair-n-is-pr*: *index-of-c-pair-n ∈ PrimRec1*

proof –

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

qed

lemma *index-of-c-pair-n-main*: *nat-to-pr (index-of-c-pair-n n) = (λ x. c-pair n x)*

proof –

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

qed

lemma *index-of-c-pair-n-inj*: *index-of-c-pair-n x1 = index-of-c-pair-n x2 ⇒ x1=x2*

proof –

assume *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

$s1-1 :: nat \Rightarrow nat \Rightarrow nat$ **where**
 $s1-1 = (\lambda n x. comp-by-index n (index-of-c-pair-n x))$

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

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

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 $\dots = (\lambda z. (nat-to-pr n) ((nat-to-pr (index-of-c-pair-n x)) z))$ **by**
(*simp add: comp-by-index-main*)

also have $\dots = (\lambda z. (nat-to-pr n) ((\lambda u. c-pair x u) z))$ **by** (*simp add: index-of-c-pair-n-main*)

finally show *?thesis* **by** *simp*

qed

lemma $s1-1-inj$: $s1-1 x1 y1 = s1-1 x2 y2 \Longrightarrow x1=x2 \wedge y1=y2$

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** (*unfold s1-1-def*)

then have $S1: x1=x2 \wedge index-of-c-pair-n y1 = index-of-c-pair-n y2$ **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 $s1-1-inj1$: $s1-1 x1 y1 = s1-1 x2 y2 \Longrightarrow x1=x2$ **by** (*frule s1-1-inj, drule conjunct1*)

lemma $s1-1-inj2$: $s1-1 x1 y1 = s1-1 x2 y2 \Longrightarrow y1=y2$ **by** (*frule s1-1-inj, drule conjunct2*)

primrec

$pr-index-enumerator :: nat \Rightarrow nat \Rightarrow nat$

where

$pr-index-enumerator n 0 = n$

| $pr-index-enumerator n (Suc m) = comp-by-index index-of-id (pr-index-enumerator n m)$

theorem $pr-index-enumerator-is-pr$: $pr-index-enumerator \in PrimRec2$

proof –

define g **where** $g x = x$ **for** $x :: nat$

```

have g-is-pr:  $g \in \text{PrimRec1}$  by (unfold g-def, rule pr-id1-1)
define h where  $h\ a\ b\ c = \text{comp-by-index index-of-id } b \text{ for } a\ b\ c :: \text{nat}$ 
from comp-by-index-is-pr have h-is-pr:  $h \in \text{PrimRec3}$  unfolding h-def by prec
let ?f = pr-index-enumerator
from g-def have f-at-0:  $\forall x. ?f\ x\ 0 = g\ x$  by auto
from h-def have f-at-Suc:  $\forall x\ y. ?f\ x\ (\text{Suc } y) = h\ x\ (?f\ x\ y)\ y$  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* $n\ m < \text{pr-index-enumerator } (n+1)\ m$

```

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

```

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

```

proof -
  let ?a = pr-index-enumerator  $n\ m$ 
  have S1: pr-index-enumerator  $n\ (m+1) = \text{comp-by-index index-of-id } ?a$  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:  $4 + \text{c-pair index-of-id } ?a \leq \text{c-pair 4 (c-pair index-of-id ?a)}$  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:  $?a \leq \text{c-pair index-of-id } ?a$  by (rule arg2-le-c-pair)
  from S4 S5 have S6:  $?a < \text{c-pair 4 (c-pair index-of-id ?a)}$  by auto
  with S1 S2 show ?thesis by auto
qed

```

```

lemma f-inc-mono: (∀ (x::nat). (f::nat⇒nat) x < f (x+1)) ⇒ ∀ (x::nat) (y::nat).
(x < y → f x < f y)
proof (rule allI, rule allI)
  fix x y assume A: ∀ (x::nat). f x < f (x+1) show x < y → f x < f y
  proof
    assume A1: x < y
    have L1: ∧ u v. f u < f (u + (v+1))
    proof –
      fix u v show f u < f (u + (v+1))
      proof (induct v)
        from A show f u < f (u + (0 + 1)) by auto
      next
        fix v n
        assume A2: f u < f (u + (n + 1))
        from A have S1: f (u + (n + 1)) < f (u + (Suc n + 1)) by auto
        from A2 S1 show f u < f (u + (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 ⇒ pr-index-enumerator n1 m <
pr-index-enumerator n2 m
proof –
  assume A: n1 < n2
  define f where f x = pr-index-enumerator x m for x
  have f-inc: ∀ x. f x < f (x+1)
  proof
    fix x show f x < f (x+1) by (unfold f-def, rule pr-index-enumerator-increase1)
  qed
  from f-inc have ∀ x y. (x < y → f x < f y) by (rule f-inc-mono)
  with A f-def show ?thesis by auto
qed

```

```

lemma pr-index-enumerator-mono2: m1 < m2 ⇒ pr-index-enumerator n m1 <
pr-index-enumerator n m2
proof –
  assume A: m1 < m2
  define f where f x = pr-index-enumerator n x for x
  have f-inc: ∀ x. f x < f (x+1)
  proof
    fix x show f x < f (x+1) by (unfold f-def, rule pr-index-enumerator-increase2)
  qed
  from f-inc have ∀ x y. (x < y → f x < f y) by (rule f-inc-mono)

```

with A f -def **show** *?thesis* **by** *auto*
qed

lemma f -mono-inj: $\forall (x::nat) (y::nat). (x < y \longrightarrow (f::nat \Rightarrow nat) x < f y) \implies \forall (x::nat) (y::nat). (f x = f y \longrightarrow x = y)$

proof (*rule allI, rule allI*)

fix $x y$ **assume** A : $\forall x y. x < y \longrightarrow f x < f y$ **show** $f x = f y \longrightarrow x = y$

proof

assume $A1$: $f x = f y$ **show** $x = y$

proof (*rule ccontr*)

assume $A2$: $x \neq y$ **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 $\neg x < y$ **with** $A2$ **have** $A4$: $y < x$ **by** *auto*

from A $A4$ **have** $f y < f x$ **by** *auto*

with $A1$ **show** *False* **by** *auto*

qed

qed

qed

qed

theorem pr -index-enumerator-inj1: pr -index-enumerator $n1$ $m = pr$ -index-enumerator $n2$ $m \implies n1 = n2$

proof $-$

assume A : pr -index-enumerator $n1$ $m = pr$ -index-enumerator $n2$ m

define f **where** $f x = pr$ -index-enumerator x m **for** x

have f -mono: $\forall x y. (x < y \longrightarrow f x < f y)$

proof (*rule allI, rule allI*)

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

qed

from f -mono **have** $\forall x y. (f x = f y \longrightarrow x = y)$ **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 $m2 \implies m1 = m2$

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: $\forall x y. (x < y \longrightarrow f x < f y)$

proof (*rule allI, rule allI*)

fix $x y$ **show** $x < y \longrightarrow f x < f y$ **by** (*unfold f-def, simp add: pr-index-enumerator-mono2*)

qed

from f -mono **have** $\forall x y. (f x = f y \longrightarrow x = y)$ **by** (*rule f-mono-inj*)

with A f -def **show** *?thesis* **by** *auto*

qed

```

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

end

```

5 Finite sets

```

theory PRecFinSet
imports PRecFun
begin

```

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

definition

```

c-in :: nat  $\Rightarrow$  nat  $\Rightarrow$  nat where
c-in = ( $\lambda$  x u. (u div ( $2^{\wedge} x$ )) mod 2)

```

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

proof –

```

from mod-is-pr power-is-pr div-is-pr have ( $\lambda$  x u. (u div ( $2^{\wedge} x$ )) mod 2)  $\in$ 
PrimRec2 by prec
with c-in-def show ?thesis by auto
qed

```

definition

```

nat-to-set :: nat  $\Rightarrow$  nat set where
nat-to-set u  $\equiv$  {x.  $2^{\wedge} x \leq u \wedge$  c-in x u = 1}

```

lemma *c-in-upper-bound*: *c-in* *x u* = 1 $\implies 2^{\wedge} x \leq u$

proof –

```

assume A: c-in x u = 1
then have S1: (u div (2^x)) mod 2 = 1 by (unfold c-in-def)
then have S2: u div (2^x) > 0 by arith
show ?thesis
proof (rule ccontr)
  assume ¬ 2^x ≤ u
  then have u < 2^x by auto
  then have u div (2^x) = 0 by (rule div-less)
  with S2 show False by auto
qed
qed

```

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

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

lemma nat-to-set-upper-bound1: $x \in \text{nat-to-set } u \implies x < u$
proof –
assume $x \in \text{nat-to-set } u$
then have S1: $2^x \leq u$ **by** (simp add: nat-to-set-def)
have S2: $x < 2^x$ **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 (nat-to-set u)
proof –
have S1: finite {i. i < u}
proof –
let ?B = {i. i < u}
let ?f = (λ (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

$\log2 :: \text{nat} \Rightarrow \text{nat}$ **where**
 $\log2 = (\lambda x. \text{Least}(\%z. x < 2^{z+1}))$

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

proof –

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

qed

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

proof –

let $?v = \log2\ 1$
have $S1: 0 \leq ?v$ **by** *auto*
have $S2: ?v = \text{Least}(\%(z::\text{nat}). (1::\text{nat}) < 2^{z+1})$ **by** (*simp add: log2-def*)
have $S3: (1::\text{nat}) < 2^{0+1}$ **by** *auto*
from $S3$ **have** $S4: \text{Least}(\%(z::\text{nat}). (1::\text{nat}) < 2^{z+1}) \leq 0$ **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*: $x > 0 \implies 2^{\log2\ x} \leq x$

proof –

assume $A: x > 0$
show *?thesis*
proof (*cases*)
assume $A1: \log2\ x = 0$
with A **show** *?thesis* **by** *auto*
next
assume $A1: \log2\ x \neq 0$
then **have** $S1: \log2\ x > 0$ **by** *auto*
define y **where** $y = \log2\ x - 1$
from $S1\ y\text{-def}$ **have** $S2: \log2\ x = y + 1$ **by** *auto*
then **have** $S3: y < \log2\ x$ **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** $\log2\ x \leq y$ **by** (*simp add: log2-def Least-le*)
with $S3$ **show** *False* **by** *auto*

qed

with $S2$ **show** *?thesis* **by** *auto*

qed

qed

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

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

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

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

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

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

lemma *log2-lm1*: $u > 0 \implies u \text{ div } 2^{\wedge}(\log 2 u) = 1$

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 $S4: (2::nat)^{\wedge}\log 2 u \text{ div } 2^{\wedge}\log 2 u = 1$ **by** *auto*
 from $S1$ **have** $S5: (2::nat)^{\wedge}\log 2 u \text{ div } 2^{\wedge}\log 2 u \leq u \text{ div } 2^{\wedge}\log 2 u$ **by** (*rule div-le-mono*)
 with $S4$ **have** $S6: 1 \leq u \text{ div } 2^{\wedge}\log 2 u$ **by** *auto*

from $S3$ **have** $S7: u \text{ div } 2^{\log_2 u} < 2$ **by** (rule *div-lt*)
from $S6$ $S7$ **show** *?thesis* **by** *auto*
qed

lemma *log2-lm2*: $u > 0 \implies c\text{-in } (\log_2 u) u = 1$

proof –

assume $A: u > 0$
then have $S1: u \text{ div } 2^{\log_2 u} = 1$ **by** (rule *log2-lm1*)
have $c\text{-in } (\log_2 u) u = (u \text{ div } 2^{\log_2 u}) \bmod 2$ **by** (*simp add: c-in-def*)
also from $S1$ **have** $\dots = 1 \bmod 2$ **by** *simp*
also have $\dots = 1$ **by** *auto*
finally show *?thesis* **by** *auto*

qed

lemma *log2-lm3*: $\log_2 u < x \implies c\text{-in } x u = 0$

proof –

assume $A: \log_2 u < x$
then have $S1: (\log_2 u) + 1 \leq x$ **by** *auto*
have $S2: 1 \leq (2::\text{nat})$ **by** *auto*
from $S1$ $S2$ **have** $S3: (2::\text{nat})^{(\log_2 u) + 1} \leq 2^x$ **by** (rule *power-increasing*)
have $S4: u < (2::\text{nat})^{(\log_2 u) + 1}$ **by** (rule *log2-gt*)
from $S3$ $S4$ **have** $S5: u < 2^x$ **by** *auto*
then have $S6: u \text{ div } 2^x = 0$ **by** (rule *div-less*)
have $c\text{-in } x u = (u \text{ div } 2^x) \bmod 2$ **by** (*simp add: c-in-def*)
also from $S6$ **have** $\dots = 0 \bmod 2$ **by** *simp*
also have $\dots = 0$ **by** *auto*
finally have *?thesis* **by** *auto*
thus *?thesis* **by** *auto*

qed

lemma *log2-lm4*: $c\text{-in } x u = 1 \implies x \leq \log_2 u$

proof –

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

qed

qed

lemma *nat-to-set-lub*: $x \in \text{nat-to-set } u \implies x \leq \log_2 u$

proof –

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

qed

lemma *log2-lm5*: $u > 0 \implies \log_2 u \in \text{nat-to-set } u$

proof –

assume A : $u > 0$

then have $c\text{-in } (\log_2 u) u = 1$ **by** (*rule log2-lm2*)

then show *?thesis* **by** (*simp add: x-in-u-eq*)

qed

lemma *pos-imp-ne*: $u > 0 \implies \text{nat-to-set } u \neq \{\}$

proof –

assume $u > 0$

then have $\log_2 u \in \text{nat-to-set } u$ **by** (*rule log2-lm5*)

thus *?thesis* **by** *auto*

qed

lemma *empty-is-zero*: $\text{nat-to-set } u = \{\} \implies u = 0$

proof (*rule ccontr*)

assume $A1$: $\text{nat-to-set } u = \{\}$

assume $A2$: $u \neq 0$ **then have** $S1$: $u > 0$ **by** *auto*

from $S1$ **have** $\text{nat-to-set } u \neq \{\}$ **by** (*rule pos-imp-ne*)

with $A1$ **show** *False* **by** *auto*

qed

lemma *log2-is-max*: $u > 0 \implies \log_2 u = \text{Max } (\text{nat-to-set } u)$

proof –

assume A : $u > 0$

then have $S1$: $\log_2 u \in \text{nat-to-set } u$ **by** (*rule log2-lm5*)

define max **where** $max = \text{Max } (\text{nat-to-set } u)$

from A **have** ne : $\text{nat-to-set } u \neq \{\}$ **by** (*rule pos-imp-ne*)

have $finite$: $finite (\text{nat-to-set } u)$ **by** (*rule nat-to-set-is-finite*)

from $max\text{-def } finite\ ne$ **have** $max\text{-in}$: $max \in \text{nat-to-set } u$ **by** *simp*

from $max\text{-in}$ **have** $S2$: $c\text{-in } max\ u = 1$ **by** (*simp add: x-in-u-eq*)

then have $S3$: $max \leq \log_2 u$ **by** (*rule log2-lm4*)

from $finite\ ne\ S1\ max\text{-def}$ **have** $S4$: $\log_2 u \leq max$ **by** *simp*

from $S3\ S4\ max\text{-def}$ **show** *?thesis* **by** *auto*

qed

lemma *zero-is-empty*: $\text{nat-to-set } 0 = \{\}$

proof –

have $S1$: $\{i. i < (0::\text{nat})\} = \{\}$ **by** *blast*

have $S2$: $\text{nat-to-set } 0 \subseteq \{i. i < 0\}$ **by** (*rule nat-to-set-upper-bound2*)

from $S1\ S2$ **show** *?thesis* **by** *auto*

qed

lemma *ne-imp-pos*: $\text{nat-to-set } u \neq \{\} \implies u > 0$

proof (*rule ccontr*)

assume $A1$: $\text{nat-to-set } u \neq \{\}$

assume $\neg 0 < u$ **then have** $u = 0$ **by** *auto*

then have $\text{nat-to-set } u = \{\}$ **by** (*simp add: zero-is-empty*)

with $A1$ **show** *False* **by** *auto*

qed

lemma *div-mod-lm*: $y < x \implies ((u + (2::nat) \hat{x}) \text{ div } (2::nat) \hat{y}) \text{ mod } 2 = (u \text{ div } (2::nat) \hat{y}) \text{ mod } 2$

proof –

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

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

proof –

assume *A*: $u < 2 \hat{x}$
 have *log2-is-x*: $\text{log2 } (u + 2 \hat{x}) = x$
 proof (*unfold log2-def, rule Least-equality*)
 from *A* **show** $u + 2 \hat{x} < 2 \hat{(x+1)}$ **by** *auto*
 next
 fix z
 assume *A1*: $u + 2 \hat{x} < 2 \hat{(z+1)}$
 show $x \leq z$
 proof (*rule ccontr*)
 assume $\neg x \leq z$
 then **have** $z < x$ **by** *auto*
 then **have** *L1*: $z+1 \leq x$ **by** *auto*
 have *L2*: $1 \leq (2::nat)$ **by** *auto*
 from *L1 L2* **have** *L3*: $(2::nat) \hat{(z+1)} \leq (2::nat) \hat{x}$ **by** (*rule power-increasing*)
 with *A1* **show** *False* **by** *auto*
 qed
 qed
 show *?thesis*

```

proof (rule subset-antisym)
  show  $\text{nat-to-set } (u + 2^{\wedge} x) \subseteq \text{nat-to-set } u \cup \{x\}$ 
  proof fix  $y$ 
    assume  $A1: y \in \text{nat-to-set } (u + 2^{\wedge} x)$ 
    show  $y \in \text{nat-to-set } u \cup \{x\}$ 
    proof
      assume  $y \notin \{x\}$  then have  $S1: y \neq x$  by auto
      from  $A1$  have  $y \leq \log 2 (u + 2^{\wedge} x)$  by (rule nat-to-set-lub)
      with log2-is-x have  $y \leq x$  by auto
      with  $S1$  have  $y\text{-lt-}x: y < x$  by auto
      from  $A1$  have  $c\text{-in } y (u + 2^{\wedge} x) = 1$  by (simp add: x-in-u-eq)
      then have  $S2: ((u + 2^{\wedge} x) \text{ div } 2^{\wedge} y) \bmod 2 = 1$  by (unfold c-in-def)
      from  $y\text{-lt-}x$  have  $((u + (2::\text{nat})^{\wedge} x) \text{ div } (2::\text{nat})^{\wedge} y) \bmod 2 = (u \text{ div } (2::\text{nat})^{\wedge} y) \bmod 2$  by (rule div-mod-lm)
      with  $S2$  have  $(u \text{ div } 2^{\wedge} y) \bmod 2 = 1$  by auto
      then have  $c\text{-in } y u = 1$  by (simp add: c-in-def)
      then show  $y \in \text{nat-to-set } u$  by (simp add: x-in-u-eq)
    qed
  qed
  next
  show  $\text{nat-to-set } u \cup \{x\} \subseteq \text{nat-to-set } (u + 2^{\wedge} x)$ 
  proof fix  $y$ 
    assume  $A1: y \in \text{nat-to-set } u \cup \{x\}$ 
    show  $y \in \text{nat-to-set } (u + 2^{\wedge} x)$ 
    proof cases
      assume  $y \in \{x\}$ 
      then have  $y=x$  by auto
      then have  $y = \log 2 (u + 2^{\wedge} x)$  by (simp add: log2-is-x)
      then show ?thesis by (simp add: log2-lm5)
    next
      assume  $y\text{-notin}: y \notin \{x\}$ 
      then have  $y\text{-ne-}x: y \neq x$  by auto
      from  $A1$   $y\text{-notin}$  have  $y\text{-in}: y \in \text{nat-to-set } u$  by auto
      have  $y\text{-lt-}x: y < x$ 
      proof (rule ccontr)
        assume  $\neg y < x$ 
        with  $y\text{-ne-}x$  have  $y\text{-gt-}x: x < y$  by auto
        have  $1 < (2::\text{nat})$  by auto
        from  $y\text{-gt-}x$  this have  $L1: (2::\text{nat})^{\wedge} x < 2^{\wedge} y$  by (rule power-strict-increasing)
        from  $y\text{-in}$  have  $L2: 2^{\wedge} y \leq u$  by (rule nat-to-set-upper-bound)
        from  $L1$   $L2$  have  $(2::\text{nat})^{\wedge} x < u$  by arith
        with  $A$  show False by auto
      qed
      from  $y\text{-in}$  have  $c\text{-in } y u = 1$  by (simp add: x-in-u-eq)
      then have  $S2: (u \text{ div } 2^{\wedge} y) \bmod 2 = 1$  by (unfold c-in-def)
      from  $y\text{-lt-}x$  have  $((u + (2::\text{nat})^{\wedge} x) \text{ div } (2::\text{nat})^{\wedge} y) \bmod 2 = (u \text{ div } (2::\text{nat})^{\wedge} y) \bmod 2$  by (rule div-mod-lm)
      with  $S2$  have  $((u + (2::\text{nat})^{\wedge} x) \text{ div } 2^{\wedge} y) \bmod 2 = 1$  by auto
      then have  $c\text{-in } y (u + (2::\text{nat})^{\wedge} x) = 1$  by (simp add: c-in-def)
    
```

```

    then show  $y \in \text{nat-to-set } (u + (2::\text{nat}) \hat{\ } x)$  by (simp add: x-in-u-eq)
  qed
qed
qed
qed
theorem nat-to-set-inj:  $\text{nat-to-set } u = \text{nat-to-set } v \implies u = v$ 
proof -
  assume A:  $\text{nat-to-set } u = \text{nat-to-set } v$ 
  let ?P =  $\lambda (n::\text{nat}). (\forall (D::\text{nat set}). \text{finite } D \wedge \text{card } D \leq n \longrightarrow (\forall u v. \text{nat-to-set } u = D \wedge \text{nat-to-set } v = D \longrightarrow u = v))$ 
  have P-at-0: ?P 0
  proof fix D show  $\text{finite } D \wedge \text{card } D \leq 0 \longrightarrow (\forall u v. \text{nat-to-set } u = D \wedge \text{nat-to-set } v = D \longrightarrow u = v)$ 
    proof (rule impI)
      assume A1:  $\text{finite } D \wedge \text{card } D \leq 0$ 
      from A1 have S1:  $\text{finite } D$  by auto
      from A1 have S2:  $\text{card } D = 0$  by auto
      from S1 S2 have S3:  $D = \{\}$  by auto
      show  $(\forall u v. \text{nat-to-set } u = D \wedge \text{nat-to-set } v = D \longrightarrow u = v)$ 
      proof (rule allI, rule allI) fix u v show  $\text{nat-to-set } u = D \wedge \text{nat-to-set } v = D \longrightarrow u = v$ 
        proof
          assume A2:  $\text{nat-to-set } u = D \wedge \text{nat-to-set } v = D$ 
          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-z:  $u = 0$  by (rule empty-is-zero)
          from L2 S3 have  $\text{nat-to-set } v = \{\}$  by auto
          then have v-z:  $v = 0$  by (rule empty-is-zero)
          from u-z v-z show  $u=v$  by auto
        qed
      qed
    qed
  have P-at-Suc:  $\bigwedge n. ?P n \implies ?P (\text{Suc } n)$ 
  proof - fix n
    assume A-n: ?P n
    show ?P (Suc n)
    proof fix D show  $\text{finite } D \wedge \text{card } D \leq \text{Suc } n \longrightarrow (\forall u v. \text{nat-to-set } u = D \wedge \text{nat-to-set } v = D \longrightarrow u = v)$ 
      proof (rule impI)
        assume A1:  $\text{finite } D \wedge \text{card } D \leq \text{Suc } n$ 
        from A1 have S1:  $\text{finite } D$  by auto
        from A1 have S2:  $\text{card } D \leq \text{Suc } n$  by auto
        show  $(\forall u v. \text{nat-to-set } u = D \wedge \text{nat-to-set } v = D \longrightarrow u = v)$ 
        proof (rule allI, rule allI, rule impI)
          fix u v
          assume A2:  $\text{nat-to-set } u = D \wedge \text{nat-to-set } v = D$ 

```

```

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-u: 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 - 2m
  define v1 where v1 = v - 2m
  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 + 2m
  proof -
    from u-pos have L1: 2log2 u ≤ u by (rule log2-le)
    with m-log-u have L2: 2m ≤ 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 < 2m 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 2m ≤ u1 by (rule nat-to-set-upper-bound)
      with u1-lt-2-m show False by auto
    qed
  qed

```

```

    from L1 m-notin have nat-to-set u1 = nat-to-set u - {m} by auto
    with d-u-d have nat-to-set u1 = D - {m} by auto
    with D1-def show ?thesis by auto
  qed
  have v-v1: v = v1 + 2^m
  proof -
    from v-pos have L1: 2 ^ log2 v ≤ v by (rule log2-le)
    with m-log-v have L2: 2 ^ m ≤ v by auto
    with v1-def show ?thesis by auto
  qed
  have v1-d1: nat-to-set v1 = D1
  proof -
    from m-log-v log2-gt have v < 2^(m+1) by auto
    with v-v1 have v1-lt-2-m: v1 < 2^m by auto
    with v-v1 have L1: nat-to-set v = nat-to-set v1 ∪ {m} by (simp add:
add-power)
    have m-notin: m ∉ nat-to-set v1
    proof (rule ccontr)
      assume ¬ m ∉ nat-to-set v1 then have m ∈ nat-to-set v1 by auto
      then have 2^m ≤ v1 by (rule nat-to-set-upper-bound)
      with v1-lt-2-m show False by auto
    qed
    from 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
  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 = ($\lambda D. \text{sum } (\lambda x. 2 \wedge x) D$)

lemma *two-power-sum*: $\text{sum } (\lambda x. (2::\text{nat}) \wedge x) \{i. i < \text{Suc } m\} = (2 \wedge \text{Suc } m) - 1$

proof (*induct m*)

show $\text{sum } (\lambda x. (2::\text{nat}) \wedge x) \{i. i < \text{Suc } 0\} = (2 \wedge \text{Suc } 0) - 1$ **by** *auto*

next

fix *n*

assume *A*: $\text{sum } (\lambda x. (2::\text{nat}) \wedge x) \{i. i < \text{Suc } n\} = (2 \wedge \text{Suc } n) - 1$

show $\text{sum } (\lambda x. (2::\text{nat}) \wedge x) \{i. i < \text{Suc } (\text{Suc } n)\} = (2 \wedge \text{Suc } (\text{Suc } n)) - 1$

proof –

let *?f* = $\lambda x. (2::\text{nat}) \wedge x$

have *S1*: $\{i. i < \text{Suc } (\text{Suc } n)\} = \{i. i \leq \text{Suc } n\}$ **by** *auto*

have *S2*: $\{i. i \leq \text{Suc } n\} = \{i. i < \text{Suc } n\} \cup \{\text{Suc } n\}$ **by** *auto*

from *S1 S2* **have** *S3*: $\{i. i < \text{Suc } (\text{Suc } n)\} = \{i. i < \text{Suc } n\} \cup \{\text{Suc } n\}$ **by**

auto

have *S4*: $\{i. i < \text{Suc } n\} = (\lambda x. x) \text{ ‘ } \{i. i < \text{Suc } n\}$ **by** *auto*

then have *S5*: *finite* $\{i. i < \text{Suc } n\}$ **by** (*rule nat-seg-image-imp-finite*)

have *S6*: $\text{Suc } n \notin \{i. i < \text{Suc } n\}$ **by** *auto*

from *S5 S6* *sum.insert* **have** *S7*: $\text{sum } ?f (\{i. i < \text{Suc } n\} \cup \{\text{Suc } n\}) = 2 \wedge \text{Suc } n + \text{sum } ?f \{i. i < \text{Suc } n\}$ **by** *auto*

from *S3* **have** $\text{sum } ?f \{i. i < \text{Suc } (\text{Suc } n)\} = \text{sum } ?f (\{i. i < \text{Suc } n\} \cup \{\text{Suc } n\})$ **by** *auto*

also from *S7* **have** $\dots = 2 \wedge \text{Suc } n + \text{sum } ?f \{i. i < \text{Suc } n\}$ **by** *auto*

also from *A* **have** $\dots = 2 \wedge \text{Suc } n + ((2::\text{nat}) \wedge \text{Suc } n) - (1::\text{nat})$ **by** *auto*

also have $\dots = (2 \wedge \text{Suc } (\text{Suc } n)) - 1$ **by** *auto*

finally show *?thesis* **by** *auto*

qed

qed

lemma *finite-interval*: *finite* $\{i. (i::\text{nat}) < m\}$

proof –

have $\{i. i < m\} = (\lambda x. x) \text{ ‘ } \{i. i < m\}$ **by** *auto*

then show *?thesis* **by** (*rule nat-seg-image-imp-finite*)

qed

lemma *set-to-nat-at-empty*: *set-to-nat* $\{\} = 0$ **by** (*unfold set-to-nat-def, rule sum.empty*)

lemma *set-to-nat-of-interval*: *set-to-nat* $\{i. (i::\text{nat}) < m\} = 2 \wedge m - 1$

proof (*induct m*)

show *set-to-nat* $\{i. i < 0\} = 2 \wedge 0 - 1$

proof –

have *S1*: $\{i. (i::\text{nat}) < 0\} = \{\}$ **by** *auto*

with *set-to-nat-at-empty* **have** *set-to-nat* $\{i. i < 0\} = 0$ **by** *auto*

thus *?thesis* **by** *auto*

qed

next

fix n **show** $set\text{-}to\text{-}nat \{i. i < Suc\ n\} = 2 \wedge Suc\ n - 1$ **by** (*unfold set-to-nat-def, rule two-power-sum*)

qed

lemma *set-to-nat-mono*: $\llbracket finite\ B; A \subseteq B \rrbracket \implies set\text{-}to\text{-}nat\ A \leq set\text{-}to\text{-}nat\ B$

proof –

assume $b\text{-}finite: finite\ B$

assume $a\text{-}le\text{-}b: A \subseteq B$

let $?f = \lambda (x::nat). (2::nat) \wedge x$

have $S1: set\text{-}to\text{-}nat\ A = sum\ ?f\ A$ **by** (*simp add: set-to-nat-def*)

have $S2: set\text{-}to\text{-}nat\ B = sum\ ?f\ B$ **by** (*simp add: set-to-nat-def*)

have $S3: \bigwedge x. x \in B - A \implies 0 \leq ?f\ x$ **by** *auto*

from $b\text{-}finite\ a\text{-}le\text{-}b\ S3$ **have** $sum\ ?f\ A \leq sum\ ?f\ B$ **by** (*rule sum-mono2*)

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

qed

theorem *nat-to-set-srj*: $finite\ (D::nat\ set) \implies nat\text{-}to\text{-}set\ (set\text{-}to\text{-}nat\ D) = D$

proof –

assume $A: finite\ D$

let $?P = \lambda (n::nat). (\forall (D::nat\ set). finite\ D \wedge card\ D = n \longrightarrow nat\text{-}to\text{-}set\ (set\text{-}to\text{-}nat\ D) = D)$

have $P\text{-}at\text{-}0: ?P\ 0$

proof (*rule allI*)

fix D

show $finite\ D \wedge card\ D = 0 \longrightarrow nat\text{-}to\text{-}set\ (set\text{-}to\text{-}nat\ D) = D$

proof

assume $A1: finite\ D \wedge card\ D = 0$

from $A1$ **have** $S1: finite\ D$ **by** *auto*

from $A1$ **have** $S2: card\ D = 0$ **by** *auto*

from $S1\ S2$ **have** $S3: D = \{\}$ **by** *auto*

with *set-to-nat-def* **have** $set\text{-}to\text{-}nat\ D = sum\ (\lambda x. 2 \wedge x)\ D$ **by** *simp*

with $S3\ sum.empty$ **have** $set\text{-}to\text{-}nat\ D = 0$ **by** *auto*

with *zero-is-empty* $S3$ **show** $nat\text{-}to\text{-}set\ (set\text{-}to\text{-}nat\ D) = D$ **by** *auto*

qed

qed

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

proof – **fix** n

assume $A\text{-}n: ?P\ n$

show $?P\ (Suc\ n)$

proof

fix D **show** $finite\ D \wedge card\ D = Suc\ n \longrightarrow nat\text{-}to\text{-}set\ (set\text{-}to\text{-}nat\ D) = D$

proof

assume $A1: finite\ D \wedge card\ D = Suc\ n$

from $A1$ **have** $S1: finite\ D$ **by** *auto*

from $A1$ **have** $S2: card\ D = Suc\ n$ **by** *auto*

define m **where** $m = Max\ D$

from $S2$ **have** $D\text{-}ne: D \neq \{\}$ **by** *auto*

with $S1\ m\text{-}def$ **have** $m\text{-}in: m \in D$ **by** *auto*

```

define D1 where  $D1 = D - \{m\}$ 
from S1 D1-def have d1-finite: finite D1 by auto
  from D1-def m-in S1 have  $\text{card } D1 = \text{card } D - 1$  by (simp add:
card-Diff-singleton)
  with S2 have card-d1:  $\text{card } D1 = n$  by auto
  from d1-finite card-d1 have  $\text{finite } D1 \wedge \text{card } D1 = n$  by auto
  with A-n have S3:  $\text{nat-to-set } (\text{set-to-nat } D1) = D1$  by auto
  define u where  $u = \text{set-to-nat } D$ 
  define u1 where  $u1 = \text{set-to-nat } D1$ 
  from S1 m-in have  $\text{sum } (\lambda (x::\text{nat}). (2::\text{nat}) ^ x) D = 2 ^ m + \text{sum } (\lambda x. 2 ^ x) (D - \{m\})$ 
    by (rule sum.remove)
  with set-to-nat-def have  $\text{set-to-nat } D = 2 ^ m + \text{set-to-nat } (D - \{m\})$  by
auto
  with u-def u1-def D1-def have  $u - u1: u = u1 + 2 ^ m$  by auto
  from S3 u1-def have d1-u1:  $\text{nat-to-set } u1 = D1$  by auto
  have u1-lt:  $u1 < 2 ^ m$ 
  proof -
    have L1:  $D1 \subseteq \{i. i < m\}$ 
    proof fix x
      assume A1:  $x \in D1$ 
      show  $x \in \{i. i < m\}$ 
      proof
        from A1 D1-def have L1-1:  $x \in D$  by auto
        from S1 D-ne L1-1 m-def have L1-2:  $x \leq m$  by auto
        with A1 L1-1 D1-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  $\text{set-to-nat } D1 \leq \text{set-to-nat } \{i. i < m\}$  by (rule set-to-nat-mono)
    with u1-def have  $u1 \leq \text{set-to-nat } \{i. i < m\}$  by auto
    with set-to-nat-of-interval have L3:  $u1 \leq 2 ^ m - 1$  by auto
    have  $0 < (2::\text{nat}) ^ m$  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  $\text{nat-to-set } (\text{set-to-nat } D) = \text{nat-to-set } u$  by auto
  also from u-u1 have  $\dots = \text{nat-to-set } (u1 + 2 ^ m)$  by auto
  also from u1-lt have  $\dots = \text{nat-to-set } u1 \cup \{m\}$  by (rule add-power)
  also from d1-u1 have  $\dots = D1 \cup \{m\}$  by auto
  also from D1-def m-in have  $\dots = D$  by auto
  finally show  $\text{nat-to-set } (\text{set-to-nat } D) = D$  by auto
  qed
qed
qed
from P-at-0 P-at-Suc have main:  $\bigwedge n. ?P n$  by (rule nat.induct)
from A main show ?thesis by auto

```

qed

theorem *nat-to-set-srj1*: $\text{finite } (D::\text{nat set}) \implies \exists u. \text{nat-to-set } u = D$

proof –

assume A : *finite* D

show $\exists u. \text{nat-to-set } u = D$

proof

from A **show** $\text{nat-to-set } (\text{set-to-nat } D) = D$ **by** (*rule nat-to-set-srj*)

qed

qed

lemma *sum-of-pr-is-pr*: $g \in \text{PrimRec1} \implies (\lambda n. \text{sum } g \{i. i < n\}) \in \text{PrimRec1}$

proof –

assume *g-is-pr*: $g \in \text{PrimRec1}$

define f **where** $f\ n = \text{sum } g \{i. i < n\}$ **for** n

from *f-def* **have** *f-at-0*: $f\ 0 = 0$ **by** *auto*

define h **where** $h\ a\ b = g\ a + b$ **for** $a\ b$

from *g-is-pr* **have** *h-is-pr*: $h \in \text{PrimRec2}$ **unfolding** *h-def* **by** *prec*

have *f-at-Suc*: $\forall y. f\ (\text{Suc } y) = h\ y\ (f\ y)$

proof

fix y **show** $f\ (\text{Suc } y) = h\ y\ (f\ y)$

proof –

from *f-def* **have** $S1$: $f\ (\text{Suc } y) = \text{sum } g \{i. i < \text{Suc } y\}$ **by** *auto*

have $S2$: $\{i. i < \text{Suc } y\} = \{i. i < y\} \cup \{y\}$ **by** *auto*

have $S3$: *finite* $\{i. i < y\}$ **by** (*rule finite-interval*)

have $S4$: $y \notin \{i. i < y\}$ **by** *auto*

from $S1\ S2$ **have** $f\ (\text{Suc } y) = \text{sum } g (\{i. (i::\text{nat}) < y\} \cup \{y\})$ **by** *auto*

also from $S3\ S4$ *sum.insert* **have** $\dots = g\ y + \text{sum } g \{i. i < y\}$ **by** *auto*

also from *f-def* **have** $\dots = g\ y + f\ y$ **by** *auto*

also from *h-def* **have** $\dots = h\ y\ (f\ y)$ **by** *auto*

finally show *?thesis* **by** *auto*

qed

qed

from *h-is-pr f-at-0 f-at-Suc* **have** *f-is-pr*: $f \in \text{PrimRec1}$ **by** (*rule pr-rec1-scheme*)

with *f-def* [*abs-def*] **show** *?thesis* **by** *auto*

qed

lemma *sum-of-pr-is-pr2*: $p \in \text{PrimRec2} \implies (\lambda n\ m. \text{sum } (\lambda x. p\ x\ m) \{i. i < n\}) \in \text{PrimRec2}$

proof –

assume *p-is-pr*: $p \in \text{PrimRec2}$

define f **where** $f\ n\ m = \text{sum } (\lambda x. p\ x\ m) \{i. i < n\}$ **for** $n\ m$

define $g :: \text{nat} \Rightarrow \text{nat}$ **where** $g\ x = 0$ **for** x

have *g-is-pr*: $g \in \text{PrimRec1}$ **by** (*unfold g-def, rule const-is-pr* [**where** *?n=0*])

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** $a\ b\ c$

```

from p-is-pr have h-is-pr:  $h \in \text{PrimRec3}$  unfolding h-def by prec
have f-at-Suc:  $\forall x y. f (\text{Suc } y) x = h y (f y x) x$ 
proof (rule allI, rule allI)
  fix  $x y$  show  $f (\text{Suc } y) x = h y (f y x) x$ 
  proof –
    from f-def have S1:  $f (\text{Suc } y) x = \text{sum } (\lambda z. p z x) \{i. i < \text{Suc } y\}$  by auto
    have S2:  $\{i. i < \text{Suc } y\} = \{i. i < y\} \cup \{y\}$  by auto
    have S3: finite  $\{i. i < y\}$  by (rule finite-interval)
    have S4:  $y \notin \{i. i < y\}$  by auto
    define g1 where  $g1 z = p z x$  for  $z$ 
    from S1 S2 g1-def have  $f (\text{Suc } y) x = \text{sum } g1 (\{i. (i::\text{nat}) < y\} \cup \{y\})$  by
auto
    also from S3 S4 sum.insert have  $\dots = g1 y + \text{sum } g1 \{i. i < y\}$  by auto
    also from f-def g1-def have  $\dots = g1 y + f y x$  by auto
    also from h-def g1-def have  $\dots = h y (f y x) x$  by auto
    finally show ?thesis by auto
  qed
qed
from g-is-pr h-is-pr f-at-0 f-at-Suc have f-is-pr:  $f \in \text{PrimRec2}$  by (rule pr-rec-scheme)
with f-def [abs-def] show ?thesis by auto
qed

```

```

lemma sum-is-pr:  $g \in \text{PrimRec1} \implies (\lambda u. \text{sum } g (\text{nat-to-set } u)) \in \text{PrimRec1}$ 
proof –
  assume g-is-pr:  $g \in \text{PrimRec1}$ 
  define g1 where  $g1 x u = (\text{if } (c\text{-in } x u = 1) \text{ then } (g x) \text{ else } 0)$  for  $x u$ 
  have g1-is-pr:  $g1 \in \text{PrimRec2}$ 
  proof (unfold g1-def, rule if-eq-is-pr2)
    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 [where ?n=1])
  next
    from g-is-pr show  $(\lambda x y. g x) \in \text{PrimRec2}$  by prec
  next
    show  $(\lambda x y. 0) \in \text{PrimRec2}$  by (rule const-is-pr-2 [where ?n=0])
  qed
  define f where  $f u = \text{sum } (\lambda x. g1 x u) \{i. (i::\text{nat}) < u\}$  for  $u$ 
  define f1 where  $f1 u v = \text{sum } (\lambda x. g1 x v) \{i. (i::\text{nat}) < u\}$  for  $u v$ 
from g1-is-pr have  $(\lambda (u::\text{nat}) v. \text{sum } (\lambda x. g1 x v) \{i. (i::\text{nat}) < u\}) \in \text{PrimRec2}$ 
by (rule sum-of-pr-is-pr2)
  with f1-def [abs-def] have f1-is-pr:  $f1 \in \text{PrimRec2}$  by auto
from f-def f1-def have f-f1:  $f = (\lambda u. f1 u u)$  by auto
from f1-is-pr have  $(\lambda u. f1 u u) \in \text{PrimRec1}$  by prec
with f-f1 have f-is-pr:  $f \in \text{PrimRec1}$  by auto
have f-is-result:  $f = (\lambda u. \text{sum } g (\text{nat-to-set } u))$ 
proof
  fix  $u$  show  $f u = \text{sum } g (\text{nat-to-set } u)$ 
  proof –
    define U where  $U = \{i. i < u\}$ 

```

```

define  $A$  where  $A = \{x \in U. c\text{-in } x \ u = 1\}$ 
define  $B$  where  $B = \{x \in U. c\text{-in } x \ u \neq 1\}$ 
have  $U\text{-finite}: \text{finite } U$  by (unfold U-def, rule finite-interval)
from  $A\text{-def } U\text{-finite}$  have  $A\text{-finite}: \text{finite } A$  by auto
from  $B\text{-def } U\text{-finite}$  have  $B\text{-finite}: \text{finite } B$  by auto
from  $U\text{-def } A\text{-def } B\text{-def}$  have  $U\text{-A-B}: U = A \cup B$  by auto
from  $U\text{-def } A\text{-def } B\text{-def}$  have  $A\text{-B}: A \cap B = \{\}$  by auto
from  $B\text{-def } g1\text{-def}$  have  $B\text{-z}: \text{sum } (\lambda x. g1 \ x \ u) \ B = 0$  by auto
have  $u\text{-in-}U: \text{nat-to-set } u \subseteq U$  by (unfold U-def, rule nat-to-set-upper-bound2)
from  $u\text{-in-}U \ x\text{-in-}u\text{-eq } A\text{-def}$  have  $A\text{-u}: A = \text{nat-to-set } u$  by auto
from  $A\text{-u } x\text{-in-}u\text{-eq } g1\text{-def}$  have  $A\text{-res}: \text{sum } (\lambda x. g1 \ x \ u) \ A = \text{sum } g \ (\text{nat-to-set } u)$  by auto
qed
from  $f\text{-def}$  have  $f \ u = \text{sum } (\lambda x. g1 \ x \ u) \ \{i. (i::\text{nat}) < u\}$  by auto
also from  $U\text{-def}$  have  $\dots = \text{sum } (\lambda x. g1 \ x \ u) \ U$  by auto
also from  $U\text{-A-B}$  have  $\dots = \text{sum } (\lambda x. g1 \ x \ u) \ (A \cup B)$  by auto
also from  $A\text{-finite } B\text{-finite } A\text{-B}$  have  $\dots = \text{sum } (\lambda x. g1 \ x \ u) \ A + \text{sum } (\lambda x. g1 \ x \ u) \ B$  by (rule sum.union-disjoint)
also from  $B\text{-z}$  have  $\dots = \text{sum } (\lambda x. g1 \ x \ u) \ A$  by auto
also from  $A\text{-res}$  have  $\dots = \text{sum } g \ (\text{nat-to-set } u)$  by auto
finally show ?thesis by auto
qed
with  $f\text{-is-pr}$  show ?thesis by auto
qed

```

definition

```

 $c\text{-card} :: \text{nat} \Rightarrow \text{nat}$  where
 $c\text{-card} = (\lambda u. \text{card } (\text{nat-to-set } u))$ 

```

theorem $c\text{-card-is-pr}: c\text{-card} \in \text{PrimRec1}$

proof –

```

define  $g :: \text{nat} \Rightarrow \text{nat}$  where  $g \ x = 1$  for  $x$ 
have  $g\text{-is-pr}: g \in \text{PrimRec1}$  by (unfold g-def, rule const-is-pr)
have  $c\text{-card} = (\lambda u. \text{sum } g \ (\text{nat-to-set } u))$ 
proof
fix  $u$  show  $c\text{-card } u = \text{sum } g \ (\text{nat-to-set } u)$  by (unfold c-card-def, unfold g-def, rule card-eq-sum)
qed
moreover from  $g\text{-is-pr}$  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

```

 $c\text{-insert} :: \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat}$  where
 $c\text{-insert} = (\lambda x \ u. \text{if } c\text{-in } x \ u = 1 \ \text{then } u \ \text{else } u + 2^{\widehat{x}})$ 

```

lemma $c\text{-insert-is-pr}: c\text{-insert} \in \text{PrimRec2}$

proof (*unfold c-insert-def, rule if-eq-is-pr2*)

```

  show  $c\text{-in} \in \text{PrimRec2}$  by (rule  $c\text{-in-is-pr}$ )
next
  show  $(\lambda x y. 1) \in \text{PrimRec2}$  by (rule  $\text{const-is-pr-2}$ )
next
  show  $(\lambda x y. y) \in \text{PrimRec2}$  by (rule  $\text{pr-id2-2}$ )
next
  from  $\text{power-is-pr}$  show  $(\lambda x y. y + 2^x) \in \text{PrimRec2}$  by  $\text{prec}$ 
qed

```

lemma $[\text{simp}]$: $\text{set-to-nat} (\text{nat-to-set } u) = u$

proof –

```

  define  $D$  where  $D = \text{nat-to-set } u$ 
  from  $D\text{-def}$   $\text{nat-to-set-is-finite}$  have  $D\text{-finite}$ :  $\text{finite } D$  by  $\text{auto}$ 
  then have  $\text{nat-to-set} (\text{set-to-nat } D) = D$  by (rule  $\text{nat-to-set-srj}$ )
  with  $D\text{-def}$  have  $\text{nat-to-set} (\text{set-to-nat } D) = \text{nat-to-set } u$  by  $\text{auto}$ 
  then have  $\text{set-to-nat } D = u$  by (rule  $\text{nat-to-set-inj}$ )
  with  $D\text{-def}$  show  $?thesis$  by  $\text{auto}$ 
qed

```

lemma insert-lemma : $x \notin \text{nat-to-set } u \implies \text{set-to-nat} (\text{nat-to-set } u \cup \{x\}) = u + 2^x$

proof –

```

  assume  $A$ :  $x \notin \text{nat-to-set } u$ 
  define  $D$  where  $D = \text{nat-to-set } u$ 
  from  $A$   $D\text{-def}$  have  $S1$ :  $x \notin D$  by  $\text{auto}$ 
  have  $\text{finite} (\text{nat-to-set } u)$  by (rule  $\text{nat-to-set-is-finite}$ )
  with  $D\text{-def}$  have  $D\text{-finite}$ :  $\text{finite } D$  by  $\text{auto}$ 
  let  $?f = \lambda (x::\text{nat}). (2::\text{nat})^x$ 
  from  $\text{set-to-nat-def}$  have  $\text{set-to-nat} (D \cup \{x\}) = \text{sum } ?f (D \cup \{x\})$  by  $\text{auto}$ 
  also from  $D\text{-finite}$   $S1$  have  $\dots = ?f x + \text{sum } ?f D$  by  $\text{simp}$ 
  also from  $\text{set-to-nat-def}$  have  $\dots = 2^x + \text{set-to-nat } D$  by  $\text{auto}$ 
  finally have  $\text{set-to-nat} (D \cup \{x\}) = \text{set-to-nat } D + 2^x$  by  $\text{auto}$ 
  with  $D\text{-def}$  show  $?thesis$  by  $\text{auto}$ 
qed

```

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

proof (rule ext , rule ext)

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

proof (cases)

```

  assume  $A$ :  $x \in \text{nat-to-set } u$ 
  then have  $\text{nat-to-set } u \cup \{x\} = \text{nat-to-set } u$  by  $\text{auto}$ 
  then have  $S1$ :  $\text{set-to-nat} (\text{nat-to-set } u \cup \{x\}) = u$  by  $\text{auto}$ 
  from  $A$  have  $c\text{-in } x u = 1$  by ( $\text{simp add: } x\text{-in-}u\text{-eq}$ )
  then have  $c\text{-insert } x u = u$  by ( $\text{unfold } c\text{-insert-def, simp}$ )
  with  $S1$  show  $?thesis$  by  $\text{auto}$ 

```

next

```

  assume  $A$ :  $x \notin \text{nat-to-set } u$ 
  then have  $S1$ :  $c\text{-in } x u \neq 1$  by ( $\text{simp add: } x\text{-in-}u\text{-eq}$ )
  then have  $S2$ :  $c\text{-insert } x u = u + 2^x$  by ( $\text{unfold } c\text{-insert-def, simp}$ )

```

from A **have** $set\text{-}to\text{-}nat\ (nat\text{-}to\text{-}set\ u \cup \{x\}) = u + 2^{\wedge}x$ **by** (*rule insert-lemma*)
with $S2$ **show** *?thesis* **by** *auto*
qed
qed

definition

$c\text{-}remove :: nat \Rightarrow nat \Rightarrow nat$ **where**
 $c\text{-}remove = (\lambda\ x\ u.\ if\ c\text{-}in\ x\ u = 0\ then\ u\ else\ u - 2^{\wedge}x)$

lemma $c\text{-}remove\text{-}is\text{-}pr$: $c\text{-}remove \in PrimRec2$

proof (*unfold c-remove-def, rule if-eq-is-pr2*)

show $c\text{-}in \in PrimRec2$ **by** (*rule c-in-is-pr*)

next

show $(\lambda x\ y.\ 0) \in PrimRec2$ **by** (*rule const-is-pr-2*)

next

show $(\lambda x\ y.\ y) \in PrimRec2$ **by** (*rule pr-id2-2*)

next

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

qed

lemma $remove\text{-}lemma$: $x \in nat\text{-}to\text{-}set\ u \implies set\text{-}to\text{-}nat\ (nat\text{-}to\text{-}set\ u - \{x\}) = u - 2^{\wedge}x$

proof –

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

define D **where** $D = nat\text{-}to\text{-}set\ u - \{x\}$

from A $D\text{-}def$ **have** $S1$: $x \notin D$ **by** *auto*

have $finite\ (nat\text{-}to\text{-}set\ u)$ **by** (*rule nat-to-set-is-finite*)

with $D\text{-}def$ **have** $D\text{-}finite$: $finite\ D$ **by** *auto*

let $?f = \lambda\ (x::nat).\ (2::nat)^{\wedge}x$

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

also from $D\text{-}finite$ $S1$ **have** $\dots = ?f\ x + sum\ ?f\ D$ **by** *simp*

also from $set\text{-}to\text{-}nat\text{-}def$ **have** $\dots = 2^{\wedge}x + set\text{-}to\text{-}nat\ D$ **by** *auto*

finally have $S2$: $set\text{-}to\text{-}nat\ (D \cup \{x\}) = set\text{-}to\text{-}nat\ D + 2^{\wedge}x$ **by** *auto*

from A $D\text{-}def$ **have** $D \cup \{x\} = nat\text{-}to\text{-}set\ u$ **by** *auto*

with $S2$ **have** $S3$: $u = set\text{-}to\text{-}nat\ D + 2^{\wedge}x$ **by** *auto*

from A **have** $S4$: $2^{\wedge}x \leq u$ **by** (*rule nat-to-set-upper-bound*)

with $S3$ $D\text{-}def$ **show** *?thesis* **by** *auto*

qed

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

proof (*rule ext, rule ext*)

fix $x\ u$ **show** $c\text{-}remove\ x\ u = set\text{-}to\text{-}nat\ (nat\text{-}to\text{-}set\ u - \{x\})$

proof (*cases*)

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

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

then have $S2$: $c\text{-}remove\ x\ u = u - 2^{\wedge}x$ **by** (*simp add: c-remove-def*)

from A **have** $set\text{-}to\text{-}nat\ (nat\text{-}to\text{-}set\ u - \{x\}) = u - 2^{\wedge}x$ **by** (*rule remove-lemma*)

with $S2$ **show** *?thesis* **by** *auto*

```

next
  assume A:  $x \notin \text{nat-to-set } u$ 
  then have S1:  $c\text{-in } x \ u \neq 1$  by (simp add: x-in-u-eq)
  then have S2:  $c\text{-remove } x \ u = u$  by (simp add: c-remove-def c-in-def)
  from A have  $\text{nat-to-set } u - \{x\} = \text{nat-to-set } u$  by auto
  with S2 show ?thesis by auto
qed
qed

definition
  c-union ::  $\text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat}$  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:  $c\text{-union} \in \text{PrimRec2}$ 
proof -
  define f where  $f \ y \ x = \text{set-to-nat } ((\text{nat-to-set } (c\text{-fst } x)) \cup \{z \in \text{nat-to-set } (c\text{-snd } x). \ z < y\})$ 
  for  $y \ x$ 
  have f-is-pr:  $f \in \text{PrimRec2}$ 
  proof -
    define g where  $g = c\text{-fst}$ 
    from c-fst-is-pr g-def have g-is-pr:  $g \in \text{PrimRec1}$  by auto
    define h where  $h \ a \ b \ c = (\text{if } c\text{-in } a \ (c\text{-snd } c) = 1 \ \text{then } c\text{-insert } a \ b \ \text{else } b)$  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:  $\forall \ x. f \ 0 \ x = g \ x$ 
    proof
      fix  $x$  show  $f \ 0 \ x = g \ x$  by (unfold f-def, unfold g-def, simp)
    qed
    have f-at-Suc:  $\forall \ x \ y. f \ (\text{Suc } y) \ x = h \ y \ (f \ y \ x) \ x$ 
    proof (rule allI, rule allI)
      fix  $x \ y$  show  $f \ (\text{Suc } y) \ x = h \ y \ (f \ y \ x) \ x$ 
      proof (cases)
        assume A:  $c\text{-in } y \ (c\text{-snd } x) = 1$ 
        then have S1:  $y \in (\text{nat-to-set } (c\text{-snd } x))$  by (simp add: x-in-u-eq)
        from A h-def have S2:  $h \ y \ (f \ y \ x) \ x = c\text{-insert } y \ (f \ y \ x)$  by auto
        from S1 have S3:  $\{z \in \text{nat-to-set } (c\text{-snd } x). \ z < \text{Suc } y\} = \{z \in \text{nat-to-set } (c\text{-snd } x). \ z < y\} \cup \{y\}$  by auto
        from nat-to-set-is-finite have S4:  $\text{finite } ((\text{nat-to-set } (c\text{-fst } x)) \cup \{z \in \text{nat-to-set } (c\text{-snd } x). \ z < y\})$  by auto
        with nat-to-set-srj f-def have S5:  $\text{nat-to-set } (f \ y \ x) = (\text{nat-to-set } (c\text{-fst } x)) \cup \{z \in \text{nat-to-set } (c\text{-snd } x). \ z < y\}$  by auto
        from f-def have S6:  $f \ (\text{Suc } y) \ x = \text{set-to-nat } ((\text{nat-to-set } (c\text{-fst } x)) \cup \{z \in \text{nat-to-set } (c\text{-snd } x). \ z < \text{Suc } y\})$  by simp
        also from S3 have ... =  $\text{set-to-nat } (((\text{nat-to-set } (c\text{-fst } x)) \cup \{z \in \text{nat-to-set } (c\text{-snd } x). \ z < y\}) \cup \{y\})$  by auto
        also from S5 have ... =  $\text{set-to-nat } (\text{nat-to-set } (f \ y \ x) \cup \{y\})$  by auto
        also have ... =  $c\text{-insert } y \ (f \ y \ x)$  by (simp add: c-insert-df)
      qed
    qed
  qed

```

finally show *?thesis* **by** (*simp add: S2*)
next
assume $A: \neg c\text{-in } y (c\text{-snd } x) = 1$
then have $S1: y \notin (\text{nat-to-set } (c\text{-snd } x))$ **by** (*simp add: x-in-u-eq*)
from A **h-def** **have** $S2: h y (f y x) x = f y x$ **by** *auto*
have $S3: \{z \in \text{nat-to-set } (c\text{-snd } x). z < \text{Suc } y\} = \{z \in \text{nat-to-set } (c\text{-snd } x).$
 $z < y\}$
proof –
have $\{z \in \text{nat-to-set } (c\text{-snd } x). z < \text{Suc } y\} = \{z \in \text{nat-to-set } (c\text{-snd } x). z$
 $< y\} \cup \{z \in \text{nat-to-set } (c\text{-snd } x). z = y\}$
by *auto*
with $S1$ **show** *?thesis* **by** *auto*
qed
from *nat-to-set-is-finite* **have** $S4: \text{finite } ((\text{nat-to-set } (c\text{-fst } x)) \cup \{z \in$
 $\text{nat-to-set } (c\text{-snd } x). z < y\})$ **by** *auto*
with *nat-to-set-srj f-def* **have** $S5: \text{nat-to-set } (f y x) = (\text{nat-to-set } (c\text{-fst } x))$
 $\cup \{z \in \text{nat-to-set } (c\text{-snd } x). z < y\}$ **by** *auto*
from *f-def* **have** $S6: f (\text{Suc } y) x = \text{set-to-nat } ((\text{nat-to-set } (c\text{-fst } x)) \cup \{z \in$
 $\text{nat-to-set } (c\text{-snd } x). z < \text{Suc } y\})$ **by** *simp*
also from $S3$ **have** $\dots = \text{set-to-nat } (((\text{nat-to-set } (c\text{-fst } x)) \cup \{z \in \text{nat-to-set}$
 $(c\text{-snd } x). z < y\}))$ **by** *auto*
also from $S5$ **have** $\dots = \text{set-to-nat } (\text{nat-to-set } (f y x))$ **by** *auto*
also have $\dots = f y x$ **by** *simp*
finally show *?thesis* **by** (*simp add: S2*)
qed
qed
from *g-is-pr h-is-pr f-at-0 f-at-Suc* **show** *?thesis* **by** (*rule pr-rec-scheme*)
qed
define *union* **where** $\text{union } u v = f v (c\text{-pair } u v)$ **for** $u v$
from *f-is-pr* **have** *union-is-pr*: $\text{union} \in \text{PrimRec2}$ **unfolding** *union-def* **by** *prec*
have $\bigwedge u v. \text{union } u v = \text{set-to-nat } (\text{nat-to-set } u \cup \text{nat-to-set } v)$
proof –
fix $u v$ **show** $\text{union } u v = \text{set-to-nat } (\text{nat-to-set } u \cup \text{nat-to-set } v)$
proof –
from *nat-to-set-upper-bound1* **have** $\{z \in \text{nat-to-set } v. z < v\} = \text{nat-to-set } v$
by *auto*
with *union-def f-def* **show** *?thesis* **by** *auto*
qed
qed
then have $\text{union} = (\lambda u v. \text{set-to-nat } (\text{nat-to-set } u \cup \text{nat-to-set } v))$ **by** (*simp*
add: ext)
with *c-union-def* **have** *c-union* = *union* **by** *simp*
with *union-is-pr* **show** *?thesis* **by** *simp*
qed
definition
c-diff $:: \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat}$ **where**
c-diff = $(\lambda u v. \text{set-to-nat } (\text{nat-to-set } u - \text{nat-to-set } v))$

```

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

```

from A *h-def* **have** $S2: h\ y\ (f\ y\ x)\ x = f\ y\ x$ **by** *auto*
have $S3: \{z \in \text{nat-to-set}\ (c\text{-snd}\ x). z < \text{Suc}\ y\} = \{z \in \text{nat-to-set}\ (c\text{-snd}\ x). z < y\}$
proof –
have $\{z \in \text{nat-to-set}\ (c\text{-snd}\ x). z < \text{Suc}\ y\} = \{z \in \text{nat-to-set}\ (c\text{-snd}\ x). z < y\} \cup \{z \in \text{nat-to-set}\ (c\text{-snd}\ x). z = y\}$
by *auto*
with $S1$ **show** *?thesis* **by** *auto*
qed
from *nat-to-set-is-finite* **have** $S4: \text{finite}\ ((\text{nat-to-set}\ (c\text{-fst}\ x)) - \{z \in \text{nat-to-set}\ (c\text{-snd}\ x). z < y\})$ **by** *auto*
with *nat-to-set-srj f-def* **have** $S5: \text{nat-to-set}\ (f\ y\ x) = (\text{nat-to-set}\ (c\text{-fst}\ x)) - \{z \in \text{nat-to-set}\ (c\text{-snd}\ x). z < y\}$ **by** *auto*
from *f-def* **have** $S6: f\ (\text{Suc}\ y)\ x = \text{set-to-nat}\ ((\text{nat-to-set}\ (c\text{-fst}\ x)) - \{z \in \text{nat-to-set}\ (c\text{-snd}\ x). z < \text{Suc}\ y\})$ **by** *simp*
also from $S3$ **have** $\dots = \text{set-to-nat}\ (((\text{nat-to-set}\ (c\text{-fst}\ x)) - \{z \in \text{nat-to-set}\ (c\text{-snd}\ x). z < y\}))$ **by** *auto*
also from $S5$ **have** $\dots = \text{set-to-nat}\ (\text{nat-to-set}\ (f\ y\ x))$ **by** *auto*
also have $\dots = f\ y\ x$ **by** *simp*
finally show *?thesis* **by** (*simp add: S2*)
qed
qed
from *g-is-pr h-is-pr f-at-0 f-at-Suc* **show** *?thesis* **by** (*rule pr-rec-scheme*)
qed
define *diff* **where** $\text{diff}\ u\ v = f\ v\ (c\text{-pair}\ u\ v)$ **for** $u\ v$
from *f-is-pr* **have** *diff-is-pr: diff* $\in \text{PrimRec2}$ **unfolding** *diff-def* **by** *prec*
have $\bigwedge u\ v. \text{diff}\ u\ v = \text{set-to-nat}\ (\text{nat-to-set}\ u - \text{nat-to-set}\ v)$
proof –
fix $u\ v$ **show** $\text{diff}\ u\ v = \text{set-to-nat}\ (\text{nat-to-set}\ u - \text{nat-to-set}\ v)$
proof –
from *nat-to-set-upper-bound1* **have** $\{z \in \text{nat-to-set}\ v. z < v\} = \text{nat-to-set}\ v$
by *auto*
with *diff-def f-def* **show** *?thesis* **by** *auto*
qed
qed
then have $\text{diff} = (\lambda u\ v. \text{set-to-nat}\ (\text{nat-to-set}\ u - \text{nat-to-set}\ v))$ **by** (*simp add: ext*)
with *c-diff-def* **have** *c-diff* $= \text{diff}$ **by** *simp*
with *diff-is-pr* **show** *?thesis* **by** *simp*
qed

definition

$c\text{-intersect} :: \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat}$ **where**
 $c\text{-intersect} = (\lambda u\ v. \text{set-to-nat}\ (\text{nat-to-set}\ u \cap \text{nat-to-set}\ v))$

theorem *c-intersect-is-pr: c-intersect* $\in \text{PrimRec2}$

proof –

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

```

from c-diff-is-pr c-union-is-pr have f-is-pr: f ∈ PrimRec2 unfolding f-def by
prec
have  $\bigwedge u v. f u v = c\text{-intersect } u v$ 
proof –
  fix u v show  $f u v = c\text{-intersect } u v$ 
proof –
  let ?A = nat-to-set u
  let ?B = nat-to-set v
  have A-fin: finite ?A by (rule nat-to-set-is-finite)
  have B-fin: finite ?B by (rule nat-to-set-is-finite)
  have S1: c-union u v = set-to-nat (?A ∪ ?B) by (simp add: c-union-def)
  have S2: c-diff u v = set-to-nat (?A - ?B) by (simp add: c-diff-def)
  have S3: c-diff v u = set-to-nat (?B - ?A) by (simp add: c-diff-def)
  from S2 A-fin B-fin have S4: nat-to-set (c-diff u v) = ?A - ?B by (simp
add: nat-to-set-srj)
  from S3 A-fin B-fin have S5: nat-to-set (c-diff v u) = ?B - ?A by (simp
add: nat-to-set-srj)
  from S4 S5 have S6: c-union (c-diff u v) (c-diff v u) = set-to-nat ((?A -
?B) ∪ (?B - ?A)) by (simp add: c-union-def)
  from S1 A-fin B-fin have S7: nat-to-set (c-union u v) = ?A ∪ ?B by (simp
add: nat-to-set-srj)
  from S6 A-fin B-fin have S8: nat-to-set (c-union (c-diff u v) (c-diff v u)) =
(?A - ?B) ∪ (?B - ?A) by (simp add: nat-to-set-srj)
  from S7 S8 have S9: f u v = set-to-nat ((?A ∪ ?B) - ((?A - ?B) ∪ (?B -
?A))) by (simp add: c-diff-def f-def)
  have S10: ?A ∩ ?B = (?A ∪ ?B) - ((?A - ?B) ∪ (?B - ?A)) by auto
  with S9 have S11: f u v = set-to-nat (?A ∩ ?B) by auto
  have c-intersect u v = set-to-nat (?A ∩ ?B) by (simp add: c-intersect-def)
  with S11 show ?thesis by auto
qed
qed
then have  $f = c\text{-intersect}$  by (simp add: ext)
with f-is-pr show ?thesis by auto
qed
end

```

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

```

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
  c-ls
)

```

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-conv-2-to-1 nat-to-pr

```

theorem *univ-is-not-pr*: $univ-for-pr \notin PrimRec1$

proof (*rule ccontr*)

assume $\neg univ-for-pr \in PrimRec1$ **then have** $A1: univ-for-pr \in PrimRec1$ **by** *simp*

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

let $?n0 = index-of-pr\ ?f$

from $A1$ **have** $S1: ?f \in PrimRec1$ **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-conv-2-to-1-def*)

qed

definition

$c\text{-is-sub-fun} :: \text{nat} \Rightarrow (\text{nat} \Rightarrow \text{nat}) \Rightarrow \text{bool}$ **where**
 $c\text{-is-sub-fun } ls\ f \iff (\forall x. c\text{-assoc-have-key } ls\ x = 0 \longrightarrow c\text{-assoc-value } ls\ x = f\ x)$

lemma $c\text{-is-sub-fun-lm-1}$: $\llbracket c\text{-is-sub-fun } ls\ f; c\text{-assoc-have-key } ls\ x = 0 \rrbracket \implies c\text{-assoc-value } ls\ x = f\ x$
apply(*unfold c-is-sub-fun-def*)
apply(*auto*)
done

lemma $c\text{-is-sub-fun-lm-2}$: $c\text{-is-sub-fun } ls\ f \implies c\text{-is-sub-fun } (c\text{-cons } (c\text{-pair } x\ (f\ x))\ ls)\ f$

proof –

assume $A1$: $c\text{-is-sub-fun } ls\ f$

show *?thesis*

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

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

proof *cases*

assume $C1$: $xa = x$

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

next

assume $C2$: $\neg xa = x$

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

from $C2$ **have** $S2$: $c\text{-assoc-value } (c\text{-cons } (c\text{-pair } x\ (f\ x))\ ls)\ xa = c\text{-assoc-value } ls\ xa$ **by** (*rule c-assoc-lm-4*)

from $A2\ S1$ **have** $S3$: $c\text{-assoc-have-key } ls\ xa = 0$ **by** *simp*

from $A1\ S3$ **have** $c\text{-assoc-value } ls\ xa = f\ xa$ **by** (*rule c-is-sub-fun-lm-1*)

with $S2$ **show** *?thesis* **by** *simp*

qed

qed

qed

lemma $mod7\text{-lm}$: $(n::\text{nat})\ mod\ 7 = 0 \vee$

$(n::\text{nat})\ mod\ 7 = 1 \vee$

$(n::\text{nat})\ mod\ 7 = 2 \vee$

$(n::\text{nat})\ mod\ 7 = 3 \vee$

$(n::\text{nat})\ mod\ 7 = 4 \vee$

$(n::\text{nat})\ mod\ 7 = 5 \vee$

$(n::\text{nat})\ mod\ 7 = 6$ **by** *arith*

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

proof –

assume A : $x > 0$

show *?thesis*

```

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

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

lemma loc-lm-1:  $c\text{-fst } n \text{ mod } 7 \neq 0 \implies n > 0$ 
proof –
  assume  $A: c\text{-fst } n \text{ mod } 7 \neq 0$ 
  have  $n = 0 \implies \text{False}$ 
  proof –
    assume  $n = 0$ 

```

then have $c\text{-fst } n \bmod 7 = 0$ **by** (*simp add: c-fst-at-0*)
with A **show** *?thesis* **by** *simp*
qed
then have $\neg n = 0$ **by** *auto*
then show *?thesis* **by** *simp*
qed

lemma *loc-lm-2*: $c\text{-fst } n \bmod 7 \neq 0 \implies \text{nat-to-sch } n = (\text{let } u=(c\text{-fst } n) \bmod 7;$
 $v=c\text{-snd } n; v1=c\text{-fst } v; v2 = c\text{-snd } v; \text{sch1}=\text{nat-to-sch } v1; \text{sch2}=\text{nat-to-sch } v2 \text{ in}$
 $\text{loc-f } u \text{ sch1 sch2})$

proof –
assume $c\text{-fst } n \bmod 7 \neq 0$
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*: $c\text{-fst } n \bmod 7 = 1 \implies \text{nat-to-sch } n = \text{Base-suc}$

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

lemma *nat-to-sch-2*: $c\text{-fst } n \bmod 7 = 2 \implies \text{nat-to-sch } n = \text{Base-fst}$

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

lemma *nat-to-sch-3*: $c\text{-fst } n \bmod 7 = 3 \implies \text{nat-to-sch } n = \text{Base-snd}$

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

lemma *nat-to-sch-4*: $c\text{-fst } n \bmod 7 = 4 \implies \text{nat-to-sch } n = \text{Comp-op } (\text{nat-to-sch}$
 $(c\text{-fst } (c\text{-snd } n))) (\text{nat-to-sch } (c\text{-snd } (c\text{-snd } n)))$

proof –
assume $A1: c\text{-fst } n \bmod 7 = 4$
then have $\text{nat-to-sch } n = (\text{let } u=(c\text{-fst } n) \bmod 7; v=c\text{-snd } n; v1=c\text{-fst } v; v2 =$
 $c\text{-snd } v; \text{sch1}=\text{nat-to-sch } v1; \text{sch2}=\text{nat-to-sch } v2 \text{ in loc-f } u \text{ sch1 sch2})$ **by** (*simp*

add: loc-lm-2)

with *A1* **show** $\text{nat-to-sch } n = \text{Comp-op } (\text{nat-to-sch } (c\text{-fst } (c\text{-snd } n))) (\text{nat-to-sch } (c\text{-snd } (c\text{-snd } n)))$ **by** (*simp add: Let-def loc-f-def*)
qed

lemma *nat-to-sch-5: $c\text{-fst } n \bmod 7 = 5 \implies \text{nat-to-sch } n = \text{Pair-op } (\text{nat-to-sch } (c\text{-fst } (c\text{-snd } n))) (\text{nat-to-sch } (c\text{-snd } (c\text{-snd } n)))$*

proof –

assume *A1: $c\text{-fst } n \bmod 7 = 5$*

then have $\text{nat-to-sch } n = (\text{let } u=(c\text{-fst } n) \bmod 7; v=c\text{-snd } n; v1=c\text{-fst } v; v2 = c\text{-snd } v; \text{sch1}=\text{nat-to-sch } v1; \text{sch2}=\text{nat-to-sch } v2 \text{ in } \text{loc-f } u \text{ sch1 sch2})$ **by** (*simp add: loc-lm-2*)

with *A1* **show** $\text{nat-to-sch } n = \text{Pair-op } (\text{nat-to-sch } (c\text{-fst } (c\text{-snd } n))) (\text{nat-to-sch } (c\text{-snd } (c\text{-snd } n)))$ **by** (*simp add: Let-def loc-f-def*)

qed

lemma *nat-to-sch-6: $c\text{-fst } n \bmod 7 = 6 \implies \text{nat-to-sch } n = \text{Rec-op } (\text{nat-to-sch } (c\text{-fst } (c\text{-snd } n))) (\text{nat-to-sch } (c\text{-snd } (c\text{-snd } n)))$*

proof –

assume *A1: $c\text{-fst } n \bmod 7 = 6$*

then have $\text{nat-to-sch } n = (\text{let } u=(c\text{-fst } n) \bmod 7; v=c\text{-snd } n; v1=c\text{-fst } v; v2 = c\text{-snd } v; \text{sch1}=\text{nat-to-sch } v1; \text{sch2}=\text{nat-to-sch } v2 \text{ in } \text{loc-f } u \text{ sch1 sch2})$ **by** (*simp add: loc-lm-2*)

with *A1* **show** $\text{nat-to-sch } n = \text{Rec-op } (\text{nat-to-sch } (c\text{-fst } (c\text{-snd } n))) (\text{nat-to-sch } (c\text{-snd } (c\text{-snd } n)))$ **by** (*simp add: Let-def loc-f-def*)

qed

lemma *nat-to-pr-lm-0: $c\text{-fst } n \bmod 7 = 0 \implies \text{nat-to-pr } n \ x = 0$*

proof –

assume *A: $c\text{-fst } n \bmod 7 = 0$*

have *S1: $\text{nat-to-pr } n \ x = \text{sch-to-pr } (\text{nat-to-sch } n) \ x$* **by** (*simp add: nat-to-pr-def*)

from *A* **have** *S2: $\text{nat-to-sch } n = \text{Base-zero}$* **by** (*rule nat-to-sch-0*)

from *S1 S2* **show** *?thesis* **by** *simp*

qed

lemma *nat-to-pr-lm-1: $c\text{-fst } n \bmod 7 = 1 \implies \text{nat-to-pr } n \ x = \text{Suc } x$*

proof –

assume *A: $c\text{-fst } n \bmod 7 = 1$*

have *S1: $\text{nat-to-pr } n \ x = \text{sch-to-pr } (\text{nat-to-sch } n) \ x$* **by** (*simp add: nat-to-pr-def*)

from *A* **have** *S2: $\text{nat-to-sch } n = \text{Base-suc}$* **by** (*rule nat-to-sch-1*)

from *S1 S2* **show** *?thesis* **by** *simp*

qed

lemma *nat-to-pr-lm-2: $c\text{-fst } n \bmod 7 = 2 \implies \text{nat-to-pr } n \ x = c\text{-fst } x$*

proof –

assume *A: $c\text{-fst } n \bmod 7 = 2$*

have *S1: $\text{nat-to-pr } n \ x = \text{sch-to-pr } (\text{nat-to-sch } n) \ x$* **by** (*simp add: nat-to-pr-def*)

from *A* **have** *S2: $\text{nat-to-sch } n = \text{Base-fst}$* **by** (*rule nat-to-sch-2*)

from *S1 S2* **show** *?thesis* **by** *simp*

qed

lemma *nat-to-pr-lm-3*: $c\text{-fst } n \text{ mod } 7 = 3 \implies \text{nat-to-pr } n \ x = c\text{-snd } x$

proof –

assume *A*: $c\text{-fst } n \text{ mod } 7 = 3$

have *S1*: $\text{nat-to-pr } n \ x = \text{sch-to-pr } (\text{nat-to-sch } n) \ x$ **by** (*simp add: nat-to-pr-def*)

from *A* **have** *S2*: $\text{nat-to-sch } n = \text{Base-snd}$ **by** (*rule nat-to-sch-3*)

from *S1 S2* **show** *?thesis* **by** *simp*

qed

lemma *nat-to-pr-lm-4*: $c\text{-fst } n \text{ mod } 7 = 4 \implies \text{nat-to-pr } n \ x = (\text{nat-to-pr } (c\text{-fst } (c\text{-snd } n)) (\text{nat-to-pr } (c\text{-snd } (c\text{-snd } n)) \ x))$

proof –

assume *A*: $c\text{-fst } n \text{ mod } 7 = 4$

have *S1*: $\text{nat-to-pr } n \ x = \text{sch-to-pr } (\text{nat-to-sch } n) \ x$ **by** (*simp add: nat-to-pr-def*)

from *A* **have** *S2*: $\text{nat-to-sch } n = \text{Comp-op } (\text{nat-to-sch } (c\text{-fst } (c\text{-snd } n))) (\text{nat-to-sch } (c\text{-snd } (c\text{-snd } n)))$ **by** (*rule nat-to-sch-4*)

from *S1 S2* **have** *S3*: $\text{nat-to-pr } n \ x = \text{sch-to-pr } (\text{Comp-op } (\text{nat-to-sch } (c\text{-fst } (c\text{-snd } n))) (\text{nat-to-sch } (c\text{-snd } (c\text{-snd } n)))) \ x$ **by** *simp*

from *S3* **have** *S4*: $\text{nat-to-pr } n \ x = (\text{sch-to-pr } (\text{nat-to-sch } (c\text{-fst } (c\text{-snd } n)))) (\text{sch-to-pr } (\text{nat-to-sch } (c\text{-snd } (c\text{-snd } n)))) \ x$ **by** *simp*

from *S4* **show** *?thesis* **by** (*simp add: nat-to-pr-def*)

qed

lemma *nat-to-pr-lm-5*: $c\text{-fst } n \text{ mod } 7 = 5 \implies \text{nat-to-pr } n \ x = (c\text{-f-pair } (\text{nat-to-pr } (c\text{-fst } (c\text{-snd } n)) (\text{nat-to-pr } (c\text{-snd } (c\text{-snd } n)))) \ x$

proof –

assume *A*: $c\text{-fst } n \text{ mod } 7 = 5$

have *S1*: $\text{nat-to-pr } n \ x = \text{sch-to-pr } (\text{nat-to-sch } n) \ x$ **by** (*simp add: nat-to-pr-def*)

from *A* **have** *S2*: $\text{nat-to-sch } n = \text{Pair-op } (\text{nat-to-sch } (c\text{-fst } (c\text{-snd } n))) (\text{nat-to-sch } (c\text{-snd } (c\text{-snd } n)))$ **by** (*rule nat-to-sch-5*)

from *S1 S2* **have** *S3*: $\text{nat-to-pr } n \ x = \text{sch-to-pr } (\text{Pair-op } (\text{nat-to-sch } (c\text{-fst } (c\text{-snd } n))) (\text{nat-to-sch } (c\text{-snd } (c\text{-snd } n)))) \ x$ **by** *simp*

from *S3* **show** *?thesis* **by** (*simp add: nat-to-pr-def*)

qed

lemma *nat-to-pr-lm-6*: $c\text{-fst } n \text{ mod } 7 = 6 \implies \text{nat-to-pr } n \ x = (\text{UnaryRecOp } (\text{nat-to-pr } (c\text{-fst } (c\text{-snd } n)) (\text{nat-to-pr } (c\text{-snd } (c\text{-snd } n)))) \ x$

proof –

assume *A*: $c\text{-fst } n \text{ mod } 7 = 6$

have *S1*: $\text{nat-to-pr } n \ x = \text{sch-to-pr } (\text{nat-to-sch } n) \ x$ **by** (*simp add: nat-to-pr-def*)

from *A* **have** *S2*: $\text{nat-to-sch } n = \text{Rec-op } (\text{nat-to-sch } (c\text{-fst } (c\text{-snd } n))) (\text{nat-to-sch } (c\text{-snd } (c\text{-snd } n)))$ **by** (*rule nat-to-sch-6*)

from *S1 S2* **have** *S3*: $\text{nat-to-pr } n \ x = \text{sch-to-pr } (\text{Rec-op } (\text{nat-to-sch } (c\text{-fst } (c\text{-snd } n))) (\text{nat-to-sch } (c\text{-snd } (c\text{-snd } n)))) \ x$ **by** *simp*

from *S3* **show** *?thesis* **by** (*simp add: nat-to-pr-def*)

qed

lemma *univ-for-pr-lm-0*: $c\text{-fst } (c\text{-fst } \text{key}) \text{ mod } 7 = 0 \implies \text{univ-for-pr } \text{key} = 0$

proof –

assume $A: c\text{-fst } (c\text{-fst key}) \bmod 7 = 0$

have $S1: \text{univ-for-pr key} = \text{nat-to-pr } (c\text{-fst key}) (c\text{-snd key})$ **by** (*simp add: univ-for-pr-def pr-conv-2-to-1-def*)

with A **show** *?thesis* **by** (*simp add: nat-to-pr-lm-0*)

qed

lemma *univ-for-pr-lm-1*: $c\text{-fst } (c\text{-fst key}) \bmod 7 = 1 \implies \text{univ-for-pr key} = \text{Suc } (c\text{-snd key})$

proof –

assume $A: c\text{-fst } (c\text{-fst key}) \bmod 7 = 1$

have $S1: \text{univ-for-pr key} = \text{nat-to-pr } (c\text{-fst key}) (c\text{-snd key})$ **by** (*simp add: univ-for-pr-def pr-conv-2-to-1-def*)

with A **show** *?thesis* **by** (*simp add: nat-to-pr-lm-1*)

qed

lemma *univ-for-pr-lm-2*: $c\text{-fst } (c\text{-fst key}) \bmod 7 = 2 \implies \text{univ-for-pr key} = c\text{-fst } (c\text{-snd key})$

proof –

assume $A: c\text{-fst } (c\text{-fst key}) \bmod 7 = 2$

have $S1: \text{univ-for-pr key} = \text{nat-to-pr } (c\text{-fst key}) (c\text{-snd key})$ **by** (*simp add: univ-for-pr-def pr-conv-2-to-1-def*)

with A **show** *?thesis* **by** (*simp add: nat-to-pr-lm-2*)

qed

lemma *univ-for-pr-lm-3*: $c\text{-fst } (c\text{-fst key}) \bmod 7 = 3 \implies \text{univ-for-pr key} = c\text{-snd } (c\text{-snd key})$

proof –

assume $A: c\text{-fst } (c\text{-fst key}) \bmod 7 = 3$

have $S1: \text{univ-for-pr key} = \text{nat-to-pr } (c\text{-fst key}) (c\text{-snd key})$ **by** (*simp add: univ-for-pr-def pr-conv-2-to-1-def*)

with A **show** *?thesis* **by** (*simp add: nat-to-pr-lm-3*)

qed

lemma *univ-for-pr-lm-4*: $c\text{-fst } (c\text{-fst key}) \bmod 7 = 4 \implies \text{univ-for-pr key} = (\text{nat-to-pr } (c\text{-fst } (c\text{-snd } (c\text{-fst key}))) (\text{nat-to-pr } (c\text{-snd } (c\text{-snd } (c\text{-fst key}))) (c\text{-snd key})))$

proof –

assume $A: c\text{-fst } (c\text{-fst key}) \bmod 7 = 4$

have $S1: \text{univ-for-pr key} = \text{nat-to-pr } (c\text{-fst key}) (c\text{-snd key})$ **by** (*simp add: univ-for-pr-def pr-conv-2-to-1-def*)

with A **show** *?thesis* **by** (*simp add: nat-to-pr-lm-4*)

qed

lemma *univ-for-pr-lm-4-1*: $c\text{-fst } (c\text{-fst key}) \bmod 7 = 4 \implies \text{univ-for-pr key} = \text{univ-for-pr } (c\text{-pair } (c\text{-fst } (c\text{-snd } (c\text{-fst key}))) (\text{univ-for-pr } (c\text{-pair } (c\text{-snd } (c\text{-snd } (c\text{-fst key}))) (c\text{-snd key}))))$

proof –

assume $A: c\text{-fst } (c\text{-fst key}) \bmod 7 = 4$

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

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

proof –

assume A : $c\text{-}fst\ (c\text{-}fst\ key)\ mod\ 7 = 5$

have $S1$: $univ\text{-}for\text{-}pr\ key = nat\text{-}to\text{-}pr\ (c\text{-}fst\ key)\ (c\text{-}snd\ key)$ **by** ($simp\ add$: $univ\text{-}for\text{-}pr\text{-}def\ pr\text{-}conv\text{-}2\text{-}to\text{-}1\text{-}def$)

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

qed

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

proof –

assume $A1$: $c\text{-}fst\ (c\text{-}fst\ key)\ mod\ 7 = 6$

assume $A2$: $c\text{-}fst\ (c\text{-}snd\ key) = 0$

have $S1$: $univ\text{-}for\text{-}pr\ key = nat\text{-}to\text{-}pr\ (c\text{-}fst\ key)\ (c\text{-}snd\ key)$ **by** ($simp\ add$: $univ\text{-}for\text{-}pr\text{-}def\ pr\text{-}conv\text{-}2\text{-}to\text{-}1\text{-}def$)

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

qed

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

proof –

assume $A1$: $c\text{-}fst\ (c\text{-}fst\ key)\ mod\ 7 = 6$

assume $A2$: $c\text{-}fst\ (c\text{-}snd\ key) = Suc\ u$

have $S1$: $univ\text{-}for\text{-}pr\ key = nat\text{-}to\text{-}pr\ (c\text{-}fst\ key)\ (c\text{-}snd\ key)$ **by** ($simp\ add$: $univ\text{-}for\text{-}pr\text{-}def\ pr\text{-}conv\text{-}2\text{-}to\text{-}1\text{-}def$)

with $A1\ A2$ **show** $?thesis$

apply ($simp\ add$: $nat\text{-}to\text{-}pr\text{-}lm\text{-}6\ UnaryRecOp\text{-}def\ univ\text{-}for\text{-}pr\text{-}def\ pr\text{-}conv\text{-}2\text{-}to\text{-}1\text{-}def$)

apply ($simp\ add$: $pr\text{-}conv\text{-}1\text{-}to\text{-}3\text{-}def$)

done

qed

lemma $univ\text{-}for\text{-}pr\text{-}lm\text{-}6\text{-}3$: $\llbracket c\text{-}fst\ (c\text{-}fst\ key)\ mod\ 7 = 6; c\text{-}fst\ (c\text{-}snd\ key) \neq 0 \rrbracket \implies univ\text{-}for\text{-}pr\ key = univ\text{-}for\text{-}pr\ (c\text{-}pair\ (c\text{-}snd\ (c\text{-}snd\ (c\text{-}fst\ key)))\ (c\text{-}pair\ (c\text{-}pair\ (c\text{-}fst\ (c\text{-}snd\ key) - 1)\ (univ\text{-}for\text{-}pr\ (c\text{-}pair\ (c\text{-}fst\ key)\ (c\text{-}pair\ (c\text{-}fst\ (c\text{-}snd\ key) - 1)\ (c\text{-}snd\ (c\text{-}snd\ key))))))\ (c\text{-}snd\ (c\text{-}snd\ key))))$

proof –
assume $A1$: $c\text{-fst } (c\text{-fst } key) \bmod 7 = 6$
assume $A2$: $c\text{-fst } (c\text{-snd } key) \neq 0$ **then have**
 $A3$: $c\text{-fst } (c\text{-snd } key) > 0$ **by** *simp*
let $?u = c\text{-fst } (c\text{-snd } key) - (1::nat)$
from $A3$ **have** $S1$: $c\text{-fst } (c\text{-snd } key) = \text{Suc } ?u$ **by** *simp*
from $A1$ $S1$ **have** $S2$: $\text{univ-for-pr } key = \text{univ-for-pr}$
 $(c\text{-pair } (c\text{-snd } (c\text{-snd } (c\text{-fst } key))))$
 $(c\text{-pair } (c\text{-pair } ?u (\text{univ-for-pr } (c\text{-pair } (c\text{-fst } key) (c\text{-pair } ?u (c\text{-snd}$
 $(c\text{-snd } key)))))) (c\text{-snd } (c\text{-snd } key))))$ **by** (rule *univ-for-pr-lm-6-2*)
thus *thesis* **by** *simp*
qed

lemma *g-comp-lm-0*: $\llbracket c\text{-fst } (c\text{-fst } key) \bmod 7 = 4; c\text{-is-sub-fun } ls \text{ univ-for-pr};$
 $g\text{-comp } ls \text{ key} \neq ls \rrbracket \implies g\text{-comp } ls \text{ key} = c\text{-cons } (c\text{-pair } key (\text{univ-for-pr } key)) \text{ ls}$
proof –

assume $A1$: $c\text{-fst } (c\text{-fst } key) \bmod 7 = 4$
assume $A2$: $c\text{-is-sub-fun } ls \text{ univ-for-pr}$
assume $A3$: $g\text{-comp } ls \text{ key} \neq ls$
let $?n = c\text{-fst } key$
let $?x = c\text{-snd } key$
let $?m = c\text{-snd } ?n$
let $?m1 = c\text{-fst } ?m$
let $?m2 = c\text{-snd } ?m$
let $?k1 = c\text{-pair } ?m2 ?x$
have $S1$: $c\text{-assoc-have-key } ls \text{ ?k1} = 0$
proof (rule *ccontr*)
assume $A1-1$: $c\text{-assoc-have-key } ls \text{ ?k1} \neq 0$
then have $g\text{-comp } ls \text{ key} = ls$ **by** (*simp add: g-comp-def*)
with $A3$ **show** *False* **by** *simp*

qed
let $?y = c\text{-assoc-value } ls \text{ ?k1}$
from $A2$ $S1$ **have** $S2$: $?y = \text{univ-for-pr } ?k1$ **by** (rule *c-is-sub-fun-lm-1*)
let $?k2 = c\text{-pair } ?m1 ?y$
have $S3$: $c\text{-assoc-have-key } ls \text{ ?k2} = 0$
proof (rule *ccontr*)
assume $A3-1$: $c\text{-assoc-have-key } ls \text{ ?k2} \neq 0$
then have $g\text{-comp } ls \text{ key} = ls$ **by** (*simp add: g-comp-def Let-def*)
with $A3$ **show** *False* **by** *simp*

qed
let $?z = c\text{-assoc-value } ls \text{ ?k2}$
from $A2$ $S3$ **have** $S4$: $?z = \text{univ-for-pr } ?k2$ **by** (rule *c-is-sub-fun-lm-1*)
from $S2$ **have** $S5$: $?k2 = c\text{-pair } ?m1 (\text{univ-for-pr } ?k1)$ **by** *simp*
from $S4$ $S5$ **have** $S6$: $?z = \text{univ-for-pr } (c\text{-pair } ?m1 (\text{univ-for-pr } ?k1))$ **by** *simp*
from $A1$ $S6$ **have** $S7$: $?z = \text{univ-for-pr } key$ **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*: $\llbracket c\text{-fst } (c\text{-fst } key) \bmod 7 = 4; c\text{-is-sub-fun } ls \text{ univ-for-pr} \rrbracket$

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

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

proof –
 assume *A1: c-fst (c-fst key) mod 7 = 5*
 assume *A2: c-is-sub-fun ls univ-for-pr*
 assume *A3: g-pair ls key \neq ls*
 let *?n = c-fst key*
 let *?x = c-snd key*
 let *?m = c-snd ?n*
 let *?m1 = c-fst ?m*
 let *?m2 = c-snd ?m*
 let *?k1 = c-pair ?m1 ?x*
 have *S1: c-assoc-have-key ls ?k1 = 0*
proof *(rule ccontr)*
 assume *A1-1: c-assoc-have-key ls ?k1 \neq 0*
 then have *g-pair ls key = ls* **by** *(simp add: g-pair-def)*
 with *A3* show *False* **by** *simp*
qed
 let *?y1 = c-assoc-value ls ?k1*
 from *A2 S1* have *S2: ?y1 = univ-for-pr ?k1* **by** *(rule c-is-sub-fun-lm-1)*
 let *?k2 = c-pair ?m2 ?x*
 have *S3: c-assoc-have-key ls ?k2 = 0*
proof *(rule ccontr)*
 assume *A3-1: c-assoc-have-key ls ?k2 \neq 0*
 then have *g-pair ls key = ls* **by** *(simp add: g-pair-def Let-def)*
 with *A3* show *False* **by** *simp*
qed
 let *?y2 = c-assoc-value ls ?k2*
 from *A2 S3* have *S4: ?y2 = univ-for-pr ?k2* **by** *(rule c-is-sub-fun-lm-1)*
 let *?z = c-pair ?y1 ?y2*
 from *S2 S4* have *S5: ?z = c-pair (univ-for-pr ?k1) (univ-for-pr ?k2)* **by** *simp*
 from *A1 S5* have *S6: ?z = univ-for-pr key* **by** *(simp add: univ-for-pr-lm-5)*
 from *S1 S3 S6* show *?thesis* **by** *(simp add: g-pair-def Let-def)*

qed

lemma *g-pair-lm-1*: $\llbracket c\text{-fst } (c\text{-fst key}) \bmod 7 = 5; c\text{-is-sub-fun } ls \text{ univ-for-pr} \rrbracket \implies c\text{-is-sub-fun } (g\text{-pair } ls \text{ key}) \text{ univ-for-pr}$

proof –

assume *A1*: $c\text{-fst } (c\text{-fst key}) \bmod 7 = 5$

assume *A2*: $c\text{-is-sub-fun } ls \text{ univ-for-pr}$

show *?thesis*

proof *cases*

assume $g\text{-pair } ls \text{ key} = ls$

with *A2* **show** $c\text{-is-sub-fun } (g\text{-pair } ls \text{ key}) \text{ univ-for-pr}$ **by** *simp*

next

assume $g\text{-pair } ls \text{ key} \neq ls$

from *A1 A2 this* **have** *S1*: $g\text{-pair } ls \text{ key} = c\text{-cons } (c\text{-pair key } (univ\text{-for-pr key}))$

ls **by** (*rule g-pair-lm-0*)

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

qed

qed

lemma *g-rec-lm-0*: $\llbracket c\text{-fst } (c\text{-fst key}) \bmod 7 = 6; c\text{-is-sub-fun } ls \text{ univ-for-pr}; g\text{-rec } ls \text{ key} \neq ls \rrbracket \implies g\text{-rec } ls \text{ key} = c\text{-cons } (c\text{-pair key } (univ\text{-for-pr key})) \text{ } ls$

proof –

assume *A1*: $c\text{-fst } (c\text{-fst key}) \bmod 7 = 6$

assume *A2*: $c\text{-is-sub-fun } ls \text{ univ-for-pr}$

assume *A3*: $g\text{-rec } ls \text{ key} \neq ls$

let *?n* = $c\text{-fst key}$

let *?x* = $c\text{-snd key}$

let *?m* = $c\text{-snd } ?n$

let *?m1* = $c\text{-fst } ?m$

let *?m2* = $c\text{-snd } ?m$

let *?y1* = $c\text{-fst } ?x$

let *?x1* = $c\text{-snd } ?x$

show *?thesis*

proof *cases*

assume *A1-1*: $?y1 = 0$

let *?k1* = $c\text{-pair } ?m1 \text{ } ?x1$

have *S1-1*: $c\text{-assoc-have-key } ls \text{ } ?k1 = 0$

proof (*rule ccontr*)

assume $c\text{-assoc-have-key } ls \text{ } ?k1 \neq 0$

with *A1-1* **have** $g\text{-rec } ls \text{ key} = ls$ **by** (*simp add: g-rec-def*)

with *A3* **show** *False* **by** *simp*

qed

let *?v* = $c\text{-assoc-value } ls \text{ } ?k1$

from *A2 S1-1* **have** *S1-2*: $?v = univ\text{-for-pr } ?k1$ **by** (*rule c-is-sub-fun-lm-1*)

from *A1 A1-1 S1-2* **have** *S1-3*: $?v = univ\text{-for-pr key}$ **by** (*simp add: univ-for-pr-lm-6-1*)

from *A1-1 S1-1 S1-3* **show** *?thesis* **by** (*simp add: g-rec-def Let-def*)

next

assume *A2-1*: $?y1 \neq 0$ **then have** *A2-2*: $?y1 > 0$ **by** *simp*

let *?y2* = $?y1 - (1::nat)$

```

let ?k2 = c-pair ?n (c-pair ?y2 ?x1)
have S2-1: c-assoc-have-key ls ?k2 = 0
proof (rule ccontr)
  assume c-assoc-have-key ls ?k2 ≠ 0
  with A2-1 have g-rec ls key = ls by (simp add: g-rec-def Let-def)
  with A3 show False by simp
qed
let ?t1 = c-assoc-value ls ?k2
from A2 S2-1 have S2-2: ?t1 = univ-for-pr ?k2 by (rule c-is-sub-fun-lm-1)
let ?t2 = c-pair (c-pair ?y2 ?t1) ?x1
let ?k3 = c-pair ?m2 ?t2
have S2-3: c-assoc-have-key ls ?k3 = 0
proof (rule ccontr)
  assume c-assoc-have-key ls ?k3 ≠ 0
  with A2-1 have g-rec ls key = ls by (simp add: g-rec-def Let-def)
  with A3 show False by simp
qed
let ?u = c-assoc-value ls ?k3
from A2 S2-3 have S2-4: ?u = univ-for-pr ?k3 by (rule c-is-sub-fun-lm-1)
from S2-4 S2-2 have S2-5: ?u = univ-for-pr (c-pair ?m2 (c-pair (c-pair ?y2
(univ-for-pr ?k2)) ?x1)) by simp
from A1 A2-1 S2-5 have S2-6: ?u = univ-for-pr key by (simp add: univ-for-pr-lm-6-3)
from A2-1 S2-1 S2-3 S2-6 show ?thesis by (simp add: g-rec-def Let-def)
qed
qed

```

```

lemma g-rec-lm-1: [ c-fst (c-fst key) mod 7 = 6; c-is-sub-fun ls univ-for-pr ] ⇒
c-is-sub-fun (g-rec ls key) univ-for-pr
proof -
  assume A1: c-fst (c-fst key) mod 7 = 6
  assume A2: c-is-sub-fun ls univ-for-pr
  show ?thesis
  proof cases
    assume g-rec ls key = ls
    with A2 show c-is-sub-fun (g-rec ls key) univ-for-pr by simp
  next
    assume g-rec ls key ≠ ls
    from A1 A2 this have S1: g-rec ls key = c-cons (c-pair key (univ-for-pr key))
ls by (rule g-rec-lm-0)
    with A2 show c-is-sub-fun (g-rec ls key) univ-for-pr by (simp add: c-is-sub-fun-lm-2)
  qed
qed

```

```

lemma g-step-lm-0: c-fst (c-fst key) mod 7 = 0 ⇒ g-step ls key = c-cons (c-pair
key 0) ls by (simp add: g-step-def)

```

```

lemma g-step-lm-1: c-fst (c-fst key) mod 7 = 1 ⇒ g-step ls key = c-cons (c-pair
key (Suc (c-snd key))) ls by (simp add: g-step-def Let-def)

```

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

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

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

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

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

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

proof –

assume *A1*: $c\text{-is-sub-fun ls univ-for-pr}$

let *?n* = $c\text{-fst key}$

let *?x* = $c\text{-snd key}$

let *?n1* = $(c\text{-fst } ?n) \bmod 7$

have *S1*: $?n1 = 0 \implies ?thesis$

proof –

assume *A*: $?n1 = 0$

then have *S1-1*: $g\text{-step ls key} = c\text{-cons } (c\text{-pair key } 0) \text{ ls by } (rule \text{ } g\text{-step-lm-0})$

from *A* **have** *S1-2*: $univ\text{-for-pr key} = 0 \text{ by } (rule \text{ } univ\text{-for-pr-lm-0})$

from *A1* **have** *S1-3*: $c\text{-is-sub-fun } (c\text{-cons } (c\text{-pair key } (univ\text{-for-pr key}))) \text{ ls) univ-for-pr by } (rule \text{ } c\text{-is-sub-fun-lm-2})$

from *S1-3 S1-1 S1-2* **show** *?thesis* **by** *simp*

qed

have *S2*: $?n1 = 1 \implies ?thesis$

proof –

assume *A*: $?n1 = 1$

then have *S2-1*: $g\text{-step ls key} = c\text{-cons } (c\text{-pair key } (Suc \text{ } (c\text{-snd key}))) \text{ ls by } (rule \text{ } g\text{-step-lm-1})$

from *A* **have** *S2-2*: $univ\text{-for-pr key} = Suc \text{ } (c\text{-snd key}) \text{ by } (rule \text{ } univ\text{-for-pr-lm-1})$

from *A1* **have** *S2-3*: $c\text{-is-sub-fun } (c\text{-cons } (c\text{-pair key } (univ\text{-for-pr key}))) \text{ ls) univ-for-pr by } (rule \text{ } c\text{-is-sub-fun-lm-2})$

from *S2-3 S2-1 S2-2* **show** *?thesis* **by** *simp*

qed

have *S3*: $?n1 = 2 \implies ?thesis$

proof –

assume *A*: $?n1 = 2$

then have *S2-1*: $g\text{-step ls key} = c\text{-cons } (c\text{-pair key } (c\text{-fst } (c\text{-snd key}))) \text{ ls by } (rule \text{ } g\text{-step-lm-2})$

from *A* **have** *S2-2*: $univ\text{-for-pr key} = c\text{-fst } (c\text{-snd key}) \text{ by } (rule \text{ } univ\text{-for-pr-lm-2})$

from *A1* **have** *S2-3*: $c\text{-is-sub-fun } (c\text{-cons } (c\text{-pair key } (univ\text{-for-pr key}))) \text{ ls) univ-for-pr by } (rule \text{ } c\text{-is-sub-fun-lm-2})$

```

    from S2-3 S2-1 S2-2 show ?thesis by simp
  qed
  have S4: ?n1 = 3  $\implies$  ?thesis
  proof -
    assume A: ?n1 = 3
    then have S2-1: g-step ls key = c-cons (c-pair key (c-snd (c-snd key))) ls by
      (rule g-step-lm-3)
    from A have S2-2: univ-for-pr key = c-snd (c-snd key) by (rule univ-for-pr-lm-3)
    from A1 have S2-3: c-is-sub-fun (c-cons (c-pair key (univ-for-pr key)) ls)
      univ-for-pr by (rule c-is-sub-fun-lm-2)
    from S2-3 S2-1 S2-2 show ?thesis by simp
  qed
  have S5: ?n1 = 4  $\implies$  ?thesis
  proof -
    assume A: ?n1 = 4
    then have S2-1: g-step ls key = g-comp ls key by (rule g-step-lm-4)
    from A A1 S2-1 show ?thesis by (simp add: g-comp-lm-1)
  qed
  have S6: ?n1 = 5  $\implies$  ?thesis
  proof -
    assume A: ?n1 = 5
    then have S2-1: g-step ls key = g-pair ls key by (rule g-step-lm-5)
    from A A1 S2-1 show ?thesis by (simp add: g-pair-lm-1)
  qed
  have S7: ?n1 = 6  $\implies$  ?thesis
  proof -
    assume A: ?n1 = 6
    then have S2-1: g-step ls key = g-rec ls key by (rule g-step-lm-6)
    from A A1 S2-1 show ?thesis by (simp add: g-rec-lm-1)
  qed
  have S8: ?n1=0  $\vee$  ?n1=1  $\vee$  ?n1=2  $\vee$  ?n1=3  $\vee$  ?n1=4  $\vee$  ?n1=5  $\vee$  ?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: $g\text{-comp } ls \text{ key} = ls \vee c\text{-tl } (g\text{-comp } ls \text{ key}) = ls$ by(simp add: g-comp-def Let-def)

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

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

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

done

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

lemma *c-assoc-have-key-pos*: $c\text{-assoc-have-key } ls \ x = 0 \implies ls > 0$

proof –

assume *A1*: $c\text{-assoc-have-key } ls \ x = 0$

thus *?thesis*

proof (*cases*)

assume *A2*: $ls = 0$

then have *S1*: $c\text{-assoc-have-key } ls \ x = 1$ **by** (*simp add: c-assoc-have-key-df*)

with *A1* **have** *S2*: *False* **by** *auto*

then show $ls > 0$ **by** *auto*

next

assume *A3*: $\neg ls = 0$

then show $ls > 0$ **by** *auto*

qed

qed

lemma *lm2*: $c\text{-assoc-have-key } (c\text{-tl } ls) \ \text{key} = 0 \implies c\text{-assoc-have-key } ls \ \text{key} = 0$

proof –

assume *A1*: $c\text{-assoc-have-key } (c\text{-tl } ls) \ \text{key} = 0$

from *A1* **have** *S1*: $c\text{-tl } ls > 0$ **by** (*rule c-assoc-have-key-pos*)

have *S2*: $c\text{-tl } ls \leq ls$ **by** (*rule c-tl-le*)

from *S1 S2* **have** *S3*: $ls \neq 0$ **by** *auto*

from *A1 S3* **show** *?thesis* **by** (*auto simp add: c-assoc-have-key-lm-1*)

qed

lemma *lm3*: $c\text{-assoc-have-key } (pr\text{-gr } x) \ \text{key} = 0 \implies c\text{-assoc-have-key } (pr\text{-gr } (Suc \ x)) \ \text{key} = 0$

proof –

assume *A1*: $c\text{-assoc-have-key } (pr\text{-gr } x) \ \text{key} = 0$

have *S1*: $pr\text{-gr } (Suc \ x) = pr\text{-gr } x \vee c\text{-tl } (pr\text{-gr } (Suc \ x)) = pr\text{-gr } x$ **by** (*rule lm1*)

from *A1* **have** *S2*: $pr\text{-gr } (Suc \ x) = pr\text{-gr } x \implies ?thesis$ **by** *auto*

have *S3*: $c\text{-tl } (pr\text{-gr } (Suc \ x)) = pr\text{-gr } x \implies ?thesis$

proof –

assume $c\text{-tl } (pr\text{-gr } (Suc \ x)) = pr\text{-gr } x$ (**is** $c\text{-tl } ?ls = -$)

with *A1* **have** $c\text{-assoc-have-key } (c\text{-tl } ?ls) \ \text{key} = 0$ **by** *auto*

then show $c\text{-assoc-have-key } ?ls \ \text{key} = 0$ **by** (*rule lm2*)

qed

from *S1 S2 S3* **show** *?thesis* **by** *auto*

qed

lemma *lm4*: $\llbracket c\text{-assoc-have-key } (pr\text{-gr } x) \ \text{key} = 0; 0 \leq y \rrbracket \implies c\text{-assoc-have-key } (pr\text{-gr } (x+y)) \ \text{key} = 0$

```

apply(induct-tac y)
apply(auto)
apply(simp add: lm3)
done

```

lemma *lm5*: $\llbracket c\text{-assoc-have-key } (pr\text{-gr } x) \text{ key} = 0; x \leq y \rrbracket \implies c\text{-assoc-have-key } (pr\text{-gr } y) \text{ key} = 0$

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*: $n = 0 \implies (c\text{-fst } n) \bmod 7 = 0$

```

apply(simp add: c-fst-at-0)

```

done

lemma *loc-upb-lm-2*: $(c\text{-fst } n) \bmod 7 > 1 \implies c\text{-snd } n < n$

proof –

```

  assume A1:  $c\text{-fst } n \bmod 7 > 1$ 
  from A1 have S1:  $1 < c\text{-fst } n$  by simp
  have S2:  $c\text{-fst } n \leq n$  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*: $(c\text{-fst } n) \bmod 7 = 4 \longrightarrow c\text{-fst } (c\text{-snd } n) < n$

proof

```

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

```

qed

lemma *loc-upb-lm-2-2*: $(c\text{-fst } n) \bmod 7 = 4 \longrightarrow c\text{-snd } (c\text{-snd } n) < n$

proof

```

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

```

qed

lemma *loc-upb-lm-2-3*: $(c\text{-fst } n) \bmod 7 = 5 \longrightarrow c\text{-fst } (c\text{-snd } n) < n$
proof

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

qed

lemma *loc-upb-lm-2-4*: $(c\text{-fst } n) \bmod 7 = 5 \longrightarrow c\text{-snd } (c\text{-snd } n) < n$
proof

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

qed

lemma *loc-upb-lm-2-5*: $(c\text{-fst } n) \bmod 7 = 6 \longrightarrow c\text{-fst } (c\text{-snd } n) < n$
proof

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

qed

lemma *loc-upb-lm-2-6*: $(c\text{-fst } n) \bmod 7 = 6 \longrightarrow c\text{-snd } (c\text{-snd } n) < n$
proof

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

qed

lemma *loc-upb-lm-2-7*: $\llbracket y2 = y1 - (1::nat); 0 < y1; x1 = c\text{-snd } x; y1 = c\text{-fst } x \rrbracket$
 $\implies c\text{-pair } y2 \ x1 < x$

proof –

assume *A1*: $y2 = y1 - (1::nat)$ **and** *A2*: $0 < y1$ **and** *A3*: $x1 = c\text{-snd } x$ **and**
 A4: $y1 = c\text{-fst } x$
 from *A1 A2* **have** *S1*: $y2 < y1$ **by** *auto*
 from *S1* **have** *S2*: $c\text{-pair } y2 \ x1 < c\text{-pair } y1 \ x1$ **by** (*rule c-pair-strict-mono1*)
 from *A3 A4* **have** *S3*: $c\text{-pair } y1 \ x1 = x$ **by** *auto*
 from *S2 S3* **show** $c\text{-pair } y2 \ x1 < x$ **by** *auto*

qed

function *loc-upb* :: $nat \Rightarrow nat \Rightarrow nat$ **where**

aa: $loc\text{-upb } n \ x = ($

```

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-fst 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 (λ m. m) <*lex*> measure (λ 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 (λ m. m)) <*lex*> (measure (λ 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

```

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

proof –

assume *A1*: $c\text{-fst } n \bmod 7 = 0$

let *?key* = $c\text{-pair } n \ x$

let *?s* = $c\text{-pair } ?key \ 0$

let *?ls* = $pr\text{-gr } ?s$

from *A1* **have** $loc\text{-upb } n \ x = ?s + 1$ **by** *simp*

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

from *A1* **have** *S2*: $g\text{-step } ?ls \ ?key = c\text{-cons } (c\text{-pair } ?key \ 0) \ ?ls$ **by** (*simp add: g-step-def*)

from *S1 S2* **have** $pr\text{-gr } (loc\text{-upb } n \ x) = c\text{-cons } (c\text{-pair } ?key \ 0) \ ?ls$ **by** *auto*

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

qed

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

proof –

assume *A1*: $c\text{-fst } n \bmod 7 = 1$

let *?key* = $c\text{-pair } n \ x$

let *?s* = $c\text{-pair } ?key \ 0$

let *?ls* = $pr\text{-gr } ?s$

from *A1* **have** $loc\text{-upb } n \ x = ?s + 1$ **by** *simp*

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

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

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

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

qed

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

proof –

assume *A1*: $c\text{-fst } n \bmod 7 = 2$

let *?key* = $c\text{-pair } n \ x$

let *?s* = $c\text{-pair } ?key \ 0$

let *?ls* = $pr\text{-gr } ?s$

from *A1* **have** $loc\text{-upb } n \ x = ?s + 1$ **by** *simp*

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

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

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

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

qed

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

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

proof –

assume $A1: c\text{-fst } n \ \text{mod } 7 = 3$

let $?key = c\text{-pair } n \ x$

let $?s = c\text{-pair } ?key \ 0$

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

from $A1$ **have** $loc\text{-upb } n \ x = ?s + 1$ **by** $simp$

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

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

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

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

qed

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

$c\text{-fst } n \ \text{mod } 7 = 4 \rrbracket \implies$

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

proof –

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

assume $A2: c\text{-fst } n \ \text{mod } 7 = 4$

let $?key = c\text{-pair } n \ x$

let $?m1 = c\text{-fst } (c\text{-snd } n)$

let $?m2 = c\text{-snd } (c\text{-snd } n)$

define $upb1$ **where** $upb1 = loc\text{-upb } ?m2 \ x$

from $A2$ **have** $m2\text{-lt-}n: ?m2 < n$ **by** $(simp \ \text{add: } loc\text{-upb-lm-2-2})$

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

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** $(simp \ \text{add: } lex\text{-p-def})$

have $T1: c\text{-is-sub-fun } (pr\text{-gr } upb1) \ \text{univ-for-pr}$ **by** $(rule \ pr\text{-gr-1})$

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** $(rule \ c\text{-is-sub-fun-lm-1})$

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** $(simp \ \text{add: } loc\text{-upb-lm-2-0})$

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

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** $(simp \ \text{add: } lex\text{-p-def})$

from $S1 \ upb1\text{-def}$ **have** $S3: c\text{-assoc-have-key } (pr\text{-gr } upb1) (c\text{-pair } ?m2 \ x) = 0$ **by** $auto$

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)) \in lex-p **by** (simp add: lex-p-eq)
then have $M1''$: (($?m1$, x), (n , x)) \in measure ($\lambda m. m$) $< *lex* >$ measure ($\lambda n. n$)
by (simp add: lex-p-def)
from $A2$ $M2'$ $M1''$ **have** $S11$: loc-upb n $x = (let$ $y = c$ -assoc-value (pr-gr (loc-upb
 $?m2$ $x))$ (c-pair $?m2$ x)
in (c-pair (c-pair n x)
(loc-upb $?m2$ $x + loc$ -upb $?m1$ $y)) + 1)$
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 $\leq ?s$ **by** (rule arg2-le-c-pair)

have $upb1$ -le-s: $upb1 \leq ?s$

proof –

have $S1$: $upb1 \leq ?sum$ -upb **by** (rule Nat.le-add1)

from $S1$ $S7$ **show** $?thesis$ **by** auto

qed

have $upb2$ -le-s: $upb2 \leq ?s$

proof –

have $S1$: $upb2 \leq ?sum$ -upb **by** (rule Nat.le-add2)

from $S1$ $S7$ **show** $?thesis$ **by** auto

qed

have $S18$: pr-gr $upb = g$ -comp $?ls$ $?key$

proof –

from $S11$ **have** $S1$: pr-gr $upb = g$ -step (pr-gr $?s$) (c-fst $?s$) **by** (simp add:
pr-gr-at-Suc)

from $A2$ **have** $S2$: g -step $?ls$ $?key = g$ -comp $?ls$ $?key$ **by** (simp add: g -step-def)

from $S1$ $S2$ **show** $?thesis$ **by** auto

qed

from $S3$ $upb1$ -le-s **have** $S19$: c -assoc-have-key $?ls$ (c-pair $?m2$ x) = 0 **by** (rule
lm5)

from $S4$ $upb2$ -le-s **have** $S20$: c -assoc-have-key $?ls$ (c-pair $?m1$ y) = 0 **by** (rule
lm5)

have T -ls: c -is-sub-fun $?ls$ univ-for-pr **by** (rule pr-gr-1)

from T -ls $S19$ **have** T -ls2: c -assoc-value $?ls$ (c-pair $?m2$ x) = univ-for-pr (c-pair
 $?m2$ x) **by** (rule c -is-sub-fun-lm-1)

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

from T -y $S19$ $S20$ **have** $S21$: g -comp $?ls$ $?key = c$ -cons (c-pair $?key$ (c-assoc-value

$?ls$ ($c\text{-pair } ?m1\ y$)) $?ls$
by($unfold\ g\text{-comp}\text{-def}$)($simp\ del: loc\text{-upb.simps}\ add: Let\text{-def}$)
from $S18\ S21$ **have** $pr\text{-gr}\ upb = c\text{-cons}\ (c\text{-pair } ?key\ (c\text{-assoc}\text{-value } ?ls\ (c\text{-pair } ?m1\ y)))\ ?ls$ **by** $auto$
with $upb\text{-def}$ **have** $pr\text{-gr}\ (loc\text{-upb}\ n\ x) = c\text{-cons}\ (c\text{-pair } ?key\ (c\text{-assoc}\text{-value } ?ls\ (c\text{-pair } ?m1\ y)))\ ?ls$ **by** $auto$
thus $?thesis$ **by** ($simp\ add: c\text{-assoc}\text{-lm}\text{-1}$)
qed

lemma $loc\text{-upb}\text{-lex}\text{-5}$: $\llbracket \bigwedge n'\ x'. ((n',x'), (n,x)) \in lex\text{-p} \implies c\text{-assoc}\text{-have}\text{-key}\ (pr\text{-gr}\ (loc\text{-upb}\ n'\ x'))\ (c\text{-pair}\ n'\ x') = 0;$
 $c\text{-fst}\ n\ \text{mod}\ 7 = 5 \rrbracket \implies$
 $c\text{-assoc}\text{-have}\text{-key}\ (pr\text{-gr}\ (loc\text{-upb}\ n\ x))\ (c\text{-pair}\ n\ x) = 0$

proof –

assume $A1$: $\bigwedge n'\ x'. ((n',x'), (n,x)) \in lex\text{-p} \implies c\text{-assoc}\text{-have}\text{-key}\ (pr\text{-gr}\ (loc\text{-upb}\ n'\ x'))\ (c\text{-pair}\ n'\ x') = 0$
assume $A2$: $c\text{-fst}\ n\ \text{mod}\ 7 = 5$
let $?key = c\text{-pair}\ n\ x$
let $?m1 = c\text{-fst}\ (c\text{-snd}\ n)$
let $?m2 = c\text{-snd}\ (c\text{-snd}\ n)$
from $A2$ **have** $?m1 < n$ **by** ($simp\ add: loc\text{-upb}\text{-lm}\text{-2}\text{-3}$)
then **have** $((?m1, x), (n,x)) \in lex\text{-p}$ **by** ($simp\ add: lex\text{-p}\text{-eq}$)
with $A1$ **have** $S1$: $c\text{-assoc}\text{-have}\text{-key}\ (pr\text{-gr}\ (loc\text{-upb}\ ?m1\ x))\ (c\text{-pair}\ ?m1\ x) = 0$
by $auto$
from $A2$ **have** $?m2 < n$ **by** ($simp\ add: loc\text{-upb}\text{-lm}\text{-2}\text{-4}$)
then **have** $((?m2, x), (n,x)) \in lex\text{-p}$ **by** ($simp\ add: lex\text{-p}\text{-eq}$)
with $A1$ **have** $S2$: $c\text{-assoc}\text{-have}\text{-key}\ (pr\text{-gr}\ (loc\text{-upb}\ ?m2\ x))\ (c\text{-pair}\ ?m2\ x) = 0$
by $auto$
define $upb1$ **where** $upb1 = loc\text{-upb}\ ?m1\ x$
define $upb2$ **where** $upb2 = loc\text{-upb}\ ?m2\ x$
from $upb1\text{-def}\ S1$ **have** $S3$: $c\text{-assoc}\text{-have}\text{-key}\ (pr\text{-gr}\ upb1)\ (c\text{-pair}\ ?m1\ x) = 0$
by $auto$
from $upb2\text{-def}\ S2$ **have** $S4$: $c\text{-assoc}\text{-have}\text{-key}\ (pr\text{-gr}\ upb2)\ (c\text{-pair}\ ?m2\ x) = 0$
by $auto$
let $?sum\text{-upb} = upb1 + upb2$
have $S5$: $upb1 \leq ?sum\text{-upb}$ **by** ($rule\ Nat.le\text{-add}\ 1$)
have $S6$: $upb2 \leq ?sum\text{-upb}$ **by** ($rule\ Nat.le\text{-add}\ 2$)
let $?s = (c\text{-pair}\ ?key\ ?sum\text{-upb})$
have $S7$: $?sum\text{-upb} \leq ?s$ **by** ($rule\ arg2\text{-le}\text{-c}\text{-pair}$)
from $S5\ S7$ **have** $S8$: $upb1 \leq ?s$ **by** $auto$
from $S6\ S7$ **have** $S9$: $upb2 \leq ?s$ **by** $auto$
let $?ls = pr\text{-gr}\ ?s$
from $A2\ upb1\text{-def}\ upb2\text{-def}$ **have** $S10$: $loc\text{-upb}\ n\ x = ?s + 1$ **by** ($simp\ add: Let\text{-def}$)
define upb **where** $upb = loc\text{-upb}\ n\ x$
from $upb\text{-def}\ S10$ **have** $S11$: $upb = ?s + 1$ **by** $auto$
from $S11$ **have** $S12$: $pr\text{-gr}\ upb = g\text{-step}\ (pr\text{-gr}\ ?s)\ (c\text{-fst}\ ?s)$ **by** ($simp\ add: pr\text{-gr}\text{-at}\text{-Suc}$)
from $S8\ S10\ upb\text{-def}$ **have** $S13$: $upb1 \leq upb$ **by** ($simp\ only:$)

from $S9\ S10$ *upb-def* **have** $S14$: $upb2 \leq upb$ **by** (*simp only*):
from $S3\ S13$ **have** $S15$: $c\text{-assoc-have-key}$ (*pr-gr upb*) ($c\text{-pair } ?m1\ x$) = 0 **by**
(*rule lm5*)
from $S4\ S14$ **have** $S16$: $c\text{-assoc-have-key}$ (*pr-gr upb*) ($c\text{-pair } ?m2\ x$) = 0 **by**
(*rule lm5*)
from $A2$ **have** $S17$: $g\text{-step } ?ls\ ?key = g\text{-pair } ?ls\ ?key$ **by** (*simp add: g-step-def*)
from $S12\ S17$ **have** $S18$: $pr\text{-gr upb} = g\text{-pair } ?ls\ ?key$ **by** *auto*
from $S3\ S8$ **have** $S19$: $c\text{-assoc-have-key } ?ls$ ($c\text{-pair } ?m1\ x$) = 0 **by** (*rule lm5*)
from $S4\ S9$ **have** $S20$: $c\text{-assoc-have-key } ?ls$ ($c\text{-pair } ?m2\ x$) = 0 **by** (*rule lm5*)
let $?y1 = c\text{-assoc-value } ?ls$ ($c\text{-pair } ?m1\ x$)
let $?y2 = c\text{-assoc-value } ?ls$ ($c\text{-pair } ?m2\ x$)
let $?y = c\text{-pair } ?y1\ ?y2$
from $S19\ S20$ **have** $S21$: $g\text{-pair } ?ls\ ?key = c\text{-cons}$ ($c\text{-pair } ?key\ ?y$) $?ls$ **by** (*unfold*
g-pair-def, simp add: Let-def)
from $S18\ S21$ **have** $S22$: $pr\text{-gr upb} = c\text{-cons}$ ($c\text{-pair } ?key\ ?y$) $?ls$ **by** *auto*
from *upb-def* $S22$ **have** $S23$: $pr\text{-gr}$ ($loc\text{-upb } n\ x$) = $c\text{-cons}$ ($c\text{-pair } ?key\ ?y$) $?ls$
by *auto*
from $S23$ **show** $?thesis$ **by** (*simp add: c-assoc-lm-1*)
qed

lemma *loc-upb-6-z*: $\llbracket c\text{-fst } n \bmod 7 = 6; c\text{-fst } x = 0 \rrbracket \implies$
 $loc\text{-upb } n\ x = c\text{-pair}$ ($c\text{-pair } n\ x$) ($loc\text{-upb}$ ($c\text{-fst}$ ($c\text{-snd } n$)) ($c\text{-snd } x$)) + 1 **by**
(*simp add: Let-def*)

lemma *loc-upb-6*: $\llbracket c\text{-fst } n \bmod 7 = 6; c\text{-fst } x \neq 0 \rrbracket \implies loc\text{-upb } n\ x = ($let\ m = c\text{-snd } n; m1 = c\text{-fst } m; m2 = c\text{-snd } m; y1 = c\text{-fst } x; x1 = c\text{-snd } x;$
 $y2 = y1 - 1;$
 $t1 = c\text{-assoc-value}$ (*pr-gr* ($loc\text{-upb } n$ ($c\text{-pair } y2\ x1$))) ($c\text{-pair } n$ ($c\text{-pair } y2\ x1$));
 $t2 = c\text{-pair}$ ($c\text{-pair } y2\ t1$) $x1$ *in*
 $c\text{-pair}$ ($c\text{-pair } n\ x$) ($loc\text{-upb } n$ ($c\text{-pair } y2\ x1$) + ($loc\text{-upb } m2\ t2$)) + 1)
by (*simp add: Let-def*)$

lemma *loc-upb-lex-6*: $\llbracket \bigwedge n'\ x'. ((n', x'), (n, x)) \in lex\text{-p} \implies c\text{-assoc-have-key}$ (*pr-gr*
($loc\text{-upb } n'\ x'$)) ($c\text{-pair } n'\ x'$) = 0;
 $c\text{-fst } n \bmod 7 = 6 \rrbracket \implies$
 $c\text{-assoc-have-key}$ (*pr-gr* ($loc\text{-upb } n\ x$)) ($c\text{-pair } n\ x$) = 0

proof –

assume $A1$: $\bigwedge n'\ x'. ((n', x'), (n, x)) \in lex\text{-p} \implies c\text{-assoc-have-key}$ (*pr-gr* ($loc\text{-upb } n'\ x'$)) ($c\text{-pair } n'\ x'$) = 0
assume $A2$: $c\text{-fst } n \bmod 7 = 6$
let $?key = c\text{-pair } n\ x$
let $?m1 = c\text{-fst}$ ($c\text{-snd } n$)
let $?m2 = c\text{-snd}$ ($c\text{-snd } n$)
let $?y1 = c\text{-fst } x$
let $?x1 = c\text{-snd } x$
define upb **where** $upb = loc\text{-upb } n\ x$

```

show ?thesis
proof (cases)
  assume A: ?y1 = 0
  from A2 A have S1: loc-upb n x = c-pair ?key (loc-upb ?m1 (c-snd x)) + 1
by (rule loc-upb-6-z)
  define upb1 where upb1 = loc-upb ?m1 (c-snd x)
  from upb1-def S1 have S2: loc-upb n x = c-pair ?key upb1 + 1 by auto
  let ?s = c-pair ?key upb1
  from S2 have S3: pr-gr (loc-upb n x) = pr-gr (Suc ?s) by simp
  have pr-gr (Suc ?s) = g-step (pr-gr ?s) (c-fst ?s) by (rule pr-gr-at-Suc)
  with S3 have S4: pr-gr (loc-upb n x) = g-step (pr-gr ?s) ?key by auto
  let ?ls = pr-gr ?s
  from A2 have g-step ?ls ?key = g-rec ?ls ?key by (simp add: g-step-def)
  with S4 have S5: pr-gr (loc-upb n x) = g-rec ?ls ?key by auto
  have S6: c-assoc-have-key ?ls (c-pair ?m1 ?x1) = 0
  proof -
    from A2 have ?m1 < n by (simp add: loc-upb-lm-2-5)
    then have ((?m1, ?x1), n, x) ∈ lex-p by (simp add: lex-p-eq)
    with A1 upb1-def have c-assoc-have-key (pr-gr upb1) (c-pair ?m1 ?x1) = 0
by auto
    also have upb1 ≤ ?s by (rule arg2-le-c-pair)
    ultimately show ?thesis by (rule lm5)
  qed
  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 = univ-for-pr (c-pair n (c-pair ?y2 ?x1))
  let ?t2 = c-pair (c-pair ?y2 ?t1) ?x1
  have S1-1: c-assoc-have-key (pr-gr (loc-upb n (c-pair ?y2 ?x1))) (c-pair n (c-pair
  ?y2 ?x1)) = 0

```

proof –
from A **have** $?y2 < ?y1$ **by** *auto*
then have $c\text{-pair } ?y2 \ ?x1 < c\text{-pair } ?y1 \ ?x1$ **by** (*rule c-pair-strict-mono1*)
then have $((n, c\text{-pair } ?y2 \ ?x1), n, x) \in \text{lex-p}$ **by** (*simp add: lex-p-eq*)
with $A1$ **show** $?thesis$ **by** *auto*
qed
have $S2: c\text{-assoc-value } (pr\text{-gr } (loc\text{-upb } n \ (c\text{-pair } ?y2 \ ?x1))) \ (c\text{-pair } n \ (c\text{-pair } ?y2 \ ?x1)) = univ\text{-for-pr } (c\text{-pair } n \ (c\text{-pair } ?y2 \ ?x1))$
proof –
have $c\text{-is-sub-fun } (pr\text{-gr } (loc\text{-upb } n \ (c\text{-pair } ?y2 \ ?x1))) \ univ\text{-for-pr}$ **by** (*rule pr-gr-1*)
with $S1\text{-1}$ **show** $?thesis$ **by** (*simp add: c-is-sub-fun-lm-1*)
qed
from $S1 \ S2$ **have** $S3: loc\text{-upb } n \ x = c\text{-pair } (c\text{-pair } n \ x) \ (loc\text{-upb } n \ (c\text{-pair } ?y2 \ ?x1) + loc\text{-upb } ?m2 \ ?t2) + 1$ **by** (*simp del: loc-upb.simps add: Let-def*)
let $?s = c\text{-pair } (c\text{-pair } n \ x) \ (loc\text{-upb } n \ (c\text{-pair } ?y2 \ ?x1) + loc\text{-upb } ?m2 \ ?t2)$
from $S3$ **have** $S4: pr\text{-gr } (loc\text{-upb } n \ x) = pr\text{-gr } (Suc \ ?s)$ **by** (*simp del: loc-upb.simps*)
have $pr\text{-gr } (Suc \ ?s) = g\text{-step } (pr\text{-gr } ?s) \ (c\text{-fst } ?s)$ **by** (*rule pr-gr-at-Suc*)
with $S4$ **have** $S5: pr\text{-gr } (loc\text{-upb } n \ x) = g\text{-step } (pr\text{-gr } ?s) \ ?key$ **by** (*simp del: loc-upb.simps*)
let $?ls = pr\text{-gr } ?s$
from $A2$ **have** $g\text{-step } ?ls \ ?key = g\text{-rec } ?ls \ ?key$ **by** (*simp add: g-step-def*)
with $S5$ **have** $S6: pr\text{-gr } (loc\text{-upb } n \ x) = g\text{-rec } ?ls \ ?key$ **by** (*simp del: loc-upb.simps*)
have $S7: c\text{-assoc-have-key } ?ls \ (c\text{-pair } n \ (c\text{-pair } ?y2 \ ?x1)) = 0$
proof –
have $loc\text{-upb } n \ (c\text{-pair } ?y2 \ ?x1) \leq loc\text{-upb } n \ (c\text{-pair } ?y2 \ ?x1) + loc\text{-upb } ?m2 \ ?t2$ **by** (*auto simp del: loc-upb.simps*)
also have $loc\text{-upb } n \ (c\text{-pair } ?y2 \ ?x1) + loc\text{-upb } ?m2 \ ?t2 \leq ?s$ **by** (*rule arg2-le-c-pair*)
ultimately have $S7\text{-1}: loc\text{-upb } n \ (c\text{-pair } ?y2 \ ?x1) \leq ?s$ **by** (*auto simp del: loc-upb.simps*)
from $S1\text{-1} \ S7\text{-1}$ **show** $?thesis$ **by** (*rule lm5*)
qed
have $S8: c\text{-assoc-value } ?ls \ (c\text{-pair } n \ (c\text{-pair } ?y2 \ ?x1)) = ?t1$
proof –
have $c\text{-is-sub-fun } ?ls \ univ\text{-for-pr}$ **by** (*rule pr-gr-1*)
with $S7$ **show** $?thesis$ **by** (*simp add: c-is-sub-fun-lm-1*)
qed
have $S9: c\text{-assoc-have-key } ?ls \ (c\text{-pair } ?m2 \ ?t2) = 0$
proof –
from $A2$ **have** $?m2 < n$ **by** (*simp add: loc-upb-lm-2-6*)
then have $((?m2, ?t2), n, x) \in \text{lex-p}$ **by** (*simp add: lex-p-eq*)
with $A1$ **have** $c\text{-assoc-have-key } (pr\text{-gr } (loc\text{-upb } ?m2 \ ?t2)) \ (c\text{-pair } ?m2 \ ?t2) = 0$ **by** *auto*
also have $loc\text{-upb } ?m2 \ ?t2 \leq ?s$
proof –
have $loc\text{-upb } ?m2 \ ?t2 \leq loc\text{-upb } n \ (c\text{-pair } ?y2 \ ?x1) + loc\text{-upb } ?m2 \ ?t2$ **by** (*auto simp del: loc-upb.simps*)
also have $loc\text{-upb } n \ (c\text{-pair } ?y2 \ ?x1) + loc\text{-upb } ?m2 \ ?t2 \leq ?s$ **by** (*rule*

```

arg2-le-c-pair)
  ultimately show ?thesis by (auto simp del: loc-upb.simps)
qed
  ultimately show ?thesis by (rule lm5)
qed
  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
qed

lemma wf-upb-step-0:
  
$$\llbracket \bigwedge n' x'. ((n', x'), (n, x)) \in \text{lex-p} \implies \text{c-assoc-have-key} (\text{pr-gr} (\text{loc-upb } n' x')) (\text{c-pair } n' x') = 0 \rrbracket \implies$$

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

proof -
  assume A1:  $\bigwedge n' x'. ((n', x'), (n, x)) \in \text{lex-p} \implies \text{c-assoc-have-key} (\text{pr-gr} (\text{loc-upb } n' x')) (\text{c-pair } n' x') = 0$ 
  let ?n1 = (c-fst n) mod 7
  have S1: ?n1 = 0  $\implies$  ?thesis
  proof -
    assume A: ?n1 = 0
    thus ?thesis by (rule loc-upb-lex-0)
  qed
  have S2: ?n1 = 1  $\implies$  ?thesis
  proof -
    assume A: ?n1 = 1
    thus ?thesis by (rule loc-upb-lex-1)
  qed
  have S3: ?n1 = 2  $\implies$  ?thesis
  proof -
    assume A: ?n1 = 2
    thus ?thesis by (rule loc-upb-lex-2)
  qed
  have S4: ?n1 = 3  $\implies$  ?thesis
  proof -
    assume A: ?n1 = 3
    thus ?thesis by (rule loc-upb-lex-3)
  qed
  have S5: ?n1 = 4  $\implies$  ?thesis
  proof -
    assume A: ?n1 = 4
    from A1 A show ?thesis by (rule loc-upb-lex-4)
  qed
  have S6: ?n1 = 5  $\implies$  ?thesis
  proof -
    assume A: ?n1 = 5
    from A1 A show ?thesis by (rule loc-upb-lex-5)
  qed
qed

```

have $S7: ?n1 = 6 \implies ?thesis$
proof –
 assume $A: ?n1 = 6$
 from $A1 A$ **show** $?thesis$ **by** (rule loc-upb-lex-6)
qed
have $S8: ?n1=0 \vee ?n1=1 \vee ?n1=2 \vee ?n1=3 \vee ?n1=4 \vee ?n1=5 \vee ?n1=6$
by (rule mod7-lm)
from $S1 S2 S3 S4 S5 S6 S7 S8$ **show** $?thesis$ **by** fast
qed

lemma *wf-upb-step*:

assumes $A1: \bigwedge p2. (p2, p1) \in lex-p \implies$
 $c-assoc-have-key (pr-gr (loc-upb (fst p2) (snd p2))) (c-pair (fst p2) (snd p2))$
 $= 0$
shows $c-assoc-have-key (pr-gr (loc-upb (fst p1) (snd p1))) (c-pair (fst p1) (snd p1)) = 0$
proof –
 let $?n = fst p1$
 let $?x = snd p1$
 from $A1$ **have** $S1: \bigwedge p2. (p2, (?n, ?x)) \in lex-p \implies$
 $c-assoc-have-key (pr-gr (loc-upb (fst p2) (snd p2))) (c-pair (fst p2) (snd p2))$
 $= 0$
 by auto
 have $S2: (\bigwedge n' x'. ((n', x'), (fst p1, snd p1)) \in lex-p$
 $\implies c-assoc-have-key (pr-gr (loc-upb n' x') (c-pair n' x') = 0) \implies$
 $c-assoc-have-key (pr-gr (loc-upb (fst p1) (snd p1))) (c-pair (fst p1) (snd p1))$
 $= 0$
 by (rule wf-upb-step-0)
 then have $S3: (\bigwedge n' x'. ((n', x'), p1) \in lex-p \implies c-assoc-have-key (pr-gr (loc-upb$
 $n' x') (c-pair n' x') = 0)$
 $\implies c-assoc-have-key (pr-gr (loc-upb (fst p1) (snd p1))) (c-pair (fst p1) (snd$
 $p1)) = 0$ **by** auto
 have $S4: \bigwedge n' x'. ((n', x'), p1) \in lex-p \implies c-assoc-have-key (pr-gr (loc-upb n'$
 $x') (c-pair n' x') = 0$
 proof –
 fix $n' x'$
 assume $A4-1: ((n', x'), p1) \in lex-p$
 let $?p2 = (n', x')$
 from $A4-1$ **have** $S4-1: (?p2, p1) \in lex-p$ **by** auto
 from $S4-1$ **have** $c-assoc-have-key (pr-gr (loc-upb (fst ?p2) (snd ?p2))) (c-pair$
 $(fst ?p2) (snd ?p2)) = 0$
 by (rule A1)
 then show $c-assoc-have-key (pr-gr (loc-upb n' x') (c-pair n' x') = 0$ **by** auto
qed
from $S4 S3$ **show** $?thesis$ **by** auto
qed

theorem *loc-upb-main*: $c-assoc-have-key (pr-gr (loc-upb n x) (c-pair n x) = 0$
proof –

have *loc-upb-lm*: $\bigwedge p. c\text{-assoc-have-key } (pr\text{-gr } (loc\text{-upb } (fst\ p) (snd\ p))) (c\text{-pair } (fst\ p) (snd\ p)) = 0$
proof – **fix** *p* **show** *c-assoc-have-key* (*pr-gr* (*loc-upb* (*fst p*) (*snd p*))) (*c-pair* (*fst p*) (*snd p*)) = 0
proof –
have *S1*: *wf lex-p* **by** (*auto simp add: lex-p-def*)
from *S1 wf-upb-step* **show** *?thesis* **by** (*rule wf-induct-rule*)
qed
qed
let *?p* = (*n,x*)
have *c-assoc-have-key* (*pr-gr* (*loc-upb* (*fst ?p*) (*snd ?p*))) (*c-pair* (*fst ?p*) (*snd ?p*)) = 0 **by** (*rule loc-upb-lm*)
thus *?thesis* **by** *simp*
qed

theorem *pr-gr-value*: *c-assoc-value* (*pr-gr* (*loc-upb n x*)) (*c-pair n x*) = *univ-for-pr* (*c-pair n x*)
by (*simp del: loc-upb.simps add: loc-upb-main pr-gr-1 c-is-sub-fun-lm-1*)

theorem *g-comp-is-pr*: *g-comp* \in *PrimRec2*
proof –
from *c-assoc-have-key-is-pr c-assoc-value-is-pr c-cons-is-pr* **have** ($\lambda x y. g\text{-comp } x y$) \in *PrimRec2*
unfolding *g-comp-def Let-def* **by** *prec*
thus *?thesis* **by** *auto*
qed

theorem *g-pair-is-pr*: *g-pair* \in *PrimRec2*
proof –
from *c-assoc-have-key-is-pr c-assoc-value-is-pr c-cons-is-pr* **have** ($\lambda x y. g\text{-pair } x y$) \in *PrimRec2*
unfolding *g-pair-def Let-def* **by** *prec*
thus *?thesis* **by** *auto*
qed

theorem *g-rec-is-pr*: *g-rec* \in *PrimRec2*
proof –
from *c-assoc-have-key-is-pr c-assoc-value-is-pr c-cons-is-pr* **have** ($\lambda x y. g\text{-rec } x y$) \in *PrimRec2*
unfolding *g-rec-def Let-def* **by** *prec*
thus *?thesis* **by** *auto*
qed

theorem *g-step-is-pr*: *g-step* \in *PrimRec2*
proof –
from *g-comp-is-pr g-pair-is-pr g-rec-is-pr mod-is-pr c-assoc-have-key-is-pr c-assoc-value-is-pr c-cons-is-pr* **have**
($\lambda ls\ key. g\text{-step } ls\ key$) \in *PrimRec2* **unfolding** *g-step-def Let-def* **by** *prec*
thus *?thesis* **by** *auto*

qed

theorem *pr-gr-is-pr*: $pr-gr \in PrimRec1$

proof –

have $S1: (\lambda x. pr-gr\ x) = PrimRecOp1\ 0\ (\lambda x\ y. g-step\ y\ (c-fst\ x))$ (**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 $S2: PrimRecOp1\ 0\ (\lambda x\ y. g-step\ y\ (c-fst\ x)) \in PrimRec1$

proof (*rule pr-rec1*)

from *g-step-is-pr* show $(\lambda x\ y. g-step\ y\ (c-fst\ x)) \in PrimRec2$ **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 :: $(nat \Rightarrow nat \Rightarrow nat) \Rightarrow nat\ set$ **where**

fn-to-set $f = \{ x. \exists y. f\ x\ y = 0 \}$

definition

ce-sets :: $(nat\ set)\ set$ **where**

ce-sets = $\{ (fn-to-set\ p) \mid p. p \in PrimRec2 \}$

7.2 Basic properties of computably enumerable sets

lemma *ce-set-lm-1*: $p \in PrimRec2 \implies fn-to-set\ p \in ce-sets$ **by** (*auto simp add: ce-sets-def*)

lemma *ce-set-lm-2*: $\llbracket p \in PrimRec2; \forall x. (x \in A) = (\exists y. p\ x\ y = 0) \rrbracket \implies A \in ce-sets$

proof –

assume *p-is-pr*: $p \in PrimRec2$

assume $\forall x. (x \in A) = (\exists y. p\ x\ y = 0)$

then have $A = fn-to-set\ p$ **by** (*unfold fn-to-set-def, auto*)

with *p-is-pr* show $A \in ce-sets$ **by** (*simp add: ce-set-lm-1*)

qed

lemma *ce-set-lm-3*: $A \in ce\text{-sets} \implies \exists p \in PrimRec2. A = fn\text{-to-set } p$

proof –

assume $A \in ce\text{-sets}$

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

thus *?thesis* **by** *auto*

qed

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

proof –

assume $A \in ce\text{-sets}$

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

then obtain p **where** $p\text{-is-pr}: p \in PrimRec2$ **and** $L1: A = fn\text{-to-set } p$..

from $p\text{-is-pr } L1$ **show** *?thesis* **by** (*unfold fn-to-set-def, auto*)

qed

lemma *ce-set-lm-5*: $\llbracket A \in ce\text{-sets}; p \in PrimRec1 \rrbracket \implies \{ x . p\ x \in A \} \in ce\text{-sets}$

proof –

assume $A1: A \in ce\text{-sets}$

assume $A2: p \in PrimRec1$

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

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

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

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

from $pA\text{-is-pr } A2$ **have** $q\text{-is-pr}: q \in PrimRec2$ **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

assume $A: p\ x \in A$

with $S2$ **obtain** y **where** $L1: pA\ (p\ x)\ y = 0$ **by** *auto*

then have $q\ x\ y = 0$ **by** (*simp add: q-def*)

thus $\exists y. q\ x\ y = 0$..

next

assume $A: \exists y. q\ x\ y = 0$

then obtain y **where** $L1: q\ x\ y = 0$..

then have $pA\ (p\ x)\ y = 0$ **by** (*simp add: q-def*)

with $S2$ **show** $p\ x \in A$ **by** *auto*

qed

qed

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

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

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

ultimately show *?thesis* **by** *auto*

qed

lemma *ce-set-lm-6*: $\llbracket A \in ce\text{-sets}; A \neq \{\} \rrbracket \implies \exists q \in PrimRec1. A = \{ q\ x \mid x. x \in UNIV \}$

proof –

```

assume A1:  $A \in ce\text{-sets}$ 
assume A2:  $A \neq \{\}$ 
from A1 have  $\exists pA \in PrimRec2. A = fn\text{-to-set } pA$  by (rule ce-set-lm-3)
then obtain pA where pA-is-pr:  $pA \in PrimRec2$  and S1:  $A = fn\text{-to-set } pA$  ..
from S1 have S2:  $A = \{ x. \exists y. pA \ x \ y = 0 \}$  by (simp add: fn-to-set-def)
from A2 obtain a where a-in:  $a \in A$  by auto
define q where  $q \ z = (if \ pA \ (c\text{-fst } z) \ (c\text{-snd } z) = 0 \ then \ c\text{-fst } z \ else \ a)$  for z
from pA-is-pr have q-is-pr:  $q \in PrimRec1$  unfolding q-def by prec
have S3:  $\forall z. q \ z \in A$ 
proof
  fix z show  $q \ z \in A$ 
  proof cases
    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-def have  $q \ z = c\text{-fst } z$  by simp
    ultimately show  $q \ z \in A$  by auto
  next
    assume A:  $pA \ (c\text{-fst } z) \ (c\text{-snd } z) \neq 0$ 
    with q-def have  $q \ z = a$  by simp
    with a-in show  $q \ z \in A$  by auto
  qed
qed
then have S4:  $\{ q \ x \mid x. x \in UNIV \} \subseteq A$  by auto
have S5:  $A \subseteq \{ q \ x \mid x. x \in UNIV \}$ 
proof
  fix x assume A:  $x \in A$  show  $x \in \{ q \ x \mid x. x \in UNIV \}$ 
  proof
    from A S2 obtain y where L1:  $pA \ x \ y = 0$  by auto
    let ?z = c-pair x y
    from L1 have  $q \ ?z = x$  by (simp add: q-def)
    then have  $\exists u. q \ u = x$  by blast
    then show  $\exists u. x = q \ u \wedge u \in UNIV$  by auto
  qed
qed
from S4 S5 have S6:  $A = \{ q \ x \mid x. x \in UNIV \}$  by auto
with q-is-pr show ?thesis by blast
qed

lemma ce-set-lm-7:  $\llbracket A \in ce\text{-sets}; p \in PrimRec1 \rrbracket \implies \{ p \ x \mid x. x \in A \} \in ce\text{-sets}$ 
proof –
  assume A1:  $A \in ce\text{-sets}$ 
  assume A2:  $p \in PrimRec1$ 
  let ?B =  $\{ p \ x \mid x. x \in A \}$ 
  fix y have S1:  $(y \in ?B) = (\exists x. x \in A \wedge (y = p \ x))$  by auto
  from A1 have  $\exists pA \in PrimRec2. A = fn\text{-to-set } pA$  by (rule ce-set-lm-3)
  then obtain pA where pA-is-pr:  $pA \in PrimRec2$  and S2:  $A = fn\text{-to-set } pA$  ..
  from S2 have S3:  $A = \{ x. \exists y. pA \ x \ y = 0 \}$  by (simp add: fn-to-set-def)
  define q where  $q \ y \ t = (if \ y = p \ (c\text{-snd } t) \ then \ pA \ (c\text{-snd } t) \ (c\text{-fst } t) \ else \ 1)$ 
for y t

```

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 ∈ ?B*
 then obtain *x0* **where** *LL-2: x0 ∈ A* **and** *LL-3: y = p x0* **by** *auto*
 from *S3* **have** *LL-4: (x0 ∈ A) = (∃ 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: ∃ z. q y z = 0*
 then obtain *z0* **where** *LL-1: q y z0 = 0* ..
 have *LL2: y = p (c-snd z0)*
 proof (*rule ccontr*)
 assume *y ≠ p (c-snd z0)*
 with *q-def LL-1* **have** *q y z0 = 1* **by** *auto*
 with *LL-1* **show** *False* **by** *auto*
 qed
 from *LL2 LL-1 q-def* **have** *LL3: pA (c-snd z0) (c-fst z0) = 0* **by** *auto*
 with *S3* **have** *LL4: c-snd z0 ∈ A* **by** *auto*
 with *LL2* **show** $y \in \{p\ x \mid x. x \in A\}$ **by** *auto*
qed
qed
 then have *L2: ?B = { y | y. ∃ z. q y z = 0 }* **by** *auto*
 with *fn-to-set-def* **have** *?B = fn-to-set q* **by** *auto*
 with *q-is-pr ce-set-lm-1* **show** *?thesis* **by** *auto*
qed

theorem *ce-empty: {} ∈ ce-sets*
proof –
 let *?f = (λ x a. (1::nat))*
 have *S1: ?f ∈ PrimRec2* **by** (*rule const-is-pr-2*)
 then have $\forall x a. ?f\ x\ a \neq 0$ **by** *simp*
 then have $\{x. \exists a. ?f\ x\ a = 0\} = \{\}$ **by** *auto*
 also have *fn-to-set ?f = ...* **by** (*simp add: fn-to-set-def*)
 with *S1* **show** *?thesis* **by** (*auto simp add: ce-sets-def*)
qed

theorem *ce-univ: UNIV ∈ ce-sets*
proof –
 let *?f = (λ x a. (0::nat))*
 have *S1: ?f ∈ PrimRec2* **by** (*rule const-is-pr-2*)
 then have $\forall x a. ?f\ x\ a = 0$ **by** *simp*
 then have $\{x. \exists a. ?f\ x\ a = 0\} = UNIV$ **by** *auto*
 also have *fn-to-set ?f = ...* **by** (*simp add: fn-to-set-def*)
 with *S1* **show** *?thesis* **by** (*auto simp add: ce-sets-def*)
qed

theorem *ce-singleton*: $\{a\} \in ce\text{-sets}$

proof –

let $?f = \lambda x y. (abs\text{-of-diff } x a) + y$
have $S1: ?f \in PrimRec2$ **using** *const-is-pr-2* [**where** $?n=a$] **by** *prec*
then have $\forall x y. (?f x y = 0) = (x=a \wedge y=0)$ **by** (*simp add: abs-of-diff-eq*)
then have $S2: \{x. \exists y. ?f x y = 0\} = \{a\}$ **by** *auto*
have *fn-to-set* $?f = \{x. \exists y. ?f x y = 0\}$ **by** (*simp add: fn-to-set-def*)
with $S2$ have *fn-to-set* $?f = \{a\}$ **by** *simp*
with $S1$ **show** *?thesis* **by** (*auto simp add: ce-sets-def*)

qed

theorem *ce-union*: $[A \in ce\text{-sets}; B \in ce\text{-sets}] \implies A \cup B \in ce\text{-sets}$

proof –

assume $A1: A \in ce\text{-sets}$
then obtain $p\text{-}a$ **where** $S2: p\text{-}a \in PrimRec2$ **and** $S3: A = \text{fn-to-set } p\text{-}a$
by (*auto simp add: ce-sets-def*)
assume $A2: B \in ce\text{-sets}$
then obtain $p\text{-}b$ **where** $S5: p\text{-}b \in PrimRec2$ **and** $S6: B = \text{fn-to-set } p\text{-}b$
by (*auto simp add: ce-sets-def*)
let $?p = (\lambda x y. (p\text{-}a x y) * (p\text{-}b x y))$
from $S2 S5$ have $S7: ?p \in PrimRec2$ **by** *prec*
have $S8: \forall x y. (?p x y = 0) = ((p\text{-}a x y = 0) \vee (p\text{-}b x y = 0))$ **by** *simp*
let $?C = \text{fn-to-set } ?p$
have $S9: ?C = \{x. \exists y. ?p x y = 0\}$ **by** (*simp add: fn-to-set-def*)
from $S3$ have $S10: A = \{x. \exists y. p\text{-}a x y = 0\}$ **by** (*simp add: fn-to-set-def*)
from $S6$ have $S11: B = \{x. \exists y. p\text{-}b x y = 0\}$ **by** (*simp add: fn-to-set-def*)
from $S10 S11 S9 S8$ have $S12: ?C = A \cup B$ **by** *auto*
from $S7$ have $?C \in ce\text{-sets}$ **by** (*auto simp add: ce-sets-def*)
with $S12$ **show** *?thesis* **by** *simp*

qed

theorem *ce-intersect*: $[A \in ce\text{-sets}; B \in ce\text{-sets}] \implies A \cap B \in ce\text{-sets}$

proof –

assume $A1: A \in ce\text{-sets}$
then obtain $p\text{-}a$ **where** $S2: p\text{-}a \in PrimRec2$ **and** $S3: A = \text{fn-to-set } p\text{-}a$
by (*auto simp add: ce-sets-def*)
assume $A2: B \in ce\text{-sets}$
then obtain $p\text{-}b$ **where** $S5: p\text{-}b \in PrimRec2$ **and** $S6: B = \text{fn-to-set } p\text{-}b$
by (*auto simp add: ce-sets-def*)
let $?p = (\lambda x y. (p\text{-}a x (c\text{-fst } y)) + (p\text{-}b x (c\text{-snd } y)))$
from $S2 S5$ have $S7: ?p \in PrimRec2$ **by** *prec*
have $S8: \forall x. (\exists y. ?p x y = 0) = ((\exists z. p\text{-}a x z = 0) \wedge (\exists z. p\text{-}b x z = 0))$
proof
fix x **show** $(\exists y. ?p x y = 0) = ((\exists z. p\text{-}a x z = 0) \wedge (\exists z. p\text{-}b x z = 0))$
proof –
have $1: (\exists y. ?p x y = 0) \implies ((\exists z. p\text{-}a x z = 0) \wedge (\exists z. p\text{-}b x z = 0))$
by *blast*
have $2: ((\exists z. p\text{-}a x z = 0) \wedge (\exists z. p\text{-}b x z = 0)) \implies (\exists y. ?p x y = 0)$

proof –
assume $((\exists z. p\text{-}a\ x\ z = 0) \wedge (\exists z. p\text{-}b\ x\ z = 0))$
then obtain $z1\ z2$ **where** $s\text{-}23: p\text{-}a\ x\ z1 = 0$ **and** $s\text{-}24: p\text{-}b\ x\ z2 = 0$ **by**
auto
let $?y1 = c\text{-}pair\ z1\ z2$
from $s\text{-}23$ **have** $s\text{-}25: p\text{-}a\ x\ (c\text{-}fst\ ?y1) = 0$ **by** *simp*
from $s\text{-}24$ **have** $s\text{-}26: p\text{-}b\ x\ (c\text{-}snd\ ?y1) = 0$ **by** *simp*
from $s\text{-}25\ s\text{-}26$ **have** $s\text{-}27: p\text{-}a\ x\ (c\text{-}fst\ ?y1) + p\text{-}b\ x\ (c\text{-}snd\ ?y1) = 0$ **by** *simp*
then show $?thesis$ **..**
qed
from $1\ 2$ **have** $(\exists y. ?p\ x\ y = 0) = ((\exists z. p\text{-}a\ x\ z = 0) \wedge (\exists z. p\text{-}b\ x\ z = 0))$
by (*rule iffI*)
then show $?thesis$ **by** *auto*
qed
qed
let $?C = fn\text{-}to\text{-}set\ ?p$
have $S9: ?C = \{x. \exists y. ?p\ x\ y = 0\}$ **by** (*simp add: fn-to-set-def*)
from $S3$ **have** $S10: A = \{x. \exists y. p\text{-}a\ x\ y = 0\}$ **by** (*simp add: fn-to-set-def*)
from $S6$ **have** $S11: B = \{x. \exists y. p\text{-}b\ x\ y = 0\}$ **by** (*simp add: fn-to-set-def*)
from $S10\ S11\ S9\ S8$ **have** $S12: ?C = A \cap B$ **by** *auto*
from $S7$ **have** $?C \in ce\text{-}sets$ **by** (*auto simp add: ce-sets-def*)
with $S12$ **show** $?thesis$ **by** *simp*
qed

7.3 Enumeration of computably enumerable sets

definition

$nat\text{-}to\text{-}ce\text{-}set :: nat \Rightarrow (nat\ set)$ **where**
 $nat\text{-}to\text{-}ce\text{-}set = (\lambda n. fn\text{-}to\text{-}set\ (pr\text{-}conv\text{-}1\text{-}to\text{-}2\ (nat\text{-}to\text{-}pr\ n)))$

lemma $nat\text{-}to\text{-}ce\text{-}set\text{-}lm\text{-}1: nat\text{-}to\text{-}ce\text{-}set\ n = \{x . \exists y. (nat\text{-}to\text{-}pr\ n)\ (c\text{-}pair\ x\ y) = 0\}$

proof –

have $S1: nat\text{-}to\text{-}ce\text{-}set\ n = fn\text{-}to\text{-}set\ (pr\text{-}conv\text{-}1\text{-}to\text{-}2\ (nat\text{-}to\text{-}pr\ n))$ **by** (*simp add: nat-to-ce-set-def*)

then have $S2: nat\text{-}to\text{-}ce\text{-}set\ n = \{x . \exists y. (pr\text{-}conv\text{-}1\text{-}to\text{-}2\ (nat\text{-}to\text{-}pr\ n))\ x\ y = 0\}$ **by** (*simp add: fn-to-set-def*)

have $S3: \bigwedge x\ y. (pr\text{-}conv\text{-}1\text{-}to\text{-}2\ (nat\text{-}to\text{-}pr\ n))\ x\ y = (nat\text{-}to\text{-}pr\ n)\ (c\text{-}pair\ x\ y)$
by (*simp add: pr-conv-1-to-2-def*)

from $S2\ S3$ **show** $?thesis$ **by** *auto*

qed

lemma $nat\text{-}to\text{-}ce\text{-}set\text{-}into\text{-}ce: nat\text{-}to\text{-}ce\text{-}set\ n \in ce\text{-}sets$

proof –

have $S1: nat\text{-}to\text{-}ce\text{-}set\ n = fn\text{-}to\text{-}set\ (pr\text{-}conv\text{-}1\text{-}to\text{-}2\ (nat\text{-}to\text{-}pr\ n))$ **by** (*simp add: nat-to-ce-set-def*)

have $(nat\text{-}to\text{-}pr\ n) \in PrimRec1$ **by** (*rule nat-to-pr-into-pr*)

then have $S2: (pr\text{-}conv\text{-}1\text{-}to\text{-}2\ (nat\text{-}to\text{-}pr\ n)) \in PrimRec2$ **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*: $A \in ce\text{-sets} \implies \exists n. A = nat\text{-to-ce-set } n$

proof –

assume $A: A \in ce\text{-sets}$

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

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

define q **where** $q = pr\text{-conv-2-to-1 } p$

from $p\text{-is-pr}$ **have** $q\text{-is-pr}: q \in PrimRec1$ **by** (*unfold q-def, rule pr-conv-2-to-1-lm*)

from $q\text{-def}$ **have** $S2: pr\text{-conv-1-to-2 } q = p$ **by** *simp*

let $?n = index\text{-of-pr } q$

from $q\text{-is-pr}$ **have** $nat\text{-to-pr } ?n = q$ **by** (*rule index-of-pr-is-real*)

with $S2$ $S1$ **have** $A = fn\text{-to-set } (pr\text{-conv-1-to-2 } (nat\text{-to-pr } ?n))$ **by** *auto*

then have $A = nat\text{-to-ce-set } ?n$ **by** (*simp add: nat-to-ce-set-def*)

thus *?thesis* ..

qed

7.4 Characteristic functions

definition

$chf :: nat\ set \Rightarrow (nat \Rightarrow nat)$ — Characteristic function **where**

$chf = (\lambda A\ x. \text{if } x \in A \text{ then } 0 \text{ else } 1)$

definition

$zero\text{-set} :: (nat \Rightarrow nat) \Rightarrow nat\ set$ **where**

$zero\text{-set} = (\lambda f. \{x. f\ x = 0\})$

lemma *chf-lm-1* [*simp*]: $zero\text{-set } (chf\ A) = A$ **by** (*unfold chf-def, unfold zero-set-def, simp*)

lemma *chf-lm-2*: $(x \in A) = (chf\ A\ x = 0)$ **by** (*unfold chf-def, simp*)

lemma *chf-lm-3*: $(x \notin A) = (chf\ A\ x = 1)$ **by** (*unfold chf-def, simp*)

lemma *chf-lm-4*: $chf\ A \in PrimRec1 \implies A \in ce\text{-sets}$

proof –

assume $A: chf\ A \in PrimRec1$

define p **where** $p = chf\ A$

from A $p\text{-def}$ **have** $p\text{-is-pr}: p \in PrimRec1$ **by** *auto*

define q **where** $q\ x\ y = p\ x$ **for** $x\ y :: nat$

from $p\text{-is-pr}$ **have** $q\text{-is-pr}: q \in PrimRec2$ **unfolding** $q\text{-def}$ **by** *prec*

have $S1: A = \{x. p(x) = 0\}$

proof –

have $zero\text{-set } p = A$ **by** (*unfold p-def, simp*)

thus *?thesis* **by** (*simp add: zero-set-def*)

qed

have $S2: fn\text{-to-set } q = \{x. \exists y. q\ x\ y = 0\}$ **by** (*simp add: fn-to-set-def*)

have $S3: \bigwedge x. (p\ x = 0) = (\exists y. q\ x\ y = 0)$ **by** (*unfold q-def, auto*)

then have $S4: \{x. p\ x = 0\} = \{x. \exists y. q\ x\ y = 0\}$ **by** *auto*

with $S1\ S2$ have $S5: fn\text{-to-set } q = A$ by *auto*
 from $q\text{-is-pr}$ have $fn\text{-to-set } q \in ce\text{-sets}$ by (rule $ce\text{-set-lm-1}$)
 with $S5$ show *?thesis* by *auto*
 qed

lemma $chf\text{-lm-5}: finite\ A \implies chf\ A \in PrimRec1$

proof –

assume $A: finite\ A$

define u where $u = set\text{-to-nat } A$

from A have $S1: nat\text{-to-set } u = A$ by (unfold $u\text{-def}$, rule $nat\text{-to-set-srj}$)

have $chf\ A = (\lambda x. sgn2\ (c\text{-in } x\ u))$

proof

fix x show $chf\ A\ x = sgn2\ (c\text{-in } x\ u)$

proof *cases*

assume $A: x \in A$

then have $S1\text{-1}: chf\ A\ x = 0$ by (simp add: $chf\text{-lm-2}$)

from $A\ S1$ have $x \in nat\text{-to-set } u$ by *auto*

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

with $S1\text{-1}$ show *?thesis* by *simp*

next

assume $A: x \notin A$

then have $S1\text{-1}: chf\ A\ x = 1$ by (simp add: $chf\text{-def}$)

from $A\ S1$ have $x \notin nat\text{-to-set } u$ by *auto*

then have $c\text{-in } x\ u = 0$ by (simp add: $x\text{-in-}u\text{-eq } c\text{-in-}def$)

with $S1\text{-1}$ show *?thesis* by *simp*

qed

qed

moreover from $c\text{-in-is-pr}$ have $(\lambda x. sgn2\ (c\text{-in } x\ u)) \in PrimRec1$ by *prec*

ultimately show *?thesis* by *auto*

qed

theorem $ce\text{-finite}: finite\ A \implies A \in ce\text{-sets}$

proof –

assume $A: finite\ A$

then have $chf\ A \in PrimRec1$ by (rule $chf\text{-lm-5}$)

then show *?thesis* by (rule $chf\text{-lm-4}$)

qed

7.5 Computably enumerable relations

definition

$ce\text{-set-to-rel} :: nat\ set \Rightarrow (nat * nat)\ set$ where

$ce\text{-set-to-rel} = (\lambda A. \{ (c\text{-fst } x, c\text{-snd } x) \mid x. x \in A \})$

definition

$ce\text{-rel-to-set} :: (nat * nat)\ set \Rightarrow nat\ set$ where

$ce\text{-rel-to-set} = (\lambda R. \{ c\text{-pair } x\ y \mid x\ y. (x,y) \in R \})$

definition

ce-rels :: ((nat * nat) set) set **where**
ce-rels = { R | R. *ce-rel-to-set* R ∈ *ce-sets* }

lemma *ce-rel-lm-1* [*simp*]: *ce-set-to-rel* (*ce-rel-to-set* r) = r

proof

show *ce-set-to-rel* (*ce-rel-to-set* r) ⊆ r

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 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

qed

lemma *ce-rel-lm-2* [*simp*]: *ce-rel-to-set* (*ce-set-to-rel* A) = A

proof

show *ce-rel-to-set* (*ce-set-to-rel* A) ⊆ A

proof fix z **show** z ∈ *ce-rel-to-set* (*ce-set-to-rel* A) ⇒ z ∈ A

proof –

assume A: z ∈ *ce-rel-to-set* (*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-rel-to-set-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

```

next
show  $A \subseteq ce\text{-rel-to-set } (ce\text{-set-to-rel } A)$ 
proof fix  $z$  show  $z \in A \implies z \in ce\text{-rel-to-set } (ce\text{-set-to-rel } A)$ 
  proof -
    assume  $A: z \in A$ 
    then have  $L1: (c\text{-fst } z, c\text{-snd } z) \in ce\text{-set-to-rel } A$  by (unfold ce-set-to-rel-def,
auto)
    define  $x$  where  $x = c\text{-fst } z$ 
    define  $y$  where  $y = c\text{-snd } z$ 
    from  $L1$  x-def y-def have  $L2: (x,y) \in ce\text{-set-to-rel } A$  by simp
    then have  $L3: c\text{-pair } x y \in ce\text{-rel-to-set } (ce\text{-set-to-rel } A)$  by (unfold ce-rel-to-set-def,
auto)
    with x-def y-def show  $z \in ce\text{-rel-to-set } (ce\text{-set-to-rel } A)$  by simp
  qed
qed
qed

```

```

lemma ce-rels-def1:  $ce\text{-rels} = \{ ce\text{-set-to-rel } A \mid A. A \in ce\text{-sets} \}$ 
proof
show  $ce\text{-rels} \subseteq \{ ce\text{-set-to-rel } A \mid A. A \in ce\text{-sets} \}$ 
proof fix  $r$  show  $r \in ce\text{-rels} \implies r \in \{ ce\text{-set-to-rel } A \mid A. A \in ce\text{-sets} \}$ 
  proof -
    assume  $A: r \in ce\text{-rels}$ 
    then have  $L1: ce\text{-rel-to-set } r \in ce\text{-sets}$  by (unfold ce-rels-def, auto)
    define  $A$  where  $A = ce\text{-rel-to-set } r$ 
    from A-def L1 have  $L2: A \in ce\text{-sets}$  by auto
    from A-def have  $L3: ce\text{-set-to-rel } A = r$  by simp
    with L2 show  $r \in \{ ce\text{-set-to-rel } A \mid A. A \in ce\text{-sets} \}$  by auto
  qed
qed

```

```

next
show  $\{ ce\text{-set-to-rel } A \mid A. A \in ce\text{-sets} \} \subseteq ce\text{-rels}$ 
proof fix  $r$  show  $r \in \{ ce\text{-set-to-rel } A \mid A. A \in ce\text{-sets} \} \implies r \in ce\text{-rels}$ 
  proof -
    assume  $A: r \in \{ ce\text{-set-to-rel } A \mid A. A \in ce\text{-sets} \}$ 
    then obtain  $A$  where  $L1: r = ce\text{-set-to-rel } A$  and  $L2: A \in ce\text{-sets}$  by auto
    from  $L1$  have  $ce\text{-rel-to-set } r = A$  by simp
    with  $L2$  show  $r \in ce\text{-rels}$  unfolding ce-rels-def by auto
  qed
qed
qed

```

```

lemma ce-rel-to-set-inj: inj ce-rel-to-set
proof (rule inj-on-inverseI)
  fix  $x$  assume  $A: (x::(nat \times nat) \text{ set}) \in UNIV$  show  $ce\text{-set-to-rel } (ce\text{-rel-to-set } x) = x$  by (rule ce-rel-lm-1)
qed

```

```

lemma ce-rel-to-set-srj: surj ce-rel-to-set

```

```

proof (rule surjI [where ?f=ce-set-to-rel])
  fix x show ce-rel-to-set (ce-set-to-rel x) = x by (rule ce-rel-lm-2)
qed

lemma ce-rel-to-set-bij: bij ce-rel-to-set
proof (rule bijI)
  show inj ce-rel-to-set by (rule ce-rel-to-set-inj)
next
  show surj ce-rel-to-set by (rule ce-rel-to-set-srj)
qed

lemma ce-set-to-rel-inj: inj ce-set-to-rel
proof (rule inj-on-inverseI)
  fix x assume A: (x::nat set) ∈ UNIV show ce-rel-to-set (ce-set-to-rel x) = x by
  (rule ce-rel-lm-2)
qed

lemma ce-set-to-rel-srj: surj ce-set-to-rel
proof (rule surjI [where ?f=ce-rel-to-set])
  fix x show ce-set-to-rel (ce-rel-to-set x) = x by (rule ce-rel-lm-1)
qed

lemma ce-set-to-rel-bij: bij ce-set-to-rel
proof (rule bijI)
  show inj ce-set-to-rel by (rule ce-set-to-rel-inj)
next
  show surj ce-set-to-rel by (rule ce-set-to-rel-srj)
qed

lemma ce-rel-lm-3: A ∈ ce-sets ⇒ ce-set-to-rel A ∈ ce-rels
proof –
  assume A: A ∈ ce-sets
  from A ce-rels-def1 show ?thesis by auto
qed

lemma ce-rel-lm-4: ce-set-to-rel A ∈ ce-rels ⇒ A ∈ ce-sets
proof –
  assume A: ce-set-to-rel A ∈ ce-rels
  from A show ?thesis by (unfold ce-rels-def, auto)
qed

lemma ce-rel-lm-5: (A ∈ ce-sets) = (ce-set-to-rel A ∈ ce-rels)
proof
  assume A ∈ ce-sets then show ce-set-to-rel A ∈ ce-rels by (rule ce-rel-lm-3)
next
  assume ce-set-to-rel A ∈ ce-rels then show A ∈ ce-sets by (rule ce-rel-lm-4)
qed

lemma ce-rel-lm-6: r ∈ ce-rels ⇒ ce-rel-to-set r ∈ ce-sets

```

proof –
 assume $A: r \in ce\text{-rels}$
 then show $?thesis$ **by** (*unfold ce-rels-def, auto*)
qed

lemma *ce-rel-lm-7*: $ce\text{-rel-to-set } r \in ce\text{-sets} \implies r \in ce\text{-rels}$
proof –
 assume $ce\text{-rel-to-set } r \in ce\text{-sets}$
 then show $?thesis$ **by** (*unfold ce-rels-def, auto*)
qed

lemma *ce-rel-lm-8*: $(r \in ce\text{-rels}) = (ce\text{-rel-to-set } r \in ce\text{-sets})$ **by** (*unfold ce-rels-def, auto*)

lemma *ce-rel-lm-9*: $(x,y) \in r \implies c\text{-pair } x\ y \in ce\text{-rel-to-set } r$ **by** (*unfold ce-rel-to-set-def, auto*)

lemma *ce-rel-lm-10*: $x \in A \implies (c\text{-fst } x, c\text{-snd } x) \in ce\text{-set-to-rel } A$ **by** (*unfold ce-set-to-rel-def, auto*)

lemma *ce-rel-lm-11*: $c\text{-pair } x\ y \in ce\text{-rel-to-set } r \implies (x,y) \in r$
proof –
 assume $A: c\text{-pair } x\ y \in ce\text{-rel-to-set } r$
 let $?z = c\text{-pair } x\ y$
 from A **have** $(c\text{-fst } ?z, c\text{-snd } ?z) \in ce\text{-set-to-rel } (ce\text{-rel-to-set } r)$ **by** (*rule ce-rel-lm-10*)
 then show $(x,y) \in r$ **by** *simp*
qed

lemma *ce-rel-lm-12*: $(c\text{-pair } x\ y \in ce\text{-rel-to-set } r) = ((x,y) \in r)$
proof
 assume $c\text{-pair } x\ y \in ce\text{-rel-to-set } r$ **then show** $(x, y) \in r$ **by** (*rule ce-rel-lm-11*)
next
 assume $(x, y) \in r$ **then show** $c\text{-pair } x\ y \in ce\text{-rel-to-set } r$ **by** (*rule ce-rel-lm-9*)
qed

lemma *ce-rel-lm-13*: $(x,y) \in ce\text{-set-to-rel } A \implies c\text{-pair } x\ y \in A$
proof –
 assume $(x,y) \in ce\text{-set-to-rel } A$
 then have $c\text{-pair } x\ y \in ce\text{-rel-to-set } (ce\text{-set-to-rel } A)$ **by** (*rule ce-rel-lm-9*)
 then show $?thesis$ **by** *simp*
qed

lemma *ce-rel-lm-14*: $c\text{-pair } x\ y \in A \implies (x,y) \in ce\text{-set-to-rel } A$
proof –
 assume $c\text{-pair } x\ y \in A$
 then have $c\text{-pair } x\ y \in ce\text{-rel-to-set } (ce\text{-set-to-rel } A)$ **by** *simp*
 then show $?thesis$ **by** (*rule ce-rel-lm-11*)
qed

lemma *ce-rel-lm-15*: $((x,y) \in ce\text{-set-to-rel } A) = (c\text{-pair } x y \in A)$

proof

assume $(x, y) \in ce\text{-set-to-rel } A$ **then show** $c\text{-pair } x y \in A$ **by** (rule *ce-rel-lm-13*)

next

assume $c\text{-pair } x y \in A$ **then show** $(x, y) \in ce\text{-set-to-rel } A$ **by** (rule *ce-rel-lm-14*)

qed

lemma *ce-rel-lm-16*: $x \in ce\text{-rel-to-set } r \implies (c\text{-fst } x, c\text{-snd } x) \in r$

proof –

assume $x \in ce\text{-rel-to-set } r$

then have $(c\text{-fst } x, c\text{-snd } x) \in ce\text{-set-to-rel } (ce\text{-rel-to-set } r)$ **by** (rule *ce-rel-lm-10*)

then show *?thesis* **by simp**

qed

lemma *ce-rel-lm-17*: $(c\text{-fst } x, c\text{-snd } x) \in ce\text{-set-to-rel } A \implies x \in A$

proof –

assume $(c\text{-fst } x, c\text{-snd } x) \in ce\text{-set-to-rel } A$

then have $c\text{-pair } (c\text{-fst } x) (c\text{-snd } x) \in A$ **by** (rule *ce-rel-lm-13*)

then show *?thesis* **by simp**

qed

lemma *ce-rel-lm-18*: $((c\text{-fst } x, c\text{-snd } x) \in ce\text{-set-to-rel } A) = (x \in A)$

proof

assume $(c\text{-fst } x, c\text{-snd } x) \in ce\text{-set-to-rel } A$ **then show** $x \in A$ **by** (rule *ce-rel-lm-17*)

next

assume $x \in A$ **then show** $(c\text{-fst } x, c\text{-snd } x) \in ce\text{-set-to-rel } A$ **by** (rule *ce-rel-lm-10*)

qed

lemma *ce-rel-lm-19*: $(c\text{-fst } x, c\text{-snd } x) \in r \implies x \in ce\text{-rel-to-set } r$

proof –

assume $(c\text{-fst } x, c\text{-snd } x) \in r$

then have $(c\text{-fst } x, c\text{-snd } x) \in ce\text{-set-to-rel } (ce\text{-rel-to-set } r)$ **by simp**

then show *?thesis* **by** (rule *ce-rel-lm-17*)

qed

lemma *ce-rel-lm-20*: $((c\text{-fst } x, c\text{-snd } x) \in r) = (x \in ce\text{-rel-to-set } r)$

proof

assume $(c\text{-fst } x, c\text{-snd } x) \in r$ **then show** $x \in ce\text{-rel-to-set } r$ **by** (rule *ce-rel-lm-19*)

next

assume $x \in ce\text{-rel-to-set } r$ **then show** $(c\text{-fst } x, c\text{-snd } x) \in r$ **by** (rule *ce-rel-lm-16*)

qed

lemma *ce-rel-lm-21*: $r \in ce\text{-rels} \implies \exists p \in PrimRec3. \forall x y. ((x,y) \in r) = (\exists u. p x y u = 0)$

proof –

assume $r\text{-ce}: r \in ce\text{-rels}$

define A **where** $A = ce\text{-rel-to-set } r$

from $r\text{-ce}$ **have** $A\text{-ce}: A \in ce\text{-sets}$ **by** (unfold $A\text{-def}$, rule *ce-rel-lm-6*)

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

then obtain q **where** q -is-pr: $q \in \text{PrimRec2}$ **and** A -def1: $A = \text{fn-to-set } q \dots$
from A -def1 **have** A -def2: $A = \{ x. \exists y. q \ x \ y = 0 \}$ **by** (*unfold fn-to-set-def*)
define p **where** $p \ x \ y \ u = q \ (c\text{-pair } x \ y) \ u$ **for** $x \ y \ u$
from q -is-pr **have** p -is-pr: $p \in \text{PrimRec3}$ **unfolding** p -def **by** *prec*
have $\bigwedge x \ y. ((x,y) \in r) = (\exists u. p \ x \ y \ u = 0)$
proof – **fix** $x \ y$ **show** $((x,y) \in r) = (\exists u. p \ x \ y \ u = 0)$
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 -pair $x \ y \in A$ **by** (*unfold z-def*)
 then have c -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** ?thesis **by** *auto*
qed

lemma *ce-rel-lm-22*: $r \in \text{ce-rels} \implies \exists p \in \text{PrimRec3}. r = \{ (x,y). \exists u. p \ x \ y \ u = 0 \}$
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** ?thesis **by** *blast*
qed

lemma *ce-rel-lm-23*: $\llbracket p \in \text{PrimRec3}; \forall x \ y. ((x,y) \in r) = (\exists u. p \ x \ y \ u = 0) \rrbracket \implies r \in \text{ce-rels}$
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*)

```

have main: A = ce-rel-to-set r
proof
  show A ⊆ ce-rel-to-set r
  proof
    fix z assume z-in-A: z ∈ A
    show z ∈ ce-rel-to-set r
    proof –
      define x where x = c-fst z
      define y where y = c-snd z
      from z-in-A A-def obtain u where L2: q z u = 0 by auto
      with x-def y-def q-def have L3: p x y u = 0 by simp
      then have ∃ u. p x y u = 0 by auto
      with A have (x,y) ∈ r by auto
      then have c-pair x y ∈ ce-rel-to-set r by (rule ce-rel-lm-9)
      with x-def y-def show ?thesis by simp
    qed
  qed
next
show ce-rel-to-set r ⊆ A
proof
  fix z assume z-in-r: z ∈ ce-rel-to-set r
  show z ∈ A
  proof –
    define x where x = c-fst z
    define y where y = c-snd z
    from z-in-r have (c-fst z, c-snd z) ∈ r by (rule ce-rel-lm-16)
    with x-def y-def have (x,y) ∈ r by simp
    with A obtain u where L1: p x y u = 0 by auto
    with x-def y-def q-def have q z u = 0 by simp
    with A-def show z ∈ A by auto
  qed
qed
qed
with A-ce have ce-rel-to-set r ∈ ce-sets by auto
then show r ∈ ce-rels by (rule ce-rel-lm-7)
qed

lemma ce-rel-lm-24: [ [ r ∈ ce-rels; s ∈ ce-rels ] ] ⇒ s O r ∈ ce-rels
proof –
  assume r-ce: r ∈ ce-rels
  assume s-ce: s ∈ ce-rels
  from r-ce have ∃ p ∈ PrimRec3. ∀ x y. ((x,y) ∈ r)=(∃ u. p x y u = 0) by
(rule ce-rel-lm-21)
  then obtain p-r where p-r-is-pr: p-r ∈ PrimRec3 and R1: ∀ x y. ((x,y) ∈
r)=(∃ u. p-r x y u = 0)
  by auto
  from s-ce have ∃ p ∈ PrimRec3. ∀ x y. ((x,y) ∈ s)=(∃ u. p x y u = 0) by
(rule ce-rel-lm-21)
  then obtain p-s where p-s-is-pr: p-s ∈ PrimRec3 and S1: ∀ x y. ((x,y) ∈

```

$s) = (\exists u. p\text{-}s\ x\ y\ u = 0)$
by auto
define p where $p\ x\ z\ u = (p\text{-}s\ x\ (c\text{-}fst\ u)\ (c\text{-}fst\ (c\text{-}snd\ u))) + (p\text{-}r\ (c\text{-}fst\ u)\ z\ (c\text{-}snd\ (c\text{-}snd\ u)))$
for $x\ z\ u$
from $p\text{-}r\text{-}is\text{-}pr\ p\text{-}s\text{-}is\text{-}pr$ **have** $p\text{-}is\text{-}pr: p \in PrimRec3$ **unfolding** $p\text{-}def$ **by** $prec$
define sr **where** $sr = s\ O\ r$
have $main: \forall x\ z. ((x,z) \in sr) = (\exists u. p\ x\ z\ u = 0)$
proof ($rule\ allI, rule\ allI$)
fix $x\ z$
show $((x, z) \in sr) = (\exists u. p\ x\ z\ u = 0)$
proof
assume $A: (x, z) \in sr$
show $\exists u. p\ x\ z\ u = 0$
proof –
from $A\ sr\text{-}def$ **obtain** y **where** $L1: (x,y) \in s$ **and** $L2: (y,z) \in r$ **by** $auto$
from $L1\ S1$ **obtain** $u\text{-}s$ **where** $L3: p\text{-}s\ x\ y\ u\text{-}s = 0$ **by** $auto$
from $L2\ R1$ **obtain** $u\text{-}r$ **where** $L4: p\text{-}r\ y\ z\ u\text{-}r = 0$ **by** $auto$
define u **where** $u = c\text{-}pair\ y\ (c\text{-}pair\ u\text{-}s\ u\text{-}r)$
from $L3\ L4$ **have** $p\ x\ z\ u = 0$ **by** ($unfold\ p\text{-}def, unfold\ u\text{-}def, simp$)
then show $?thesis$ **by** $auto$
qed
next
assume $A: \exists u. p\ x\ z\ u = 0$
show $(x, z) \in sr$
proof –
from A **obtain** u **where** $L1: p\ x\ z\ u = 0$ **by** $auto$
then have $L2: (p\text{-}s\ x\ (c\text{-}fst\ u)\ (c\text{-}fst\ (c\text{-}snd\ u))) + (p\text{-}r\ (c\text{-}fst\ u)\ z\ (c\text{-}snd\ (c\text{-}snd\ u))) = 0$ **by** ($unfold\ p\text{-}def$)
from $L2$ **have** $L3: p\text{-}s\ x\ (c\text{-}fst\ u)\ (c\text{-}fst\ (c\text{-}snd\ u)) = 0$ **by** $auto$
from $L2$ **have** $L4: p\text{-}r\ (c\text{-}fst\ u)\ z\ (c\text{-}snd\ (c\text{-}snd\ u)) = 0$ **by** $auto$
from $L3\ S1$ **have** $L5: (x, (c\text{-}fst\ u)) \in s$ **by** $auto$
from $L4\ R1$ **have** $L6: ((c\text{-}fst\ u), z) \in r$ **by** $auto$
from $L5\ L6$ **have** $(x, z) \in s\ O\ r$ **by** $auto$
with $sr\text{-}def$ **show** $?thesis$ **by** $auto$
qed
qed
qed
from $p\text{-}is\text{-}pr\ main$ **have** $sr \in ce\text{-}rels$ **by** ($rule\ ce\text{-}rel\text{-}lm\text{-}23$)
then show $?thesis$ **by** ($unfold\ sr\text{-}def$)
qed

lemma $ce\text{-}rel\text{-}lm\text{-}25: r \in ce\text{-}rels \implies r^{\widehat{-}1} \in ce\text{-}rels$
proof –
assume $r\text{-}ce: r \in ce\text{-}rels$
have $r^{\widehat{-}1} = \{(y,x). (x,y) \in r\}$ **by** $auto$
then have $L1: \forall x\ y. ((x,y) \in r) = ((y,x) \in r^{\widehat{-}1})$ **by** $auto$
from $r\text{-}ce$ **have** $\exists p \in PrimRec3. \forall x\ y. ((x,y) \in r) = (\exists u. p\ x\ y\ u = 0)$ **by** ($rule\ ce\text{-}rel\text{-}lm\text{-}21$)

then obtain p **where** p -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 -is-pr **have** q -is-pr: $q \in \text{PrimRec3}$ **unfolding** q -def **by** *prec*
from $L1 R1$ **have** $L2: \forall x y. ((x,y) \in r^{-1}) = (\exists u. p y x u = 0)$ **by** *auto*
with q -def **have** $L3: \forall x y. ((x,y) \in r^{-1}) = (\exists u. q x y u = 0)$ **by** *auto*
with q -is-pr **show** *?thesis* **by** (rule *ce-rel-lm-23*)
qed

lemma *ce-rel-lm-26*: $r \in \text{ce-rels} \implies \text{Domain } r \in \text{ce-sets}$

proof –

assume r -ce: $r \in \text{ce-rels}$
have $L1: \forall x. (x \in \text{Domain } r) = (\exists y. (x,y) \in r)$ **by** *auto*
define A **where** $A = \text{ce-rel-to-set } r$
from r -ce **have** $\text{ce-rel-to-set } r \in \text{ce-sets}$ **by** (rule *ce-rel-lm-6*)
then have A -ce: $A \in \text{ce-sets}$ **by** (*unfold A-def*)
have $\forall x y. ((x,y) \in r) = (\text{c-pair } x y \in \text{ce-rel-to-set } r)$ **by** (*simp add: ce-rel-lm-12*)
then have $L2: \forall x y. ((x,y) \in r) = (\text{c-pair } x y \in A)$ **by** (*unfold A-def*)
from A -ce c-fst-is-pr **have** $L3: \{ \text{c-fst } z \mid z. z \in A \} \in \text{ce-sets}$ **by** (rule *ce-set-lm-7*)
have $L4: \forall x. (x \in \{ \text{c-fst } z \mid z. z \in A \}) = (\exists y. \text{c-pair } x y \in A)$
proof **fix** x **show** $(x \in \{ \text{c-fst } z \mid z. z \in A \}) = (\exists y. \text{c-pair } x y \in A)$

proof

assume $A: x \in \{ \text{c-fst } z \mid z. z \in A \}$
then obtain z **where** z -in- $A: z \in A$ **and** x - $z: x = \text{c-fst } z$ **by** *auto*
from x - z **have** $z = \text{c-pair } x (\text{c-snd } z)$ **by** *simp*
with z -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 -1: $\text{c-pair } x y \in A$ **by** *auto*
define z **where** $z = \text{c-pair } x y$
from y -1 **have** z -in- $A: z \in A$ **by** (*unfold z-def*)
from z -def **have** x - $z: x = \text{c-fst } z$ **by** (*unfold z-def, simp*)
from z -in- A x - z **show** $x \in \{ \text{c-fst } z \mid z. z \in A \}$ **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** $L6: \forall x. (x \in \text{Domain } r) = (x \in \{ \text{c-fst } z \mid z. z \in A \})$ **by** *auto*

then have $\text{Domain } r = \{ \text{c-fst } z \mid z. z \in A \}$ **by** *auto*

with $L3$ **show** $\text{Domain } r \in \text{ce-sets}$ **by** *auto*

qed

lemma *ce-rel-lm-27*: $r \in \text{ce-rels} \implies \text{Range } r \in \text{ce-sets}$

proof –

assume r -ce: $r \in \text{ce-rels}$
then have $r^{-1} \in \text{ce-rels}$ **by** (rule *ce-rel-lm-25*)
then have $\text{Domain } (r^{-1}) \in \text{ce-sets}$ **by** (rule *ce-rel-lm-26*)
then show *?thesis* **by** (*unfold Domain-converse [symmetric]*)

qed

lemma *ce-rel-lm-28*: $r \in ce\text{-rels} \implies Field\ r \in ce\text{-sets}$

proof –

assume *r-ce*: $r \in ce\text{-rels}$

from *r-ce* have *L1*: $Domain\ r \in ce\text{-sets}$ by (rule *ce-rel-lm-26*)

from *r-ce* have *L2*: $Range\ r \in ce\text{-sets}$ by (rule *ce-rel-lm-27*)

from *L1 L2* have *L3*: $Domain\ r \cup Range\ r \in ce\text{-sets}$ by (rule *ce-union*)

then show *?thesis* by (unfold *Field-def*)

qed

lemma *ce-rel-lm-29*: $\llbracket A \in ce\text{-sets}; B \in ce\text{-sets} \rrbracket \implies A \times B \in ce\text{-rels}$

proof –

assume *A-ce*: $A \in ce\text{-sets}$

assume *B-ce*: $B \in ce\text{-sets}$

define *r-a* where $r-a = \{(x,(0::nat)) \mid x. x \in A\}$

define *r-b* where $r-b = \{((0::nat),z) \mid z. z \in B\}$

have *L1*: $r-a\ O\ r-b = A \times B$ by (unfold *r-a-def*, *unfold r-b-def*, *auto*)

have *r-a-ce*: $r-a \in ce\text{-rels}$

proof –

have *loc1*: $ce\text{-rel-to-set}\ r-a = \{c\text{-pair}\ x\ 0 \mid x. x \in A\}$ by (unfold *r-a-def*, *unfold ce-rel-to-set-def*, *auto*)

define *p* where $p\ x = c\text{-pair}\ x\ 0$ for x

have *p-is-pr*: $p \in PrimRec1$ unfolding *p-def* by *prec*

from *A-ce p-is-pr* have $\{c\text{-pair}\ x\ 0 \mid x. x \in A\} \in ce\text{-sets}$

unfolding *p-def* by (*simp add: ce-set-lm-7*)

with *loc1* have $ce\text{-rel-to-set}\ r-a \in ce\text{-sets}$ by *auto*

then show *?thesis* by (rule *ce-rel-lm-7*)

qed

have *r-b-ce*: $r-b \in ce\text{-rels}$

proof –

have *loc1*: $ce\text{-rel-to-set}\ r-b = \{c\text{-pair}\ 0\ z \mid z. z \in B\}$

by (unfold *r-b-def*, *unfold ce-rel-to-set-def*, *auto*)

define *p* where $p\ z = c\text{-pair}\ 0\ z$ for z

have *p-is-pr*: $p \in PrimRec1$ unfolding *p-def* by *prec*

from *B-ce p-is-pr* have $\{c\text{-pair}\ 0\ z \mid z. z \in B\} \in ce\text{-sets}$

unfolding *p-def* by (*simp add: ce-set-lm-7*)

with *loc1* have $ce\text{-rel-to-set}\ r-b \in ce\text{-sets}$ by *auto*

then show *?thesis* by (rule *ce-rel-lm-7*)

qed

from *r-b-ce r-a-ce* have $r-a\ O\ r-b \in ce\text{-rels}$ by (rule *ce-rel-lm-24*)

with *L1* show *?thesis* by *auto*

qed

lemma *ce-rel-lm-30*: $\{\} \in ce\text{-rels}$

proof –

have $ce\text{-rel-to-set}\ \{\} = \{\}$ by (unfold *ce-rel-to-set-def*, *auto*)

with *ce-empty* have $ce\text{-rel-to-set}\ \{\} \in ce\text{-sets}$ by *auto*

then show *?thesis* by (rule *ce-rel-lm-7*)

qed

lemma *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 *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 *ce-rel-lm-33*: $\llbracket r \in ce-rels; s \in ce-rels \rrbracket \implies r \cup s \in ce-rels$

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 *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 $ce-rel-to-set r \cap ce-rel-to-set s \subseteq ce-rel-to-set (r \cap s)$

proof fix x **assume** $A: x \in ce-rel-to-set r \cap ce-rel-to-set s$

from A **have** $L1: x \in ce-rel-to-set r$ **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 *ce-rel-lm-35*: $\llbracket r \in ce-rels; s \in ce-rels \rrbracket \implies r \cap s \in ce-rels$

proof –

assume $r \in ce\text{-rels}$
then have $r\text{-ce}: ce\text{-rel-to-set } r \in ce\text{-sets}$ **by** (rule *ce-rel-lm-6*)
assume $s \in ce\text{-rels}$
then have $s\text{-ce}: ce\text{-rel-to-set } s \in ce\text{-sets}$ **by** (rule *ce-rel-lm-6*)
have $ce\text{-rel-to-set } (r \cap s) = (ce\text{-rel-to-set } r) \cap (ce\text{-rel-to-set } s)$ **by** (rule *ce-rel-lm-34*)
moreover from $r\text{-ce } s\text{-ce}$ **have** $(ce\text{-rel-to-set } r) \cap (ce\text{-rel-to-set } s) \in ce\text{-sets}$ **by**
(rule *ce-intersect*)
ultimately have $ce\text{-rel-to-set } (r \cap s) \in ce\text{-sets}$ **by** *auto*
then show *?thesis* **by** (rule *ce-rel-lm-7*)
qed

lemma *ce-rel-lm-36*: $ce\text{-set-to-rel } (A \cup B) = (ce\text{-set-to-rel } A) \cup (ce\text{-set-to-rel } B)$
by (*unfold ce-set-to-rel-def, auto*)

lemma *ce-rel-lm-37*: $ce\text{-set-to-rel } (A \cap B) = (ce\text{-set-to-rel } A) \cap (ce\text{-set-to-rel } B)$

proof –
define f **where** $f\ x = (c\text{-fst } x, c\text{-snd } x)$ **for** x
have $f\text{-inj}: inj\ f$
proof (*unfold f-def, rule inj-on-inverseI [where ?g= $\lambda (u,v). c\text{-pair } u\ v]$)
fix $x :: nat$
assume $x \in UNIV$
show $case\text{-prod } c\text{-pair } (c\text{-fst } x, c\text{-snd } x) = x$ **by** *simp*
qed
from $f\text{-inj}$ **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*: $\llbracket r \in ce\text{-rels}; A \in ce\text{-sets} \rrbracket \implies r\ ' A \in ce\text{-sets}$

proof –
assume $r\text{-ce}: r \in ce\text{-rels}$
assume $A\text{-ce}: A \in ce\text{-sets}$
have $L1: r\ ' A = Range\ (r \cap A \times UNIV)$ **by** *blast*
have $L2: Range\ (r \cap A \times UNIV) \in ce\text{-sets}$
proof (rule *ce-rel-lm-27*)
show $r \cap A \times UNIV \in ce\text{-rels}$
proof (rule *ce-rel-lm-35*)
show $r \in ce\text{-rels}$ **by** (rule *r-ce*)
next
show $A \times UNIV \in ce\text{-rels}$
proof (rule *ce-rel-lm-29*)
show $A \in ce\text{-sets}$ **by** (rule *A-ce*)
next
show $UNIV \in ce\text{-sets}$ **by** (rule *ce-univ*)
qed
qed
qed
from $L1\ L2$ **show** *?thesis* **by** *auto*
qed

7.6 Total computable functions

definition

$graph :: (nat \Rightarrow nat) \Rightarrow (nat \times nat) \text{ set}$ **where**
 $graph = (\lambda f. \{ (x, f x) \mid x. x \in UNIV \})$

lemma $graph-lm-1$: $(x,y) \in graph\ f \Longrightarrow y = f\ x$ **by** (*unfold graph-def, auto*)

lemma $graph-lm-2$: $y = f\ x \Longrightarrow (x,y) \in graph\ f$ **by** (*unfold graph-def, auto*)

lemma $graph-lm-3$: $((x,y) \in graph\ f) = (y = f\ x)$ **by** (*unfold graph-def, auto*)

lemma $graph-lm-4$: $graph\ (f\ o\ g) = (graph\ g)\ O\ (graph\ f)$ **by** (*unfold graph-def, auto*)

definition

$c-graph :: (nat \Rightarrow nat) \Rightarrow nat\ \text{set}$ **where**
 $c-graph = (\lambda f. \{ c-pair\ x\ (f\ x) \mid x. x \in UNIV \})$

lemma $c-graph-lm-1$: $c-pair\ x\ y \in c-graph\ f \Longrightarrow y = f\ x$

proof –

assume A : $c-pair\ x\ y \in c-graph\ f$

have $S1$: $c-graph\ f = \{ c-pair\ x\ (f\ x) \mid x. x \in UNIV \}$ **by** (*simp add: c-graph-def*)

from A $S1$ **obtain** z **where** $S2$: $c-pair\ x\ y = c-pair\ z\ (f\ z)$ **by** *auto*

then have $x = z$ **by** (*rule c-pair-inj1*)

moreover from $S2$ **have** $y = f\ z$ **by** (*rule c-pair-inj2*)

ultimately show *?thesis* **by** *auto*

qed

lemma $c-graph-lm-2$: $y = f\ x \Longrightarrow c-pair\ x\ y \in c-graph\ f$ **by** (*unfold c-graph-def, auto*)

lemma $c-graph-lm-3$: $(c-pair\ x\ y \in c-graph\ f) = (y = f\ x)$

proof

assume $c-pair\ x\ y \in c-graph\ f$ **then show** $y = f\ x$ **by** (*rule c-graph-lm-1*)

next

assume $y = f\ x$ **then show** $c-pair\ x\ y \in c-graph\ f$ **by** (*rule c-graph-lm-2*)

qed

lemma $c-graph-lm-4$: $c-graph\ f = ce-rel-to-set\ (graph\ f)$ **by** (*unfold c-graph-def 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

$total-recursive :: (nat \Rightarrow nat) \Rightarrow bool$ **where**
 $total-recursive = (\lambda f. graph\ f \in ce-rels)$

lemma $total-recursive-def1$: $total-recursive = (\lambda f. c-graph\ f \in ce-sets)$

proof (*rule ext*) **fix** f **show** $total-recursive\ f = (c-graph\ f \in ce-sets)$

proof
 assume A : *total-recursive* f
 then have $\text{graph } f \in \text{ce-rels}$ **by** (*unfold total-recursive-def*)
 then have $\text{ce-rel-to-set } (\text{graph } f) \in \text{ce-sets}$ **by** (*rule ce-rel-lm-6*)
 then show $c\text{-graph } f \in \text{ce-sets}$ **by** (*simp add: c-graph-lm-4*)
next
 assume $c\text{-graph } f \in \text{ce-sets}$
 then have $\text{ce-rel-to-set } (\text{graph } f) \in \text{ce-sets}$ **by** (*simp add: c-graph-lm-4*)
 then have $\text{graph } f \in \text{ce-rels}$ **by** (*rule ce-rel-lm-7*)
 then show *total-recursive* f **by** (*unfold total-recursive-def*)
qed
qed

theorem *pr-is-total-rec*: $f \in \text{PrimRec1} \implies \text{total-recursive } f$
proof –
 assume A : $f \in \text{PrimRec1}$
 define p where $p\ x = c\text{-pair } x\ (f\ x)$ **for** x
 from A have $p\text{-is-pr}$: $p \in \text{PrimRec1}$ **unfolding** $p\text{-def}$ **by** *prec*
 let $?U = \{ p\ x \mid x. x \in \text{UNIV} \}$
 from *ce-univ p-is-pr* have $U\text{-ce}$: $?U \in \text{ce-sets}$ **by** (*rule ce-set-lm-7*)
 have $U\text{-1}$: $?U = \{ c\text{-pair } x\ (f\ x) \mid x. x \in \text{UNIV} \}$ **by** (*simp add: p-def*)
 with $U\text{-ce}$ have $S1$: $\{ c\text{-pair } x\ (f\ x) \mid x. x \in \text{UNIV} \} \in \text{ce-sets}$ **by** *simp*
 with *c-graph-def* have $c\text{-graph-}f\text{-is-}ce$: $c\text{-graph } f \in \text{ce-sets}$ **by** (*unfold c-graph-def, auto*)
 then show *?thesis* **by** (*unfold total-recursive-def1, auto*)
qed

theorem *comp-tot-rec*: $\llbracket \text{total-recursive } f; \text{total-recursive } g \rrbracket \implies \text{total-recursive } (f\ o\ g)$
proof –
 assume *total-recursive* f
 then have $f\text{-ce}$: $\text{graph } f \in \text{ce-rels}$ **by** (*unfold total-recursive-def*)
 assume *total-recursive* g
 then have $g\text{-ce}$: $\text{graph } g \in \text{ce-rels}$ **by** (*unfold total-recursive-def*)
 from $f\text{-ce}$ $g\text{-ce}$ have $\text{graph } g\ O\ \text{graph } f \in \text{ce-rels}$ **by** (*rule ce-rel-lm-24*)
 then have $\text{graph } (f\ o\ g) \in \text{ce-rels}$ **by** (*simp add: graph-lm-4*)
 then show *?thesis* **by** (*unfold total-recursive-def*)
qed

lemma *univ-for-pr-tot-rec-lm*: $c\text{-graph } \text{univ-for-pr} \in \text{ce-sets}$
proof –
 define A where $A = c\text{-graph } \text{univ-for-pr}$
 from $A\text{-def}$ have $S1$: $A = \{ c\text{-pair } x\ (\text{univ-for-pr } x) \mid x. x \in \text{UNIV} \}$
 by (*simp add: c-graph-def*)
 from $S1$ have $S2$: $A = \{ z. \exists x. z = c\text{-pair } x\ (\text{univ-for-pr } x) \}$ **by** *auto*
 have $S3$: $\bigwedge z. (\exists x. (z = c\text{-pair } x\ (\text{univ-for-pr } x))) = (\text{univ-for-pr } (c\text{-fst } z)) = c\text{-snd } z$
proof –
 fix z show $(\exists x. (z = c\text{-pair } x\ (\text{univ-for-pr } x))) = (\text{univ-for-pr } (c\text{-fst } z)) =$

```

c-snd z)
proof
  assume A:  $\exists x. z = c\text{-pair } x \text{ (univ-for-pr } x)$ 
  then obtain x where S3-1:  $z = c\text{-pair } x \text{ (univ-for-pr } x)$  ..
  then show  $\text{univ-for-pr } (c\text{-fst } z) = c\text{-snd } z$  by simp
next
  assume A:  $\text{univ-for-pr } (c\text{-fst } z) = c\text{-snd } z$ 
  from A have  $z = c\text{-pair } (c\text{-fst } z) \text{ (univ-for-pr } (c\text{-fst } z))$  by simp
  thus  $\exists x. z = c\text{-pair } x \text{ (univ-for-pr } x)$  ..
qed
qed
with S2 have S4:  $A = \{ z . \text{univ-for-pr } (c\text{-fst } z) = c\text{-snd } z \}$  by auto
define p where  $p \ x \ y =$ 
  (if c-assoc-have-key (pr-gr y) (c-fst x) = 0 then
    (if c-assoc-value (pr-gr y) (c-fst x) = c-snd x then (0::nat) else 1)
  else 1) for x y
from c-assoc-have-key-is-pr c-assoc-value-is-pr pr-gr-is-pr have p-is-pr:  $p \in$ 
PrimRec2
  unfolding p-def by prec
have S5:  $\bigwedge z. (\text{univ-for-pr } (c\text{-fst } z) = c\text{-snd } z) = (\exists y. p \ z \ y = 0)$ 
proof –
  fix z show  $(\text{univ-for-pr } (c\text{-fst } z) = c\text{-snd } z) = (\exists y. p \ z \ y = 0)$ 
proof
  assume A:  $\text{univ-for-pr } (c\text{-fst } z) = c\text{-snd } z$ 
  let ?n = c-fst (c-fst z)
  let ?x = c-snd (c-fst z)
  let ?y = loc-upb ?n ?x
have S5-1:  $c\text{-assoc-have-key } (pr\text{-gr } ?y) \ (c\text{-pair } ?n \ ?x) = 0$  by (rule loc-upb-main)
  have S5-2:  $c\text{-assoc-value } (pr\text{-gr } ?y) \ (c\text{-pair } ?n \ ?x) = \text{univ-for-pr } (c\text{-pair } ?n$ 
?x) by (rule pr-gr-value)
  from S5-1 have S5-3:  $c\text{-assoc-have-key } (pr\text{-gr } ?y) \ (c\text{-fst } z) = 0$  by simp
  from S5-2 A have S5-4:  $c\text{-assoc-value } (pr\text{-gr } ?y) \ (c\text{-fst } z) = c\text{-snd } z$  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:  $c\text{-assoc-have-key } (pr\text{-gr } y) \ (c\text{-fst } z) = 0$ 
proof (rule ccontr)
  assume A-1:  $c\text{-assoc-have-key } (pr\text{-gr } y) \ (c\text{-fst } z) \neq 0$ 
  then have  $p \ z \ y = 1$  by (simp add: p-def)
  with S5-1 show False by auto
qed
then have S5-3:  $p \ z \ y = (\text{if } c\text{-assoc-value } (pr\text{-gr } y) \ (c\text{-fst } z) = c\text{-snd } z \text{ then}$ 
(0::nat) else 1) by (simp add: p-def)
have S5-4:  $c\text{-assoc-value } (pr\text{-gr } y) \ (c\text{-fst } z) = c\text{-snd } z$ 
proof (rule ccontr)
  assume A-2:  $c\text{-assoc-value } (pr\text{-gr } y) \ (c\text{-fst } z) \neq c\text{-snd } z$ 
  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
  qed
  from S5 S4 have A = {z.  $\exists y. p z y = 0$ } 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  $\in$  ce-sets by (rule ce-set-lm-1)
  ultimately have A  $\in$  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  $\in$  ce-sets by (rule univ-for-pr-tot-rec-lm)
  then show ?thesis by (unfold total-recursive-def1, auto)
qed

```

7.7 Computable sets, Post's theorem

definition

```

computable :: nat set  $\Rightarrow$  bool where
computable = ( $\lambda A. A \in$  ce-sets  $\wedge$   $\neg A \in$  ce-sets)

```

lemma computable-complement-1: $computable A \implies computable (\neg A)$

proof –

```

  assume computable A
  then show ?thesis by (unfold computable-def, auto)

```

qed

lemma computable-complement-2: $computable (\neg A) \implies computable A$

proof –

```

  assume computable ( $\neg A$ )
  then show ?thesis by (unfold computable-def, auto)

```

qed

lemma computable-complement-3: $(computable A) = (computable (\neg A))$ by (unfold computable-def, auto)

theorem comp-impl-tot-rec: $computable A \implies total-recursive (chf A)$

proof –

```

  assume A: computable A
  from A have A1: A  $\in$  ce-sets by (unfold computable-def, simp)
  from A have A2:  $\neg A \in$  ce-sets by (unfold computable-def, simp)
  define p where p x = c-pair x 0 for x
  define q where q x = c-pair x 1 for x

```

```

from  $p$ -def have  $p$ -is-pr:  $p \in \text{PrimRec1}$  unfolding  $p$ -def by prec
from  $q$ -def have  $q$ -is-pr:  $q \in \text{PrimRec1}$  unfolding  $q$ -def by prec
define  $U0$  where  $U0 = \{p\ x \mid x. x \in A\}$ 
define  $U1$  where  $U1 = \{q\ x \mid x. x \in -A\}$ 
from  $A1$   $p$ -is-pr have  $U0$ -ce:  $U0 \in \text{ce-sets}$  by(unfold  $U0$ -def, rule ce-set-lm-7)
from  $A2$   $q$ -is-pr have  $U1$ -ce:  $U1 \in \text{ce-sets}$  by(unfold  $U1$ -def, rule ce-set-lm-7)
define  $U$  where  $U = U0 \cup U1$ 
from  $U0$ -ce  $U1$ -ce have  $U$ -ce:  $U \in \text{ce-sets}$  by (unfold  $U$ -def, rule ce-union)
define  $V$  where  $V = \text{c-graph } (\text{chf } A)$ 
have  $V$ -1:  $V = \{ \text{c-pair } x\ y\ (\text{chf } A\ x) \mid x. x \in \text{UNIV} \}$  by (simp add:  $V$ -def
c-graph-def)
from  $U0$ -def  $p$ -def have  $U0$ -1:  $U0 = \{ \text{c-pair } x\ y \mid x\ y. x \in A \wedge y=0 \}$  by auto
from  $U1$ -def  $q$ -def have  $U1$ -1:  $U1 = \{ \text{c-pair } x\ y \mid x\ y. x \notin A \wedge y=1 \}$  by auto
from  $U0$ -1  $U1$ -1  $U$ -def have  $U$ -1:  $U = \{ \text{c-pair } x\ y \mid x\ y. (x \in A \wedge y=0) \vee (x \notin A \wedge y=1) \}$  by auto
from  $V$ -1 have  $V$ -2:  $V = \{ \text{c-pair } x\ y \mid x\ y. y = \text{chf } A\ x \}$  by auto
have  $L1$ :  $\bigwedge x\ y. ((x \in A \wedge y=0) \vee (x \notin A \wedge y=1)) = (y = \text{chf } A\ x)$ 
proof -
  fix  $x\ y$ 
  show  $((x \in A \wedge y=0) \vee (x \notin A \wedge y=1)) = (y = \text{chf } A\ x)$  by(unfold chf-def,
auto)
qed
from  $V$ -2  $U$ -1  $L1$  have  $U=V$  by simp
with  $U$ -ce have  $V$ -ce:  $V \in \text{ce-sets}$  by auto
with  $V$ -def have  $\text{c-graph } (\text{chf } A) \in \text{ce-sets}$  by auto
then show ?thesis by (unfold total-recursive-def1)
qed

theorem tot-rec-impl-comp: total-recursive ( $\text{chf } A$ )  $\implies$  computable  $A$ 
proof -
  assume  $A$ : total-recursive ( $\text{chf } A$ )
  then have  $A1$ :  $\text{c-graph } (\text{chf } A) \in \text{ce-sets}$  by (unfold total-recursive-def1)
  let  $?U = \text{c-graph } (\text{chf } A)$ 
  have  $L1$ :  $?U = \{ \text{c-pair } x\ (\text{chf } A\ x) \mid x. x \in \text{UNIV} \}$  by (simp add: c-graph-def)
  have  $L2$ :  $\bigwedge x\ y. ((x \in A \wedge y=0) \vee (x \notin A \wedge y=1)) = (y = \text{chf } A\ x)$ 
  proof - fix  $x\ y$  show  $((x \in A \wedge y=0) \vee (x \notin A \wedge y=1)) = (y = \text{chf } A\ x)$ 
by(unfold chf-def, auto)
  qed
  from  $L1$   $L2$  have  $L3$ :  $?U = \{ \text{c-pair } x\ y \mid x\ y. (x \in A \wedge y=0) \vee (x \notin A \wedge y=1) \}$  by auto
  define  $p$  where  $p\ x = \text{c-pair } x\ 0$  for  $x$ 
  define  $q$  where  $q\ x = \text{c-pair } x\ 1$  for  $x$ 
  have  $p$ -is-pr:  $p \in \text{PrimRec1}$  unfolding  $p$ -def by prec
  have  $q$ -is-pr:  $q \in \text{PrimRec1}$  unfolding  $q$ -def by prec
  define  $V$  where  $V = \{ \text{c-pair } x\ y \mid x\ y. (x \in A \wedge y=0) \vee (x \notin A \wedge y=1) \}$ 
from  $V$ -def  $L3$   $A1$  have  $V$ -ce:  $V \in \text{ce-sets}$  by auto
from  $V$ -def have  $L4$ :  $\forall z. (z \in V) = (\exists x\ y. z = \text{c-pair } x\ y \wedge ((x \in A \wedge y=0) \vee (x \notin A \wedge y=1)))$  by blast
  have  $L5$ :  $\bigwedge x. (p\ x \in V) = (x \in A)$ 

```

proof – **fix** x **show** $(p\ x \in V) = (x \in A)$
proof
 assume $A: p\ x \in V$
 then have $c\text{-pair}\ x\ 0 \in V$ **by** (*unfold p-def*)
 with $V\text{-def}$ **obtain** $x1\ y1$ **where** $L5\text{-2}: c\text{-pair}\ x\ 0 = c\text{-pair}\ x1\ y1$
 and $L5\text{-3}: ((x1 \in A \wedge y1=0) \vee (x1 \notin A \wedge y1=1))$ **by** *auto*
 from $L5\text{-2}$ **have** $X\text{-eq}\text{-}X1: x=x1$ **by** (*rule c-pair-inj1*)
 from $L5\text{-2}$ **have** $Y1\text{-eq}\text{-}0: 0=y1$ **by** (*rule c-pair-inj2*)
 from $L5\text{-3}$ $X\text{-eq}\text{-}X1\ Y1\text{-eq}\text{-}0$ **show** $x \in A$ **by** *auto*
next
 assume $A: x \in A$
 let $?z = c\text{-pair}\ x\ 0$
 from A **have** $L5\text{-1}: \exists\ x1\ y1. c\text{-pair}\ x\ 0 = c\text{-pair}\ x1\ y1 \wedge ((x1 \in A \wedge y1=0) \vee (x1 \notin A \wedge y1=1))$ **by** *auto*
 with $V\text{-def}$ **have** $c\text{-pair}\ x\ 0 \in V$ **by** *auto*
 with $p\text{-def}$ **show** $p\ x \in V$ **by** *simp*
qed
qed
then have $A\text{-eq}: A = \{x. p\ x \in V\}$ **by** *auto*
from $V\text{-ce}\ p\text{-is}\text{-pr}$ **have** $\{x. p\ x \in V\} \in ce\text{-sets}$ **by** (*rule ce-set-lm-5*)
with $A\text{-eq}$ **have** $A\text{-ce}: A \in ce\text{-sets}$ **by** *simp*
have $CA\text{-eq}: \neg A = \{x. q\ x \in V\}$
proof –
 have $\bigwedge x. (q\ x \in V) = (x \notin A)$
 proof – **fix** x **show** $(q\ x \in V) = (x \notin A)$
 proof
 assume $A: q\ x \in V$
 then have $c\text{-pair}\ x\ 1 \in V$ **by** (*unfold q-def*)
 with $V\text{-def}$ **obtain** $x1\ y1$ **where** $L5\text{-2}: c\text{-pair}\ x\ 1 = c\text{-pair}\ x1\ y1$
 and $L5\text{-3}: ((x1 \in A \wedge y1=0) \vee (x1 \notin A \wedge y1=1))$ **by** *auto*
 from $L5\text{-2}$ **have** $X\text{-eq}\text{-}X1: x=x1$ **by** (*rule c-pair-inj1*)
 from $L5\text{-2}$ **have** $Y1\text{-eq}\text{-}1: 1=y1$ **by** (*rule c-pair-inj2*)
 from $L5\text{-3}$ $X\text{-eq}\text{-}X1\ Y1\text{-eq}\text{-}1$ **show** $x \notin A$ **by** *auto*
 next
 assume $A: x \notin A$
 from A **have** $L5\text{-1}: \exists\ x1\ y1. c\text{-pair}\ x\ 1 = c\text{-pair}\ x1\ y1 \wedge ((x1 \in A \wedge y1=0) \vee (x1 \notin A \wedge y1=1))$ **by** *auto*
 with $V\text{-def}$ **have** $c\text{-pair}\ x\ 1 \in V$ **by** *auto*
 with $q\text{-def}$ **show** $q\ x \in V$ **by** *simp*
 qed
 qed
 then show $?thesis$ **by** *auto*
qed
from $V\text{-ce}\ q\text{-is}\text{-pr}$ **have** $\{x. q\ x \in V\} \in ce\text{-sets}$ **by** (*rule ce-set-lm-5*)
with $CA\text{-eq}$ **have** $CA\text{-ce}: \neg A \in ce\text{-sets}$ **by** *simp*
from $A\text{-ce}\ CA\text{-ce}$ **show** $?thesis$ **by** (*simp add: computable-def*)
qed

theorem $post\text{-th}\text{-}0: (computable\ A) = (total\text{-recursive}\ (chf\ A))$

proof
 assume *computable A* then show *total-recursive (chf A)* by (rule *comp-impl-tot-rec*)
 next
 assume *total-recursive (chf A)* then show *computable A* by (rule *tot-rec-impl-comp*)
 qed

7.8 Universal computably enumerable set

definition

univ-ce :: *nat set* where
univ-ce = { *c-pair n x* | *n x. x ∈ nat-to-ce-set n* }

lemma *univ-for-pr-lm*: *univ-for-pr (c-pair n x) = (nat-to-pr n) x*
 by (*simp add: univ-for-pr-def pr-conv-2-to-1-def*)

theorem *univ-is-ce*: *univ-ce ∈ ce-sets*

proof –

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 $\exists pA \in \text{PrimRec2}. A = \text{fn-to-set } pA$ by (rule *ce-set-lm-3*)
 then obtain *pA* where *pA-is-pr*: *pA ∈ PrimRec2* and *S1*: *A = fn-to-set pA*
 by *auto*

from *S1* have *S2*: *A = { x. $\exists y. pA\ x\ y = 0$ }* by (*simp add: fn-to-set-def*)
 define *p* where *p z y = pA (c-pair (c-pair (c-fst z) (c-pair (c-snd z) (c-fst y)))*
0) (c-snd y)

for *z y*
 from *pA-is-pr* have *p-is-pr*: *p ∈ PrimRec2* unfolding *p-def* by *prec*
 have $\bigwedge z. (\exists n\ x. z = \text{c-pair } n\ x \wedge x \in \text{nat-to-ce-set } n) = (\text{c-snd } z \in \text{nat-to-ce-set } (\text{c-fst } z))$

proof –

fix *z* show $(\exists n\ x. z = \text{c-pair } n\ x \wedge x \in \text{nat-to-ce-set } n) = (\text{c-snd } z \in \text{nat-to-ce-set } (\text{c-fst } z))$

proof

assume *A*: $\exists n\ x. z = \text{c-pair } n\ x \wedge x \in \text{nat-to-ce-set } n$
 then obtain *n x* where *L1*: *z = c-pair n x* and *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* and *?x ∈ nat-to-ce-set ?n* by *auto*
 thus $\exists n\ x. z = \text{c-pair } n\ x \wedge x \in \text{nat-to-ce-set } n$ by *blast*

qed

qed

then have $\{ c\text{-pair } n \ x \mid n \ x. \ x \in \text{nat-to-ce-set } n \} = \{ z. \ c\text{-snd } z \in \text{nat-to-ce-set } (c\text{-fst } z) \}$ **by** *auto*
then have $S3: \text{univ-ce} = \{ z. \ c\text{-snd } z \in \text{nat-to-ce-set } (c\text{-fst } z) \}$ **by** (*simp add: univ-ce-def*)
have $S4: \bigwedge z. (c\text{-snd } z \in \text{nat-to-ce-set } (c\text{-fst } z)) = (\exists y. \ p \ z \ y = 0)$
proof –
fix z **show** $(c\text{-snd } z \in \text{nat-to-ce-set } (c\text{-fst } z)) = (\exists y. \ p \ z \ y = 0)$
proof
assume $A: c\text{-snd } z \in \text{nat-to-ce-set } (c\text{-fst } z)$
have $\text{nat-to-ce-set } (c\text{-fst } z) = \{ x. \ \exists y. (\text{nat-to-pr } (c\text{-fst } z)) (c\text{-pair } x \ y) = 0 \}$ **by** (*simp add: nat-to-ce-set-lm-1*)
with A **obtain** u **where** $S4\text{-}1: (\text{nat-to-pr } (c\text{-fst } z)) (c\text{-pair } (c\text{-snd } z) \ u) = 0$
by *auto*
then have $S4\text{-}2: \text{univ-for-pr } (c\text{-pair } (c\text{-fst } z) (c\text{-pair } (c\text{-snd } z) \ u)) = 0$ **by** (*simp add: univ-for-pr-lm*)
from $A\text{-def}$ **have** $S4\text{-}3: A = \{ c\text{-pair } x \ (\text{univ-for-pr } x) \mid x. \ x \in \text{UNIV} \}$ **by** (*simp add: c-graph-def*)
then have $S4\text{-}4: \bigwedge x. \ c\text{-pair } x \ (\text{univ-for-pr } x) \in A$ **by** *auto*
then have $c\text{-pair } (c\text{-pair } (c\text{-fst } z) (c\text{-pair } (c\text{-snd } z) \ u)) (\text{univ-for-pr } (c\text{-pair } (c\text{-fst } z) (c\text{-pair } (c\text{-snd } z) \ u))) \in A$ **by** *auto*
with $S4\text{-}2$ **have** $S4\text{-}5: c\text{-pair } (c\text{-pair } (c\text{-fst } z) (c\text{-pair } (c\text{-snd } z) \ u)) \ 0 \in A$ **by** *auto*
with $S2$ **obtain** v **where** $S4\text{-}6: pA (c\text{-pair } (c\text{-pair } (c\text{-fst } z) (c\text{-pair } (c\text{-snd } z) \ u)) \ 0) \ v = 0$
by *auto*
define y **where** $y = c\text{-pair } u \ v$
from $y\text{-def}$ **have** $S4\text{-}7: u = c\text{-fst } y$ **by** *simp*
from $y\text{-def}$ **have** $S4\text{-}8: v = c\text{-snd } y$ **by** *simp*
from $S4\text{-}6 \ S4\text{-}7 \ S4\text{-}8 \ p\text{-def}$ **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** $S4\text{-}1: p \ z \ y = 0 \ ..$
from $S4\text{-}1 \ p\text{-def}$ **have** $S4\text{-}2: pA (c\text{-pair } (c\text{-pair } (c\text{-fst } z) (c\text{-pair } (c\text{-snd } z) (c\text{-fst } y))) \ 0) (c\text{-snd } y) = 0$ **by** *simp*
with $S2$ **have** $S4\text{-}3: c\text{-pair } (c\text{-pair } (c\text{-fst } z) (c\text{-pair } (c\text{-snd } z) (c\text{-fst } y))) \ 0 \in A$ **by** *auto*
with $A\text{-def}$ **have** $c\text{-pair } (c\text{-pair } (c\text{-fst } z) (c\text{-pair } (c\text{-snd } z) (c\text{-fst } y))) \ 0 \in c\text{-graph } \text{univ-for-pr}$ **by** *simp*
then have $S4\text{-}4: 0 = \text{univ-for-pr } (c\text{-pair } (c\text{-fst } z) (c\text{-pair } (c\text{-snd } z) (c\text{-fst } y)))$ **by** (*rule c-graph-lm-1*)
then have $S4\text{-}5: \text{univ-for-pr } (c\text{-pair } (c\text{-fst } z) (c\text{-pair } (c\text{-snd } z) (c\text{-fst } y))) = 0$ **by** *auto*
then have $S4\text{-}6: (\text{nat-to-pr } (c\text{-fst } z)) (c\text{-pair } (c\text{-snd } z) (c\text{-fst } y)) = 0$ **by** (*simp add: univ-for-pr-lm*)
then have $S4\text{-}7: \exists y. (\text{nat-to-pr } (c\text{-fst } z)) (c\text{-pair } (c\text{-snd } z) \ y) = 0 \ ..$
have $S4\text{-}8: \text{nat-to-ce-set } (c\text{-fst } z) = \{ x. \ \exists y. (\text{nat-to-pr } (c\text{-fst } z)) (c\text{-pair } x \ y) = 0 \}$ **by** (*simp add: nat-to-ce-set-lm-1*)
from $S4\text{-}7$ **have** $S4\text{-}9: c\text{-snd } z \in \{ x. \ \exists y. (\text{nat-to-pr } (c\text{-fst } z)) (c\text{-pair } x \ y) = 0 \}$

= 0 } by auto
 with $S4-8$ show $c\text{-snd } z \in \text{nat-to-ce-set } (c\text{-fst } z)$ by auto
 qed
 qed
 with $S3$ have $\text{univ-ce} = \{z. \exists y. p \ z \ y = 0\}$ by auto
 then have $\text{univ-ce} = \text{fn-to-set } p$ by (simp add: fn-to-set-def)
 moreover from $p\text{-is-pr}$ have $\text{fn-to-set } p \in \text{ce-sets}$ by (rule ce-set-lm-1)
 ultimately show $\text{univ-ce} \in \text{ce-sets}$ by auto
 qed

lemma univ-ce-lm-1 : $(c\text{-pair } n \ x \in \text{univ-ce}) = (x \in \text{nat-to-ce-set } n)$
proof –
 from univ-ce-def have $S1$: $\text{univ-ce} = \{z. \exists n \ x. z = c\text{-pair } n \ x \wedge x \in \text{nat-to-ce-set } n\}$ by auto
 have $S2$: $(\exists n1 \ x1. c\text{-pair } n \ x = c\text{-pair } n1 \ x1 \wedge x1 \in \text{nat-to-ce-set } n1) = (x \in \text{nat-to-ce-set } n)$
proof
 assume $\exists n1 \ x1. c\text{-pair } n \ x = c\text{-pair } n1 \ x1 \wedge x1 \in \text{nat-to-ce-set } n1$
 then obtain $n1 \ x1$ where $L1$: $c\text{-pair } n \ x = c\text{-pair } n1 \ x1$ and $L2$: $x1 \in \text{nat-to-ce-set } n1$ by auto
 from $L1$ have $L3$: $n = n1$ by (rule $c\text{-pair-inj1}$)
 from $L1$ have $L4$: $x = x1$ by (rule $c\text{-pair-inj2}$)
 from $L2 \ L3 \ L4$ show $x \in \text{nat-to-ce-set } n$ by auto
next
 assume A : $x \in \text{nat-to-ce-set } n$
 then have $c\text{-pair } n \ x = c\text{-pair } n \ x \wedge x \in \text{nat-to-ce-set } n$ by auto
 thus $\exists n1 \ x1. c\text{-pair } n \ x = c\text{-pair } n1 \ x1 \wedge x1 \in \text{nat-to-ce-set } n1$ by blast
 qed
 with $S1$ show ?thesis by auto
 qed

theorem $\text{univ-ce-is-not-comp1}$: $-\ \text{univ-ce} \notin \text{ce-sets}$
proof (rule $c\text{contr}$)
 assume $\neg - \text{univ-ce} \notin \text{ce-sets}$
 then have A : $-\ \text{univ-ce} \in \text{ce-sets}$ by auto
 define p where $p \ x = c\text{-pair } x \ x$ for x
 have $p\text{-is-pr}$: $p \in \text{PrimRec1}$ unfolding $p\text{-def}$ by prec
 define A where $A = \{x. p \ x \in - \text{univ-ce}\}$
 from $A \ p\text{-is-pr}$ have $\{x. p \ x \in - \text{univ-ce}\} \in \text{ce-sets}$ by (rule ce-set-lm-5)
 with $A\text{-def}$ have $S1$: $A \in \text{ce-sets}$ by auto
 then have $\exists n. A = \text{nat-to-ce-set } n$ by (rule nat-to-ce-set-srj)
 then obtain n where $S2$: $A = \text{nat-to-ce-set } n$..
 from $A\text{-def}$ have $(n \in A) = (p \ n \in - \text{univ-ce})$ by auto
 with $p\text{-def}$ have $(n \in A) = (c\text{-pair } n \ n \notin \text{univ-ce})$ by auto
 with $\text{univ-ce-def} \ \text{univ-ce-lm-1}$ have $(n \in A) = (n \notin \text{nat-to-ce-set } n)$ by auto
 with $S2$ have $(n \in A) = (n \notin A)$ by auto
 thus False by auto
 qed

theorem *univ-ce-is-not-comp2*: \neg *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 \neg *univ-ce* \in *ce-sets* **by** (*unfold computable-def, auto*)

with *univ-ce-is-not-comp1* **show** *False* **by** *auto*

qed

theorem *univ-ce-is-not-comp3*: \neg *computable univ-ce*

proof (*rule ccontr*)

assume \neg \neg *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* \Rightarrow *nat* \Rightarrow *nat* **where**

s-ce == (λ *e* *x*. *s1-1* (*comp-by-index* *e* *index-of-r-to-l*) *x*)

lemma *s-ce-is-pr*: *s-ce* \in *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* \implies *e1*=*e2* \wedge *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 $L1: \text{comp-by-index } e1 \ ?n1 = \text{comp-by-index } e2 \ ?n1 \wedge x1=x2$ **by** (rule $s1-1\text{-inj}$)
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 $?thesis$ **by** *auto*
qed

lemma $s\text{-ce-inj1}: s\text{-ce } e1 \ x = s\text{-ce } e2 \ x \implies e1=e2$
proof –
assume $s\text{-ce } e1 \ x = s\text{-ce } e2 \ x$
then have $e1=e2 \wedge x=x$ **by** (rule $s\text{-ce-inj}$)
then show $e1=e2$ **by** *auto*
qed

lemma $s\text{-ce-inj2}: s\text{-ce } e \ x1 = s\text{-ce } e \ x2 \implies x1=x2$
proof –
assume $s\text{-ce } e \ x1 = s\text{-ce } e \ x2$
then have $e=e \wedge x1=x2$ **by** (rule $s\text{-ce-inj}$)
then show $x1=x2$ **by** *auto*
qed

theorem $s1-1\text{-th1}: \forall \ n \ x \ y. ((\text{nat-to-pr } n) (\text{c-pair } x \ y)) = (\text{nat-to-pr } (s1-1 \ n \ x)) \ y$
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 $(\lambda \ y. (\text{nat-to-pr } n) (\text{c-pair } x \ y)) = \text{nat-to-pr } (s1-1 \ n \ x)$ **by** (rule $s1-1\text{-th}$)
then show $?thesis$ **by** (*simp add: fun-eq-iff*)
qed
qed

lemma $s\text{-lm}: (\text{nat-to-pr } (s\text{-ce } e \ x)) (\text{c-pair } y \ z) = (\text{nat-to-pr } e) (\text{c-pair } (\text{c-pair } x \ y) \ z)$
proof –
let $?n1 = \text{index-of-r-to-l}$
have $(\text{nat-to-pr } (s\text{-ce } e \ x)) (\text{c-pair } y \ z) = \text{nat-to-pr } (s1-1 (\text{comp-by-index } e \ ?n1) \ x) (\text{c-pair } y \ z)$ **by** (*unfold s-ce-def, simp*)
also have $\dots = (\text{nat-to-pr } (\text{comp-by-index } e \ ?n1)) (\text{c-pair } x (\text{c-pair } y \ z))$ **by** (*simp add: s1-1-th1*)
also have $\dots = (\text{nat-to-pr } e) ((\text{nat-to-pr } ?n1) (\text{c-pair } x (\text{c-pair } y \ z)))$ **by** (*simp add: comp-by-index-main*)
finally show $?thesis$ **by** (*simp add: index-of-r-to-l-lm*)
qed

theorem $s\text{-ce-1-1-th}: (\text{c-pair } x \ y \in \text{nat-to-ce-set } e) = (y \in \text{nat-to-ce-set } (s\text{-ce } e \ x))$
proof
assume $A: \text{c-pair } x \ y \in \text{nat-to-ce-set } e$
then obtain z **where** $L1: (\text{nat-to-pr } e) (\text{c-pair } (\text{c-pair } x \ y) \ z) = 0$
by (*auto simp add: nat-to-ce-set-lm-1*)

have $(\text{nat-to-pr } (s\text{-ce } e \ x)) \ (c\text{-pair } y \ z) = 0$ **by** $(\text{simp add: } s\text{-lm } L1)$
with $\text{nat-to-ce-set-lm-1}$ **show** $y \in \text{nat-to-ce-set } (s\text{-ce } e \ x)$ **by** auto
next
assume $A: y \in \text{nat-to-ce-set } (s\text{-ce } e \ x)$
then obtain z **where** $L1: (\text{nat-to-pr } (s\text{-ce } e \ x)) \ (c\text{-pair } y \ z) = 0$
by $(\text{auto simp add: } \text{nat-to-ce-set-lm-1})$
then have $(\text{nat-to-pr } e) \ (c\text{-pair } (c\text{-pair } x \ y) \ z) = 0$ **by** $(\text{simp add: } s\text{-lm})$
with $\text{nat-to-ce-set-lm-1}$ **show** $c\text{-pair } x \ y \in \text{nat-to-ce-set } e$ **by** auto
qed

definition

$\text{one-reducible-to-via} :: (\text{nat set}) \Rightarrow (\text{nat set}) \Rightarrow (\text{nat} \Rightarrow \text{nat}) \Rightarrow \text{bool}$ **where**
 $\text{one-reducible-to-via} = (\lambda \ A \ B \ f. \ \text{total-recursive } f \wedge \text{inj } f \wedge (\forall \ x. \ (x \in A) = (f \ x \in B)))$

definition

$\text{one-reducible-to} :: (\text{nat set}) \Rightarrow (\text{nat set}) \Rightarrow \text{bool}$ **where**
 $\text{one-reducible-to} = (\lambda \ A \ B. \ \exists \ f. \ \text{one-reducible-to-via } A \ B \ f)$

definition

$\text{many-reducible-to-via} :: (\text{nat set}) \Rightarrow (\text{nat set}) \Rightarrow (\text{nat} \Rightarrow \text{nat}) \Rightarrow \text{bool}$ **where**
 $\text{many-reducible-to-via} = (\lambda \ A \ B \ f. \ \text{total-recursive } f \wedge (\forall \ x. \ (x \in A) = (f \ x \in B)))$

definition

$\text{many-reducible-to} :: (\text{nat set}) \Rightarrow (\text{nat set}) \Rightarrow \text{bool}$ **where**
 $\text{many-reducible-to} = (\lambda \ A \ B. \ \exists \ f. \ \text{many-reducible-to-via } A \ B \ f)$

lemma $\text{one-reducible-to-via-trans}: \llbracket \text{one-reducible-to-via } A \ B \ f; \text{one-reducible-to-via } B \ C \ g \rrbracket \Longrightarrow \text{one-reducible-to-via } A \ C \ (g \ o \ f)$

proof –

assume $A1: \text{one-reducible-to-via } A \ B \ f$
assume $A2: \text{one-reducible-to-via } B \ C \ g$
from $A1$ **have** $f\text{-tr}: \text{total-recursive } f$ **by** $(\text{unfold one-reducible-to-via-def, auto})$
from $A1$ **have** $f\text{-inj}: \text{inj } f$ **by** $(\text{unfold one-reducible-to-via-def, auto})$
from $A1$ **have** $L1: \forall \ x. \ (x \in A) = (f \ x \in B)$ **by** $(\text{unfold one-reducible-to-via-def, auto})$
from $A2$ **have** $g\text{-tr}: \text{total-recursive } g$ **by** $(\text{unfold one-reducible-to-via-def, auto})$
from $A2$ **have** $g\text{-inj}: \text{inj } g$ **by** $(\text{unfold one-reducible-to-via-def, auto})$
from $A2$ **have** $L2: \forall \ x. \ (x \in B) = (g \ x \in C)$ **by** $(\text{unfold one-reducible-to-via-def, auto})$
from $g\text{-tr } f\text{-tr}$ **have** $fg\text{-tr}: \text{total-recursive } (g \ o \ f)$ **by** $(\text{rule comp-tot-rec})$
from $g\text{-inj } f\text{-inj}$ **have** $fg\text{-inj}: \text{inj } (g \ o \ f)$ **by** $(\text{rule inj-compose})$
from $L1 \ L2$ **have** $L3: (\forall \ x. \ (x \in A) = ((g \ o \ f) \ x \in C))$ **by** auto
with $fg\text{-tr } fg\text{-inj}$ **show** $?thesis$ **by** $(\text{unfold one-reducible-to-via-def, auto})$
qed

lemma $\text{one-reducible-to-trans}: \llbracket \text{one-reducible-to } A \ B; \text{one-reducible-to } B \ C \rrbracket \Longrightarrow \text{one-reducible-to } A \ C$

proof –
assume *one-reducible-to* $A B$
then obtain f **where** $A1: \text{one-reducible-to-via } A B f$ **unfolding** *one-reducible-to-def*
by *auto*
assume *one-reducible-to* $B C$
then obtain g **where** $A2: \text{one-reducible-to-via } B C g$ **unfolding** *one-reducible-to-def*
by *auto*
from $A1 A2$ **have** *one-reducible-to-via* $A C (g \circ f)$ **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* $A A (\lambda x. x)$

proof –
have *is-pr*: $(\lambda x. x) \in \text{PrimRec1}$ **by** (*rule pr-id1-1*)
then have *is-tr*: *total-recursive* $(\lambda x. x)$ **by** (*rule pr-is-total-rec*)
have *is-inj*: *inj* $(\lambda x. x)$ **by** *simp*
have $L1: \forall x. (x \in A) = ((\lambda x. x) x) \in A$ **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* $A A (\lambda x. x)$ **by** (*rule one-reducible-to-via-refl*)
then show *?thesis* **by** (*unfold one-reducible-to-def, auto*)
qed

lemma *many-reducible-to-via-trans*: $\llbracket \text{many-reducible-to-via } A B f; \text{many-reducible-to-via } B C g \rrbracket \implies \text{many-reducible-to-via } A C (g \circ f)$

proof –
assume $A1: \text{many-reducible-to-via } A B f$
assume $A2: \text{many-reducible-to-via } B C g$
from $A1$ **have** *f-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*)
from $A2$ **have** *g-tr*: *total-recursive* g **by** (*unfold many-reducible-to-via-def, auto*)
from $A2$ **have** $L2: \forall x. (x \in B) = (g x \in C)$ **by** (*unfold many-reducible-to-via-def, auto*)
from *g-tr f-tr* **have** *fg-tr*: *total-recursive* $(g \circ f)$ **by** (*rule comp-tot-rec*)
from $L1 L2$ **have** $L3: (\forall x. (x \in A) = ((g \circ f) x \in C))$ **by** *auto*
with *fg-tr* **show** *?thesis* **by** (*unfold many-reducible-to-via-def, auto*)
qed

lemma *many-reducible-to-trans*: $\llbracket \text{many-reducible-to } A B; \text{many-reducible-to } B C \rrbracket \implies \text{many-reducible-to } A C$

proof –
assume *many-reducible-to* $A B$
then obtain f **where** $A1: \text{many-reducible-to-via } A B f$
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* $A C (g \circ f)$ **by** (*rule many-reducible-to-via-trans*)
then show *?thesis unfolding many-reducible-to-def* **by** *auto*
qed

lemma *one-reducibility-via-is-many*: *one-reducible-to-via* $A B f \implies$ *many-reducible-to-via* $A B f$

proof –
assume A : *one-reducible-to-via* $A B f$
from A **have** f -tr: *total-recursive* f **by** (*unfold one-reducible-to-via-def, auto*)
from A **have** $\forall x. (x \in A) = (f x \in B)$ **by** (*unfold one-reducible-to-via-def, auto*)
with f -tr **show** *?thesis* **by** (*unfold many-reducible-to-via-def, auto*)
qed

lemma *one-reducibility-is-many*: *one-reducible-to* $A B \implies$ *many-reducible-to* $A B$

proof –
assume *one-reducible-to* $A B$
then obtain f **where** A : *one-reducible-to-via* $A B f$
unfolding *one-reducible-to-def* **by** *auto*
then have *many-reducible-to-via* $A B f$ **by** (*rule one-reducibility-via-is-many*)
then show *?thesis unfolding many-reducible-to-def* **by** *auto*
qed

lemma *many-reducible-to-via-refl*: *many-reducible-to-via* $A A (\lambda x. x)$

proof –
have *one-reducible-to-via* $A A (\lambda x. x)$ **by** (*rule one-reducible-to-via-refl*)
then show *?thesis* **by** (*rule one-reducibility-via-is-many*)
qed

lemma *many-reducible-to-refl*: *many-reducible-to* $A A$

proof –
have *one-reducible-to* $A A$ **by** (*rule one-reducible-to-refl*)
then show *?thesis* **by** (*rule one-reducibility-is-many*)
qed

theorem *m-red-to-comp*: \llbracket *many-reducible-to* $A B$; *computable* B $\rrbracket \implies$ *computable* A

proof –
assume *many-reducible-to* $A B$
then obtain f **where** $A1$: *many-reducible-to-via* $A B f$
unfolding *many-reducible-to-def* **by** *auto*
from $A1$ **have** f -tr: *total-recursive* f **by** (*unfold many-reducible-to-via-def, auto*)
from $A1$ **have** $L1$: $\forall x. (x \in A) = (f x \in B)$ **by** (*unfold many-reducible-to-via-def, auto*)
assume *computable* B
then have $L2$: *total-recursive* $(chf B)$ **by** (*rule comp-impl-tot-rec*)
have $L3$: $chf A = (chf B) \circ f$
proof **fix** x

```

have chf A x = (chf B) (f x)
proof cases
  assume A: x ∈ A
  then have L3-1: chf A x = 0 by (simp add: chf-lm-2)
  from A L1 have f x ∈ B by auto
  then have L3-2: (chf B) (f x) = 0 by (simp add: chf-lm-2)
  from L3-1 L3-2 show chf A x = (chf B) (f x) by auto
next
  assume A: x ∉ A
  then have L3-1: chf A x = 1 by (simp add: chf-lm-3)
  from A L1 have f x ∉ B by auto
  then have L3-2: (chf B) (f x) = 1 by (simp add: chf-lm-3)
  from L3-1 L3-2 show chf A x = (chf B) (f x) by auto
qed
then show chf A x = (chf B ∘ f) x by auto
qed
from L2 f-tr have total-recursive (chf B ∘ f) by (rule comp-tot-rec)
with L3 have total-recursive (chf A) by auto
then show ?thesis by (rule tot-rec-impl-comp)
qed

lemma many-reducible-lm-1: many-reducible-to univ-ce A ⇒ ¬ computable A
proof (rule ccontr)
  assume A1: many-reducible-to univ-ce A
  assume ¬ ¬ computable A
  then have A2: computable A by auto
  from A1 A2 have computable univ-ce by (rule m-red-to-comp)
  with univ-ce-is-not-comp3 show False by auto
qed

lemma one-reducible-lm-1: one-reducible-to univ-ce A ⇒ ¬ computable A
proof -
  assume one-reducible-to univ-ce A
  then have many-reducible-to univ-ce A by (rule one-reducibility-is-many)
  then show ?thesis by (rule many-reducible-lm-1)
qed

lemma one-reducible-lm-2: one-reducible-to-via (nat-to-ce-set n) univ-ce (λ x. c-pair
n x)
proof -
  define f where f x = c-pair n x for x
  have f-is-pr: f ∈ PrimRec1 unfolding f-def by prec
  then have f-tr: total-recursive f by (rule pr-is-total-rec)
  have f-inj: inj f
  proof (rule injI)
    fix x y assume A: f x = f y
    then have c-pair n x = c-pair n y by (unfold f-def)
    then show x = y by (rule c-pair-inj2)
  qed

```

have $\forall x. (x \in (\text{nat-to-ce-set } n)) = (f x \in \text{univ-ce})$
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 $f\text{-tr } f\text{-inj}$ **show** *?thesis* **by** (*unfold f-def, unfold one-reducible-to-via-def, auto*)
qed

lemma *one-reducible-lm-3: one-reducible-to (nat-to-ce-set n) univ-ce*
proof –
have *one-reducible-to-via (nat-to-ce-set n) univ-ce ($\lambda x. c\text{-pair } n x$)* **by** (*rule one-reducible-lm-2*)
then show *?thesis* **by** (*unfold one-reducible-to-def, auto*)
qed

lemma *one-reducible-lm-4: $A \in \text{ce-sets} \implies \text{one-reducible-to } A \text{ univ-ce}$*
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** *?thesis* **by** *auto*
qed

7.10 One-complete sets

definition
one-complete :: *nat set* \Rightarrow *bool* **where**
one-complete = ($\lambda A. A \in \text{ce-sets} \wedge (\forall B. B \in \text{ce-sets} \longrightarrow \text{one-reducible-to } B A)$)

theorem *univ-is-complete: one-complete univ-ce*
proof (*unfold one-complete-def*)
show $\text{univ-ce} \in \text{ce-sets} \wedge (\forall B. B \in \text{ce-sets} \longrightarrow \text{one-reducible-to } B \text{ univ-ce})$
proof
show $\text{univ-ce} \in \text{ce-sets}$ **by** (*rule univ-is-ce*)
next
show $\forall B. B \in \text{ce-sets} \longrightarrow \text{one-reducible-to } B \text{ univ-ce}$
proof (*rule allI, rule impI*)
fix B **assume** $B \in \text{ce-sets}$ **then show** $\text{one-reducible-to } B \text{ univ-ce}$ **by** (*rule one-reducible-lm-4*)
qed
qed
qed

7.11 Index sets, Rice's theorem

definition
index-set :: *nat set* \Rightarrow *bool* **where**
index-set = ($\lambda A. \forall n m. n \in A \wedge (\text{nat-to-ce-set } n = \text{nat-to-ce-set } m) \longrightarrow m \in A$)

lemma *index-set-lm-1*: $\llbracket \text{index-set } A; n \in A; \text{nat-to-ce-set } n = \text{nat-to-ce-set } m \rrbracket \implies m \in A$
proof –
 assume *A1*: *index-set* *A*
 assume *A2*: $n \in A$
 assume *A3*: $\text{nat-to-ce-set } n = \text{nat-to-ce-set } m$
 from *A2* *A3* **have** *L1*: $n \in A \wedge (\text{nat-to-ce-set } n = \text{nat-to-ce-set } m)$ **by** *auto*
 from *A1* **have** *L2*: $\forall n m. n \in A \wedge (\text{nat-to-ce-set } n = \text{nat-to-ce-set } m) \longrightarrow m \in A$ **by** (*unfold index-set-def*)
 from *L1* *L2* **show** *?thesis* **by** *auto*
qed

lemma *index-set-lm-2*: $\text{index-set } A \implies \text{index-set } (\neg A)$
proof –
 assume *A*: *index-set* *A*
 show *index-set* $(\neg A)$
 proof (*unfold index-set-def*)
 show $\forall n m. n \in \neg A \wedge \text{nat-to-ce-set } n = \text{nat-to-ce-set } m \longrightarrow m \in \neg A$
 proof (*rule allI, rule allI, rule impI*)
 fix *n m* **assume** *A1*: $n \in \neg A \wedge \text{nat-to-ce-set } n = \text{nat-to-ce-set } m$
 from *A1* **have** *A2*: $n \in \neg A$ **by** *auto*
 from *A1* **have** *A3*: $\text{nat-to-ce-set } m = \text{nat-to-ce-set } n$ **by** *auto*
 show $m \in \neg A$
 proof
 assume $m \in A$
 from *A* *this* *A3* **have** $n \in A$ **by** (*rule index-set-lm-1*)
 with *A2* **show** *False* **by** *auto*
 qed
 qed
 qed
qed

lemma *Rice-lm-1*: $\llbracket \text{index-set } A; A \neq \{\}; A \neq \text{UNIV}; \exists n \in A. \text{nat-to-ce-set } n = \{\} \rrbracket \implies \text{one-reducible-to univ-ce } (\neg A)$
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 $\text{nat-to-ce-set } e-0 \neq \text{nat-to-ce-set } e-1$
 proof
 assume $\text{nat-to-ce-set } e-0 = \text{nat-to-ce-set } e-1$
 with *A1* *e-0-in-A* **have** $e-1 \in A$ **by** (*rule index-set-lm-1*)
 with *e-1-not-in-A* **show** *False* **by** *auto*

qed
with $e-0\text{-empty}$ **have** $e1\text{-not-empty}$: $\text{nat-to-ce-set } e-1 \neq \{\}$ **by** *auto*
define $we-1$ **where** $we-1 = \text{nat-to-ce-set } e-1$
from $e1\text{-not-empty}$ **have** $we-1\text{-not-empty}$: $we-1 \neq \{\}$ **by** (*unfold we-1-def*)
define r **where** $r = \text{univ-ce} \times we-1$
have $loc\text{-lm-1}$: $\bigwedge x. x \in \text{univ-ce} \implies \forall y. (y \in we-1) = ((x,y) \in r)$ **by** (*unfold r-def, auto*)
have $loc\text{-lm-2}$: $\bigwedge x. x \notin \text{univ-ce} \implies \forall y. (y \in \{\}) = ((x,y) \in r)$ **by** (*unfold r-def, auto*)
have $r\text{-ce}$: $r \in \text{ce-rels}$
proof (*unfold r-def, rule ce-rel-lm-29*)
show $\text{univ-ce} \in \text{ce-sets}$ **by** (*rule univ-is-ce*)
show $we-1 \in \text{ce-sets}$ **by** (*unfold we-1-def, rule nat-to-ce-set-into-ce*)
qed
define $we-n$ **where** $we-n = \text{ce-rel-to-set } r$
from $r\text{-ce}$ **have** $we-n\text{-ce}$: $we-n \in \text{ce-sets}$ **by** (*unfold we-n-def, rule ce-rel-lm-6*)
then **have** $\exists n. we-n = \text{nat-to-ce-set } n$ **by** (*rule nat-to-ce-set-srj*)
then **obtain** n **where** $we-n\text{-df1}$: $we-n = \text{nat-to-ce-set } n$ **by** *auto*
define f **where** $f x = s\text{-ce } n x$ **for** x
from $s\text{-ce-is-pr}$ **have** $f\text{-is-pr}$: $f \in \text{PrimRec1}$ **unfolding** $f\text{-def}$ **by** *prec*
then **have** $f\text{-tr}$: *total-recursive* f **by** (*rule pr-is-total-rec*)
have $f\text{-inj}$: *inj* f
proof (*rule injI*)
fix $x y$
assume $f x = f y$
then **have** $s\text{-ce } n x = s\text{-ce } n y$ **by** (*unfold f-def*)
then **show** $x = y$ **by** (*rule s-ce-inj2*)
qed
have $loc\text{-lm-3}$: $\forall x y. (c\text{-pair } x y \in we-n) = (y \in \text{nat-to-ce-set } (f x))$
proof (*rule allI, rule allI*)
fix $x y$ **show** $(c\text{-pair } x y \in we-n) = (y \in \text{nat-to-ce-set } (f x))$ **by** (*unfold f-def, unfold we-n-df1, simp add: s-ce-1-1-th*)
qed
from $A1$ **have** $loc\text{-lm-4}$: *index-set* $(- A)$ **by** (*rule index-set-lm-2*)
have $loc\text{-lm-5}$: $\forall x. (x \in \text{univ-ce}) = (f x \in -A)$
proof **fix** x **show** $(x \in \text{univ-ce}) = (f x \in -A)$
proof
assume A : $x \in \text{univ-ce}$
then **have** $S1$: $\forall y. (y \in we-1) = ((x,y) \in r)$ **by** (*rule loc-lm-1*)
from $ce\text{-rel-lm-12}$ **have** $\forall y. (c\text{-pair } x y \in ce\text{-rel-to-set } r) = ((x,y) \in r)$ **by** *auto*
then **have** $\forall y. ((x,y) \in r) = (c\text{-pair } x y \in we-n)$ **by** (*unfold we-n-def, auto*)
with $S1$ **have** $\forall y. (y \in we-1) = (c\text{-pair } x y \in we-n)$ **by** *auto*
with $loc\text{-lm-3}$ **have** $\forall y. (y \in we-1) = (y \in \text{nat-to-ce-set } (f x))$ **by** *auto*
then **have** $S2$: $we-1 = \text{nat-to-ce-set } (f x)$ **by** *auto*
then **have** $\text{nat-to-ce-set } e-1 = \text{nat-to-ce-set } (f x)$ **by** (*unfold we-1-def*)
with $loc\text{-lm-4}$ $e-1\text{-not-in-}A$ **show** $f x \in -A$ **by** (*rule index-set-lm-1*)
next
show $f x \in -A \implies x \in \text{univ-ce}$

```

proof (rule ccontr)
  assume fx-in-A:  $f x \in - A$ 
  assume x-not-in-univ:  $x \notin \text{univ-}ce$ 
  then have S1:  $\forall y. (y \in \{\}) = ((x,y) \in r)$  by (rule loc-lm-2)
  from ce-rel-lm-12 have  $\forall y. (c\text{-pair } x y \in ce\text{-rel-to-set } r) = ((x,y) \in r)$  by
auto
  then have  $\forall y. ((x,y) \in r) = (c\text{-pair } x y \in we\text{-}n)$  by (unfold we-n-def,
auto)
  with S1 have  $\forall y. (y \in \{\}) = (c\text{-pair } x y \in we\text{-}n)$  by auto
  with loc-lm-3 have  $\forall y. (y \in \{\}) = (y \in \text{nat-to-}ce\text{-set } (f x))$  by auto
  then have S2:  $\{\} = \text{nat-to-}ce\text{-set } (f x)$  by auto
  then have  $\text{nat-to-}ce\text{-set } e\text{-}0 = \text{nat-to-}ce\text{-set } (f x)$  by (unfold e-0-empty)
  with A1 e-0-in-A have  $f x \in A$  by (rule index-set-lm-1)
  with fx-in-A show False by auto
qed
qed
qed
with f-tr f-inj have one-reducible-to-via univ-ce  $(-A) f$  by (unfold one-reducible-to-via-def,
auto)
  then show ?thesis by (unfold one-reducible-to-def, auto)
qed

```

lemma *Rice-lm-2*: $\llbracket \text{index-set } A; A \neq \{\}; A \neq UNIV; n \in A; \text{nat-to-}ce\text{-set } n = \{\} \rrbracket \implies \text{one-reducible-to univ-}ce (-A)$

```

proof -
  assume A1: index-set A
  assume A2:  $A \neq \{\}$ 
  assume A3:  $A \neq UNIV$ 
  assume A4:  $n \in A$ 
  assume A5:  $\text{nat-to-}ce\text{-set } n = \{\}$ 
  from A4 A5 have S1:  $\exists n \in A. \text{nat-to-}ce\text{-set } n = \{\}$  by auto
  from A1 A2 A3 S1 show ?thesis by (rule Rice-lm-1)
qed

```

theorem *Rice-1*: $\llbracket \text{index-set } A; A \neq \{\}; A \neq UNIV \rrbracket \implies \text{one-reducible-to univ-}ce A \vee \text{one-reducible-to univ-}ce (-A)$

```

proof -
  assume A1: index-set A
  assume A2:  $A \neq \{\}$ 
  assume A3:  $A \neq UNIV$ 
  from ce-empty have  $\exists n. \{\} = \text{nat-to-}ce\text{-set } n$  by (rule nat-to-}ce\text{-set-srj)
  then obtain n where n-empty:  $\text{nat-to-}ce\text{-set } n = \{\}$  by auto
  show ?thesis
proof cases
  assume A:  $n \in A$ 
  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  $A: n \in - A$  by auto
    from  $A1$  have  $S1: \text{index-set } (- A)$  by (rule index-set-lm-2)
    from  $A3$  have  $S2: - A \neq \{\}$  by auto
    from  $A2$  have  $S3: - A \neq UNIV$  by auto
    from  $S1 S2 S3 A$  n-empty have one-reducible-to univ-ce  $(- (- A))$  by (rule
    Rice-lm-2)
    then have one-reducible-to univ-ce  $A$  by simp
    then show ?thesis by auto
  qed
qed

```

theorem *Rice-2*: $\llbracket \text{index-set } A; A \neq \{\}; A \neq UNIV \rrbracket \implies \neg \text{computable } A$

proof –

```

  assume  $A1: \text{index-set } A$ 
  assume  $A2: A \neq \{\}$ 
  assume  $A3: A \neq UNIV$ 
  from  $A1 A2 A3$  have one-reducible-to univ-ce  $A \vee$  one-reducible-to univ-ce  $(- A)$  by (rule Rice-1)
  then have  $S1: \neg \text{one-reducible-to univ-ce } A \longrightarrow \text{one-reducible-to univ-ce } (- A)$ 
  by auto
  show ?thesis
  proof cases
    assume one-reducible-to univ-ce  $A$ 
    then show  $\neg \text{computable } A$  by (rule one-reducible-lm-1)
  next
    assume  $\neg \text{one-reducible-to univ-ce } A$ 
    with  $S1$  have one-reducible-to univ-ce  $(- A)$  by auto
    then have  $\neg \text{computable } (- A)$  by (rule one-reducible-lm-1)
    with computable-complement-3 show  $\neg \text{computable } A$  by auto
  qed
qed

```

theorem *Rice-3*: $\llbracket C \subseteq \text{ce-sets}; \text{computable } \{ n. \text{nat-to-ce-set } n \in C \} \rrbracket \implies C = \{\} \vee C = \text{ce-sets}$

proof (rule *ccontr*)

```

  assume  $A1: C \subseteq \text{ce-sets}$ 
  assume  $A2: \text{computable } \{ n. \text{nat-to-ce-set } n \in C \}$ 
  assume  $A3: \neg (C = \{\} \vee C = \text{ce-sets})$ 
  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 (unfold index-set-def)
    show  $\forall n m. n \in A \wedge \text{nat-to-ce-set } n = \text{nat-to-ce-set } m \longrightarrow m \in A$ 
    proof (rule allI, rule allI, rule impI)
      fix  $n m$  assume  $A1-1: n \in A \wedge \text{nat-to-ce-set } n = \text{nat-to-ce-set } m$ 
      from  $A1-1$  have  $n \in A$  by auto
      then have  $S1-1: \text{nat-to-ce-set } n \in C$  by (unfold A-def, auto)
      from  $A1-1$  have  $\text{nat-to-ce-set } n = \text{nat-to-ce-set } m$  by auto
    qed
  qed

```

```

    with S1-1 have nat-to-ce-set m ∈ C by auto
    then show m ∈ A by (unfold A-def, auto)
  qed
qed
have S2: A ≠ {}
proof -
  from A4 obtain B where S2-1: B ∈ C by auto
  with A1 have B ∈ ce-sets by auto
  then have ∃ n. B = nat-to-ce-set n by (rule nat-to-ce-set-srj)
  then obtain n where B = nat-to-ce-set n ..
  with S2-1 have nat-to-ce-set n ∈ C by auto
  then show ?thesis by (unfold A-def, auto)
qed
have S3: A ≠ UNIV
proof -
  from A1 A5 obtain B where S2-1: B ∉ C and S2-2: B ∈ ce-sets by auto
  from S2-2 have ∃ n. B = nat-to-ce-set n by (rule nat-to-ce-set-srj)
  then obtain n where B = nat-to-ce-set n ..
  with S2-1 have nat-to-ce-set n ∉ C by auto
  then show ?thesis by (unfold A-def, auto)
qed
from S1 S2 S3 have ¬ computable A by (rule Rice-2)
with A2 show False unfolding A-def by auto
qed
end

```

References

- [1] Rogers. *Theory of recursive functions and effective computability*. 1967.
- [2] Soare. *Recursively enumerable sets and degrees*. 1987.