

Recursion Theory I

Michael Nedzelsky

December 12, 2009

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	4
2	Primitive recursive functions	6
2.1	Basic definitions	6
2.2	Bounded least operator	10
2.3	Examples	12
3	Primitive recursive coding of lists of natural numbers	13
4	Primitive recursive functions of one variable	20
4.1	Alternative definition of primitive recursive functions of one variable	20
4.2	The scheme datatype	22
4.3	Indexes of primitive recursive functions of one variables	24
4.4	s-1-1 theorem for primitive recursive functions of one variable	26
5	Finite sets	28
6	The function which is universal for primitive recursive functions of one variable	32

7	Computably enumerable sets of natural numbers	42
7.1	Basic definitions	42
7.2	Basic properties of computably enumerable sets	42
7.3	Enumeration of computably enumerable sets	43
7.4	Characteristic functions	43
7.5	Computably enumerable relations	44
7.6	Total computable functions	47
7.7	Computable sets, Post's theorem	48
7.8	Universal computably enumerable set	48
7.9	s-1-1 theorem, one-one and many-one reducibilities	49
7.10	One-complete sets	51
7.11	Index sets, Rice's theorem	51

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 *<proof>*

lemma *sf-at-1*: *sf* 1 = 1 *<proof>*

lemma *sf-at-Suc*: *sf* (x+1) = *sf* x + x + 1
<proof>

lemma *arg-le-sf*: x ≤ *sf* x
<proof>

lemma *sf-mono*: x ≤ y ⇒ *sf* x ≤ *sf* y
<proof>

lemma *sf-strict-mono*: x < y ⇒ *sf* x < *sf* y

<proof>

lemma *sf-posI*: $x > 0 \implies sf(x) > 0$
<proof>

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

lemma *sf-eq-arg*: $sf\ x = x \implies x \leq 1$
<proof>

lemma *sf-le-sfD*: $sf\ x \leq sf\ y \implies x \leq y$
<proof>

lemma *sf-less-sfD*: $sf\ x < sf\ y \implies x < y$
<proof>

lemma *sf-inj*: $sf\ x = sf\ y \implies x = y$
<proof>

Auxiliary lemmas

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

lemma *sf-aux2*: $sf(z) \leq sf(x+y) + x \implies z \leq x+y$
<proof>

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

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

Basic properties of *c_pair* function

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

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

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

lemma *c-pair-sum-mono*: $(x1::nat) + y1 < x2 + y2 \implies c_pair\ x1\ y1 < c_pair\ x2\ y2$
<proof>

lemma *c-pair-sum-inj*: $c_pair\ x1\ y1 = c_pair\ x2\ y2 \implies x1 + y1 = x2 + y2$
<proof>

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

lemma *c-pair-inj1*: $c\text{-pair } x1 \ y1 = c\text{-pair } x2 \ y2 \implies x1 = x2$ (proof)

lemma *c-pair-inj2*: $c\text{-pair } x1 \ y1 = c\text{-pair } x2 \ y2 \implies y1 = y2$ (proof)

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

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

lemma *c-pair-strict-mono2*: $y1 < y2 \implies c\text{-pair } x \ y1 < c\text{-pair } x \ y2$
(proof)

lemma *c-pair-mono2*: $y1 \leq y2 \implies c\text{-pair } x \ y1 \leq c\text{-pair } x \ y2$
(proof)

1.2 Inverse mapping

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

definition

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

definition

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

definition

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

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

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

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

lemma *c-sum-le-arg*: $c\text{-sum } u \leq u$
(proof)

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

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

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

lemma *c-pair-at-0*: $c\text{-pair } 0 \ 0 = 0$ *<proof>*

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

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

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

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

lemma *c-snd-le-c-sum*: $c\text{-snd } u \leq c\text{-sum } u$ *<proof>*

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

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

lemma *c-sum-is-sum*: $c\text{-sum } u = c\text{-fst } u + c\text{-snd } u$ *<proof>*

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>

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

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

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

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

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

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

lemma *c-snd-less-arg*: $x > 1 \implies c\text{-snd } x < x$
(*proof*)

end

2 Primitive recursive functions

theory *PRecFun* **imports** *CPair*
uses (*Utils.ML*)
begin

This theory contains definition of the primitive recursive functions.

2.1 Basic definitions

primrec

PrimRecOp :: $(\text{nat} \Rightarrow \text{nat}) \Rightarrow (\text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat}) \Rightarrow (\text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat})$

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

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

where

PrimRecOp1 *a h* 0 = *a*

| *PrimRecOp1* *a h* (*Suc* *y*) = *h* *y* (*PrimRecOp1* *a h* *y*)

inductive-set

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

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

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

where

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

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

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

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

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

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

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

$| id3-3: (\lambda x y z. z) \in PrimRec3$
 $| comp1-1: \llbracket f \in PrimRec1; g \in PrimRec1 \rrbracket \implies (\lambda x. f (g x)) \in PrimRec1$
 $| comp1-2: \llbracket f \in PrimRec1; g \in PrimRec2 \rrbracket \implies (\lambda x y. f (g x y)) \in PrimRec2$
 $| comp1-3: \llbracket f \in PrimRec1; g \in PrimRec3 \rrbracket \implies (\lambda x y z. f (g x y z)) \in PrimRec3$
 $| comp2-1: \llbracket f \in PrimRec2; g \in PrimRec1; h \in PrimRec1 \rrbracket \implies (\lambda x. f (g x) (h x)) \in PrimRec1$
 $| comp3-1: \llbracket f \in PrimRec3; g \in PrimRec1; h \in PrimRec1; k \in PrimRec1 \rrbracket \implies (\lambda x. f (g x) (h x) (k x)) \in PrimRec1$
 $| comp2-2: \llbracket f \in PrimRec2; g \in PrimRec2; h \in PrimRec2 \rrbracket \implies (\lambda x y. f (g x y) (h x y)) \in PrimRec2$
 $| comp2-3: \llbracket f \in PrimRec2; g \in PrimRec3; h \in PrimRec3 \rrbracket \implies (\lambda x y z. f (g x y z) (h x y z)) \in PrimRec3$
 $| comp3-2: \llbracket f \in PrimRec3; g \in PrimRec2; h \in PrimRec2; k \in PrimRec2 \rrbracket \implies (\lambda x y. f (g x y) (h x y) (k x y)) \in PrimRec2$
 $| comp3-3: \llbracket f \in PrimRec3; g \in PrimRec3; h \in PrimRec3; k \in PrimRec3 \rrbracket \implies (\lambda x y z. f (g x y z) (h x y z) (k x y z)) \in PrimRec3$
 $| prim-rec: \llbracket g \in PrimRec1; h \in PrimRec3 \rrbracket \implies PrimRecOp g h \in PrimRec2$

lemmas *pr-zero* = *PrimRec1-PrimRec2-PrimRec3.zero*
lemmas *pr-suc* = *PrimRec1-PrimRec2-PrimRec3.suc*
lemmas *pr-id1-1* = *PrimRec1-PrimRec2-PrimRec3.id1-1*
lemmas *pr-id2-1* = *PrimRec1-PrimRec2-PrimRec3.id2-1*
lemmas *pr-id2-2* = *PrimRec1-PrimRec2-PrimRec3.id2-2*
lemmas *pr-id3-1* = *PrimRec1-PrimRec2-PrimRec3.id3-1*
lemmas *pr-id3-2* = *PrimRec1-PrimRec2-PrimRec3.id3-2*
lemmas *pr-id3-3* = *PrimRec1-PrimRec2-PrimRec3.id3-3*
lemmas *pr-comp1-1* = *PrimRec1-PrimRec2-PrimRec3.comp1-1*
lemmas *pr-comp1-2* = *PrimRec1-PrimRec2-PrimRec3.comp1-2*
lemmas *pr-comp1-3* = *PrimRec1-PrimRec2-PrimRec3.comp1-3*
lemmas *pr-comp2-1* = *PrimRec1-PrimRec2-PrimRec3.comp2-1*
lemmas *pr-comp2-2* = *PrimRec1-PrimRec2-PrimRec3.comp2-2*
lemmas *pr-comp2-3* = *PrimRec1-PrimRec2-PrimRec3.comp2-3*
lemmas *pr-comp3-1* = *PrimRec1-PrimRec2-PrimRec3.comp3-1*
lemmas *pr-comp3-2* = *PrimRec1-PrimRec2-PrimRec3.comp3-2*
lemmas *pr-comp3-3* = *PrimRec1-PrimRec2-PrimRec3.comp3-3*
lemmas *pr-rec* = *PrimRec1-PrimRec2-PrimRec3.prim-rec*

$\langle ML \rangle$

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*: $f \in PrimRec2 \implies (\lambda x y. f y x) \in PrimRec2$ $\langle proof \rangle$

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

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

$\langle proof \rangle$

lemma *op-mult-is-pr* [*prec*]: $(\lambda x y. x * y) \in PrimRec2$
 $\langle proof \rangle$

lemma *const-is-pr*: $(\lambda x. (n :: nat)) \in PrimRec1$
 $\langle proof \rangle$

lemma *const-is-pr-2*: $(\lambda x y. (n :: nat)) \in PrimRec2$
 $\langle proof \rangle$

lemma *const-is-pr-3*: $(\lambda x y z. (n :: nat)) \in PrimRec3$
 $\langle proof \rangle$

theorem *pr-rec-last*: $\llbracket g \in PrimRec1; h \in PrimRec3 \rrbracket \implies PrimRecOp-last\ g\ h \in PrimRec2$
 $\langle proof \rangle$

theorem *pr-rec1*: $h \in PrimRec2 \implies PrimRecOp1\ (a :: nat)\ h \in PrimRec1$
 $\langle proof \rangle$

theorem *pr-rec1-scheme*: $\llbracket h \in PrimRec2; f\ 0 = a; \forall y. f\ (Suc\ y) = h\ y\ (f\ y) \rrbracket \implies f \in PrimRec1$
 $\langle proof \rangle$

lemma *pred-is-pr*: $(\lambda x. x - (1 :: nat)) \in PrimRec1$
 $\langle proof \rangle$

lemma *op-sub-is-pr* [*prec*]: $(\lambda x y. x - y) \in PrimRec2$
 $\langle proof \rangle$

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 :: $nat \Rightarrow nat$ **where**
sgn1 $x = (case\ x\ of\ 0 \Rightarrow 0 \mid Suc\ y \Rightarrow 1)$

definition

sgn2 :: $nat \Rightarrow nat$ **where**
sgn2 $x \equiv (case\ x\ of\ 0 \Rightarrow 1 \mid Suc\ y \Rightarrow 0)$

definition

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

lemma [*simp*]: *sgn1* 0 = 0 $\langle proof \rangle$

lemma $[simp]$: $sgn1 (Suc y) = 1$ $\langle proof \rangle$

lemma $[simp]$: $sgn2 0 = 1$ $\langle proof \rangle$

lemma $[simp]$: $sgn2 (Suc y) = 0$ $\langle proof \rangle$

lemma $[simp]$: $x \neq 0 \implies sgn1 x = 1$ $\langle proof \rangle$

lemma $[simp]$: $x \neq 0 \implies sgn2 x = 0$ $\langle proof \rangle$

lemma $sgn1-nz-impl-arg-pos$: $sgn1 x \neq 0 \implies x > 0$ $\langle proof \rangle$

lemma $sgn1-zero-impl-arg-zero$: $sgn1 x = 0 \implies x = 0$ $\langle proof \rangle$

lemma $sgn2-nz-impl-arg-zero$: $sgn2 x \neq 0 \implies x = 0$ $\langle proof \rangle$

lemma $sgn2-zero-impl-arg-pos$: $sgn2 x = 0 \implies x > 0$ $\langle proof \rangle$

lemma $sgn1-nz-eq-arg-pos$: $(sgn1 x \neq 0) = (x > 0)$ $\langle proof \rangle$

lemma $sgn1-zero-eq-arg-zero$: $(sgn1 x = 0) = (x = 0)$ $\langle proof \rangle$

lemma $sgn2-nz-eq-arg-pos$: $(sgn2 x \neq 0) = (x = 0)$ $\langle proof \rangle$

lemma $sgn2-zero-eq-arg-zero$: $(sgn2 x = 0) = (x > 0)$ $\langle proof \rangle$

lemma $sgn1-pos-eq-one$: $sgn1 x > 0 \implies sgn1 x = 1$ $\langle proof \rangle$

lemma $sgn2-pos-eq-one$: $sgn2 x > 0 \implies sgn2 x = 1$ $\langle proof \rangle$

lemma $sgn2-eq-1-sub-arg$: $sgn2 = (\lambda x. 1 - x)$
 $\langle proof \rangle$

lemma $sgn1-eq-1-sub-sgn2$: $sgn1 = (\lambda x. 1 - (sgn2 x))$
 $\langle proof \rangle$

lemma $sgn2-is-pr$ $[prec]$: $sgn2 \in PrimRec1$
 $\langle proof \rangle$

lemma $sgn1-is-pr$ $[prec]$: $sgn1 \in PrimRec1$
 $\langle proof \rangle$

lemma $abs-of-diff-is-pr$ $[prec]$: $abs-of-diff \in PrimRec2$ $\langle proof \rangle$

lemma $abs-of-diff-eq$: $(abs-of-diff x y = 0) = (x = y)$ $\langle proof \rangle$

lemma $sf-is-pr$ $[prec]$: $sf \in PrimRec1$
 $\langle proof \rangle$

lemma $c-pair-is-pr$ $[prec]$: $c-pair \in PrimRec2$
 $\langle proof \rangle$

lemma $if-is-pr$: $\llbracket p \in PrimRec1; q1 \in PrimRec1; q2 \in PrimRec1 \rrbracket \implies (\lambda x. \text{if } (p x = 0) \text{ then } (q1 x) \text{ else } (q2 x)) \in PrimRec1$
 $\langle proof \rangle$

lemma $if-eq-is-pr$ $[prec]$: $\llbracket p1 \in PrimRec1; p2 \in PrimRec1; q1 \in PrimRec1; q2 \in PrimRec1 \rrbracket \implies (\lambda x. \text{if } (p1 x = p2 x) \text{ then } (q1 x) \text{ else } (q2 x)) \in PrimRec1$
 $\langle proof \rangle$

lemma *if-is-pr2* [prec]: $\llbracket p \in \text{PrimRec2}; q1 \in \text{PrimRec2}; q2 \in \text{PrimRec2} \rrbracket \implies (\lambda x y. \text{if } (p x y = 0) \text{ then } (q1 x y) \text{ else } (q2 x y)) \in \text{PrimRec2}$
 ⟨proof⟩

lemma *if-eq-is-pr2*: $\llbracket p1 \in \text{PrimRec2}; p2 \in \text{PrimRec2}; q1 \in \text{PrimRec2}; q2 \in \text{PrimRec2} \rrbracket \implies (\lambda x y. \text{if } (p1 x y = p2 x y) \text{ then } (q1 x y) \text{ else } (q2 x y)) \in \text{PrimRec2}$
 ⟨proof⟩

lemma *if-is-pr3* [prec]: $\llbracket p \in \text{PrimRec3}; q1 \in \text{PrimRec3}; q2 \in \text{PrimRec3} \rrbracket \implies (\lambda x y z. \text{if } (p x y z = 0) \text{ then } (q1 x y z) \text{ else } (q2 x y z)) \in \text{PrimRec3}$
 ⟨proof⟩

lemma *if-eq-is-pr3*: $\llbracket p1 \in \text{PrimRec3}; p2 \in \text{PrimRec3}; q1 \in \text{PrimRec3}; q2 \in \text{PrimRec3} \rrbracket \implies (\lambda x y z. \text{if } (p1 x y z = p2 x y z) \text{ then } (q1 x y z) \text{ else } (q2 x y z)) \in \text{PrimRec3}$
 ⟨proof⟩

⟨ML⟩

2.2 Bounded least operator

definition

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

definition

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

lemma *b-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⟩

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

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

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

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

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⟩

lemma *less-b-least-impl-zero2*: $\llbracket y < x; b\text{-least } f x = x \rrbracket \implies f x y = 0$ *<proof>*

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

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>

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>

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

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

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

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>

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

lemma *less-b-least2-impl-zero2*: $\llbracket z < y; b\text{-least2 } f x y = y \rrbracket \implies f x z = 0$
<proof>

lemma *nz-b-least2-impl-less*: $\llbracket z < y; (f x z) \neq 0 \rrbracket \implies (b\text{-least2 } f x y) < y$
<proof>

lemma *b-least2-less-impl-eq2*: $\llbracket z < y; (f x z) \neq 0 \rrbracket \implies (b\text{-least2 } f x y) = (\text{Least } (\%z. (f x z) \neq 0))$
<proof>

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>

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>

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

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

Suc y
<proof>

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

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

lemma *b-least-def1*: $b\text{-least } f = (\lambda x. b\text{-least2 } f \ x \ x)$ *<proof>*

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

2.3 Examples

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

theorem *c-sum-is-pr*: $c\text{-sum} \in \text{PrimRec1}$
<proof>

theorem *c-fst-is-pr* [*prec*]: $c\text{-fst} \in \text{PrimRec1}$
<proof>

theorem *c-snd-is-pr* [*prec*]: $c\text{-snd} \in \text{PrimRec1}$
<proof>

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

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

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*]: $\text{pr-conv-1-to-2 } (\text{pr-conv-2-to-1 } f) = f$ *<proof>*

lemma [*simp*]: $\text{pr-conv-2-to-1 } (\text{pr-conv-1-to-2 } f) = f$ *<proof>*

lemma [*simp*]: $\text{pr-conv-1-to-3 } (\text{pr-conv-3-to-1 } f) = f$ *<proof>*

lemma [*simp*]: $\text{pr-conv-3-to-1 } (\text{pr-conv-1-to-3 } f) = f$ *<proof>*

lemma [*simp*]: $\text{pr-conv-3-to-2 } (\text{pr-conv-2-to-3 } f) = f$ *<proof>*

lemma [*simp*]: $\text{pr-conv-2-to-3 } (\text{pr-conv-3-to-2 } f) = f$ *<proof>*

lemma *pr-conv-1-to-2-lm*: $f \in \text{PrimRec1} \implies \text{pr-conv-1-to-2 } f \in \text{PrimRec2}$ *<proof>*

lemma *pr-conv-1-to-3-lm*: $f \in \text{PrimRec1} \implies \text{pr-conv-1-to-3 } f \in \text{PrimRec3}$ *<proof>*
lemma *pr-conv-2-to-1-lm*: $f \in \text{PrimRec2} \implies \text{pr-conv-2-to-1 } f \in \text{PrimRec1}$ *<proof>*
lemma *pr-conv-3-to-1-lm*: $f \in \text{PrimRec3} \implies \text{pr-conv-3-to-1 } f \in \text{PrimRec1}$ *<proof>*
lemma *pr-conv-3-to-2-lm*: $f \in \text{PrimRec3} \implies \text{pr-conv-3-to-2 } f \in \text{PrimRec2}$
<proof>
lemma *pr-conv-2-to-3-lm*: $f \in \text{PrimRec2} \implies \text{pr-conv-2-to-3 } f \in \text{PrimRec3}$
<proof>

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

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

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

theorem *mod-is-pr*: $(\lambda a b. a \text{ mod } b) \in \text{PrimRec2}$
<proof>

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>

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

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
 $c\text{-len} :: \text{nat} \Rightarrow \text{nat}$ **where**
 $c\text{-len} = (\lambda (u::\text{nat}). (\text{sgn1 } u) * (c\text{-fst}(u-(1::\text{nat}))+1))$

lemma *c-len-1*: $c\text{-len } u = (\text{case } u \text{ of } 0 \Rightarrow 0 \mid \text{Suc } v \Rightarrow c\text{-fst}(v)+1)$ *<proof>*

lemma *c-len-is-pr*: $c-len \in PrimRec1$ $\langle proof \rangle$

lemma [*simp*]: $c-len\ 0 = 0$ $\langle proof \rangle$

lemma *c-len-2*: $u \neq 0 \implies c-len\ u = c-fst(u-(1::nat))+1$ $\langle proof \rangle$

lemma *c-len-3*: $u > 0 \implies c-len\ u > 0$ $\langle proof \rangle$

lemma *c-len-4*: $c-len\ u = 0 \implies u = 0$
 $\langle proof \rangle$

lemma *c-len-5*: $c-len\ u > 0 \implies u > 0$
 $\langle proof \rangle$

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*: $ls \neq [] \implies c-fold\ (x\#ls) = c-pair\ x\ (c-fold\ ls)$
 $\langle proof \rangle$

primrec

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

where

c-unfold 0 u = []
 | *c-unfold* (Suc k) u = (if k = 0 then [u] else ((c-fst u) # (c-unfold k (c-snd u))))

lemma *c-fold-1*: $c-unfold\ 1\ (c-fold\ [x]) = [x]$ $\langle proof \rangle$

lemma *c-fold-2*: $c-fold\ (c-unfold\ 1\ u) = u$ $\langle proof \rangle$

lemma *c-unfold-1*: $c-unfold\ 1\ u = [u]$ $\langle proof \rangle$

lemma *c-unfold-2*: $c-unfold\ (Suc\ 1)\ u = (c-fst\ u) \# (c-unfold\ 1\ (c-snd\ u))$ $\langle proof \rangle$

lemma *c-unfold-3*: $c-unfold\ (Suc\ 1)\ u = [c-fst\ u, c-snd\ u]$ $\langle proof \rangle$

lemma *c-unfold-4*: $k > 0 \implies c-unfold\ (Suc\ k)\ u = (c-fst\ u) \# (c-unfold\ k\ (c-snd\ u))$ $\langle proof \rangle$

lemma *c-unfold-4-1*: $k > 0 \implies c-unfold\ (Suc\ k)\ u \neq []$ $\langle proof \rangle$

lemma *two*: $(2::nat) = Suc\ 1$ $\langle proof \rangle$

lemma *c-unfold-5*: $c-unfold\ 2\ u = [c-fst\ u, c-snd\ u]$ $\langle proof \rangle$

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

<proof>

lemma *th-lm-1*: $k=1 \implies (\forall u. c\text{-fold } (c\text{-unfold } k \ u) = u)$ *<proof>*

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

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>

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

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

lemma *c-fold-3*: $c\text{-unfold } 2 \ (c\text{-fold } [x, y]) = [x, y]$ *<proof>*

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

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>

lemma *th-3-lm-1*: $\llbracket c\text{-unfold } (\text{length } \ ls) \ (c\text{-fold } \ ls) = \ ls; \ ls = a \ \# \ ls1 \rrbracket \implies c\text{-unfold } (\text{length } (x \ \# \ ls)) \ (c\text{-fold } (x \ \# \ ls)) = x \ \# \ ls$
<proof>

lemma *th-3-lm-2*: $c\text{-unfold } (\text{length } \ ls) \ (c\text{-fold } \ ls) = \ ls \implies c\text{-unfold } (\text{length } (x \ \# \ ls)) \ (c\text{-fold } (x \ \# \ ls)) = x \ \# \ ls$
<proof>

theorem *th-3*: $c\text{-unfold } (\text{length } \ ls) \ (c\text{-fold } \ ls) = \ ls$
<proof>

definition

list-to-nat :: $nat \ list \Rightarrow nat$ **where**

list-to-nat = $(\lambda \ ls. \text{if } \ ls = [] \ \text{then } 0 \ \text{else } (c\text{-pair } ((\text{length } \ ls) - 1) \ (c\text{-fold } \ ls)) + 1)$

definition

nat-to-list :: $nat \Rightarrow nat \ list$ **where**

nat-to-list = $(\lambda \ u. \text{if } u = 0 \ \text{then } [] \ \text{else } (c\text{-unfold } (c\text{-len } \ u) \ (c\text{-snd } (u - (1 :: nat))))))$

lemma *nat-to-list-of-pos*: $u > 0 \implies \text{nat-to-list } u = c\text{-unfold } (c\text{-len } \ u) \ (c\text{-snd } (u - (1 :: nat)))$
<proof>

theorem *list-to-nat-th* [simp]: $list\text{-to-nat} (nat\text{-to-list } u) = u$
<proof>

theorem *nat-to-list-th* [simp]: $nat\text{-to-list} (list\text{-to-nat } ls) = ls$
<proof>

lemma [simp]: $list\text{-to-nat } [] = 0$ <proof>

lemma [simp]: $nat\text{-to-list } 0 = []$ <proof>

theorem *c-len-th-1*: $c\text{-len} (list\text{-to-nat } ls) = length\ ls$
<proof>

theorem *length (nat-to-list u) = c-len u*
<proof>

definition

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

definition

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

definition

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

lemma [simp]: $c\text{-hd } 0 = 0$ <proof>

lemma *c-hd-aux0*: $c\text{-len } u = 1 \implies nat\text{-to-list } u = [c\text{-snd } (u-(1::nat))]$ <proof>

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

lemma *c-hd-aux2*: $c\text{-len } u > 1 \implies c\text{-hd } u = c\text{-fst } (c\text{-snd } (u-(1::nat)))$
<proof>

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

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

lemma *c-hd-is-pr*: $c\text{-hd} \in PrimRec1$
<proof>

lemma [simp]: $c\text{-tl } 0 = 0$ \langle proof \rangle

lemma $c\text{-tl-eq-tl}$: $c\text{-tl } (\text{list-to-nat } ls) = \text{list-to-nat } (\text{tl } ls)$ \langle proof \rangle

lemma $tl\text{-eq-c-tl}$: $tl (\text{nat-to-list } x) = \text{nat-to-list } (c\text{-tl } x)$ \langle proof \rangle

lemma $c\text{-tl-aux1}$: $c\text{-len } u = 1 \implies c\text{-tl } u = 0$ \langle proof \rangle

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

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

lemma $c\text{-tl-less}$: $u > 0 \implies c\text{-tl } u < u$ \langle proof \rangle

lemma $c\text{-tl-le}$: $c\text{-tl } u \leq u$ \langle proof \rangle

theorem $c\text{-tl-is-pr}$: $c\text{-tl} \in \text{PrimRec1}$ \langle proof \rangle

lemma $c\text{-cons-aux1}$: $c\text{-cons } x \ 0 = (c\text{-pair } 0 \ x) + 1$ \langle proof \rangle

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

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

lemma $c\text{-cons-pos}$: $c\text{-cons } x \ u > 0$ \langle proof \rangle

theorem $c\text{-cons-is-pr}$: $c\text{-cons} \in \text{PrimRec2}$ \langle proof \rangle

definition

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

lemma $c\text{-drop-at-0}$ [simp]: $c\text{-drop } 0 \ x = x$ \langle proof \rangle

lemma *c-drop-at-Suc*: $c\text{-drop } (Suc\ y)\ x = c\text{-tl } (c\text{-drop } y\ x)$ *<proof>*

theorem *c-drop-is-pr*: $c\text{-drop} \in PrimRec2$
<proof>

lemma *c-tl-c-drop*: $c\text{-tl } (c\text{-drop } y\ x) = c\text{-drop } y\ (c\text{-tl } x)$
<proof>

lemma *c-drop-at-Suc1*: $c\text{-drop } (Suc\ y)\ x = c\text{-drop } y\ (c\text{-tl } x)$
<proof>

lemma *c-drop-df*: $\forall\ ls.\ drop\ n\ ls = nat\text{-to-list } (c\text{-drop } n\ (list\text{-to-nat } ls))$
<proof>

definition

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

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

lemma *c-nth-at-0*: $c\text{-nth } x\ 0 = c\text{-hd } x$ *<proof>*

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

lemma *c-tl-c-cons [simp]*: $c\text{-tl } (c\text{-cons } x\ y) = y$ *<proof>*

definition

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

lemma *c-f-list-at-0*: $c\text{-f-list } f\ 0\ x = c\text{-cons } (f\ 0\ x)\ 0$ *<proof>*

lemma *c-f-list-at-Suc*: $c\text{-f-list } f\ (Suc\ y)\ x = c\text{-cons } (f\ (Suc\ y)\ x)\ (c\text{-f-list } f\ y\ x)$
<proof>

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

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

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

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

<proof>

lemma *c-f-list-lm-1*: $c\text{-nth } (c\text{-cons } x \ y) \ (Suc \ z) = c\text{-nth } y \ z$ *<proof>*

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>

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

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

theorem *th-rec*: $\llbracket g \in PrimRec1; \alpha \in PrimRec2; h \in PrimRec3; (\forall x \ y. \alpha \ y \ x \leq y); (\forall x. (f \ 0 \ x) = (g \ x)); (\forall x \ y. (f \ (Suc \ y) \ x) = h \ y \ (f \ (\alpha \ y \ x) \ x) \ x) \rrbracket \implies f \in PrimRec2$
<proof>

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

fun *c-assoc-have-key* :: $nat \Rightarrow nat \Rightarrow nat$ **where**

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

lemma *c-assoc-have-key-lm-1*: $y \neq 0 \implies c\text{-assoc-have-key } y \ x = (\text{if } c\text{-fst } (c\text{-hd } y) = x \text{ then } 0 \text{ else } c\text{-assoc-have-key } (c\text{-tl } y) \ x)$ *<proof>*

theorem *c-assoc-have-key-is-pr*: $c\text{-assoc-have-key} \in PrimRec2$
<proof>

fun *c-assoc-value* :: $nat \Rightarrow nat \Rightarrow nat$ **where**

c-assoc-value-df [*simp del*]: $c\text{-assoc-value } y \ x = (\text{if } y = 0 \text{ then } 0 \text{ else } (\text{if } c\text{-fst } (c\text{-hd } y) = x \text{ then } c\text{-snd } (c\text{-hd } y) \text{ else } c\text{-assoc-value } (c\text{-tl } y) \ x))$

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

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

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

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

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-assoc-have-key $z\ x1$
<proof>

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

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*: $\llbracket g \in PrimRec1; h \in PrimRec1 \rrbracket \implies UnaryRecOp\ g\ h \in PrimRec1$ *<proof>*

definition

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

lemma *c-f-pair-to-pr*: $\llbracket f \in PrimRec1; g \in PrimRec1 \rrbracket \implies c\text{-f-pair } f\ g \in PrimRec1$ *<proof>*

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*: $\llbracket f \in PrimRec1'; g \in PrimRec1' \rrbracket \implies (\lambda\ x.\ f\ (g\ x)) \in PrimRec1'$
| *pair*: $\llbracket f \in PrimRec1'; g \in PrimRec1' \rrbracket \implies c\text{-f-pair } f\ g \in PrimRec1'$
| *un-rec*: $\llbracket f \in PrimRec1'; g \in PrimRec1' \rrbracket \implies UnaryRecOp\ f\ g \in PrimRec1'$

lemma *primrec'-into-primrec*: $f \in PrimRec1' \implies f \in PrimRec1$ *<proof>*

lemma *pr-id1-1'*: $(\lambda\ x.\ x) \in PrimRec1'$ *<proof>*

lemma *pr-id2-1'*: $pr\text{-conv-2-to-1 } (\lambda\ x\ y.\ x) \in PrimRec1'$ *<proof>*

lemma *pr-id2-2'*: *pr-conv-2-to-1* $(\lambda x y. y) \in \text{PrimRec1}'$ $\langle \text{proof} \rangle$

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

lemma *pr-id3-2'*: *pr-conv-3-to-1* $(\lambda x y z. y) \in \text{PrimRec1}'$
 $\langle \text{proof} \rangle$

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

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

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

lemma *pr-comp1-2'*: $\llbracket f \in \text{PrimRec1}'; \text{pr-conv-2-to-1 } g \in \text{PrimRec1}' \rrbracket \implies \text{pr-conv-2-to-1 } (\lambda x y. f (g x y)) \in \text{PrimRec1}'$
 $\langle \text{proof} \rangle$

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

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

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

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

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

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

$g1 \in \text{PrimRec1}' \wedge (h1 \in \text{PrimRec3} \longrightarrow \text{pr-conv-3-to-1 } h1 \in \text{PrimRec1}')$
 ⟨proof⟩

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

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)* = ($\lambda x. (\text{sch-to-pr } t1) ((\text{sch-to-pr } t2) x)$)
 | *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* ⟨proof⟩

lemma *sch-to-pr-srj*: $f \in \text{PrimRec1} \implies (\exists \text{sch}. f = \text{sch-to-pr sch})$
 ⟨proof⟩

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 = ($\lambda x. x \text{ mod } 7$)

lemma *c-snd-snd-lt* [*termination-simp*]: *c-snd (c-snd (Suc (Suc x)))* < *Suc (Suc x)*
 ⟨proof⟩

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

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

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

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

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

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

lemma *nat-to-sch-at-2*: $nat\text{-to-sch } 2 = Base\text{-suc}$
 ⟨*proof*⟩

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

lemma *loc-srj-2*: $nat\text{-to-sch } (c\text{-pair } 3\ 0) = Base\text{-snd}$
 ⟨*proof*⟩

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

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

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

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

lemma *loc-srj-5*: $\llbracket \text{nat-to-sch } (\text{sch-to-nat } \text{sch1}) = \text{sch1}; \text{nat-to-sch } (\text{sch-to-nat } \text{sch2}) = \text{sch2} \rrbracket$
 $\implies \text{nat-to-sch } (\text{c-pair } 6 (\text{c-pair } (\text{sch-to-nat } \text{sch1}) (\text{sch-to-nat } \text{sch2}))) =$
Rec-op sch1 sch2
 $\langle \text{proof} \rangle$

lemma *loc-srj-5-1*: $\text{nat-to-sch } (\text{c-pair } 6 (\text{c-pair } n1 n2)) = \text{Rec-op } (\text{nat-to-sch } n1)$
 $(\text{nat-to-sch } n2)$
 $\langle \text{proof} \rangle$

theorem *nat-to-sch-srj*: $\text{nat-to-sch } (\text{sch-to-nat } \text{sch}) = \text{sch}$
 $\langle \text{proof} \rangle$

4.3 Indexes of primitive recursive functions of one variables

definition

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

theorem *nat-to-pr-into-pr*: $\text{nat-to-pr } n \in \text{PrimRec1}$ $\langle \text{proof} \rangle$

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

lemma *nat-to-pr-at-0*: $\text{nat-to-pr } 0 = (\lambda x. 0)$ $\langle \text{proof} \rangle$

definition

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

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

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**

$pair\text{-}by\text{-}index = (\lambda n1\ n2. c\text{-}pair\ 5\ (c\text{-}pair\ n1\ n2))$

definition

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

lemma $comp\text{-}by\text{-}index\text{-}is\text{-}pr$: $comp\text{-}by\text{-}index \in PrimRec2$
 $\langle proof \rangle$

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

lemma $comp\text{-}by\text{-}index\text{-}inj1$: $comp\text{-}by\text{-}index\ x1\ y1 = comp\text{-}by\text{-}index\ x2\ y2 \implies x1 = x2$ $\langle proof \rangle$

lemma $comp\text{-}by\text{-}index\text{-}inj2$: $comp\text{-}by\text{-}index\ x1\ y1 = comp\text{-}by\text{-}index\ x2\ y2 \implies y1 = y2$ $\langle proof \rangle$

lemma $comp\text{-}by\text{-}index\text{-}main$: $nat\text{-}to\text{-}pr\ (comp\text{-}by\text{-}index\ n1\ n2) = (\lambda x. (nat\text{-}to\text{-}pr\ n1)\ ((nat\text{-}to\text{-}pr\ n2)\ x))$ $\langle proof \rangle$

lemma $pair\text{-}by\text{-}index\text{-}is\text{-}pr$: $pair\text{-}by\text{-}index \in PrimRec2$ $\langle proof \rangle$

lemma $pair\text{-}by\text{-}index\text{-}inj$: $pair\text{-}by\text{-}index\ x1\ y1 = pair\text{-}by\text{-}index\ x2\ y2 \implies x1=x2 \wedge y1=y2$
 $\langle proof \rangle$

lemma $pair\text{-}by\text{-}index\text{-}inj1$: $pair\text{-}by\text{-}index\ x1\ y1 = pair\text{-}by\text{-}index\ x2\ y2 \implies x1 = x2$ $\langle proof \rangle$

lemma $pair\text{-}by\text{-}index\text{-}inj2$: $pair\text{-}by\text{-}index\ x1\ y1 = pair\text{-}by\text{-}index\ x2\ y2 \implies y1 = y2$ $\langle proof \rangle$

lemma $pair\text{-}by\text{-}index\text{-}main$: $nat\text{-}to\text{-}pr\ (pair\text{-}by\text{-}index\ n1\ n2) = c\text{-}f\text{-}pair\ (nat\text{-}to\text{-}pr\ n1)\ (nat\text{-}to\text{-}pr\ n2)$ $\langle proof \rangle$

lemma $nat\text{-}to\text{-}sch\text{-}of\text{-}pair\text{-}by\text{-}index$ [simp]: $nat\text{-}to\text{-}sch\ (pair\text{-}by\text{-}index\ n1\ n2) = Pair\text{-}op\ (nat\text{-}to\text{-}sch\ n1)\ (nat\text{-}to\text{-}sch\ n2)$
 $\langle proof \rangle$

lemma $rec\text{-}by\text{-}index\text{-}is\text{-}pr$: $rec\text{-}by\text{-}index \in PrimRec2$ $\langle proof \rangle$

lemma $rec\text{-}by\text{-}index\text{-}inj$: $rec\text{-}by\text{-}index\ x1\ y1 = rec\text{-}by\text{-}index\ x2\ y2 \implies x1=x2 \wedge y1=y2$
 $\langle proof \rangle$

lemma $rec\text{-}by\text{-}index\text{-}inj1$: $rec\text{-}by\text{-}index\ x1\ y1 = rec\text{-}by\text{-}index\ x2\ y2 \implies x1 = x2$ $\langle proof \rangle$

lemma *rec-by-index-inj2*: $\text{rec-by-index } x1 \ y1 = \text{rec-by-index } x2 \ y2 \implies y1 = y2$
<proof>

lemma *rec-by-index-main*: $\text{nat-to-pr } (\text{rec-by-index } n1 \ n2) = \text{UnaryRecOp } (\text{nat-to-pr } n1) \ (\text{nat-to-pr } n2)$ *<proof>*

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

definition

index-of-const :: $\text{nat} \Rightarrow \text{nat}$ **where**
index-of-const = *PrimRecOp1* 0 ($\lambda x \ y. \text{c-pair } 4 \ (\text{c-pair } 2 \ y)$)

lemma *index-of-const-is-pr*: $\text{index-of-const} \in \text{PrimRec1}$
<proof>

lemma *index-of-const-at-0*: $\text{index-of-const } 0 = 0$ *<proof>*

lemma *index-of-const-at-suc*: $\text{index-of-const } (\text{Suc } u) = \text{c-pair } 4 \ (\text{c-pair } 2 \ (\text{index-of-const } u))$ *<proof>*

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

lemma *index-of-const-lm-1*: $(\text{nat-to-pr } (\text{index-of-const } n)) \ 0 = n$ *<proof>*

lemma *index-of-const-inj*: $\text{index-of-const } n1 = \text{index-of-const } n2 \implies n1 = n2$
<proof>

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*: $\text{nat-to-pr } \text{index-of-zero} = (\lambda x. 0)$ *<proof>*

lemma *index-of-suc-main*: $\text{nat-to-pr } \text{index-of-suc} = \text{Suc}$
<proof>

lemma *index-of-c-fst-main*: $\text{nat-to-pr } \text{index-of-c-fst} = \text{c-fst}$ *<proof>*

lemma [*simp*]: $\text{nat-to-sch } \text{index-of-c-fst} = \text{Base-fst}$ *<proof>*

lemma *index-of-c-snd-main*: $\text{nat-to-pr } \text{index-of-c-snd} = \text{c-snd}$ *<proof>*

lemma [*simp*]: $\text{nat-to-sch } \text{index-of-c-snd} = \text{Base-snd}$ *<proof>*

lemma *index-of-id-main*: $\text{nat-to-pr } \text{index-of-id} = (\lambda x. x) \langle \text{proof} \rangle$

definition

index-of-c-pair-n :: $\text{nat} \Rightarrow \text{nat}$ **where**
index-of-c-pair-n = $(\lambda n. \text{pair-by-index } (\text{index-of-const } n) \text{ index-of-id})$

lemma *index-of-c-pair-n-is-pr*: $\text{index-of-c-pair-n} \in \text{PrimRec1}$
 $\langle \text{proof} \rangle$

lemma *index-of-c-pair-n-main*: $\text{nat-to-pr } (\text{index-of-c-pair-n } n) = (\lambda x. \text{c-pair } n x)$
 $\langle \text{proof} \rangle$

lemma *index-of-c-pair-n-inj*: $\text{index-of-c-pair-n } x1 = \text{index-of-c-pair-n } x2 \implies x1=x2$
 $\langle \text{proof} \rangle$

definition

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

lemma *s1-1-is-pr*: $s1-1 \in \text{PrimRec2}$ $\langle \text{proof} \rangle$

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

lemma *s1-1-inj*: $s1-1 x1 y1 = s1-1 x2 y2 \implies x1=x2 \wedge y1=y2$
 $\langle \text{proof} \rangle$

lemma *s1-1-inj1*: $s1-1 x1 y1 = s1-1 x2 y2 \implies x1=x2$ $\langle \text{proof} \rangle$

lemma *s1-1-inj2*: $s1-1 x1 y1 = s1-1 x2 y2 \implies y1=y2$ $\langle \text{proof} \rangle$

primrec

pr-index-enumerator :: $\text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat}$

where

pr-index-enumerator n $0 = n$
 $| \text{pr-index-enumerator } n (\text{Suc } m) = \text{comp-by-index } \text{index-of-id } (\text{pr-index-enumerator } n m)$

theorem *pr-index-enumerator-is-pr*: $\text{pr-index-enumerator} \in \text{PrimRec2}$
 $\langle \text{proof} \rangle$

lemma *pr-index-enumerator-increase1*: $\text{pr-index-enumerator } n m < \text{pr-index-enumerator } (n+1) m$
 $\langle \text{proof} \rangle$

lemma *pr-index-enumerator-increase2*: $\text{pr-index-enumerator } n m < \text{pr-index-enumerator } n (m + 1)$
 $\langle \text{proof} \rangle$

lemma *f-inc-mono*: $(\forall (x::nat). (f::nat \Rightarrow nat) x < f (x+1)) \implies \forall (x::nat) (y::nat). (x < y \implies f x < f y)$
 $\langle proof \rangle$

lemma *pr-index-enumerator-mono1*: $n1 < n2 \implies pr\text{-index-enumerator } n1\ m < pr\text{-index-enumerator } n2\ m$
 $\langle proof \rangle$

lemma *pr-index-enumerator-mono2*: $m1 < m2 \implies pr\text{-index-enumerator } n\ m1 < pr\text{-index-enumerator } n\ m2$
 $\langle proof \rangle$

lemma *f-mono-inj*: $\forall (x::nat) (y::nat). (x < y \implies (f::nat \Rightarrow nat) x < f y) \implies \forall (x::nat) (y::nat). (f x = f y \implies x = y)$
 $\langle proof \rangle$

theorem *pr-index-enumerator-inj1*: $pr\text{-index-enumerator } n1\ m = pr\text{-index-enumerator } n2\ m \implies n1 = n2$
 $\langle proof \rangle$

theorem *pr-index-enumerator-inj2*: $pr\text{-index-enumerator } n\ m1 = pr\text{-index-enumerator } n\ m2 \implies m1 = m2$
 $\langle proof \rangle$

theorem *pr-index-enumerator-main*: $nat\text{-to-pr } n = nat\text{-to-pr } (pr\text{-index-enumerator } n\ m)$
 $\langle proof \rangle$

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

$c\text{-in} :: nat \Rightarrow nat \Rightarrow nat$ **where**
 $c\text{-in} = (\lambda x\ u. (u \text{ div } (2 \wedge x)) \text{ mod } 2)$

lemma *c-in-is-pr*: $c\text{-in} \in PrimRec2$
 $\langle proof \rangle$

definition

nat-to-set :: nat \Rightarrow nat set **where**
nat-to-set u \equiv {x. $2^x \leq u \wedge c\text{-in } x \ u = 1$ }

lemma *c-in-upper-bound*: $c\text{-in } x \ u = 1 \implies 2^x \leq u$
 <proof>

lemma *nat-to-set-upper-bound*: $x \in \text{nat-to-set } u \implies 2^x \leq u$ <proof>

lemma *x-lt-2-x*: $x < 2^x$ <proof>

lemma *nat-to-set-upper-bound1*: $x \in \text{nat-to-set } u \implies x < u$
 <proof>

lemma *nat-to-set-upper-bound2*: $\text{nat-to-set } u \subseteq \{i. i < u\}$
 <proof>

lemma *nat-to-set-is-finite*: finite (nat-to-set u)
 <proof>

lemma *x-in-u-eq*: $(x \in \text{nat-to-set } u) = (c\text{-in } x \ u = 1)$ <proof>

definition

log2 :: nat \Rightarrow nat **where**
log2 = ($\lambda x. \text{Least}(\%z. x < 2^{z+1})$)

lemma *log2-at-0*: $\text{log2 } 0 = 0$
 <proof>

lemma *log2-at-1*: $\text{log2 } 1 = 0$
 <proof>

lemma *log2-le*: $x > 0 \implies 2^{\text{log2 } x} \leq x$
 <proof>

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

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

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

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

lemma *log2-lm1*: $u > 0 \implies u \text{ div } 2^{(\text{log2 } u)} = 1$
 <proof>

lemma *log2-lm2*: $u > 0 \implies c\text{-in } (\text{log2 } u) \ u = 1$
 <proof>

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

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

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

lemma *log2-lm5*: $u > 0 \implies \log_2 u \in \text{nat-to-set } u$
(proof)

lemma *pos-imp-ne*: $u > 0 \implies \text{nat-to-set } u \neq \{\}$
(proof)

lemma *empty-is-zero*: $\text{nat-to-set } u = \{\} \implies u = 0$
(proof)

lemma *log2-is-max*: $u > 0 \implies \log_2 u = \text{Max } (\text{nat-to-set } u)$
(proof)

lemma *zero-is-empty*: $\text{nat-to-set } 0 = \{\}$
(proof)

lemma *ne-imp-pos*: $\text{nat-to-set } u \neq \{\} \implies u > 0$
(proof)

lemma *div-mod-lm*: $y < x \implies ((u + (2::\text{nat})^x) \text{ div } (2::\text{nat})^y) \text{ mod } 2 = (u \text{ div } (2::\text{nat})^y) \text{ mod } 2$
(proof)

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

theorem *nat-to-set-inj*: $\text{nat-to-set } u = \text{nat-to-set } v \implies u = v$
(proof)

definition

set-to-nat :: $\text{nat set} \Rightarrow \text{nat}$ **where**
set-to-nat = $(\lambda D. \text{setsum } (\lambda x. 2^x) D)$

lemma *two-power-sum*: $\text{setsum } (\lambda x. (2::\text{nat})^x) \{i. i < \text{Suc } m\} = (2^{\text{Suc } m}) - 1$
(proof)

lemma *finite-interval*: $\text{finite } \{i. (i::\text{nat}) < m\}$
(proof)

lemma *set-to-nat-at-empty*: $\text{set-to-nat } \{\} = 0$ (proof)

lemma *set-to-nat-of-interval*: $set\text{-to-nat } \{i. (i::nat) < m\} = 2^m - 1$
 ⟨proof⟩

lemma *set-to-nat-mono*: $\llbracket finite\ B; A \subseteq B \rrbracket \implies set\text{-to-nat } A \leq set\text{-to-nat } B$
 ⟨proof⟩

theorem *nat-to-set-srj*: $finite\ (D::nat\ set) \implies nat\text{-to-set } (set\text{-to-nat } D) = D$
 ⟨proof⟩

theorem *nat-to-set-srj1*: $finite\ (D::nat\ set) \implies \exists u. nat\text{-to-set } u = D$
 ⟨proof⟩

lemma *sum-of-pr-is-pr*: $g \in PrimRec1 \implies (\lambda n. setsum\ g\ \{i. i < n\}) \in PrimRec1$
 ⟨proof⟩

lemma *sum-of-pr-is-pr2*: $p \in PrimRec2 \implies (\lambda n\ m. setsum\ (\lambda x. p\ x\ m)\ \{i. i < n\}) \in PrimRec2$
 ⟨proof⟩

lemma *setsum-is-pr*: $g \in PrimRec1 \implies (\lambda u. setsum\ g\ (nat\text{-to-set } u)) \in PrimRec1$
 ⟨proof⟩

definition

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

theorem *c-card-is-pr*: $c\text{-card} \in PrimRec1$
 ⟨proof⟩

definition

c-insert :: $nat \Rightarrow nat \Rightarrow nat$ **where**
c-insert = $(\lambda x\ u. if\ c\text{-in } x\ u = 1\ then\ u\ else\ u + 2^x)$

lemma *c-insert-is-pr*: $c\text{-insert} \in PrimRec2$
 ⟨proof⟩

lemma [*simp*]: $set\text{-to-nat } (nat\text{-to-set } u) = u$
 ⟨proof⟩

lemma *insert-lemma*: $x \notin nat\text{-to-set } u \implies set\text{-to-nat } (nat\text{-to-set } u \cup \{x\}) = u + 2^x$
 ⟨proof⟩

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

definition

c-remove :: $nat \Rightarrow nat \Rightarrow nat$ **where**

$c\text{-remove} = (\lambda x u. \text{if } c\text{-in } x u = 0 \text{ then } u \text{ else } u - 2^x)$

lemma *c-remove-is-pr*: $c\text{-remove} \in \text{PrimRec2}$
<proof>

lemma *remove-lemma*: $x \in \text{nat-to-set } u \implies \text{set-to-nat } (\text{nat-to-set } u - \{x\}) = u - 2^x$
<proof>

lemma *c-remove-df*: $c\text{-remove} = (\lambda x u. \text{set-to-nat } ((\text{nat-to-set } u) - \{x\}))$
<proof>

definition

$c\text{-union} :: \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat}$ **where**
 $c\text{-union} = (\lambda u v. \text{set-to-nat } (\text{nat-to-set } u \cup \text{nat-to-set } v))$

theorem *c-union-is-pr*: $c\text{-union} \in \text{PrimRec2}$
<proof>

definition

$c\text{-diff} :: \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat}$ **where**
 $c\text{-diff} = (\lambda u v. \text{set-to-nat } (\text{nat-to-set } u - \text{nat-to-set } v))$

theorem *c-diff-is-pr*: $c\text{-diff} \in \text{PrimRec2}$
<proof>

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*: $c\text{-intersect} \in \text{PrimRec2}$
<proof>

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

$g\text{-comp} :: \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat}$ **where**
 $g\text{-comp } c\text{-ls } \text{key} = (\text{let } n = c\text{-fst } \text{key}; x = c\text{-snd } \text{key}; m = c\text{-snd } n;$

```

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*: $pr-gr\ 0 = 0$ *<proof>*

lemma *pr-gr-at-Suc*: $pr-gr\ (Suc\ x) = g-step\ (pr-gr\ x)\ (c-fst\ x)$ *<proof>*

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>

definition

```

c-is-sub-fun :: nat => (nat => nat) => bool where
c-is-sub-fun ls f <math>\iff (\forall x. c-assoc-have-key\ ls\ x = 0 \implies c-assoc-value\ ls\ x = f\ x)</math>

```

lemma *c-is-sub-fun-lm-1*: $\llbracket c-is-sub-fun\ ls\ f; c-assoc-have-key\ ls\ x = 0 \rrbracket \implies c-assoc-value\ ls\ x = f\ x$
<proof>

lemma *c-is-sub-fun-lm-2*: $c-is-sub-fun\ ls\ f \implies c-is-sub-fun\ (c-cons\ (c-pair\ x\ (f\ x))\ ls)\ f$
<proof>

lemma *mod7-lm*: $(n::nat)\ mod\ 7 = 0 \vee$

$(n::\text{nat}) \bmod 7 = 1 \vee$
 $(n::\text{nat}) \bmod 7 = 2 \vee$
 $(n::\text{nat}) \bmod 7 = 3 \vee$
 $(n::\text{nat}) \bmod 7 = 4 \vee$
 $(n::\text{nat}) \bmod 7 = 5 \vee$
 $(n::\text{nat}) \bmod 7 = 6 \langle \text{proof} \rangle$

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

lemma *nat-to-sch-0*: $c\text{-fst } n \bmod 7 = 0 \implies \text{nat-to-sch } n = \text{Base-zero}$
 $\langle \text{proof} \rangle$

lemma *loc-lm-1*: $c\text{-fst } n \bmod 7 \neq 0 \implies n > 0$
 $\langle \text{proof} \rangle$

lemma *loc-lm-2*: $c\text{-fst } n \bmod 7 \neq 0 \implies \text{nat-to-sch } n = (\text{let } u=(c\text{-fst } n) \bmod 7;$
 $v=c\text{-snd } n; v1=c\text{-fst } v; v2 = c\text{-snd } v; sch1=\text{nat-to-sch } v1; sch2=\text{nat-to-sch } v2 \text{ in}$
 $loc\text{-f } u \text{ } sch1 \text{ } sch2)$
 $\langle \text{proof} \rangle$

lemma *nat-to-sch-1*: $c\text{-fst } n \bmod 7 = 1 \implies \text{nat-to-sch } n = \text{Base-suc}$
 $\langle \text{proof} \rangle$

lemma *nat-to-sch-2*: $c\text{-fst } n \bmod 7 = 2 \implies \text{nat-to-sch } n = \text{Base-fst}$
 $\langle \text{proof} \rangle$

lemma *nat-to-sch-3*: $c\text{-fst } n \bmod 7 = 3 \implies \text{nat-to-sch } n = \text{Base-snd}$
 $\langle \text{proof} \rangle$

lemma *nat-to-sch-4*: $c\text{-fst } n \bmod 7 = 4 \implies \text{nat-to-sch } n = \text{Comp-op } (\text{nat-to-sch}$
 $(c\text{-fst } (c\text{-snd } n))) (\text{nat-to-sch } (c\text{-snd } (c\text{-snd } n)))$
 $\langle \text{proof} \rangle$

lemma *nat-to-sch-5*: $c\text{-fst } n \bmod 7 = 5 \implies \text{nat-to-sch } n = \text{Pair-op } (\text{nat-to-sch}$
 $(c\text{-fst } (c\text{-snd } n))) (\text{nat-to-sch } (c\text{-snd } (c\text{-snd } n)))$
 $\langle \text{proof} \rangle$

lemma *nat-to-sch-6*: $c\text{-fst } n \bmod 7 = 6 \implies \text{nat-to-sch } n = \text{Rec-op } (\text{nat-to-sch}$
 $(c\text{-fst } (c\text{-snd } n))) (\text{nat-to-sch } (c\text{-snd } (c\text{-snd } n)))$
 $\langle \text{proof} \rangle$

lemma *nat-to-pr-lm-0*: $c\text{-fst } n \bmod 7 = 0 \implies \text{nat-to-pr } n \text{ } x = 0$
 $\langle \text{proof} \rangle$

lemma *nat-to-pr-lm-1*: $c\text{-fst } n \bmod 7 = 1 \implies \text{nat-to-pr } n \text{ } x = \text{Suc } x$
 $\langle \text{proof} \rangle$

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}

lemma *nat-to-pr-lm-3*: $c\text{-fst } n \bmod 7 = 3 \implies \text{nat-to-pr } n \ x = c\text{-snd } x$
{proof}

lemma *nat-to-pr-lm-4*: $c\text{-fst } n \bmod 7 = 4 \implies \text{nat-to-pr } n \ x = (\text{nat-to-pr } (c\text{-fst } (c\text{-snd } n))) (\text{nat-to-pr } (c\text{-snd } (c\text{-snd } n))) x)$
{proof}

lemma *nat-to-pr-lm-5*: $c\text{-fst } n \bmod 7 = 5 \implies \text{nat-to-pr } n \ x = (c\text{-f-pair } (\text{nat-to-pr } (c\text{-fst } (c\text{-snd } n)))) (\text{nat-to-pr } (c\text{-snd } (c\text{-snd } n)))) x$
{proof}

lemma *nat-to-pr-lm-6*: $c\text{-fst } n \bmod 7 = 6 \implies \text{nat-to-pr } n \ x = (\text{UnaryRecOp } (\text{nat-to-pr } (c\text{-fst } (c\text{-snd } n))) (\text{nat-to-pr } (c\text{-snd } (c\text{-snd } n)))) x$
{proof}

lemma *univ-for-pr-lm-0*: $c\text{-fst } (c\text{-fst } \text{key}) \bmod 7 = 0 \implies \text{univ-for-pr } \text{key} = 0$
{proof}

lemma *univ-for-pr-lm-1*: $c\text{-fst } (c\text{-fst } \text{key}) \bmod 7 = 1 \implies \text{univ-for-pr } \text{key} = \text{Suc } (c\text{-snd } \text{key})$
{proof}

lemma *univ-for-pr-lm-2*: $c\text{-fst } (c\text{-fst } \text{key}) \bmod 7 = 2 \implies \text{univ-for-pr } \text{key} = c\text{-fst } (c\text{-snd } \text{key})$
{proof}

lemma *univ-for-pr-lm-3*: $c\text{-fst } (c\text{-fst } \text{key}) \bmod 7 = 3 \implies \text{univ-for-pr } \text{key} = c\text{-snd } (c\text{-snd } \text{key})$
{proof}

lemma *univ-for-pr-lm-4*: $c\text{-fst } (c\text{-fst } \text{key}) \bmod 7 = 4 \implies \text{univ-for-pr } \text{key} = (\text{nat-to-pr } (c\text{-fst } (c\text{-snd } (c\text{-fst } \text{key})))) (\text{nat-to-pr } (c\text{-snd } (c\text{-snd } (c\text{-fst } \text{key})))) (c\text{-snd } \text{key}))$
{proof}

lemma *univ-for-pr-lm-4-1*: $c\text{-fst } (c\text{-fst } \text{key}) \bmod 7 = 4 \implies \text{univ-for-pr } \text{key} = \text{univ-for-pr } (c\text{-pair } (c\text{-fst } (c\text{-snd } (c\text{-fst } \text{key})))) (\text{univ-for-pr } (c\text{-pair } (c\text{-snd } (c\text{-snd } (c\text{-fst } \text{key})))) (c\text{-snd } \text{key}))))$
{proof}

lemma *univ-for-pr-lm-5*: $c\text{-fst } (c\text{-fst } \text{key}) \bmod 7 = 5 \implies \text{univ-for-pr } \text{key} = c\text{-pair } (\text{univ-for-pr } (c\text{-pair } (c\text{-fst } (c\text{-snd } (c\text{-fst } \text{key})))) (c\text{-snd } \text{key}))) (\text{univ-for-pr } (c\text{-pair } (c\text{-snd } (c\text{-snd } (c\text{-fst } \text{key})))) (c\text{-snd } \text{key})))$
{proof}

lemma *univ-for-pr-lm-6-1*: $\llbracket c\text{-fst } (c\text{-fst } \text{key}) \bmod 7 = 6; c\text{-fst } (c\text{-snd } \text{key}) = 0 \rrbracket$

$\implies \text{univ-for-pr key} = \text{univ-for-pr } (c\text{-pair } (c\text{-fst } (c\text{-snd } (c\text{-fst key}))) (c\text{-snd } (c\text{-snd key})))$
 $\langle \text{proof} \rangle$

lemma *univ-for-pr-lm-6-2*: $\llbracket c\text{-fst } (c\text{-fst key}) \bmod 7 = 6; c\text{-fst } (c\text{-snd key}) = \text{Suc } u \rrbracket \implies \text{univ-for-pr key} = \text{univ-for-pr}$
 $(c\text{-pair } (c\text{-snd } (c\text{-snd } (c\text{-fst key})))$
 $(c\text{-pair } (c\text{-pair } u (\text{univ-for-pr } (c\text{-pair } (c\text{-fst key}) (c\text{-pair } u (c\text{-snd } (c\text{-snd key})))))) (c\text{-snd } (c\text{-snd key})))$
 $\langle \text{proof} \rangle$

lemma *univ-for-pr-lm-6-3*: $\llbracket c\text{-fst } (c\text{-fst key}) \bmod 7 = 6; c\text{-fst } (c\text{-snd key}) \neq 0 \rrbracket \implies \text{univ-for-pr key} = \text{univ-for-pr}$
 $(c\text{-pair } (c\text{-snd } (c\text{-snd } (c\text{-fst key})))$
 $(c\text{-pair } (c\text{-pair } (c\text{-fst } (c\text{-snd key}) - 1) (\text{univ-for-pr } (c\text{-pair } (c\text{-fst key})$
 $(c\text{-pair } (c\text{-fst } (c\text{-snd key}) - 1) (c\text{-snd } (c\text{-snd key})))))) (c\text{-snd } (c\text{-snd key})))$
 $\langle \text{proof} \rangle$

lemma *g-comp-lm-0*: $\llbracket c\text{-fst } (c\text{-fst key}) \bmod 7 = 4; c\text{-is-sub-fun } ls \text{ univ-for-pr}; g\text{-comp } ls \text{ key} \neq ls \rrbracket \implies g\text{-comp } ls \text{ key} = c\text{-cons } (c\text{-pair key } (\text{univ-for-pr key})) ls$
 $\langle \text{proof} \rangle$

lemma *g-comp-lm-1*: $\llbracket c\text{-fst } (c\text{-fst key}) \bmod 7 = 4; c\text{-is-sub-fun } ls \text{ univ-for-pr} \rrbracket \implies c\text{-is-sub-fun } (g\text{-comp } ls \text{ key}) \text{ univ-for-pr}$
 $\langle \text{proof} \rangle$

lemma *g-pair-lm-0*: $\llbracket c\text{-fst } (c\text{-fst key}) \bmod 7 = 5; c\text{-is-sub-fun } ls \text{ univ-for-pr}; g\text{-pair } ls \text{ key} \neq ls \rrbracket \implies g\text{-pair } ls \text{ key} = c\text{-cons } (c\text{-pair key } (\text{univ-for-pr key})) ls$
 $\langle \text{proof} \rangle$

lemma *g-pair-lm-1*: $\llbracket c\text{-fst } (c\text{-fst key}) \bmod 7 = 5; c\text{-is-sub-fun } ls \text{ univ-for-pr} \rrbracket \implies c\text{-is-sub-fun } (g\text{-pair } ls \text{ key}) \text{ univ-for-pr}$
 $\langle \text{proof} \rangle$

lemma *g-rec-lm-0*: $\llbracket c\text{-fst } (c\text{-fst key}) \bmod 7 = 6; c\text{-is-sub-fun } ls \text{ univ-for-pr}; g\text{-rec } ls \text{ key} \neq ls \rrbracket \implies g\text{-rec } ls \text{ key} = c\text{-cons } (c\text{-pair key } (\text{univ-for-pr key})) ls$
 $\langle \text{proof} \rangle$

lemma *g-rec-lm-1*: $\llbracket c\text{-fst } (c\text{-fst key}) \bmod 7 = 6; c\text{-is-sub-fun } ls \text{ univ-for-pr} \rrbracket \implies c\text{-is-sub-fun } (g\text{-rec } ls \text{ key}) \text{ univ-for-pr}$
 $\langle \text{proof} \rangle$

lemma *g-step-lm-0*: $c\text{-fst } (c\text{-fst key}) \bmod 7 = 0 \implies g\text{-step } ls \text{ key} = c\text{-cons } (c\text{-pair key } 0) ls$ $\langle \text{proof} \rangle$

lemma *g-step-lm-1*: $c\text{-fst } (c\text{-fst key}) \bmod 7 = 1 \implies g\text{-step } ls \text{ key} = c\text{-cons } (c\text{-pair key } (\text{Suc } (c\text{-snd key}))) ls$ $\langle \text{proof} \rangle$

lemma *g-step-lm-2*: $c\text{-fst } (c\text{-fst key}) \bmod 7 = 2 \implies g\text{-step } ls \text{ key} = c\text{-cons } (c\text{-pair$

$key (c\text{-fst } (c\text{-snd } key))) ls \langle proof \rangle$

lemma *g-step-lm-3*: $c\text{-fst } (c\text{-fst } key) \bmod 7 = 3 \implies g\text{-step } ls \ key = c\text{-cons } (c\text{-pair } key (c\text{-snd } (c\text{-snd } key))) ls \langle proof \rangle$

lemma *g-step-lm-4*: $c\text{-fst } (c\text{-fst } key) \bmod 7 = 4 \implies g\text{-step } ls \ key = g\text{-comp } ls \ key \langle proof \rangle$

lemma *g-step-lm-5*: $c\text{-fst } (c\text{-fst } key) \bmod 7 = 5 \implies g\text{-step } ls \ key = g\text{-pair } ls \ key \langle proof \rangle$

lemma *g-step-lm-6*: $c\text{-fst } (c\text{-fst } key) \bmod 7 = 6 \implies g\text{-step } ls \ key = g\text{-rec } ls \ key \langle proof \rangle$

lemma *g-step-lm-7*: $c\text{-is-sub-fun } ls \ univ\text{-for-pr} \implies c\text{-is-sub-fun } (g\text{-step } ls \ key) \ univ\text{-for-pr} \langle proof \rangle$

theorem *pr-gr-1*: $c\text{-is-sub-fun } (pr\text{-gr } x) \ univ\text{-for-pr} \langle proof \rangle$

lemma *comp-next*: $g\text{-comp } ls \ key = ls \vee c\text{-tl } (g\text{-comp } ls \ key) = ls \langle proof \rangle$

lemma *pair-next*: $g\text{-pair } ls \ key = ls \vee c\text{-tl } (g\text{-pair } ls \ key) = ls \langle proof \rangle$

lemma *rec-next*: $g\text{-rec } ls \ key = ls \vee c\text{-tl } (g\text{-rec } ls \ key) = ls \langle proof \rangle$

lemma *step-next*: $g\text{-step } ls \ key = ls \vee c\text{-tl } (g\text{-step } ls \ key) = ls \langle proof \rangle$

lemma *lm1*: $pr\text{-gr } (Suc \ x) = pr\text{-gr } x \vee c\text{-tl } (pr\text{-gr } (Suc \ x)) = pr\text{-gr } x \langle proof \rangle$

lemma *c-assoc-have-key-pos*: $c\text{-assoc-have-key } ls \ x = 0 \implies ls > 0 \langle proof \rangle$

lemma *lm2*: $c\text{-assoc-have-key } (c\text{-tl } ls) \ key = 0 \implies c\text{-assoc-have-key } ls \ key = 0 \langle proof \rangle$

lemma *lm3*: $c\text{-assoc-have-key } (pr\text{-gr } x) \ key = 0 \implies c\text{-assoc-have-key } (pr\text{-gr } (Suc \ x)) \ key = 0 \langle proof \rangle$

lemma *lm4*: $\llbracket c\text{-assoc-have-key } (pr\text{-gr } x) \ key = 0; 0 \leq y \rrbracket \implies c\text{-assoc-have-key } (pr\text{-gr } (x+y)) \ key = 0 \langle proof \rangle$

lemma *lm5*: $\llbracket c\text{-assoc-have-key } (pr\text{-gr } x) \ key = 0; x \leq y \rrbracket \implies c\text{-assoc-have-key } (pr\text{-gr } y) \ key = 0 \langle proof \rangle$

lemma *loc-upb-lm-1*: $n = 0 \implies (c\text{-fst } n) \bmod 7 = 0$

<proof>

lemma *loc-upb-lm-2*: $(c\text{-fst } n) \bmod 7 > 1 \implies c\text{-snd } n < n$
<proof>

lemma *loc-upb-lm-2-0*: $(c\text{-fst } n) \bmod 7 = 4 \implies c\text{-fst } (c\text{-snd } n) < n$
<proof>

lemma *loc-upb-lm-2-2*: $(c\text{-fst } n) \bmod 7 = 4 \implies c\text{-snd } (c\text{-snd } n) < n$
<proof>

lemma *loc-upb-lm-2-3*: $(c\text{-fst } n) \bmod 7 = 5 \implies c\text{-fst } (c\text{-snd } n) < n$
<proof>

lemma *loc-upb-lm-2-4*: $(c\text{-fst } n) \bmod 7 = 5 \implies c\text{-snd } (c\text{-snd } n) < n$
<proof>

lemma *loc-upb-lm-2-5*: $(c\text{-fst } n) \bmod 7 = 6 \implies c\text{-fst } (c\text{-snd } n) < n$
<proof>

lemma *loc-upb-lm-2-6*: $(c\text{-fst } n) \bmod 7 = 6 \implies c\text{-snd } (c\text{-snd } n) < n$
<proof>

lemma *loc-upb-lm-2-7*: $\llbracket y2 = y1 - (1::\text{nat}); 0 < y1; x1 = c\text{-snd } x; y1 = c\text{-fst } x \rrbracket \implies c\text{-pair } y2 \ x1 < x$
<proof>

consts *loc-upb* :: $\text{nat} \times \text{nat} \Rightarrow \text{nat}$

recdef *loc-upb measure* $(\lambda m. m) <*\text{lex}*> \text{measure } (\lambda n. n)$

aa: *loc-upb* $(n, x) =$
 let $n1 = (c\text{-fst } n) \bmod 7$ *in*
 if $n1 = 0$ *then* $(c\text{-pair } (c\text{-pair } n \ x) \ 0) + 1$ *else*
 if $n1 = 1$ *then* $(c\text{-pair } (c\text{-pair } n \ x) \ 0) + 1$ *else*
 if $n1 = 2$ *then* $(c\text{-pair } (c\text{-pair } n \ x) \ 0) + 1$ *else*
 if $n1 = 3$ *then* $(c\text{-pair } (c\text{-pair } n \ x) \ 0) + 1$ *else*
 if $n1 = 4$ *then* (
 let $m = c\text{-snd } n; m1 = c\text{-fst } m; m2 = c\text{-snd } m;$
 $y = c\text{-assoc-value } (pr\text{-gr } (loc\text{-upb } (m2, x))) (c\text{-pair } m2 \ x)$ *in*
 $(c\text{-pair } (c\text{-pair } n \ x) (loc\text{-upb } (m2, x) + loc\text{-upb}(m1, y))) + 1$
) *else*
 if $n1 = 5$ *then* (
 let $m = c\text{-snd } n; m1 = c\text{-fst } m; m2 = c\text{-snd } m$ *in*
 $(c\text{-pair } (c\text{-pair } n \ x) (loc\text{-upb } (m1, x) + loc\text{-upb}(m2, x))) + 1$
) *else*
 if $n1 = 6$ *then* (
 let $m = c\text{-snd } n; m1 = c\text{-fst } m; m2 = c\text{-snd } m; y1 = c\text{-fst } x; x1 = c\text{-snd } x$
 in
 if $y1 = 0$ *then* (
 $(c\text{-pair } (c\text{-pair } n \ x) (loc\text{-upb } (m1, x1))) + 1$
)

$$\begin{aligned}
& \text{) else (} \\
& \quad \text{let } y2 = y1 - (1 :: \text{nat}); \\
& \quad \quad t1 = \text{c-assoc-value (pr-gr (loc-upb (n, (c-pair y2 x1)))) (c-pair n (c-pair} \\
& \text{y2 x1)); } t2 = \text{c-pair (c-pair y2 t1) x1 in} \\
& \quad \quad (\text{c-pair (c-pair n x) (loc-upb (n, (c-pair y2 x1)) + loc-upb(m2, t2))) + 1} \\
& \quad \text{)} \\
& \text{)} \\
& \text{else 0} \\
& \text{)} \\
& \text{(hints recdef-simp add: loc-upb-lm-2-0 loc-upb-lm-2-2 loc-upb-lm-2-3 loc-upb-lm-2-4} \\
& \text{loc-upb-lm-2-5 loc-upb-lm-2-6 loc-upb-lm-2-7)}
\end{aligned}$$

definition

$$\begin{aligned}
\text{lex-p} &:: ((\text{nat} \times \text{nat}) \times \text{nat} \times \text{nat}) \text{ set where} \\
\text{lex-p} &= ((\text{measure } (\lambda m. m)) <*\text{lex}*> (\text{measure } (\lambda n. n)))
\end{aligned}$$

lemma *wf-lex-p*: $\text{wf}(\text{lex-p})$
 $\langle \text{proof} \rangle$

lemma *lex-p-eq*: $((n', x'), (n, x)) \in \text{lex-p} = (n' < n \vee n' = n \wedge x' < x)$
 $\langle \text{proof} \rangle$

lemma *loc-upb-lex-0*: $\text{c-fst } n \bmod 7 = 0 \implies \text{c-assoc-have-key (pr-gr (loc-upb (n, x))) (c-pair n x)} = 0$
 $\langle \text{proof} \rangle$

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

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

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

lemma *loc-upb-lex-4*: $\llbracket \bigwedge n' x'. ((n', x'), (n, x)) \in \text{lex-p} \implies \text{c-assoc-have-key (pr-gr (loc-upb (n', x'))) (c-pair n' x')} = 0; \text{c-fst } n \bmod 7 = 4 \rrbracket \implies$
 $\text{c-assoc-have-key (pr-gr (loc-upb (n, x))) (c-pair n x)} = 0$
 $\langle \text{proof} \rangle$

lemma *loc-upb-lex-5*: $\llbracket \bigwedge n' x'. ((n', x'), (n, x)) \in \text{lex-p} \implies \text{c-assoc-have-key (pr-gr (loc-upb (n', x'))) (c-pair n' x')} = 0; \text{c-fst } n \bmod 7 = 5 \rrbracket \implies$
 $\text{c-assoc-have-key (pr-gr (loc-upb (n, x))) (c-pair n x)} = 0$
 $\langle \text{proof} \rangle$

lemma *loc-upb-6-z*: $\llbracket c\text{-fst } n \text{ mod } 7 = 6; c\text{-fst } x = 0 \rrbracket \implies \text{loc-upb } (n, x) = c\text{-pair}$
 $(c\text{-pair } n \ x) (\text{loc-upb } (c\text{-snd } n), c\text{-snd } x) + 1$ *<proof>*

lemma *loc-upb-6*: $\llbracket c\text{-fst } n \text{ mod } 7 = 6; c\text{-fst } x \neq 0 \rrbracket \implies \text{loc-upb } (n, x) =$
 $\text{let } m = c\text{-snd } n; m1 = c\text{-fst } m; m2 = c\text{-snd } m; y1 = c\text{-fst}$
 $x; x1 = c\text{-snd } x;$

$y2 = y1 - 1;$
 $t1 = c\text{-assoc-value } (pr\text{-gr } (\text{loc-upb } (n, c\text{-pair } y2 \ x1)))$
 $(c\text{-pair } n \ (c\text{-pair } y2 \ x1));$
 $t2 = c\text{-pair } (c\text{-pair } y2 \ t1) \ x1 \text{ in}$
 $c\text{-pair } (c\text{-pair } n \ x) (\text{loc-upb } (n, c\text{-pair } y2 \ x1) + (\text{loc-upb}$
 $(m2, t2))) + 1$
<proof>

lemma *loc-upb-lex-6*: $\llbracket \bigwedge n' \ x'. ((n', x'), (n, x)) \in \text{lex-p} \implies c\text{-assoc-have-key } (pr\text{-gr}$
 $(\text{loc-upb } (n', x')) (c\text{-pair } n' \ x') = 0;$
 $c\text{-fst } n \text{ mod } 7 = 6 \rrbracket \implies$
 $c\text{-assoc-have-key } (pr\text{-gr } (\text{loc-upb } (n, x))) (c\text{-pair } n \ x) = 0$
<proof>

lemma *wf-upb-step-0*:

$\llbracket \bigwedge n' \ x'. ((n', x'), (n, x)) \in \text{lex-p} \implies c\text{-assoc-have-key } (pr\text{-gr } (\text{loc-upb } (n', x'))$
 $(c\text{-pair } n' \ x') = 0 \rrbracket \implies$
 $c\text{-assoc-have-key } (pr\text{-gr } (\text{loc-upb } (n, x))) (c\text{-pair } n \ x) = 0$
<proof>

lemma *wf-upb-step*:

$\llbracket \bigwedge p2. (p2, p1) \in \text{lex-p} \implies c\text{-assoc-have-key } (pr\text{-gr } (\text{loc-upb } (fst \ p2, snd \ p2)))$
 $(c\text{-pair } (fst \ p2) \ (snd \ p2)) = 0 \rrbracket \implies$
 $c\text{-assoc-have-key } (pr\text{-gr } (\text{loc-upb } (fst \ p1, snd \ p1))) (c\text{-pair } (fst \ p1) \ (snd \ p1))$
 $= 0$
<proof>

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

theorem *pr-gr-value*: $c\text{-assoc-value } (pr\text{-gr } (\text{loc-upb } (n, x))) (c\text{-pair } n \ x) = \text{univ-for-pr}$
 $(c\text{-pair } n \ x)$
<proof>

theorem *g-comp-is-pr*: $g\text{-comp} \in \text{PrimRec2}$
<proof>

theorem *g-pair-is-pr*: $g\text{-pair} \in \text{PrimRec2}$
<proof>

theorem *g-rec-is-pr*: $g\text{-rec} \in \text{PrimRec2}$
<proof>

theorem *g-step-is-pr*: $g\text{-step} \in \text{PrimRec2}$
<proof>

theorem *pr-gr-is-pr*: $pr\text{-gr} \in \text{PrimRec1}$
<proof>

end

7 Computationally enumerable sets of natural numbers

theory *RecEnSet*
imports *PRecList PRecFun2 PRecFinSet PRecUnGr*
begin

7.1 Basic definitions

definition
fn-to-set :: $(\text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat}) \Rightarrow \text{nat set}$ **where**
fn-to-set $f = \{ x. \exists y. f\ x\ y = 0 \}$

definition
ce-sets :: $(\text{nat set}) \text{ set}$ **where**
ce-sets = $\{ (fn\text{-to-set}\ p) \mid p. p \in \text{PrimRec2} \}$

7.2 Basic properties of computably enumerable sets

lemma *ce-set-lm-1*: $p \in \text{PrimRec2} \implies fn\text{-to-set}\ p \in \text{ce-sets}$ *<proof>*

lemma *ce-set-lm-2*: $\llbracket p \in \text{PrimRec2}; \forall x. (x \in A) = (\exists y. p\ x\ y = 0) \rrbracket \implies A \in \text{ce-sets}$
<proof>

lemma *ce-set-lm-3*: $A \in \text{ce-sets} \implies \exists p \in \text{PrimRec2}. A = fn\text{-to-set}\ p$
<proof>

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

lemma *ce-set-lm-5*: $\llbracket A \in \text{ce-sets}; p \in \text{PrimRec1} \rrbracket \implies \{ x . p\ x \in A \} \in \text{ce-sets}$
<proof>

lemma *ce-set-lm-6*: $\llbracket A \in \text{ce-sets}; A \neq \{\} \rrbracket \implies \exists q \in \text{PrimRec1}. A = \{ q\ x \mid x. x \in UNIV \}$
<proof>

lemma *ce-set-lm-7*: $\llbracket A \in \text{ce-sets}; p \in \text{PrimRec1} \rrbracket \implies \{ p x \mid x. x \in A \} \in \text{ce-sets}$
 $\langle \text{proof} \rangle$

theorem *ce-empty*: $\{\} \in \text{ce-sets}$
 $\langle \text{proof} \rangle$

theorem *ce-univ*: $UNIV \in \text{ce-sets}$
 $\langle \text{proof} \rangle$

theorem *ce-singleton*: $\{a\} \in \text{ce-sets}$
 $\langle \text{proof} \rangle$

theorem *ce-union*: $\llbracket A \in \text{ce-sets}; B \in \text{ce-sets} \rrbracket \implies A \cup B \in \text{ce-sets}$
 $\langle \text{proof} \rangle$

theorem *ce-intersect*: $\llbracket A \in \text{ce-sets}; B \in \text{ce-sets} \rrbracket \implies A \cap B \in \text{ce-sets}$
 $\langle \text{proof} \rangle$

7.3 Enumeration of computably enumerable sets

definition

nat-to-ce-set :: $\text{nat} \Rightarrow (\text{nat set})$ **where**
nat-to-ce-set = $(\lambda n. \text{fn-to-set } (\text{pr-conv-1-to-2 } (\text{nat-to-pr } n)))$

lemma *nat-to-ce-set-lm-1*: $\text{nat-to-ce-set } n = \{ x . \exists y. (\text{nat-to-pr } n) (c\text{-pair } x y) = 0 \}$
 $\langle \text{proof} \rangle$

lemma *nat-to-ce-set-into-ce*: $\text{nat-to-ce-set } n \in \text{ce-sets}$
 $\langle \text{proof} \rangle$

lemma *nat-to-ce-set-srj*: $A \in \text{ce-sets} \implies \exists n. A = \text{nat-to-ce-set } n$
 $\langle \text{proof} \rangle$

7.4 Characteristic functions

definition

chf :: $\text{nat set} \Rightarrow (\text{nat} \Rightarrow \text{nat})$ — Characteristic function **where**
chf = $(\lambda A x. \text{if } x \in A \text{ then } 0 \text{ else } 1)$

definition

zero-set :: $(\text{nat} \Rightarrow \text{nat}) \Rightarrow \text{nat set}$ **where**
zero-set = $(\lambda f. \{ x. f x = 0 \})$

lemma *chf-lm-1* [*simp*]: $\text{zero-set } (\text{chf } A) = A$ $\langle \text{proof} \rangle$

lemma *chf-lm-2*: $(x \in A) = (\text{chf } A x = 0)$ $\langle \text{proof} \rangle$

lemma *chf-lm-3*: $(x \notin A) = (\text{chf } A x = 1)$ $\langle \text{proof} \rangle$

lemma *chf-lm-4*: $\text{chf } A \in \text{PrimRec1} \implies A \in \text{ce-sets}$
(proof)

lemma *chf-lm-5*: $\text{finite } A \implies \text{chf } A \in \text{PrimRec1}$
(proof)

theorem *ce-finite*: $\text{finite } A \implies A \in \text{ce-sets}$
(proof)

7.5 Computably enumerable relations

definition

ce-set-to-rel :: $\text{nat set} \Rightarrow (\text{nat} * \text{nat}) \text{ set}$ **where**
 $\text{ce-set-to-rel} = (\lambda A. \{ (c\text{-fst } x, c\text{-snd } x) \mid x. x \in A \})$

definition

ce-rel-to-set :: $(\text{nat} * \text{nat}) \text{ set} \Rightarrow \text{nat set}$ **where**
 $\text{ce-rel-to-set} = (\lambda R. \{ c\text{-pair } x y \mid x y. (x,y) \in R \})$

definition

ce-rels :: $((\text{nat} * \text{nat}) \text{ set}) \text{ set}$ **where**
 $\text{ce-rels} = \{ R \mid R. \text{ce-rel-to-set } R \in \text{ce-sets} \}$

lemma *ce-rel-lm-1* [simp]: $\text{ce-set-to-rel } (\text{ce-rel-to-set } r) = r$
(proof)

lemma *ce-rel-lm-2* [simp]: $\text{ce-rel-to-set } (\text{ce-set-to-rel } A) = A$
(proof)

lemma *ce-rels-def1*: $\text{ce-rels} = \{ \text{ce-set-to-rel } A \mid A. A \in \text{ce-sets} \}$
(proof)

lemma *ce-rel-to-set-inj*: *inj ce-rel-to-set*
(proof)

lemma *ce-rel-to-set-surj*: *surj ce-rel-to-set*
(proof)

lemma *ce-rel-to-set-bij*: *bij ce-rel-to-set*
(proof)

lemma *ce-set-to-rel-inj*: *inj ce-set-to-rel*
(proof)

lemma *ce-set-to-rel-surj*: *surj ce-set-to-rel*
(proof)

lemma *ce-set-to-rel-bij*: *bij ce-set-to-rel*
(proof)

lemma *ce-rel-lm-3*: $A \in ce\text{-sets} \implies ce\text{-set-to-rel } A \in ce\text{-rels}$
<proof>

lemma *ce-rel-lm-4*: $ce\text{-set-to-rel } A \in ce\text{-rels} \implies A \in ce\text{-sets}$
<proof>

lemma *ce-rel-lm-5*: $(A \in ce\text{-sets}) = (ce\text{-set-to-rel } A \in ce\text{-rels})$
<proof>

lemma *ce-rel-lm-6*: $r \in ce\text{-rels} \implies ce\text{-rel-to-set } r \in ce\text{-sets}$
<proof>

lemma *ce-rel-lm-7*: $ce\text{-rel-to-set } r \in ce\text{-sets} \implies r \in ce\text{-rels}$
<proof>

lemma *ce-rel-lm-8*: $(r \in ce\text{-rels}) = (ce\text{-rel-to-set } r \in ce\text{-sets})$ *<proof>*

lemma *ce-rel-lm-9*: $(x,y) \in r \implies c\text{-pair } x\ y \in ce\text{-rel-to-set } r$ *<proof>*

lemma *ce-rel-lm-10*: $x \in A \implies (c\text{-fst } x, c\text{-snd } x) \in ce\text{-set-to-rel } A$ *<proof>*

lemma *ce-rel-lm-11*: $c\text{-pair } x\ y \in ce\text{-rel-to-set } r \implies (x,y) \in r$
<proof>

lemma *ce-rel-lm-12*: $(c\text{-pair } x\ y \in ce\text{-rel-to-set } r) = ((x,y) \in r)$
<proof>

lemma *ce-rel-lm-13*: $(x,y) \in ce\text{-set-to-rel } A \implies c\text{-pair } x\ y \in A$
<proof>

lemma *ce-rel-lm-14*: $c\text{-pair } x\ y \in A \implies (x,y) \in ce\text{-set-to-rel } A$
<proof>

lemma *ce-rel-lm-15*: $((x,y) \in ce\text{-set-to-rel } A) = (c\text{-pair } x\ y \in A)$
<proof>

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>

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>

lemma *ce-rel-lm-18*: $((c\text{-fst } x, c\text{-snd } x) \in ce\text{-set-to-rel } A) = (x \in A)$
<proof>

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>

lemma *ce-rel-lm-20*: $((c\text{-fst } x, c\text{-snd } x) \in r) = (x \in ce\text{-rel-to-set } r)$
<proof>

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>

lemma *ce-rel-lm-22*: $r \in ce\text{-rels} \implies \exists p \in PrimRec3. r = \{ (x,y). \exists u. p x y u = 0 \}$
<proof>

lemma *ce-rel-lm-23*: $\llbracket p \in PrimRec3; \forall x y. ((x,y) \in r) = (\exists u. p x y u = 0) \rrbracket \implies r \in ce\text{-rels}$
<proof>

lemma *ce-rel-lm-24*: $\llbracket r \in ce\text{-rels}; s \in ce\text{-rels} \rrbracket \implies s \circ r \in ce\text{-rels}$
<proof>

lemma *ce-rel-lm-25*: $r \in ce\text{-rels} \implies r^{-1} \in ce\text{-rels}$
<proof>

lemma *ce-rel-lm-26*: $r \in ce\text{-rels} \implies Domain r \in ce\text{-sets}$
<proof>

lemma *ce-rel-lm-27*: $r \in ce\text{-rels} \implies Range r \in ce\text{-sets}$
<proof>

lemma *ce-rel-lm-28*: $r \in ce\text{-rels} \implies Field r \in ce\text{-sets}$
<proof>

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>

lemma *ce-rel-lm-30*: $\{\} \in ce\text{-rels}$
<proof>

lemma *ce-rel-lm-31*: $UNIV \in ce\text{-rels}$
<proof>

lemma *ce-rel-lm-32*: $ce\text{-rel-to-set } (r \cup s) = (ce\text{-rel-to-set } r) \cup (ce\text{-rel-to-set } s)$
<proof>

lemma *ce-rel-lm-33*: $\llbracket r \in ce\text{-rels}; s \in ce\text{-rels} \rrbracket \implies r \cup s \in ce\text{-rels}$
<proof>

lemma *ce-rel-lm-34*: $ce\text{-rel-to-set } (r \cap s) = (ce\text{-rel-to-set } r) \cap (ce\text{-rel-to-set } s)$
<proof>

lemma *ce-rel-lm-35*: $\llbracket r \in ce\text{-rels}; s \in ce\text{-rels} \rrbracket \implies r \cap s \in ce\text{-rels}$

<proof>

lemma *ce-rel-lm-36*: $ce\text{-set-to-rel } (A \cup B) = (ce\text{-set-to-rel } A) \cup (ce\text{-set-to-rel } B)$
<proof>

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>

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>

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*: $(x,y) \in graph\ f \implies y = f\ x$ *<proof>*

lemma *graph-lm-2*: $y = f\ x \implies (x,y) \in graph\ f$ *<proof>*

lemma *graph-lm-3*: $((x,y) \in graph\ f) = (y = f\ x)$ *<proof>*

lemma *graph-lm-4*: $graph\ (f \circ g) = (graph\ g) \circ (graph\ f)$ *<proof>*

definition

$c\text{-graph} :: (nat \Rightarrow nat) \Rightarrow nat \text{ set}$ **where**
 $c\text{-graph} = (\lambda f. \{ c\text{-pair } x\ (f\ x) \mid x. x \in UNIV \})$

lemma *c-graph-lm-1*: $c\text{-pair } x\ y \in c\text{-graph } f \implies y = f\ x$
<proof>

lemma *c-graph-lm-2*: $y = f\ x \implies c\text{-pair } x\ y \in c\text{-graph } f$ *<proof>*

lemma *c-graph-lm-3*: $(c\text{-pair } x\ y \in c\text{-graph } f) = (y = f\ x)$
<proof>

lemma *c-graph-lm-4*: $c\text{-graph } f = ce\text{-rel-to-set } (graph\ f)$ *<proof>*

lemma *c-graph-lm-5*: $graph\ f = ce\text{-set-to-rel } (c\text{-graph } f)$ *<proof>*

definition

$total\text{-recursive} :: (nat \Rightarrow nat) \Rightarrow bool$ **where**
 $total\text{-recursive} = (\lambda f. graph\ f \in ce\text{-rels})$

lemma *total-recursive-def1*: $total\text{-recursive} = (\lambda f. c\text{-graph } f \in ce\text{-sets})$
<proof>

theorem *pr-is-total-rec*: $f \in PrimRec1 \implies total\text{-recursive } f$

<proof>

theorem *comp-tot-rec*: $\llbracket \text{total-recursive } f; \text{total-recursive } g \rrbracket \implies \text{total-recursive } (f \text{ o } g)$
<proof>

lemma *univ-for-pr-tot-rec-lm*: $c\text{-graph } \text{univ-for-pr} \in \text{ce-sets}$
<proof>

theorem *univ-for-pr-tot-rec*: $\text{total-recursive } \text{univ-for-pr}$
<proof>

7.7 Computable sets, Post's theorem

definition

computable :: $\text{nat set} \Rightarrow \text{bool}$ **where**
computable = $(\lambda A. A \in \text{ce-sets} \wedge \neg A \in \text{ce-sets})$

lemma *computable-complement-1*: $\text{computable } A \implies \text{computable } (\neg A)$
<proof>

lemma *computable-complement-2*: $\text{computable } (\neg A) \implies \text{computable } A$
<proof>

lemma *computable-complement-3*: $(\text{computable } A) = (\text{computable } (\neg A))$ *<proof>*

theorem *comp-impl-tot-rec*: $\text{computable } A \implies \text{total-recursive } (\text{chf } A)$
<proof>

theorem *tot-rec-impl-comp*: $\text{total-recursive } (\text{chf } A) \implies \text{computable } A$
<proof>

theorem *post-th-0*: $(\text{computable } A) = (\text{total-recursive } (\text{chf } A))$
<proof>

7.8 Universal computably enumerable set

definition

univ-ce :: nat set **where**
univ-ce = $\{ c\text{-pair } n \ x \mid n \ x. x \in \text{nat-to-ce-set } n \}$

lemma *univ-for-pr-lm*: $\text{univ-for-pr } (c\text{-pair } n \ x) = (\text{nat-to-pr } n) \ x$ *<proof>*

theorem *univ-is-ce*: $\text{univ-ce} \in \text{ce-sets}$
<proof>

lemma *univ-ce-lm-1*: $(c\text{-pair } n \ x \in \text{univ-ce}) = (x \in \text{nat-to-ce-set } n)$
<proof>

theorem *univ-ce-is-not-comp1*: $\neg \text{univ-ce} \notin \text{ce-sets}$

<proof>

theorem *univ-ce-is-not-comp2*: \neg *total-recursive* (*chf univ-ce*)
<proof>

theorem *univ-ce-is-not-comp3*: \neg *computable univ-ce*
<proof>

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*
<proof>

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*
<proof>

lemma *s-ce-inj*: *s-ce* *e1* *x1* = *s-ce* *e2* *x2* \implies *e1=e2* \wedge *x1=x2*
<proof>

lemma *s-ce-inj1*: *s-ce* *e1* *x* = *s-ce* *e2* *x* \implies *e1=e2*
<proof>

lemma *s-ce-inj2*: *s-ce* *e* *x1* = *s-ce* *e* *x2* \implies *x1=x2*
<proof>

theorem *s1-1-th1*: \forall *n* *x* *y*. ((*nat-to-pr* *n*) (*c-pair* *x* *y*)) = (*nat-to-pr* (*s1-1* *n* *x*)) *y*
<proof>

lemma *s-lm*: (*nat-to-pr* (*s-ce* *e* *x*)) (*c-pair* *y* *z*) = (*nat-to-pr* *e*) (*c-pair* (*c-pair* *x*
y) *z*)
<proof>

theorem *s-ce-1-1-th*: (*c-pair* *x* *y* \in *nat-to-ce-set* *e*) = (*y* \in *nat-to-ce-set* (*s-ce* *e*
x))
<proof>

definition

one-reducible-to-via :: (nat set) ⇒ (nat set) ⇒ (nat ⇒ nat) ⇒ bool **where**
one-reducible-to-via = (λ A B f. total-recursive f ∧ inj f ∧ (∀ x. (x ∈ A) = (f x ∈ B)))

definition

one-reducible-to :: (nat set) ⇒ (nat set) ⇒ bool **where**
one-reducible-to = (λ A B. ∃ f. *one-reducible-to-via* A B f)

definition

many-reducible-to-via :: (nat set) ⇒ (nat set) ⇒ (nat ⇒ nat) ⇒ bool **where**
many-reducible-to-via = (λ A B f. total-recursive f ∧ (∀ x. (x ∈ A) = (f x ∈ B)))

definition

many-reducible-to :: (nat set) ⇒ (nat set) ⇒ bool **where**
many-reducible-to = (λ A B. ∃ f. *many-reducible-to-via* A B f)

lemma *inj-comp*: [inj f; inj g] ⇒ inj (f o g)
 ⟨proof⟩

lemma *one-reducible-to-via-trans*: [*one-reducible-to-via* A B f; *one-reducible-to-via* B C g] ⇒ *one-reducible-to-via* A C (g o f)
 ⟨proof⟩

lemma *one-reducible-to-trans*: [*one-reducible-to* A B; *one-reducible-to* B C] ⇒ *one-reducible-to* A C
 ⟨proof⟩

lemma *one-reducible-to-via-refl*: *one-reducible-to-via* A A (λ x. x)
 ⟨proof⟩

lemma *one-reducible-to-refl*: *one-reducible-to* A A
 ⟨proof⟩

lemma *many-reducible-to-via-trans*: [*many-reducible-to-via* A B f; *many-reducible-to-via* B C g] ⇒ *many-reducible-to-via* A C (g o f)
 ⟨proof⟩

lemma *many-reducible-to-trans*: [*many-reducible-to* A B; *many-reducible-to* B C] ⇒ *many-reducible-to* A C
 ⟨proof⟩

lemma *one-reducibility-via-is-many*: *one-reducible-to-via* A B f ⇒ *many-reducible-to-via* A B f
 ⟨proof⟩

lemma *one-reducibility-is-many*: *one-reducible-to* A B ⇒ *many-reducible-to* A B
 ⟨proof⟩

lemma *many-reducible-to-via-refl: many-reducible-to-via A A* ($\lambda x. x$)
(proof)

lemma *many-reducible-to-refl: many-reducible-to A A*
(proof)

theorem *m-red-to-comp: [many-reducible-to A B; computable B] \implies computable A*
(proof)

lemma *many-reducible-lm-1: many-reducible-to univ-ce A \implies \neg computable A*
(proof)

lemma *one-reducible-lm-1: one-reducible-to univ-ce A \implies \neg computable A*
(proof)

lemma *one-reducible-lm-2: one-reducible-to-via (nat-to-ce-set n) univ-ce ($\lambda x. c\text{-pair } n x$)*
(proof)

lemma *one-reducible-lm-3: one-reducible-to (nat-to-ce-set n) univ-ce*
(proof)

lemma *one-reducible-lm-4: A \in ce-sets \implies one-reducible-to A univ-ce*
(proof)

7.10 One-complete sets

definition

one-complete :: nat set \implies 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)

7.11 Index sets, Rice's theorem

definition

index-set :: nat set \implies 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: [index-set A; n \in A; nat-to-ce-set n = nat-to-ce-set m] \implies m \in A*
(proof)

lemma *index-set-lm-2: index-set A \implies index-set ($\neg A$)*
(proof)

lemma *Rice-lm-1*: $\llbracket \text{index-set } A; A \neq \{\}; A \neq UNIV; \exists n \in A. \text{nat-to-ce-set } n = \{\} \rrbracket \implies \text{one-reducible-to univ-ce } (\neg A)$
<proof>

lemma *Rice-lm-2*: $\llbracket \text{index-set } A; A \neq \{\}; A \neq UNIV; n \in A; \text{nat-to-ce-set } n = \{\} \rrbracket \implies \text{one-reducible-to univ-ce } (\neg A)$
<proof>

theorem *Rice-1*: $\llbracket \text{index-set } A; A \neq \{\}; A \neq UNIV \rrbracket \implies \text{one-reducible-to univ-ce } A \vee \text{one-reducible-to univ-ce } (\neg A)$
<proof>

theorem *Rice-2*: $\llbracket \text{index-set } A; A \neq \{\}; A \neq UNIV \rrbracket \implies \neg \text{computable } A$
<proof>

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>

end

References

- [1] Rogers. *Theory of recursive functions and effective computability*. 1967.
- [2] Soare. *Recursively enumerable sets and degrees*. 1987.