

Infinite Lists

David Trachtenherz

May 31, 2023

Abstract

We introduce a theory of infinite lists in HOL formalized as functions over naturals (folder `ListInf`, theories `ListInf` and `ListInf_Prefix`). It also provides additional results for finite lists (theory `ListInf/List2`), natural numbers (folder `CommonArith`, esp. division/modulo, naturals with infinity), sets (folder `CommonSet`, esp. cutting/truncating sets, traversing sets of naturals).

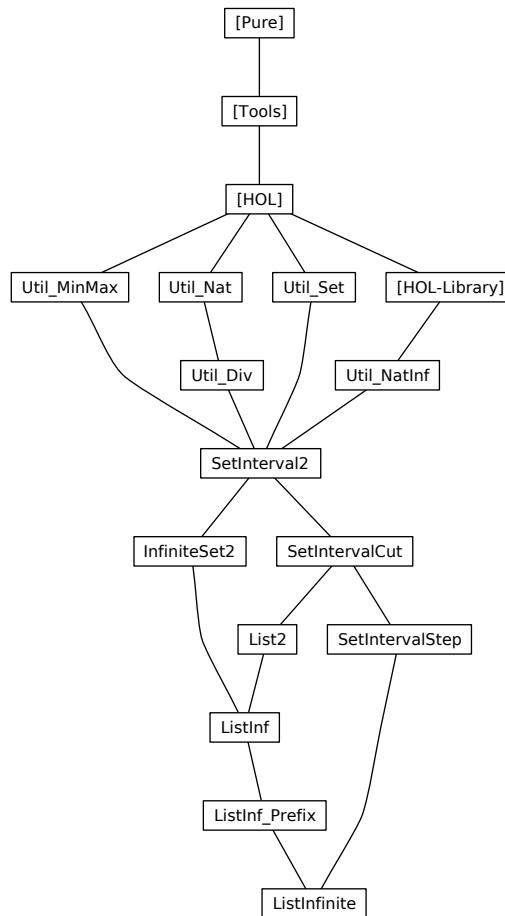
Contents

1	Convenience results for set quantifiers	3
1.1	Some auxiliary results for HOL rules	3
1.1.1	Some auxiliary results for <i>Let</i>	3
1.1.2	Some auxiliary <i>if</i> -rules	3
1.1.3	Some auxiliary rules for function composition	3
1.2	Some auxiliary lemmata for quantifiers	3
1.2.1	Auxiliary results for universal and existential quantifiers	3
1.2.2	Auxiliary results for <i>empty</i> sets	4
1.2.3	Some auxiliary results for subset and membership relation	4
2	Order and linear order: min and max	4
2.1	Additional lemmata about <i>min</i> and <i>max</i>	4
3	Results for natural arithmetics with infinity	5
3.1	Arithmetic operations with <i>enat</i>	5
3.1.1	Additional definitions	5
3.1.2	Addition, difference, order	6
3.1.3	Multiplication and division	7
4	Results for natural arithmetics	10
4.1	Some convenience arithmetic lemmata	10
4.2	Additional facts about inequalities	12
4.3	Inequalities for <code>Suc</code> and <code>pred</code>	13
4.4	Additional facts about cancellation in (in-)equalities	13

5	Results for division and modulo operators on integers	16
5.1	Additional (in-)equalities with <i>div</i> and <i>mod</i>	16
5.2	Additional results for addition and subtraction with <i>mod</i>	17
5.2.1	Divisor subtraction with <i>div</i> and <i>mod</i>	19
5.2.2	Modulo equality and modulo of difference	20
5.3	Some additional lemmata about integer <i>div</i> and <i>mod</i>	20
5.4	Some further (in-)equality results for <i>div</i> and <i>mod</i>	22
5.5	Additional multiplication results for <i>mod</i> and <i>div</i>	23
5.6	Some factor distribution facts for <i>mod</i>	23
5.7	More results about quotient <i>div</i> with addition and subtraction	24
5.8	Further results about <i>div</i> and <i>mod</i>	25
5.8.1	Some auxiliary facts about <i>mod</i>	25
5.8.2	Some auxiliary facts about <i>div</i>	26
6	Sets of natural numbers	30
6.1	Auxiliary results for monotonic, injective and surjective functions over sets	30
6.1.1	Monotonicity	30
6.1.2	Injectivity	31
6.1.3	Surjectivity	32
6.1.4	Induction over natural sets	33
6.1.5	Monotonicity and injectivity of arithmetic operators	34
6.2	<i>Min</i> and <i>Max</i> elements of a set	35
6.2.1	Basic results, as for <i>Least</i>	35
6.2.2	<i>Max</i> for sets over <i>enat</i>	39
6.2.3	<i>Min</i> and <i>Max</i> for set operations	40
6.3	Some auxiliary results for set operations	42
6.3.1	Some additional abbreviations for relations	42
6.3.2	Auxiliary results for <i>singletons</i>	43
6.3.3	Auxiliary results for <i>finite</i> and <i>infinite</i> sets	44
6.3.4	Some auxiliary results for disjoint sets	45
6.3.5	Some auxiliary results for subset relation	46
6.3.6	Auxiliary results for intervals from <i>SetInterval</i>	46
6.3.7	Auxiliary results for <i>card</i>	49
7	Cutting linearly ordered and natural sets	50
7.1	Set restriction	50
7.2	Cut operators for sets/intervals	52
7.2.1	Definitions and basic lemmata for cut operators	52
7.2.2	Basic results for cut operators	54
7.2.3	Relations between cut operators	61
7.2.4	Function images with cut operators	61
7.2.5	Finiteness and cardinality with cut operators	62
7.2.6	Cutting a set at <i>Min</i> or <i>Max</i> element	63

7.2.7	Cut operators with intervals from <code>SetInterval</code>	65
7.2.8	Mirroring finite natural sets between their <i>Min</i> and <i>Max</i> element	66
8	Stepping through sets of natural numbers	71
8.1	Function <i>inext</i> and <i>iprev</i> for stepping through natural sets . .	71
8.2	<i>inext-nth</i> and <i>iprev-nth</i> – nth element of a natural set	83
8.3	Induction over arbitrary natural sets using the functions <i>inext</i> and <i>iprev</i>	87
8.4	Natural intervals with <i>inext</i> and <i>iprev</i>	88
8.5	Further result for <i>inext-nth</i> and <i>iprev-nth</i>	89
9	Additional definitions and results for lists	90
9.1	Additional definitions and results for lists	90
9.1.1	Additional lemmata about list emptiness	91
9.1.2	Additional lemmata about <i>take</i> , <i>drop</i> , <i>hd</i> , <i>last</i> , <i>nth</i> and <i>filter</i>	91
9.1.3	Ordered lists	94
9.1.4	Additional definitions and results for sublists	97
9.1.5	Natural set images with lists	100
9.1.6	Mapping lists of functions to lists	102
9.1.7	Mapping functions with two arguments to lists	104
10	Set operations with results of type <code>enat</code>	107
10.1	Set operations with <i>enat</i>	107
10.1.1	Basic definitions	107
10.2	Results for <i>icard</i>	107
11	Additional definitions and results for lists	110
11.1	Infinite lists	111
11.1.1	Appending a functions to a list	111
11.1.2	<i>take</i> and <i>drop</i> for infinite lists	115
11.1.3	<i>zip</i> for infinite lists	121
11.1.4	Mapping functions with two arguments to infinite lists	122
11.2	Generalised lists as combination of finite and infinite lists . .	123
11.2.1	Basic definitions	123
11.2.2	<i>glength</i>	124
11.2.3	@ _{<i>g</i>} – <i>gappend</i>	125
11.2.4	<i>gmap</i>	125
11.2.5	<i>gset</i>	126
11.2.6	! _{<i>g</i>} – <i>gnth</i>	126
11.2.7	<i>gtake</i> and <i>gdrop</i>	127

12 Prefices on finite and infinite lists	128
12.1 Additional list prefix results	128
12.2 Counting equal pairs	129
12.3 Prefix length	131
12.4 Prefix infimum	133
12.5 Prefices for infinite lists	135



1 Convenience results for set quantifiers

```
theory Util-Set
imports Main
begin
```

1.1 Some auxiliary results for HOL rules

```
lemma conj-disj-absorb:  $(P \wedge Q \vee Q) = Q$  <proof>
lemma disj-eq-distribL:  $((a \vee b) = (a \vee c)) = (a \vee (b = c))$  <proof>
lemma disj-eq-distribR:  $((a \vee c) = (b \vee c)) = ((a = b) \vee c)$  <proof>
```

1.1.1 Some auxiliary results for Let

```
lemma Let-swap:  $f$  (let  $x=a$  in  $g$   $x$ ) = (let  $x=a$  in  $f$  ( $g$   $x$ )) <proof>
```

1.1.2 Some auxiliary if-rules

```
lemma if-P':  $\llbracket P; x = z \rrbracket \Longrightarrow (if\ P\ then\ x\ else\ y) = z$  <proof>
lemma if-not-P':  $\llbracket \neg P; y = z \rrbracket \Longrightarrow (if\ P\ then\ x\ else\ y) = z$  <proof>

lemma if-P-both:  $\llbracket Q\ x; Q\ y \rrbracket \Longrightarrow Q$  (if  $P$  then  $x$  else  $y$ ) <proof>
lemma if-P-both-in-set:  $\llbracket x \in s; y \in s \rrbracket \Longrightarrow (if\ P\ then\ x\ else\ y) \in s$  <proof>
```

1.1.3 Some auxiliary rules for function composition

```
lemma comp2-conv:  $f1 \circ f2 = (\lambda x. f1\ (f2\ x))$  <proof>
lemma comp3-conv:  $f1 \circ f2 \circ f3 = (\lambda x. f1\ (f2\ (f3\ x)))$  <proof>
```

1.2 Some auxiliary lemmata for quantifiers

1.2.1 Auxiliary results for universal and existential quantifiers

```
lemma ball-cong2:
 $\llbracket I \subseteq A; \forall x \in A. f\ x = g\ x \rrbracket \Longrightarrow (\forall x \in I. P\ (f\ x)) = (\forall x \in I. P\ (g\ x))$  <proof>
lemma bex-cong2:
 $\llbracket I \subseteq A; \forall x \in I. f\ x = g\ x \rrbracket \Longrightarrow (\exists x \in I. P\ (f\ x)) = (\exists x \in I. P\ (g\ x))$  <proof>
lemma ball-all-cong:
 $\forall x. f\ x = g\ x \Longrightarrow (\forall x \in I. P\ (f\ x)) = (\forall x \in I. P\ (g\ x))$  <proof>
lemma bex-all-cong:
 $\forall x. f\ x = g\ x \Longrightarrow (\exists x \in I. P\ (f\ x)) = (\exists x \in I. P\ (g\ x))$  <proof>
lemma all-cong:
 $\forall x. f\ x = g\ x \Longrightarrow (\forall x. P\ (f\ x)) = (\forall x. P\ (g\ x))$  <proof>
lemma ex-cong:
 $\forall x. f\ x = g\ x \Longrightarrow (\exists x. P\ (f\ x)) = (\exists x. P\ (g\ x))$  <proof>
```

```
lemmas all-eqI = iff-allI
```

```
lemmas ex-eqI = iff-exI
```

lemma *all-imp-eqI*:

$$\llbracket P = P'; \bigwedge x. P x \implies Q x = Q' x \rrbracket \implies$$

$$(\forall x. P x \longrightarrow Q x) = (\forall x. P' x \longrightarrow Q' x)$$
<proof>

lemma *ex-imp-eqI*:

$$\llbracket P = P'; \bigwedge x. P x \implies Q x = Q' x \rrbracket \implies$$

$$(\exists x. P x \wedge Q x) = (\exists x. P' x \wedge Q' x)$$
<proof>

1.2.2 Auxiliary results for empty sets

lemma *empty-imp-not-in*: $x \notin \{\}$ *<proof>*
lemma *ex-imp-not-empty*: $\exists x. x \in A \implies A \neq \{\}$ *<proof>*
lemma *in-imp-not-empty*: $x \in A \implies A \neq \{\}$ *<proof>*
lemma *not-empty-imp-ex*: $A \neq \{\} \implies \exists x. x \in A$ *<proof>*
lemma *not-ex-in-conv*: $(\neg (\exists x. x \in A)) = (A = \{\})$ *<proof>*

1.2.3 Some auxiliary results for subset and membership relation

lemma *beX-subset-imp-beX*: $\llbracket \exists x \in A. P x; A \subseteq B \rrbracket \implies \exists x \in B. P x$ *<proof>*
lemma *beX-imp-ex*: $\exists x \in A. P x \implies \exists x. P x$ *<proof>*
lemma *ball-subset-imp-ball*: $\llbracket \forall x \in B. P x; A \subseteq B \rrbracket \implies \forall x \in A. P x$ *<proof>*
lemma *all-imp-ball*: $\forall x. P x \implies \forall x \in A. P x$ *<proof>*

lemma *mem-Collect-eq-not*: $(a \notin \{x. P x\}) = (\neg P a)$ *<proof>*
lemma *Collect-not-in-imp-not*: $a \notin \{x. P x\} \implies \neg P a$ *<proof>*
lemma *Collect-not-imp-not-in*: $\neg P a \implies a \notin \{x. P x\}$ *<proof>*
lemma *Collect-is-subset*: $\{x \in A. P x\} \subseteq A$ *<proof>*

end

2 Order and linear order: min and max

theory *Util-MinMax*
imports *Main*
begin

2.1 Additional lemmata about *min* and *max*

lemma *min-less-imp-conj*: $(z::'a::linorder) < \min x y \implies z < x \wedge z < y$ *<proof>*
lemma *conj-less-imp-min*: $\llbracket z < x; z < y \rrbracket \implies (z::'a::linorder) < \min x y$ *<proof>*

lemmas *min-le-iff-conj* = *min.bounded-iff*
lemma *min-le-imp-conj*: $(z::'a::linorder) \leq \min x y \implies z \leq x \wedge z \leq y$ *<proof>*

lemmas *conj-le-imp-min* = *min.boundedI*

lemmas *min-eqL* = *min.absorb1*

lemmas *min-eqR* = *min.absorb2*

lemmas *min-eq* = *min-eqL min-eqR*

lemma *max-less-imp-conj*: $\max x y < b \implies x < (b::('a::linorder)) \wedge y < b$ *<proof>*

lemma *conj-less-imp-max*: $\llbracket x < (b::('a::linorder)); y < b \rrbracket \implies \max x y < b$ *<proof>*

lemmas *max-le-iff-conj* = *max.bounded-iff*

lemma *max-le-imp-conj*: $\max x y \leq b \implies x \leq (b::('a::linorder)) \wedge y \leq b$ *<proof>*

lemmas *conj-le-imp-max* = *max.boundedI*

lemmas *max-eqL* = *max.absorb1*

lemmas *max-eqR* = *max.absorb2*

lemmas *max-eq* = *max-eqL max-eqR*

lemmas *le-minI1* = *min.cobounded1*

lemmas *le-minI2* = *min.cobounded2*

lemma

min-le-monoR: $(a::'a::linorder) \leq b \implies \min x a \leq \min x b$ **and**

min-le-monoL: $(a::'a::linorder) \leq b \implies \min a x \leq \min b x$

<proof>

lemma

max-le-monoR: $(a::'a::linorder) \leq b \implies \max x a \leq \max x b$ **and**

max-le-monoL: $(a::'a::linorder) \leq b \implies \max a x \leq \max b x$

<proof>

end

3 Results for natural arithmetics with infinity

theory *Util-NatInf*

imports *HOL-Library.Extended-Nat*

begin

3.1 Arithmetic operations with *enat*

3.1.1 Additional definitions

instantiation *enat* :: *modulo*

begin

definition

div-enat-def [code del]:

$a \text{ div } b \equiv (\text{case } a \text{ of}$
 $(\text{enat } x) \Rightarrow (\text{case } b \text{ of } (\text{enat } y) \Rightarrow \text{enat } (x \text{ div } y) \mid \infty \Rightarrow 0) \mid$
 $\infty \Rightarrow (\text{case } b \text{ of } (\text{enat } y) \Rightarrow ((\text{case } y \text{ of } 0 \Rightarrow 0 \mid \text{Suc } n \Rightarrow \infty)) \mid \infty \Rightarrow \infty))$)

definition

mod-enat-def [code del]:

$a \text{ mod } b \equiv (\text{case } a \text{ of}$
 $(\text{enat } x) \Rightarrow (\text{case } b \text{ of } (\text{enat } y) \Rightarrow \text{enat } (x \text{ mod } y) \mid \infty \Rightarrow a) \mid$
 $\infty \Rightarrow \infty)$

instance $\langle \text{proof} \rangle$

end

lemmas *enat-arith-defs* =

zero-enat-def one-enat-def

plus-enat-def diff-enat-def times-enat-def div-enat-def mod-enat-def

declare *zero-enat-def*[simp]

lemmas *ineq0-conv-enat*[simp] = *i0-less*[symmetric, unfolded *zero-enat-def*]

lemmas *iless-eSuc0-enat*[simp] = *iless-eSuc0*[unfolded *zero-enat-def*]

3.1.2 Addition, difference, order

lemma *diff-eq-conv-nat*: $(x - y = (z::nat)) = (\text{if } y < x \text{ then } x = y + z \text{ else } z = 0)$

$\langle \text{proof} \rangle$

lemma *idiff-eq-conv*:

$(x - y = (z::enat)) =$

$(\text{if } y < x \text{ then } x = y + z \text{ else if } x \neq \infty \text{ then } z = 0 \text{ else } z = \infty)$

$\langle \text{proof} \rangle$

lemmas *idiff-eq-conv-enat* = *idiff-eq-conv*[unfolded *zero-enat-def*]

lemma *less-eq-idiff-eq-sum*: $y \leq (x::enat) \implies (z \leq x - y) = (z + y \leq x)$

$\langle \text{proof} \rangle$

lemma *eSuc-pred*: $0 < n \implies \text{eSuc } (n - \text{eSuc } 0) = n$

$\langle \text{proof} \rangle$

lemma *eSuc-pred-enat* = *eSuc-pred*[unfolded *zero-enat-def*]

lemmas *iadd-0-enat*[simp] = *add-0-left*[**where** 'a = enat, unfolded *zero-enat-def*]

lemmas *iadd-0-right-enat*[simp] = *add-0-right*[**where** 'a=enat, unfolded *zero-enat-def*]

lemma *ile-add1*: $(n::enat) \leq n + m$

<proof>

lemma *ile-add2*: $(n::\text{enat}) \leq m + n$

<proof>

lemma *iadd-iless-mono*: $\llbracket (i::\text{enat}) < j; k < l \rrbracket \implies i + k < j + l$

<proof>

lemma *trans-ile-iadd1*: $i \leq (j::\text{enat}) \implies i \leq j + m$

<proof>

lemma *trans-ile-iadd2*: $i \leq (j::\text{enat}) \implies i \leq m + j$

<proof>

lemma *trans-iless-iadd1*: $i < (j::\text{enat}) \implies i < j + m$

<proof>

lemma *trans-iless-iadd2*: $i < (j::\text{enat}) \implies i < m + j$

<proof>

lemma *iadd-ileD1*: $m + k \leq (n::\text{enat}) \implies m \leq n$

<proof>

lemma *iadd-ileD2*: $m + k \leq (n::\text{enat}) \implies k \leq n$

<proof>

lemma *idiff-ile-mono*: $m \leq (n::\text{enat}) \implies m - l \leq n - l$

<proof>

lemma *idiff-ile-mono2*: $m \leq (n::\text{enat}) \implies l - n \leq l - m$

<proof>

lemma *idiff-iless-mono*: $\llbracket m < (n::\text{enat}); l \leq m \rrbracket \implies m - l < n - l$

<proof>

lemma *idiff-iless-mono2*: $\llbracket m < (n::\text{enat}); m < l \rrbracket \implies l - n \leq l - m$

<proof>

3.1.3 Multiplication and division

lemmas *imult-infinity-enat[simp]* = *imult-infinity[unfolded zero-enat-def]*

lemmas *imult-infinity-right-enat[simp]* = *imult-infinity-right[unfolded zero-enat-def]*

lemma *idiv-enat-enat[simp, code]*: $\text{enat } a \text{ div enat } b = \text{enat } (a \text{ div } b)$

<proof>

lemma *idiv-infinity*: $0 < n \implies (\infty::\text{enat}) \text{ div } n = \infty$

<proof>

lemmas *idiv-infinity-enat[simp]* = *idiv-infinity[unfolded zero-enat-def]*

lemma *idiv-infinity-right*[simp]: $n \neq \infty \implies n \text{ div } (\infty::\text{enat}) = 0$
 ⟨proof⟩

lemma *idiv-infinity-if*: $n \text{ div } \infty = (\text{if } n = \infty \text{ then } \infty \text{ else } 0::\text{enat})$
 ⟨proof⟩

lemmas *idiv-infinity-if-enat* = *idiv-infinity-if*[unfolded zero-enat-def]

lemmas *imult-0-enat*[simp] = *mult-zero-left*[**where** 'a=enat,unfolding zero-enat-def]

lemmas *imult-0-right-enat*[simp] = *mult-zero-right*[**where** 'a=enat,unfolding zero-enat-def]

lemmas *imult-is-0-enat* = *imult-is-0*[unfolding zero-enat-def]

lemmas *enat-0-less-mult-iff-enat* = *enat-0-less-mult-iff*[unfolding zero-enat-def]

lemma *imult-infinity-if*: $\infty * n = (\text{if } n = 0 \text{ then } 0 \text{ else } \infty::\text{enat})$
 ⟨proof⟩

lemma *imult-infinity-right-if*: $n * \infty = (\text{if } n = 0 \text{ then } 0 \text{ else } \infty::\text{enat})$
 ⟨proof⟩

lemmas *imult-infinity-if-enat* = *imult-infinity-if*[unfolding zero-enat-def]

lemmas *imult-infinity-right-if-enat* = *imult-infinity-right-if*[unfolding zero-enat-def]

lemmas *imult-is-infinity-enat* = *imult-is-infinity*[unfolding zero-enat-def]

lemma *idiv-by-0*: $(a::\text{enat}) \text{ div } 0 = 0$
 ⟨proof⟩

lemmas *idiv-by-0-enat*[simp, code] = *idiv-by-0*[unfolding zero-enat-def]

lemma *idiv-0*: $0 \text{ div } (a::\text{enat}) = 0$
 ⟨proof⟩

lemmas *idiv-0-enat*[simp, code] = *idiv-0*[unfolding zero-enat-def]

lemma *imod-by-0*: $(a::\text{enat}) \text{ mod } 0 = a$
 ⟨proof⟩

lemmas *imod-by-0-enat*[simp, code] = *imod-by-0*[unfolding zero-enat-def]

lemma *imod-0*: $0 \text{ mod } (a::\text{enat}) = 0$
 ⟨proof⟩

lemmas *imod-0-enat*[simp, code] = *imod-0*[unfolding zero-enat-def]

lemma *imod-enat-enat*[simp, code]: $\text{enat } a \text{ mod enat } b = \text{enat } (a \text{ mod } b)$
 ⟨proof⟩

lemma *imod-infinity*[simp, code]: $\infty \text{ mod } n = (\infty::\text{enat})$
 ⟨proof⟩

lemma *imod-infinity-right*[simp, code]: $n \text{ mod } (\infty::\text{enat}) = n$
 ⟨proof⟩

lemma *idiv-self*: $\llbracket 0 < (n::\text{enat}); n \neq \infty \rrbracket \implies n \text{ div } n = 1$
 ⟨proof⟩

lemma *imod-self*: $n \neq \infty \implies (n::\text{enat}) \text{ mod } n = 0$

<proof>

lemma *idiv-iless*: $m < (n::\text{enat}) \implies m \text{ div } n = 0$

<proof>

lemma *imod-iless*: $m < (n::\text{enat}) \implies m \text{ mod } n = m$

<proof>

lemma *imod-iless-divisor*: $\llbracket 0 < (n::\text{enat}); m \neq \infty \rrbracket \implies m \text{ mod } n < n$

<proof>

lemma *imod-ile-dividend*: $(m::\text{enat}) \text{ mod } n \leq m$

<proof>

lemma *idiv-ile-dividend*: $(m::\text{enat}) \text{ div } n \leq m$

<proof>

lemma *idiv-imult2-eq*: $(a::\text{enat}) \text{ div } (b * c) = a \text{ div } b \text{ div } c$

<proof>

lemma *imult-ile-mono*: $\llbracket (i::\text{enat}) \leq j; k \leq l \rrbracket \implies i * k \leq j * l$

<proof>

lemma *imult-ile-mono1*: $(i::\text{enat}) \leq j \implies i * k \leq j * k$

<proof>

lemma *imult-ile-mono2*: $(i::\text{enat}) \leq j \implies k * i \leq k * j$

<proof>

lemma *imult-iless-mono1*: $\llbracket (i::\text{enat}) < j; 0 < k; k \neq \infty \rrbracket \implies i * k \leq j * k$

<proof>

lemma *imult-iless-mono2*: $\llbracket (i::\text{enat}) < j; 0 < k; k \neq \infty \rrbracket \implies k * i \leq k * j$

<proof>

lemma *imod-1*: $(\text{enat } m) \text{ mod } \text{eSuc } 0 = 0$

<proof>

lemmas *imod-1-enat*[*simp*, *code*] = *imod-1*[*unfolded zero-enat-def*]

lemma *imod-iadd-self2*: $(m + \text{enat } n) \text{ mod } (\text{enat } n) = m \text{ mod } (\text{enat } n)$

<proof>

lemma *imod-iadd-self1*: $(\text{enat } n + m) \text{ mod } (\text{enat } n) = m \text{ mod } (\text{enat } n)$

<proof>

lemma *idiv-imod-equality*: $(m::\text{enat}) \text{ div } n * n + m \text{ mod } n + k = m + k$

<proof>

lemma *imod-idiv-equality*: $(m::\text{enat}) \text{ div } n * n + m \text{ mod } n = m$

<proof>

lemma *idiv-ile-mono*: $m \leq (n::\text{enat}) \implies m \text{ div } k \leq n \text{ div } k$

<proof>

lemma *idiv-ile-mono2*: $\llbracket 0 < m; m \leq (n::\text{enat}) \rrbracket \implies k \text{ div } n \leq k \text{ div } m$

<proof>

end

4 Results for natural arithmetics

theory *Util-Nat*

imports *Main*

begin

4.1 Some convenience arithmetic lemmata

lemma *add-1-Suc-conv*: $m + 1 = \text{Suc } m$ *<proof>*

lemma *sub-Suc0-sub-Suc-conv*: $b - a - \text{Suc } 0 = b - \text{Suc } a$ *<proof>*

lemma *Suc-diff-Suc*: $m < n \implies \text{Suc } (n - \text{Suc } m) = n - m$
<proof>

lemma *nat-grSuc0-conv*: $(\text{Suc } 0 < n) = (n \neq 0 \wedge n \neq \text{Suc } 0)$
<proof>

lemma *nat-geSucSuc0-conv*: $(\text{Suc } (\text{Suc } 0) \leq n) = (n \neq 0 \wedge n \neq \text{Suc } 0)$
<proof>

lemma *nat-lessSucSuc0-conv*: $(n < \text{Suc } (\text{Suc } 0)) = (n = 0 \vee n = \text{Suc } 0)$
<proof>

lemma *nat-leSuc0-conv*: $(n \leq \text{Suc } 0) = (n = 0 \vee n = \text{Suc } 0)$
<proof>

lemma *mult-pred*: $(m - \text{Suc } 0) * n = m * n - n$
<proof>

lemma *mult-pred-right*: $m * (n - \text{Suc } 0) = m * n - m$
<proof>

lemma *gr-implies-gr0*: $m < (n::\text{nat}) \implies 0 < n$ *<proof>*

corollary *mult-cancel1-gr0*:

$(0::\text{nat}) < k \implies (k * m = k * n) = (m = n)$ *<proof>*

corollary *mult-cancel2-gr0*:

$(0::\text{nat}) < k \implies (m * k = n * k) = (m = n)$ *<proof>*

corollary *mult-le-cancel1-gr0*:

$(0::\text{nat}) < k \implies (k * m \leq k * n) = (m \leq n)$ *<proof>*

corollary *mult-le-cancel2-gr0*:

$(0::\text{nat}) < k \implies (m * k \leq n * k) = (m \leq n)$ *<proof>*

lemma *gr0-imp-self-le-mult1*: $0 < (k::\text{nat}) \implies m \leq m * k$

<proof>

lemma *gr0-imp-self-le-mult2*: $0 < (k::nat) \implies m \leq k * m$
<proof>

lemma *less-imp-Suc-mult-le*: $m < n \implies Suc\ m * k \leq n * k$
<proof>

lemma *less-imp-Suc-mult-pred-less*: $\llbracket m < n; 0 < k \rrbracket \implies Suc\ m * k - Suc\ 0 < n * k$
<proof>

lemma *ord-zero-less-diff*: $(0 < (b::'a::ordered-ab-group-add) - a) = (a < b)$
<proof>

lemma *ord-zero-le-diff*: $(0 \leq (b::'a::ordered-ab-group-add) - a) = (a \leq b)$
<proof>

diff-diff-right in rule format

lemmas *diff-diff-right* = *Nat.diff-diff-right*[*rule-format*]

lemma *less-add1*: $(0::nat) < j \implies i < i + j$ *<proof>*

lemma *less-add2*: $(0::nat) < j \implies i < j + i$ *<proof>*

lemma *add-lessD2*: $i + j < (k::nat) \implies j < k$ *<proof>*

lemma *add-le-mono2*: $i \leq (j::nat) \implies k + i \leq k + j$ *<proof>*

lemma *add-less-mono2*: $i < (j::nat) \implies k + i < k + j$ *<proof>*

lemma *diff-less-self*: $\llbracket (0::nat) < i; 0 < j \rrbracket \implies i - j < i$ *<proof>*

lemma

ge-less-req-conv: $((a::'a::linorder) \leq n) = (\forall x. x < a \longrightarrow n \neq x)$ **and**

le-greater-req-conv: $(n \leq (a::'a::linorder)) = (\forall x. a < x \longrightarrow n \neq x)$

<proof>

lemma

greater-le-req-conv: $((a::'a::linorder) < n) = (\forall x. x \leq a \longrightarrow n \neq x)$ **and**

less-ge-req-conv: $(n < (a::'a::linorder)) = (\forall x. a \leq x \longrightarrow n \neq x)$

<proof>

Lemmas for @termabs function

lemma *leq-pos-imp-abs-leq*: $\llbracket 0 \leq (a::'a::ordered-ab-group-add-abs); a \leq b \rrbracket \implies |a| \leq |b|$

<proof>

lemma *leq-neg-imp-abs-geq*: $\llbracket (a::'a::ordered-ab-group-add-abs) \leq 0; b \leq a \rrbracket \implies |a| \leq |b|$

<proof>

lemma *abs-range*: $\llbracket 0 \leq (a::'a::\{\text{ordered-ab-group-add-abs,abs-if}\}); -a \leq x; x \leq a \rrbracket \implies |x| \leq a$

<proof>

Lemmas for @term_{sgn} function

lemma *sgn-abs*: $(x::'a::\text{linordered-idom}) \neq 0 \implies |\text{sgn } x| = 1$

<proof>

lemma *sgn-mult-abs*: $|x| * |\text{sgn } (a::'a::\text{linordered-idom})| = |x * \text{sgn } a|$

<proof>

lemma *abs-imp-sgn-abs*: $|a| = |b| \implies |\text{sgn } (a::'a::\text{linordered-idom})| = |\text{sgn } b|$

<proof>

lemma *sgn-mono*: $a \leq b \implies \text{sgn } (a::'a::\{\text{linordered-idom,linordered-semidom}\}) \leq \text{sgn } b$

<proof>

4.2 Additional facts about inequalities

lemma *add-diff-le*: $k \leq n \implies m + k - n \leq (m::\text{nat})$

<proof>

lemma *less-add-diff*: $k < (n::\text{nat}) \implies m < n + m - k$

<proof>

lemma *add-diff-less*: $\llbracket k < n; 0 < m \rrbracket \implies m + k - n < (m::\text{nat})$

<proof>

lemma *add-le-imp-le-diff1*: $i + k \leq j \implies i \leq j - (k::\text{nat})$

<proof>

lemma *add-le-imp-le-diff2*: $k + i \leq j \implies i \leq j - (k::\text{nat})$ *<proof>*

lemma *diff-less-imp-less-add*: $j - (k::\text{nat}) < i \implies j < i + k$ *<proof>*

lemma *diff-less-conv*: $0 < i \implies (j - (k::\text{nat}) < i) = (j < i + k)$

<proof>

lemma *le-diff-swap*: $\llbracket i \leq (k::\text{nat}); j \leq k \rrbracket \implies (k - j \leq i) = (k - i \leq j)$

<proof>

lemma *diff-less-imp-swap*: $\llbracket 0 < (i::\text{nat}); k - i < j \rrbracket \implies (k - j < i)$ *<proof>*

lemma *diff-less-swap*: $\llbracket 0 < (i::\text{nat}); 0 < j \rrbracket \implies (k - j < i) = (k - i < j)$

<proof>

lemma *less-diff-imp-less*: $(i::\text{nat}) < j - m \implies i < j$ *<proof>*

lemma *le-diff-imp-le*: $(i::\text{nat}) \leq j - m \implies i \leq j$ *<proof>*

lemma *less-diff-le-imp-less*: $\llbracket (i::\text{nat}) < j - m; n \leq m \rrbracket \implies i < j - n$ *<proof>*

lemma *le-diff-le-imp-le*: $\llbracket (i::\text{nat}) \leq j - m; n \leq m \rrbracket \implies i \leq j - n$ *<proof>*

lemma *le-imp-diff-le*: $(j::\text{nat}) \leq k \implies j - n \leq k$ *<proof>*

4.3 Inequalities for Suc and pred

corollary *less-eq-le-pred*: $0 < (n::\text{nat}) \implies (m < n) = (m \leq n - \text{Suc } 0)$
<proof>

corollary *less-imp-le-pred*: $m < n \implies m \leq n - \text{Suc } 0$ *<proof>*

corollary *le-pred-imp-less*: $\llbracket 0 < n; m \leq n - \text{Suc } 0 \rrbracket \implies m < n$ *<proof>*

corollary *pred-less-eq-le*: $0 < m \implies (m - \text{Suc } 0 < n) = (m \leq n)$
<proof>

corollary *pred-less-imp-le*: $m - \text{Suc } 0 < n \implies m \leq n$ *<proof>*

corollary *le-imp-pred-less*: $\llbracket 0 < m; m \leq n \rrbracket \implies m - \text{Suc } 0 < n$ *<proof>*

lemma *diff-add-inverse-Suc*: $n < m \implies n + (m - \text{Suc } n) = m - \text{Suc } 0$ *<proof>*

lemma *pred-mono*: $\llbracket m < n; 0 < m \rrbracket \implies m - \text{Suc } 0 < n - \text{Suc } 0$ *<proof>*

corollary *pred-Suc-mono*: $\llbracket m < \text{Suc } n; 0 < m \rrbracket \implies m - \text{Suc } 0 < n$ *<proof>*

lemma *Suc-less-pred-conv*: $(\text{Suc } m < n) = (m < n - \text{Suc } 0)$ *<proof>*

lemma *Suc-le-pred-conv*: $0 < n \implies (\text{Suc } m \leq n) = (m \leq n - \text{Suc } 0)$ *<proof>*

lemma *Suc-le-imp-le-pred*: $\text{Suc } m \leq n \implies m \leq n - \text{Suc } 0$ *<proof>*

4.4 Additional facts about cancellation in (in-)equalities

lemma *diff-cancel-imp-eq*: $\llbracket 0 < (n::\text{nat}); n + i - j = n \rrbracket \implies i = j$ *<proof>*

lemma *nat-diff-left-cancel-less*: $k - m < k - (n::\text{nat}) \implies n < m$ *<proof>*

lemma *nat-diff-right-cancel-less*: $n - k < (m::\text{nat}) - k \implies n < m$ *<proof>*

lemma *nat-diff-left-cancel-le1*: $\llbracket k - m \leq k - (n::\text{nat}); m < k \rrbracket \implies n \leq m$
<proof>

lemma *nat-diff-left-cancel-le2*: $\llbracket k - m \leq k - (n::\text{nat}); n \leq k \rrbracket \implies n \leq m$ *<proof>*

lemma *nat-diff-right-cancel-le1*: $\llbracket m - k \leq n - (k::\text{nat}); k < m \rrbracket \implies m \leq n$
<proof>

lemma *nat-diff-right-cancel-le2*: $\llbracket m - k \leq n - (k::\text{nat}); k \leq n \rrbracket \implies m \leq n$
<proof>

lemma *nat-diff-left-cancel-eq1*: $\llbracket k - m = k - (n::\text{nat}); m < k \rrbracket \implies m = n$
<proof>

lemma *nat-diff-left-cancel-eq2*: $\llbracket k - m = k - (n::\text{nat}); n < k \rrbracket \implies m = n$ *<proof>*

lemma *nat-diff-right-cancel-eq1*: $\llbracket m - k = n - (k::\text{nat}); k < m \rrbracket \implies m = n$
<proof>

lemma *nat-diff-right-cancel-eq2*: $\llbracket m - k = n - (k::nat); k < n \rrbracket \implies m = n$
<proof>

lemma *eq-diff-left-iff*: $\llbracket (m::nat) \leq k; n \leq k \rrbracket \implies (k - m = k - n) = (m = n)$
<proof>

lemma *eq-imp-diff-eq*: $m = (n::nat) \implies m - k = n - k$ *<proof>*

List of definitions and lemmas

thm

Nat.add-Suc-right
add-1-Suc-conv
sub-Suc0-sub-Suc-conv

thm

Nat.mult-cancel1
Nat.mult-cancel2
mult-cancel1-gr0
mult-cancel2-gr0

thm

Nat.add-lessD1
add-lessD2

thm

Nat.zero-less-diff
ord-zero-less-diff
ord-zero-le-diff

thm

le-add-diff
add-diff-le
less-add-diff
add-diff-less

thm

Nat.le-diff-conv le-diff-conv2
Nat.less-diff-conv
diff-less-imp-less-add
diff-less-conv

thm

le-diff-swap
diff-less-imp-swap
diff-less-swap

thm

less-diff-imp-less
le-diff-imp-le

thm

less-diff-le-imp-less
le-diff-le-imp-le

thm

Nat.less-imp-diff-less
le-imp-diff-le

thm

Nat.less-Suc-eq-le
less-eq-le-pred
less-imp-le-pred
le-pred-imp-less

thm

Nat.Suc-le-eq
pred-less-eq-le
pred-less-imp-le
le-imp-pred-less

thm

diff-cancel-imp-eq

thm

diff-add-inverse-Suc

thm

Nat.nat-add-left-cancel-less
Nat.nat-add-left-cancel-le
add-right-cancel
add-left-cancel
Nat.eq-diff-iff
Nat.less-diff-iff
Nat.le-diff-iff

thm

nat-diff-left-cancel-less
nat-diff-right-cancel-less

thm

nat-diff-left-cancel-le1
nat-diff-left-cancel-le2
nat-diff-right-cancel-le1
nat-diff-right-cancel-le2

thm

nat-diff-left-cancel-eq1
nat-diff-left-cancel-eq2
nat-diff-right-cancel-eq1
nat-diff-right-cancel-eq2

thm

Nat.eq-diff-iff

eq-diff-left-iff

thm

add-right-cancel add-left-cancel

Nat.diff-le-mono

eq-imp-diff-eq

end

5 Results for division and modulo operators on integers

theory *Util-Div*

imports *Util-Nat*

begin

5.1 Additional (in-)equalities with *div* and *mod*

corollary *Suc-mod-le-divisor*: $0 < m \implies \text{Suc } (n \text{ mod } m) \leq m$

<proof>

lemma *mod-less-dividend*: $\llbracket 0 < m; m \leq n \rrbracket \implies n \text{ mod } m < (n::\text{nat})$

<proof>

lemmas *mod-le-dividend = mod-less-eq-dividend*

lemma *diff-mod-le*: $(t - r) \text{ mod } m \leq (t::\text{nat})$

<proof>

lemmas *div-mult-cancel = minus-mod-eq-div-mult* [*symmetric*]

lemma *mod-0-div-mult-cancel*: $(n \text{ mod } (m::\text{nat}) = 0) = (n \text{ div } m * m = n)$

<proof>

lemma *div-mult-le*: $(n::\text{nat}) \text{ div } m * m \leq n$

<proof>

lemma *less-div-Suc-mult*: $0 < (m::\text{nat}) \implies n < \text{Suc } (n \text{ div } m) * m$

<proof>

lemma *nat-ge2-conv*: $((2::\text{nat}) \leq n) = (n \neq 0 \wedge n \neq 1)$

<proof>

lemma *Suc0-mod*: $m \neq \text{Suc } 0 \implies \text{Suc } 0 \text{ mod } m = \text{Suc } 0$

<proof>

corollary *Suc0-mod-subst*:

$$\llbracket m \neq \text{Suc } 0; P (\text{Suc } 0) \rrbracket \implies P (\text{Suc } 0 \text{ mod } m)$$

<proof>

corollary *Suc0-mod-cong*:

$$m \neq \text{Suc } 0 \implies f (\text{Suc } 0 \text{ mod } m) = f (\text{Suc } 0)$$

<proof>

5.2 Additional results for addition and subtraction with *mod*

lemma *mod-Suc-conv*:

$$((\text{Suc } a) \text{ mod } m = (\text{Suc } b) \text{ mod } m) = (a \text{ mod } m = b \text{ mod } m)$$

<proof>

lemma *mod-Suc'*:

$$0 < n \implies \text{Suc } m \text{ mod } n = (\text{if } m \text{ mod } n < n - \text{Suc } 0 \text{ then } \text{Suc } (m \text{ mod } n) \text{ else } 0)$$

<proof>

lemma *mod-add*:

$$\begin{aligned} ((a + k) \text{ mod } m = (b + k) \text{ mod } m) = \\ ((a::\text{nat}) \text{ mod } m = b \text{ mod } m) \end{aligned}$$

<proof>

corollary *mod-sub-add*:

$$\begin{aligned} k \leq (a::\text{nat}) \implies \\ ((a - k) \text{ mod } m = b \text{ mod } m) = (a \text{ mod } m = (b + k) \text{ mod } m) \end{aligned}$$

<proof>

lemma *mod-sub-eq-mod-0-conv*:

$$\begin{aligned} a + b \leq (n::\text{nat}) \implies \\ ((n - a) \text{ mod } m = b \text{ mod } m) = ((n - (a + b)) \text{ mod } m = 0) \end{aligned}$$

<proof>

lemma *mod-sub-eq-mod-swap*:

$$\begin{aligned} \llbracket a \leq (n::\text{nat}); b \leq n \rrbracket \implies \\ ((n - a) \text{ mod } m = b \text{ mod } m) = ((n - b) \text{ mod } m = a \text{ mod } m) \end{aligned}$$

<proof>

lemma *le-mod-greater-imp-div-less*:

$$\llbracket a \leq (b::\text{nat}); a \text{ mod } m > b \text{ mod } m \rrbracket \implies a \text{ div } m < b \text{ div } m$$

<proof>

lemma *less-mod-ge-imp-div-less*: $\llbracket a < (b::\text{nat}); a \text{ mod } m \geq b \text{ mod } m \rrbracket \implies a \text{ div } m < b \text{ div } m$

<proof>

corollary *less-mod-0-imp-div-less*: $\llbracket a < (b::\text{nat}); b \text{ mod } m = 0 \rrbracket \implies a \text{ div } m < b \text{ div } m$

<proof>

lemma *mod-diff-right-eq*:

$$(a::nat) \leq b \implies (b - a) \text{ mod } m = (b - a \text{ mod } m) \text{ mod } m$$

<proof>

corollary *mod-eq-imp-diff-mod-eq*:

$$\llbracket x \text{ mod } m = y \text{ mod } m; x \leq (t::nat); y \leq t \rrbracket \implies (t - x) \text{ mod } m = (t - y) \text{ mod } m$$

<proof>

lemma *mod-eq-imp-diff-mod-eq2*:

$$\llbracket x \text{ mod } m = y \text{ mod } m; (t::nat) \leq x; t \leq y \rrbracket \implies (x - t) \text{ mod } m = (y - t) \text{ mod } m$$

<proof>

lemma *divisor-add-diff-mod-if*:

$$(m + b \text{ mod } m - a \text{ mod } m) \text{ mod } (m::nat) = (\text{if } a \text{ mod } m \leq b \text{ mod } m \text{ then } (b \text{ mod } m - a \text{ mod } m) \text{ else } (m + b \text{ mod } m - a \text{ mod } m))$$

<proof>

corollary *divisor-add-diff-mod-eq1*:

$$a \text{ mod } m \leq b \text{ mod } m \implies (m + b \text{ mod } m - a \text{ mod } m) \text{ mod } (m::nat) = b \text{ mod } m - a \text{ mod } m$$

<proof>

corollary *divisor-add-diff-mod-eq2*:

$$b \text{ mod } m < a \text{ mod } m \implies (m + b \text{ mod } m - a \text{ mod } m) \text{ mod } (m::nat) = m + b \text{ mod } m - a \text{ mod } m$$

<proof>

lemma *mod-add-mod-if*:

$$(a \text{ mod } m + b \text{ mod } m) \text{ mod } (m::nat) = (\text{if } a \text{ mod } m + b \text{ mod } m < m \text{ then } a \text{ mod } m + b \text{ mod } m \text{ else } a \text{ mod } m + b \text{ mod } m - m)$$

<proof>

corollary *mod-add-mod-eq1*:

$$a \text{ mod } m + b \text{ mod } m < m \implies (a \text{ mod } m + b \text{ mod } m) \text{ mod } (m::nat) = a \text{ mod } m + b \text{ mod } m$$

<proof>

corollary *mod-add-mod-eq2*:

$$m \leq a \text{ mod } m + b \text{ mod } m \implies (a \text{ mod } m + b \text{ mod } m) \text{ mod } (m::nat) = a \text{ mod } m + b \text{ mod } m - m$$

<proof>

lemma *mod-add1-eq-if*:

$$(a + b) \text{ mod } (m::nat) = (\text{if } (a \text{ mod } m + b \text{ mod } m < m) \text{ then } a \text{ mod } m + b \text{ mod } m \text{ else } a \text{ mod } m + b \text{ mod } m - m)$$

<proof>

lemma *mod-add-eq-mod-conv*: $0 < (m::nat) \implies$

$((x + a) \bmod m = b \bmod m) =$
 $(x \bmod m = (m + b \bmod m - a \bmod m) \bmod m)$
 <proof>

lemma *mod-diff1-eq*:

$(a::nat) \leq b \implies (b - a) \bmod m = (m + b \bmod m - a \bmod m) \bmod m$
 <proof>

corollary *mod-diff1-eq-if*:

$(a::nat) \leq b \implies (b - a) \bmod m =$
 $\text{if } a \bmod m \leq b \bmod m \text{ then } b \bmod m - a \bmod m$
 $\text{else } m + b \bmod m - a \bmod m)$
 <proof>

corollary *mod-diff1-eq1*:

$\llbracket (a::nat) \leq b; a \bmod m \leq b \bmod m \rrbracket$
 $\implies (b - a) \bmod m = b \bmod m - a \bmod m$
 <proof>

corollary *mod-diff1-eq2*:

$\llbracket (a::nat) \leq b; b \bmod m < a \bmod m \rrbracket$
 $\implies (b - a) \bmod m = m + b \bmod m - a \bmod m$
 <proof>

5.2.1 Divisor subtraction with *div* and *mod*

lemma *mod-diff-self1*:

$0 < (n::nat) \implies (m - n) \bmod m = m - n$
 <proof>

lemma *mod-diff-self2*:

$m \leq (n::nat) \implies (n - m) \bmod m = n \bmod m$
 <proof>

lemma *mod-diff-mult-self1*:

$k * m \leq (n::nat) \implies (n - k * m) \bmod m = n \bmod m$
 <proof>

lemma *mod-diff-mult-self2*:

$m * k \leq (n::nat) \implies (n - m * k) \bmod m = n \bmod m$
 <proof>

lemma *div-diff-self1*: $0 < (n::nat) \implies (m - n) \text{ div } m = 0$

<proof>

lemma *div-diff-self2*: $(n - m) \text{ div } m = n \text{ div } m - \text{Suc } 0$

<proof>

lemma *div-diff-mult-self1*:

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

lemma *div-diff-mult-self2*:

$(n - m * k) \text{ div } m = n \text{ div } m - (k::nat)$

<proof>

5.2.2 Modulo equality and modulo of difference

lemma *mod-eq-imp-diff-mod-0*:

$(a::nat) \text{ mod } m = b \text{ mod } m \implies (b - a) \text{ mod } m = 0$
(is $?P \implies ?Q$)

<proof>

corollary *mod-eq-imp-diff-dvd*:

$(a::nat) \text{ mod } m = b \text{ mod } m \implies m \text{ dvd } b - a$

<proof>

lemma *mod-neq-imp-diff-mod-neq0*:

$\llbracket (a::nat) \text{ mod } m \neq b \text{ mod } m; a \leq b \rrbracket \implies 0 < (b - a) \text{ mod } m$

<proof>

corollary *mod-neq-imp-diff-not-dvd*:

$\llbracket (a::nat) \text{ mod } m \neq b \text{ mod } m; a \leq b \rrbracket \implies \neg m \text{ dvd } b - a$

<proof>

lemma *diff-mod-0-imp-mod-eq*:

$\llbracket (b - a) \text{ mod } m = 0; a \leq b \rrbracket \implies (a::nat) \text{ mod } m = b \text{ mod } m$

<proof>

corollary *diff-dvd-imp-mod-eq*:

$\llbracket m \text{ dvd } b - a; a \leq b \rrbracket \implies (a::nat) \text{ mod } m = b \text{ mod } m$

<proof>

lemma *mod-eq-diff-mod-0-conv*:

$a \leq (b::nat) \implies (a \text{ mod } m = b \text{ mod } m) = ((b - a) \text{ mod } m = 0)$

<proof>

corollary *mod-eq-diff-dvd-conv*:

$a \leq (b::nat) \implies (a \text{ mod } m = b \text{ mod } m) = (m \text{ dvd } b - a)$

<proof>

5.3 Some additional lemmata about integer *div* and *mod*

lemma *zmod-eq-imp-diff-mod-0*:

$a \text{ mod } m = b \text{ mod } m \implies (b - a) \text{ mod } m = 0$ **for** $a \ b \ m :: \text{int}$

<proof>

lemmas *int-mod-distrib = zmod-int*

lemma *zdiff-mod-0-imp-mod-eq--pos*:

$\llbracket (b - a) \text{ mod } m = 0; 0 < (m::int) \rrbracket \implies a \text{ mod } m = b \text{ mod } m$

(is $\llbracket ?P; ?Pm \rrbracket \implies ?Q$)

<proof>

lemma *zmod-zminus-eq-conv-pos*:

$0 < (m::int) \implies (a \text{ mod } -m = b \text{ mod } -m) = (a \text{ mod } m = b \text{ mod } m)$
 ⟨proof⟩

lemma *zmod-zminus-eq-conv*:

$((a::int) \text{ mod } -m = b \text{ mod } -m) = (a \text{ mod } m = b \text{ mod } m)$
 ⟨proof⟩

lemma *zdiff-mod-0-imp-mod-eq*:

$(b - a) \text{ mod } m = 0 \implies (a::int) \text{ mod } m = b \text{ mod } m$
 ⟨proof⟩

lemma *zmod-eq-diff-mod-0-conv*:

$((a::int) \text{ mod } m = b \text{ mod } m) = ((b - a) \text{ mod } m = 0)$
 ⟨proof⟩

lemma $\neg(\exists (a::int) b m. (b - a) \text{ mod } m = 0 \wedge a \text{ mod } m \neq b \text{ mod } m)$
 ⟨proof⟩

lemma $\exists (a::nat) b m. (b - a) \text{ mod } m = 0 \wedge a \text{ mod } m \neq b \text{ mod } m$
 ⟨proof⟩

lemma *zmult-div-leq-mono*:

$\llbracket (0::int) \leq x; a \leq b; 0 < d \rrbracket \implies x * a \text{ div } d \leq x * b \text{ div } d$
 ⟨proof⟩

lemma *zmult-div-leq-mono-neg*:

$\llbracket x \leq (0::int); a \leq b; 0 < d \rrbracket \implies x * b \text{ div } d \leq x * a \text{ div } d$
 ⟨proof⟩

lemma *zmult-div-pos-le*:

$\llbracket (0::int) \leq a; 0 \leq b; b \leq c \rrbracket \implies a * b \text{ div } c \leq a$
 ⟨proof⟩

lemma *zmult-div-neg-le*:

$\llbracket a \leq (0::int); 0 < c; c \leq b \rrbracket \implies a * b \text{ div } c \leq a$
 ⟨proof⟩

lemma *zmult-div-ge-0*: $\llbracket (0::int) \leq x; 0 \leq a; 0 < c \rrbracket \implies 0 \leq a * x \text{ div } c$
 ⟨proof⟩

corollary *zmult-div-plus-ge-0*:

$\llbracket (0::int) \leq x; 0 \leq a; 0 \leq b; 0 < c \rrbracket \implies 0 \leq a * x \text{ div } c + b$
 ⟨proof⟩

lemma *zmult-div-abs-ge*:

$\llbracket (0::int) \leq b; b \leq b'; 0 \leq a; 0 < c \rrbracket \implies$
 $|a * b \text{ div } c| \leq |a * b' \text{ div } c|$
 ⟨proof⟩

lemma *zmult-div-plus-abs-ge*:

$$\begin{aligned} & \llbracket (0::\text{int}) \leq b; b \leq b'; 0 \leq a; 0 < c \rrbracket \implies \\ & |a * b \text{ div } c + a| \leq |a * b' \text{ div } c + a| \\ & \langle \text{proof} \rangle \end{aligned}$$

5.4 Some further (in-)equality results for *div* and *mod*

lemma *less-mod-eq-imp-add-divisor-le*:

$$\llbracket (x::\text{nat}) < y; x \text{ mod } m = y \text{ mod } m \rrbracket \implies x + m \leq y$$

$\langle \text{proof} \rangle$

lemma *less-div-imp-mult-add-divisor-le*:

$$(x::\text{nat}) < n \text{ div } m \implies x * m + m \leq n$$

$\langle \text{proof} \rangle$

lemma *mod-add-eq-imp-mod-0*:

$$((n + k) \text{ mod } (m::\text{nat}) = n \text{ mod } m) = (k \text{ mod } m = 0)$$

$\langle \text{proof} \rangle$

lemma *between-imp-mod-between*:

$$\begin{aligned} & \llbracket b < (m::\text{nat}); m * k + a \leq n; n \leq m * k + b \rrbracket \implies \\ & a \leq n \text{ mod } m \wedge n \text{ mod } m \leq b \\ & \langle \text{proof} \rangle \end{aligned}$$

corollary *between-imp-mod-le*:

$$\llbracket b < (m::\text{nat}); m * k \leq n; n \leq m * k + b \rrbracket \implies n \text{ mod } m \leq b$$

$\langle \text{proof} \rangle$

corollary *between-imp-mod-gr0*:

$$\llbracket (m::\text{nat}) * k < n; n < m * k + m \rrbracket \implies 0 < n \text{ mod } m$$

$\langle \text{proof} \rangle$

corollary *le-less-div-conv*:

$$0 < m \implies (k * m \leq n \wedge n < \text{Suc } k * m) = (n \text{ div } m = k)$$

$\langle \text{proof} \rangle$

lemma *le-less-imp-div*:

$$\llbracket k * m \leq n; n < \text{Suc } k * m \rrbracket \implies n \text{ div } m = k$$

$\langle \text{proof} \rangle$

lemma *div-imp-le-less*:

$$\llbracket n \text{ div } m = k; 0 < m \rrbracket \implies k * m \leq n \wedge n < \text{Suc } k * m$$

$\langle \text{proof} \rangle$

lemma *div-le-mod-le-imp-le*:

$$\llbracket (a::\text{nat}) \text{ div } m \leq b \text{ div } m; a \text{ mod } m \leq b \text{ mod } m \rrbracket \implies a \leq b$$

$\langle \text{proof} \rangle$

lemma *le-mod-add-eq-imp-add-mod-le*:

$\llbracket a \leq b; (a + k) \bmod m = (b::\text{nat}) \bmod m \rrbracket \implies a + k \bmod m \leq b$
 $\langle \text{proof} \rangle$

corollary *mult-divisor-le-mod-ge-imp-ge*:

$\llbracket (m::\text{nat}) * k \leq n; r \leq n \bmod m \rrbracket \implies m * k + r \leq n$
 $\langle \text{proof} \rangle$

5.5 Additional multiplication results for *mod* and *div*

lemma *mod-0-imp-mod-mult-right-0*:

$n \bmod m = (0::\text{nat}) \implies n * k \bmod m = 0$
 $\langle \text{proof} \rangle$

lemma *mod-0-imp-mod-mult-left-0*:

$n \bmod m = (0::\text{nat}) \implies k * n \bmod m = 0$
 $\langle \text{proof} \rangle$

lemma *mod-0-imp-div-mult-left-eq*:

$n \bmod m = (0::\text{nat}) \implies k * n \text{ div } m = k * (n \text{ div } m)$
 $\langle \text{proof} \rangle$

lemma *mod-0-imp-div-mult-right-eq*:

$n \bmod m = (0::\text{nat}) \implies n * k \text{ div } m = k * (n \text{ div } m)$
 $\langle \text{proof} \rangle$

lemma *mod-0-imp-mod-factor-0-left*:

$n \bmod (m * m') = (0::\text{nat}) \implies n \bmod m = 0$
 $\langle \text{proof} \rangle$

lemma *mod-0-imp-mod-factor-0-right*:

$n \bmod (m * m') = (0::\text{nat}) \implies n \bmod m' = 0$
 $\langle \text{proof} \rangle$

5.6 Some factor distribution facts for *mod*

lemma *mod-eq-mult-distrib*:

$(a::\text{nat}) \bmod m = b \bmod m \implies$
 $a * k \bmod (m * k) = b * k \bmod (m * k)$
 $\langle \text{proof} \rangle$

lemma *mod-mult-eq-imp-mod-eq*:

$(a::\text{nat}) \bmod (m * k) = b \bmod (m * k) \implies a \bmod m = b \bmod m$
 $\langle \text{proof} \rangle$

corollary *mod-eq-mod-0-imp-mod-eq*:

$\llbracket (a::\text{nat}) \bmod m' = b \bmod m'; m' \bmod m = 0 \rrbracket$
 $\implies a \bmod m = b \bmod m$
 $\langle \text{proof} \rangle$

lemma *mod-factor-imp-mod-0*:

$\llbracket (x::\text{nat}) \bmod (m * k) = y * k \bmod (m * k) \rrbracket \implies x \bmod k = 0$
 $(\text{is } \llbracket ?P1 \rrbracket \implies ?Q)$

<proof>

corollary *mod-factor-div*:

$\llbracket (x::\text{nat}) \text{ mod } (m * k) = y * k \text{ mod } (m * k) \rrbracket \implies x \text{ div } k * k = x$
<proof>

lemma *mod-factor-div-mod*:

$\llbracket (x::\text{nat}) \text{ mod } (m * k) = y * k \text{ mod } (m * k); 0 < k \rrbracket$
 $\implies x \text{ div } k \text{ mod } m = y \text{ mod } m$
 (is $\llbracket ?P1; ?P2 \rrbracket \implies ?L = ?R$)
<proof>

5.7 More results about quotient *div* with addition and subtraction

lemma *div-add1-eq-if*: $0 < m \implies$

$(a + b) \text{ div } (m::\text{nat}) = a \text{ div } m + b \text{ div } m +$
if $a \text{ mod } m + b \text{ mod } m < m$ *then* 0 *else* $\text{Suc } 0$

<proof>

corollary *div-add1-eq1*:

$a \text{ mod } m + b \text{ mod } m < (m::\text{nat}) \implies$
 $(a + b) \text{ div } (m::\text{nat}) = a \text{ div } m + b \text{ div } m$

<proof>

corollary *div-add1-eq1-mod-0-left*:

$a \text{ mod } m = 0 \implies (a + b) \text{ div } (m::\text{nat}) = a \text{ div } m + b \text{ div } m$

<proof>

corollary *div-add1-eq1-mod-0-right*:

$b \text{ mod } m = 0 \implies (a + b) \text{ div } (m::\text{nat}) = a \text{ div } m + b \text{ div } m$

<proof>

corollary *div-add1-eq2*:

$\llbracket 0 < m; (m::\text{nat}) \leq a \text{ mod } m + b \text{ mod } m \rrbracket \implies$
 $(a + b) \text{ div } (m::\text{nat}) = \text{Suc } (a \text{ div } m + b \text{ div } m)$

<proof>

lemma *div-Suc*:

$0 < n \implies \text{Suc } m \text{ div } n = (\text{if } \text{Suc } (m \text{ mod } n) = n \text{ then } \text{Suc } (m \text{ div } n) \text{ else } m \text{ div } n)$

<proof>

lemma *div-Suc'*:

$0 < n \implies \text{Suc } m \text{ div } n = (\text{if } m \text{ mod } n < n - \text{Suc } 0 \text{ then } m \text{ div } n \text{ else } \text{Suc } (m \text{ div } n))$

<proof>

lemma *div-diff1-eq-if*:

$(b - a) \text{ div } (m::\text{nat}) =$
 $b \text{ div } m - a \text{ div } m - (\text{if } a \text{ mod } m \leq b \text{ mod } m \text{ then } 0 \text{ else } \text{Suc } 0)$

<proof>

corollary *div-diff1-eq*:

$(b - a) \text{ div } (m::\text{nat}) =$

$b \operatorname{div} m - a \operatorname{div} m - (m + a \operatorname{mod} m - \operatorname{Suc} (b \operatorname{mod} m)) \operatorname{div} m$
 ⟨proof⟩

corollary *div-diff1-eq1*:

$a \operatorname{mod} m \leq b \operatorname{mod} m \implies$
 $(b - a) \operatorname{div} (m::\operatorname{nat}) = b \operatorname{div} m - a \operatorname{div} m$
 ⟨proof⟩

corollary *div-diff1-eq1-mod-0*:

$a \operatorname{mod} m = 0 \implies$
 $(b - a) \operatorname{div} (m::\operatorname{nat}) = b \operatorname{div} m - a \operatorname{div} m$
 ⟨proof⟩

corollary *div-diff1-eq2*:

$b \operatorname{mod} m < a \operatorname{mod} m \implies$
 $(b - a) \operatorname{div} (m::\operatorname{nat}) = b \operatorname{div} m - \operatorname{Suc} (a \operatorname{div} m)$
 ⟨proof⟩

5.8 Further results about *div* and *mod*

5.8.1 Some auxiliary facts about *mod*

lemma *diff-less-divisor-imp-sub-mod-eq*:

$\llbracket (x::\operatorname{nat}) \leq y; y - x < m \rrbracket \implies x = y - (y - x) \operatorname{mod} m$
 ⟨proof⟩

lemma *diff-ge-divisor-imp-sub-mod-less*:

$\llbracket (x::\operatorname{nat}) \leq y; m \leq y - x; 0 < m \rrbracket \implies x < y - (y - x) \operatorname{mod} m$
 ⟨proof⟩

lemma *le-imp-sub-mod-le*:

$(x::\operatorname{nat}) \leq y \implies x \leq y - (y - x) \operatorname{mod} m$
 ⟨proof⟩

lemma *mod-less-diff-mod*:

$\llbracket n \operatorname{mod} m < r; r \leq m; r \leq (n::\operatorname{nat}) \rrbracket \implies$
 $(n - r) \operatorname{mod} m = m + n \operatorname{mod} m - r$
 ⟨proof⟩

lemma *mod-0-imp-mod-pred*:

$\llbracket 0 < (n::\operatorname{nat}); n \operatorname{mod} m = 0 \rrbracket \implies$
 $(n - \operatorname{Suc} 0) \operatorname{mod} m = m - \operatorname{Suc} 0$
 ⟨proof⟩

lemma *mod-pred*:

$0 < n \implies$
 $(n - \operatorname{Suc} 0) \operatorname{mod} m = ($
 $\text{if } n \operatorname{mod} m = 0 \text{ then } m - \operatorname{Suc} 0 \text{ else } n \operatorname{mod} m - \operatorname{Suc} 0)$
 ⟨proof⟩

corollary *mod-pred-Suc-mod*:

$0 < n \implies \operatorname{Suc} ((n - \operatorname{Suc} 0) \operatorname{mod} m) \operatorname{mod} m = n \operatorname{mod} m$
 ⟨proof⟩

corollary *diff-mod-pred*:

$a < b \implies$
 $(b - \text{Suc } a) \text{ mod } m =$
 $\text{if } a \text{ mod } m = b \text{ mod } m \text{ then } m - \text{Suc } 0 \text{ else } (b - a) \text{ mod } m - \text{Suc } 0$
 <proof>

corollary *diff-mod-pred-Suc-mod*:

$a < b \implies \text{Suc } ((b - \text{Suc } a) \text{ mod } m) \text{ mod } m = (b - a) \text{ mod } m$
 <proof>

lemma *mod-eq-imp-diff-mod-eq-divisor*:

$\llbracket a < b; 0 < m; a \text{ mod } m = b \text{ mod } m \rrbracket \implies$
 $\text{Suc } ((b - \text{Suc } a) \text{ mod } m) = m$
 <proof>

lemma *sub-diff-mod-eq*:

$r \leq t \implies (t - (t - r) \text{ mod } m) \text{ mod } (m::\text{nat}) = r \text{ mod } m$
 <proof>

lemma *sub-diff-mod-eq'*:

$r \leq t \implies (k * m + t - (t - r) \text{ mod } m) \text{ mod } (m::\text{nat}) = r \text{ mod } m$
 <proof>

lemma *mod-eq-Suc-0-conv*: $\text{Suc } 0 < k \implies ((x + k - \text{Suc } 0) \text{ mod } k = 0) = (x \text{ mod } k = \text{Suc } 0)$
 <proof>

lemma *mod-eq-divisor-minus-Suc-0-conv*: $\text{Suc } 0 < k \implies (x \text{ mod } k = k - \text{Suc } 0) = (\text{Suc } x \text{ mod } k = 0)$
 <proof>

5.8.2 Some auxiliary facts about *div*

lemma *sub-mod-div-eq-div*: $((n::\text{nat}) - n \text{ mod } m) \text{ div } m = n \text{ div } m$
 <proof>

lemma *mod-less-imp-diff-div-conv*:

$\llbracket n \text{ mod } m < r; r \leq m + n \text{ mod } m \rrbracket \implies (n - r) \text{ div } m = n \text{ div } m - \text{Suc } 0$
 <proof>

corollary *mod-0-le-imp-diff-div-conv*:

$\llbracket n \text{ mod } m = 0; 0 < r; r \leq m \rrbracket \implies (n - r) \text{ div } m = n \text{ div } m - \text{Suc } 0$
 <proof>

corollary *mod-0-less-imp-diff-Suc-div-conv*:

$\llbracket n \text{ mod } m = 0; r < m \rrbracket \implies (n - \text{Suc } r) \text{ div } m = n \text{ div } m - \text{Suc } 0$
 <proof>

corollary *mod-0-imp-diff-Suc-div-conv*:

$(n - r) \text{ mod } m = 0 \implies (n - \text{Suc } r) \text{ div } m = (n - r) \text{ div } m - \text{Suc } 0$
 <proof>

corollary *mod-0-imp-sub-1-div-conv*:

$n \bmod m = 0 \implies (n - \text{Suc } 0) \text{ div } m = n \text{ div } m - \text{Suc } 0$
 ⟨proof⟩

corollary *sub-Suc-mod-div-conv*:

$(n - \text{Suc } (n \bmod m)) \text{ div } m = n \text{ div } m - \text{Suc } 0$
 ⟨proof⟩

lemma *div-le-conv*: $0 < m \implies n \text{ div } m \leq k = (n \leq \text{Suc } k * m - \text{Suc } 0)$
 ⟨proof⟩

lemma *le-div-conv*: $0 < (m::\text{nat}) \implies (n \leq k \text{ div } m) = (n * m \leq k)$
 ⟨proof⟩

lemma *less-mult-imp-div-less*: $n < k * m \implies n \text{ div } m < (k::\text{nat})$
 ⟨proof⟩

lemma *div-less-imp-less-mult*: $\llbracket 0 < (m::\text{nat}); n \text{ div } m < k \rrbracket \implies n < k * m$
 ⟨proof⟩

lemma *div-less-conv*: $0 < (m::\text{nat}) \implies (n \text{ div } m < k) = (n < k * m)$
 ⟨proof⟩

lemma *div-eq-0-conv*: $(n \text{ div } (m::\text{nat}) = 0) = (m = 0 \vee n < m)$
 ⟨proof⟩

lemma *div-eq-0-conv'*: $0 < m \implies (n \text{ div } (m::\text{nat}) = 0) = (n < m)$
 ⟨proof⟩

corollary *div-gr-imp-gr-divisor*: $x < n \text{ div } (m::\text{nat}) \implies m \leq n$
 ⟨proof⟩

lemma *mod-0-less-div-conv*:

$n \bmod (m::\text{nat}) = 0 \implies (k * m < n) = (k < n \text{ div } m)$
 ⟨proof⟩

lemma *add-le-divisor-imp-le-Suc-div*:

$\llbracket x \text{ div } m \leq n; y \leq m \rrbracket \implies (x + y) \text{ div } m \leq \text{Suc } n$
 ⟨proof⟩

List of definitions and lemmas

thm

minus-mod-eq-mult-div [symmetric]

mod-0-div-mult-cancel

div-mult-le

less-div-Suc-mult

thm

Suc0-mod

Suc0-mod-subst

Suc0-mod-cong

thm

mod-Suc-conv

thm

mod-add

mod-sub-add

thm

mod-sub-eq-mod-0-conv

mod-sub-eq-mod-swap

thm

le-mod-greater-imp-div-less

thm

mod-diff-right-eq

mod-eq-imp-diff-mod-eq

thm

divisor-add-diff-mod-if

divisor-add-diff-mod-eq1

divisor-add-diff-mod-eq2

thm

mod-add-eq

mod-add1-eq-if

thm

mod-diff1-eq-if

mod-diff1-eq

mod-diff1-eq1

mod-diff1-eq2

thm

nat-mod-distrib

int-mod-distrib

thm

zmod-zminus-eq-conv

thm

mod-eq-imp-diff-mod-0

zmod-eq-imp-diff-mod-0

thm

mod-neq-imp-diff-mod-neq0

diff-mod-0-imp-mod-eq

zdiff-mod-0-imp-mod-eq

thm

zmod-eq-diff-mod-0-conv

mod-eq-diff-mod-0-conv

thm
less-mod-eq-imp-add-divisor-le

thm
mod-add-eq-imp-mod-0

thm
mod-eq-mult-distrib
mod-factor-imp-mod-0
mod-factor-div
mod-factor-div-mod

thm
mod-diff-self1
mod-diff-self2
mod-diff-mult-self1
mod-diff-mult-self2

thm
div-diff-self1
div-diff-self2
div-diff-mult-self1
div-diff-mult-self2

thm
le-less-imp-div
div-imp-le-less

thm
le-less-div-conv

thm
diff-less-divisor-imp-sub-mod-eq
diff-ge-divisor-imp-sub-mod-less
le-imp-sub-mod-le

thm
sub-mod-div-eq-div

thm
mod-less-imp-diff-div-conv
mod-0-le-imp-diff-div-conv
mod-0-less-imp-diff-Suc-div-conv
mod-0-imp-sub-1-div-conv

thm
sub-Suc-mod-div-conv

thm

mod-less-diff-mod
mod-0-imp-mod-pred

thm

mod-pred
mod-pred-Suc-mod

thm

mod-eq-imp-diff-mod-eq-divisor

thm

diff-mod-le
sub-diff-mod-eq
sub-diff-mod-eq'

thm

div-diff1-eq-if
div-diff1-eq
div-diff1-eq1
div-diff1-eq2

thm

div-le-conv

end

6 Sets of natural numbers

theory *SetInterval2*

imports

HOL-Library.Infinite-Set
Util-Set
../CommonArith/Util-MinMax
../CommonArith/Util-NatInf
../CommonArith/Util-Div

begin

6.1 Auxiliary results for monotonic, injective and surjective functions over sets

6.1.1 Monotonicity

definition *mono-on* :: ('a::order ⇒ 'b::order) ⇒ 'a set ⇒ bool

where *mono-on* *f* *A* ≡ $\forall a \in A. \forall b \in A. a \leq b \longrightarrow f\ a \leq f\ b$

definition *strict-mono-on* :: ('a::order ⇒ 'b::order) ⇒ 'a set ⇒ bool

where *strict-mono-on* *f* *A* ≡ $\forall a \in A. \forall b \in A. a < b \longrightarrow f\ a < f\ b$

lemma *mono-on-subset*: $\llbracket \text{mono-on } f \ A ; B \subseteq A \rrbracket \implies \text{mono-on } f \ B$
 ⟨proof⟩

lemma *strict-mono-on-subset*: $\llbracket \text{strict-mono-on } f \ A ; B \subseteq A \rrbracket \implies \text{strict-mono-on } f \ B$
 ⟨proof⟩

lemma *mono-on-UNIV-mono-conv*: $\text{mono-on } f \ \text{UNIV} = \text{mono } f$
 ⟨proof⟩

lemma *strict-mono-on-UNIV-strict-mono-conv*:
 $\text{strict-mono-on } f \ \text{UNIV} = \text{strict-mono } f$
 ⟨proof⟩

lemma *mono-imp-mono-on*: $\text{mono } f \implies \text{mono-on } f \ A$
 ⟨proof⟩

lemma *strict-mono-imp-strict-mono-on*: $\text{strict-mono } f \implies \text{strict-mono-on } f \ A$
 ⟨proof⟩

lemma *strict-mono-on-imp-mono-on*: $\text{strict-mono-on } f \ A \implies \text{mono-on } f \ A$
 ⟨proof⟩

6.1.2 Injectivity

lemma *inj-imp-inj-on*: $\text{inj } f \implies \text{inj-on } f \ A$
 ⟨proof⟩

lemma *strict-mono-on-imp-inj-on*:
 $\text{strict-mono-on } f \ (A::'a::\text{linorder set}) \implies \text{inj-on } f \ A$
 ⟨proof⟩

lemma *strict-mono-imp-inj*: $\text{strict-mono } (f::('a::\text{linorder} \Rightarrow 'b::\text{order})) \implies \text{inj } f$
 ⟨proof⟩

lemma *strict-mono-on-mono-on-conv*:
 $\text{strict-mono-on } f \ (A::'a::\text{linorder set}) = (\text{mono-on } f \ A \wedge \text{inj-on } f \ A)$
 ⟨proof⟩

corollary *strict-mono-mono-conv*:
 $\text{strict-mono } (f::('a::\text{linorder} \Rightarrow 'b::\text{order})) = (\text{mono } f \wedge \text{inj } f)$
 ⟨proof⟩

lemma *inj-on-image-mem-iff*:
 $\llbracket \text{inj-on } f \ A ; B \subseteq A ; a \in A \rrbracket \implies (f \ a \in f \ 'B) = (a \in B)$
 ⟨proof⟩

corollary *inj-on-union-image-Int:*

$inj\text{-on } f (A \cup B) \implies f ' (A \cap B) = f ' A \cap f ' B$
 ⟨proof⟩

6.1.3 Surjectivity

definition *surj-on* :: ('a ⇒ 'b) ⇒ 'a set ⇒ 'b set ⇒ bool
 where $surj\text{-on } f A B \equiv \forall b \in B. \exists a \in A. b = f a$

lemma *surj-on-conv*: $(surj\text{-on } f A B) = (\forall b \in B. \exists a \in A. b = f a)$
 ⟨proof⟩

lemma *surj-on-image-conv*: $(surj\text{-on } f A B) = (B \subseteq f ' A)$
 ⟨proof⟩

lemma *surj-on-id*: $surj\text{-on } id A A$
 ⟨proof⟩

lemma
surj-onI: $\llbracket \forall b \in B. \exists a \in A. b = f a \rrbracket \implies surj\text{-on } f A B$ **and**
surj-onD2: $surj\text{-on } f A B \implies \forall b \in B. \exists a \in A. b = f a$ **and**
surj-onD: $\llbracket surj\text{-on } f A B; b \in B \rrbracket \implies \exists a \in A. b = f a$
 ⟨proof⟩

lemma *comp-surj-on*:
 $\llbracket surj\text{-on } f A B; surj\text{-on } g B C \rrbracket \implies surj\text{-on } (g \circ f) A C$
 ⟨proof⟩

lemma *surj-on-Un-right*: $surj\text{-on } f A (B1 \cup B2) = (surj\text{-on } f A B1 \wedge surj\text{-on } f A B2)$
 ⟨proof⟩

lemma *surj-on-Un-left*:
 $surj\text{-on } f (A1 \cup A2) B =$
 $(\exists B1. \exists B2. B \subseteq B1 \cup B2 \wedge surj\text{-on } f A1 B1 \wedge surj\text{-on } f A2 B2)$
 ⟨proof⟩

lemma *surj-on-diff-right*: $surj\text{-on } f A B \implies surj\text{-on } f A (B - B')$
 ⟨proof⟩

lemma *surj-on-empty-right*: $surj\text{-on } f A \{\}$
 ⟨proof⟩

lemma *surj-on-empty-left*: $surj\text{-on } f \{\} B = (B = \{\})$
 ⟨proof⟩

lemma *surj-on-imageI*: $surj\text{-on } (g \circ f) A B \implies surj\text{-on } g (f ' A) B$
 ⟨proof⟩

lemma *surj-on-insert-right*: $\text{surj-on } f \ A \ (\text{insert } b \ B) = (\text{surj-on } f \ A \ B \wedge \text{surj-on } f \ A \ \{b\})$
 ⟨proof⟩

lemma *surj-on-insert-left*: $\text{surj-on } f \ (\text{insert } a \ A) \ B = (\text{surj-on } f \ A \ (B - \{f \ a\}))$
 ⟨proof⟩

lemma *surj-on-subset-right*: $\llbracket \text{surj-on } f \ A \ B; B' \subseteq B \rrbracket \implies \text{surj-on } f \ A \ B'$
 ⟨proof⟩

lemma *surj-on-subset-left*: $\llbracket \text{surj-on } f \ A \ B; A \subseteq A' \rrbracket \implies \text{surj-on } f \ A' \ B$
 ⟨proof⟩

lemma *bij-betw-imp-surj-on*: $\text{bij-betw } f \ A \ B \implies \text{surj-on } f \ A \ B$
 ⟨proof⟩

lemma *bij-betw-inj-on-surj-on-conv*:
 $\text{bij-betw } f \ A \ B = (\text{inj-on } f \ A \wedge \text{surj-on } f \ A \ B \wedge f \ ' \ A \subseteq B)$
 ⟨proof⟩

6.1.4 Induction over natural sets

lemma *image-nat-induct*:
 $\llbracket P \ (f \ 0); \bigwedge n. P \ (f \ n) \implies P \ (f \ (\text{Suc } n)); \text{surj-on } f \ \text{UNIV } I; a \in I \rrbracket \implies P \ a$
 ⟨proof⟩

lemma *nat-induct*^[rule-format]:
 $\llbracket P \ n0; \bigwedge n. \llbracket n0 \leq n; P \ n \rrbracket \implies P \ (\text{Suc } n); n0 \leq n \rrbracket \implies P \ n$
 ⟨proof⟩

lemma *enat-induct*:
 $\llbracket P \ 0; P \ \infty; \bigwedge n. P \ n \implies P \ (\text{eSuc } n) \rrbracket \implies P \ n$
 ⟨proof⟩

lemma *eSuc-imp-Suc-aux-0*:
 $\llbracket \bigwedge n. P \ n \implies P \ (\text{eSuc } n); n0' \leq n'; P \ (\text{enat } n') \rrbracket \implies P \ (\text{enat } (\text{Suc } n'))$
 ⟨proof⟩

lemma *eSuc-imp-Suc-aux-n0*:
 $\llbracket \bigwedge n. \llbracket \text{enat } n0' \leq n; P \ n \rrbracket \implies P \ (\text{eSuc } n); n0' \leq n'; P \ (\text{enat } n') \rrbracket \implies P \ (\text{enat } (\text{Suc } n'))$
 ⟨proof⟩

lemma *enat-induct'*:
 $\llbracket P \ (n0::\text{enat}); P \ \infty; \bigwedge n. \llbracket n0 \leq n; P \ n \rrbracket \implies P \ (\text{eSuc } n); n0 \leq n \rrbracket \implies P \ n$
 ⟨proof⟩

lemma *wf-less-interval*:

$wf \{ (x,y). x \in (I::nat\ set) \wedge y \in I \wedge x < y \}$
 $\langle proof \rangle$

lemma *interval-induct*:

$\llbracket \bigwedge x. \forall y. (x \in (I::nat\ set) \wedge y \in I \wedge y < x \longrightarrow P\ y) \Longrightarrow P\ x \rrbracket$
 $\Longrightarrow P\ a$
(is $\llbracket \bigwedge x. \forall y. ?IA\ x\ y \Longrightarrow P\ x \rrbracket \Longrightarrow P\ a$)
 $\langle proof \rangle$

corollary *interval-induct-rule*:

$\llbracket \bigwedge x. (\bigwedge y. (x \in (I::nat\ set) \wedge y \in I \wedge y < x \Longrightarrow P\ y)) \Longrightarrow P\ x \rrbracket$
 $\Longrightarrow P\ a$
 $\langle proof \rangle$

6.1.5 Monotonicity and injectivity of arithmetic operators

lemma *add-left-inj*: $inj\ (\lambda x. n + (x::'a::cancel-ab-semigroup-add))$
 $\langle proof \rangle$

lemma *add-right-inj*: $inj\ (\lambda x. x + (n::'a::cancel-ab-semigroup-add))$
 $\langle proof \rangle$

lemma *mult-left-inj*: $0 < n \Longrightarrow inj\ (\lambda x. (n::nat) * x)$
 $\langle proof \rangle$

lemma *mult-right-inj*: $0 < n \Longrightarrow inj\ (\lambda x. x * (n::nat))$
 $\langle proof \rangle$

lemma *sub-left-inj-on*: $inj-on\ (\lambda x. (x::nat) - k) \{k..\}$
 $\langle proof \rangle$

lemma *sub-right-inj-on*: $inj-on\ (\lambda x. k - (x::nat)) \{..k\}$
 $\langle proof \rangle$

lemma *add-left-strict-mono*: $strict-mono\ (\lambda x. n + (x::'a::ordered-cancel-ab-semigroup-add))$
 $\langle proof \rangle$

lemma *add-right-strict-mono*: $strict-mono\ (\lambda x. x + (n::'a::ordered-cancel-ab-semigroup-add))$
 $\langle proof \rangle$

lemma *mult-left-strict-mono*: $0 < n \Longrightarrow strict-mono\ (\lambda x. n * (x::nat))$
 $\langle proof \rangle$

lemma *mult-right-strict-mono*: $0 < n \Longrightarrow strict-mono\ (\lambda x. x * (n::nat))$
 $\langle proof \rangle$

lemma *sub-left-strict-mono-on*: $strict-mono-on\ (\lambda x. (x::nat) - k) \{k..\}$
 $\langle proof \rangle$

lemma *div-right-strict-mono-on*:

$\llbracket 0 < (k::\text{nat}); \forall x \in I. \forall y \in I. x < y \longrightarrow x + k \leq y \rrbracket \implies$
strict-mono-on $(\lambda x. x \text{ div } k) I$
 ⟨proof⟩

lemma *mod-eq-div-right-strict-mono-on*:

$\llbracket 0 < (k::\text{nat}); \forall x \in I. \forall y \in I. x \text{ mod } k = y \text{ mod } k \rrbracket \implies$
strict-mono-on $(\lambda x. x \text{ div } k) I$
 ⟨proof⟩

corollary *div-right-inj-on*:

$\llbracket 0 < (k::\text{nat}); \forall x \in I. \forall y \in I. x < y \longrightarrow x + k \leq y \rrbracket \implies$
inj-on $(\lambda x. x \text{ div } k) I$
 ⟨proof⟩

corollary *mod-eq-imp-div-right-inj-on*:

$\llbracket 0 < (k::\text{nat}); \forall x \in I. \forall y \in I. x \text{ mod } k = y \text{ mod } k \rrbracket \implies$
inj-on $(\lambda x. x \text{ div } k) I$
 ⟨proof⟩

6.2 *Min* and *Max* elements of a set

A special minimum operator is required for dealing with infinite wellordered sets because the standard operator *Min* is usable only with finite sets.

definition *iMin* :: 'a::wellorder set \Rightarrow 'a

where *iMin* *I* \equiv *LEAST* *x*. *x* \in *I*

6.2.1 Basic results, as for *Least*

lemma *iMinI*: $k \in I \implies iMin I \in I$

⟨proof⟩

lemma *iMinI-ex*: $\exists x. x \in I \implies iMin I \in I$

⟨proof⟩

corollary *iMinI-ex2*: $I \neq \{\} \implies iMin I \in I$

⟨proof⟩

lemma *iMinI2*: $\llbracket k \in I; \bigwedge x. x \in I \implies P x \rrbracket \implies P (iMin I)$

⟨proof⟩

lemma *iMinI2-ex*: $\llbracket \exists x. x \in I; \bigwedge x. x \in I \implies P x \rrbracket \implies P (iMin I)$

⟨proof⟩

lemma *iMinI2-ex2*: $\llbracket I \neq \{\}; \bigwedge x. x \in I \implies P x \rrbracket \implies P (iMin I)$

⟨proof⟩

lemma *iMin-le[dest]*: $k \in I \implies iMin I \leq k$

$\langle \text{proof} \rangle$

lemma *iMin-neq-imp-greater*[*dest*]: $\llbracket k \in I; k \neq iMin\ I \rrbracket \implies iMin\ I < k$
 $\langle \text{proof} \rangle$

lemma *iMin-mono*:

$\llbracket \text{mono}\ f; \exists x. x \in I \rrbracket \implies iMin\ (f\ 'I) = f\ (iMin\ I)$
 $\langle \text{proof} \rangle$

corollary *iMin-mono2*:

$\llbracket \text{mono}\ f; I \neq \{\} \rrbracket \implies iMin\ (f\ 'I) = f\ (iMin\ I)$
 $\langle \text{proof} \rangle$

lemma *not-less-iMin*: $k < iMin\ I \implies k \notin I$
 $\langle \text{proof} \rangle$

lemma *Collect-not-less-iMin*: $k < iMin\ \{x. P\ x\} \implies \neg P\ k$
 $\langle \text{proof} \rangle$

lemma *Collect-iMin-le*: $P\ k \implies iMin\ \{x. P\ x\} \leq k$
 $\langle \text{proof} \rangle$

lemma *Collect-minI*: $\llbracket k \in I; P\ (k::('a::wellorder)) \rrbracket \implies \exists x \in I. P\ x \wedge (\forall y \in I. y < x \longrightarrow \neg P\ y)$
 $\langle \text{proof} \rangle$

corollary *Collect-minI-ex*: $\exists k \in I. P\ (k::('a::wellorder)) \implies \exists x \in I. P\ x \wedge (\forall y \in I. y < x \longrightarrow \neg P\ y)$
 $\langle \text{proof} \rangle$

corollary *Collect-minI-ex2*: $\{k \in I. P\ (k::('a::wellorder))\} \neq \{\} \implies \exists x \in I. P\ x \wedge (\forall y \in I. y < x \longrightarrow \neg P\ y)$
 $\langle \text{proof} \rangle$

lemma *iMin-the*: $iMin\ I = (THE\ x. x \in I \wedge (\forall y. y \in I \longrightarrow x \leq y))$
 $\langle \text{proof} \rangle$

lemma *iMin-the2*: $iMin\ I = (THE\ x. x \in I \wedge (\forall y \in I. x \leq y))$
 $\langle \text{proof} \rangle$

lemma *iMin-equality*:

$\llbracket k \in I; \bigwedge x. x \in I \implies k \leq x \rrbracket \implies iMin\ I = k$
 $\langle \text{proof} \rangle$

lemma *iMin-mono-on*:

$\llbracket \text{mono-on}\ f\ I; \exists x. x \in I \rrbracket \implies iMin\ (f\ 'I) = f\ (iMin\ I)$
 $\langle \text{proof} \rangle$

lemma *iMin-mono-on2*:

$\llbracket \text{mono-on } f \ I; I \neq \{\} \rrbracket \implies iMin (f \ ' \ I) = f (iMin \ I)$
 <proof>

lemma *iMinI2-order*:

$\llbracket k \in I; \bigwedge y. y \in I \implies k \leq y; \bigwedge x. \llbracket x \in I; \forall y \in I. x \leq y \rrbracket \implies P \ x \rrbracket \implies$
 $P (iMin \ I)$
 <proof>

lemma *wellorder-iMin-lemma*:

$k \in I \implies iMin \ I \in I \wedge iMin \ I \leq k$
 <proof>

lemma *iMin-Min-conv*:

$\llbracket \text{finite } I; I \neq \{\} \rrbracket \implies iMin \ I = Min \ I$
 <proof>

lemma *Min-neq-imp-greater[dest]*: $\llbracket \text{finite } I; k \in I; k \neq Min \ I \rrbracket \implies Min \ I < k$
 <proof>

lemma *Max-neq-imp-less[dest]*: $\llbracket \text{finite } I; k \in I; k \neq Max \ I \rrbracket \implies k < Max \ I$
 <proof>

lemma *nat-Least-mono*:

$\llbracket A \neq \{\}; \text{mono } (f::(\text{nat} \Rightarrow \text{nat})) \rrbracket \implies$
 $(LEAST \ x. x \in f \ ' \ A) = f (LEAST \ x. x \in A)$
 <proof>

lemma *Least-disj*:

$\llbracket \exists x. P \ x; \exists x. Q \ x \rrbracket \implies$
 $(LEAST \ (x::'a::\text{wellorder}). (P \ x \vee Q \ x)) = \min (LEAST \ x. P \ x) (LEAST \ x. Q \ x)$
 <proof>

lemma *Least-imp-le*:

$\llbracket \exists x. P \ x; \bigwedge x. P \ x \implies Q \ x \rrbracket \implies$
 $(LEAST \ (x::'a::\text{wellorder}). Q \ x) \leq (LEAST \ x. P \ x)$
 <proof>

lemma *Least-imp-disj-eq*:

$\llbracket \exists x. P \ x; \bigwedge x. P \ x \implies Q \ x \rrbracket \implies$
 $(LEAST \ (x::'a::\text{wellorder}). P \ x \vee Q \ x) = (LEAST \ x. Q \ x)$
 <proof>

lemma *Least-le-imp-le*:

$\llbracket \exists x. P \ x; \exists x. Q \ x; \bigwedge x \ y. \llbracket P \ x; Q \ y \rrbracket \implies x \leq y \rrbracket \implies$
 $(LEAST \ (x::'a::\text{wellorder}). P \ x) \leq (LEAST \ (x::'a::\text{wellorder}). Q \ x)$
 <proof>

lemma *Least-le-imp-le-disj*:

$\llbracket \exists x. P x; \bigwedge x y. \llbracket P x; Q y \rrbracket \implies x \leq y \rrbracket \implies$
 $(LEAST (x::'a::wellorder). P x \vee Q x) = (LEAST (x::'a::wellorder). P x)$
 <proof>

lemma *Max-equality*: $\llbracket (k::'a::linorder) \in A; \text{finite } A; \bigwedge x. x \in A \implies x \leq k \rrbracket \implies$
 $Max A = k$
 <proof>

lemma *not-greater-Max*: $\llbracket \text{finite } A; Max A < k \rrbracket \implies k \notin A$
 <proof>

lemma *Collect-not-greater-Max*: $\llbracket \text{finite } \{x. P x\}; Max \{x. P x\} < k \rrbracket \implies \neg P k$
 <proof>

lemma *Collect-Max-ge*: $\llbracket \text{finite } \{x. P x\}; P k \rrbracket \implies k \leq Max \{x. P x\}$
 <proof>

lemma *MaxI*: $\llbracket k \in A; \text{finite } A \rrbracket \implies Max A \in A$
 <proof>

lemma *MaxI-ex*: $\llbracket \exists x. x \in A; \text{finite } A \rrbracket \implies Max A \in A$
 <proof>

lemma *MaxI-ex2*: $\llbracket A \neq \{\}; \text{finite } A \rrbracket \implies Max A \in A$
 <proof>

lemma *MaxI2*: $\llbracket k \in A; \bigwedge x. x \in A \implies P x; \text{finite } A \rrbracket \implies P (Max A)$
 <proof>

lemma *MaxI2-ex*: $\llbracket \exists x. x \in A; \bigwedge x. x \in A \implies P x; \text{finite } A \rrbracket \implies P (Max A)$
 <proof>

lemma *MaxI2-ex2*: $\llbracket A \neq \{\}; \bigwedge x. x \in A \implies P x; \text{finite } A \rrbracket \implies P (Max A)$
 <proof>

lemma *Max-mono*: $\llbracket \text{mono } f; \exists x. x \in A; \text{finite } A \rrbracket \implies Max (f ' A) = f (Max A)$
 <proof>

lemma *Max-mono2*: $\llbracket \text{mono } f; A \neq \{\}; \text{finite } A \rrbracket \implies Max (f ' A) = f (Max A)$
 <proof>

lemma *Max-mono-on*: $\llbracket \text{mono-on } f A; \exists x. x \in A; \text{finite } A \rrbracket \implies Max (f ' A) = f (Max A)$
 <proof>

lemma *Max-mono-on2*:
 $\llbracket \text{mono-on } f A; A \neq \{\}; \text{finite } A \rrbracket \implies Max (f ' A) = f (Max A)$

<proof>

lemma *Max-the*:

$\llbracket A \neq \{\}; \text{finite } A \rrbracket \implies$

$\text{Max } A = (\text{THE } x. x \in A \wedge (\forall y. y \in A \longrightarrow y \leq x))$

<proof>

lemma *Max-the2*: $\llbracket A \neq \{\}; \text{finite } A \rrbracket \implies$

$\text{Max } A = (\text{THE } x. x \in A \wedge (\forall y \in A. y \leq x))$

<proof>

lemma *wellorder-Max-lemma*: $\llbracket k \in A; \text{finite } A \rrbracket \implies \text{Max } A \in A \wedge k \leq \text{Max } A$

<proof>

lemma *MaxI2-order*: $\llbracket k \in A; \text{finite } A; \bigwedge y. y \in A \implies y \leq k; \bigwedge x. \llbracket x \in A; \forall y \in A. y \leq x \rrbracket \implies P x \rrbracket$

$\implies P (\text{Max } A)$

<proof>

lemma *Min-le-Max*: $\llbracket \text{finite } A; A \neq \{\} \rrbracket \implies \text{Min } A \leq \text{Max } A$

<proof>

lemma *iMin-le-Max*: $\llbracket \text{finite } A; A \neq \{\} \rrbracket \implies \text{iMin } A \leq \text{Max } A$

<proof>

6.2.2 Max for sets over *enat*

definition *iMax* :: *nat set* \Rightarrow *enat*

where *iMax* *i* \equiv *if* (*finite* *i*) *then* (*enat* (*Max* *i*)) *else* ∞

lemma *iMax-finite-conv*: *finite* *I* = (*iMax* *I* \neq ∞)

<proof>

lemma *iMax-infinite-conv*: *infinite* *I* = (*iMax* *I* = ∞)

<proof>

lemma *class.distrib-lattice* (*min*::('a::linorder \Rightarrow 'a \Rightarrow 'a)) (\leq) ($<$) *max*

<proof>

print-locale *Lattices.distrib-lattice*

lemma *max-Min-eq-Min-max*[*rule-format*]:

finite *A* \implies

$A \neq \{\} \longrightarrow$

$\text{max } x (\text{Min } A) = \text{Min } \{\text{max } x a \mid a. a \in A\}$

<proof>

lemma *max-iMin-eq-iMin-max*:

$\llbracket \text{finite } A; A \neq \{\} \rrbracket \implies$

$\max x (iMin A) = iMin \{ \max x a \mid a. a \in A \}$
 ⟨proof⟩

lemma $\llbracket \text{finite } A; A \neq \{\} \rrbracket \implies \forall x \in A. x \leq Max A$
 ⟨proof⟩

6.2.3 *Min and Max for set operations*

lemma *iMin-subset*: $\llbracket A \neq \{\}; A \subseteq B \rrbracket \implies iMin B \leq iMin A$
 ⟨proof⟩

lemma *Max-subset*: $\llbracket A \neq \{\}; A \subseteq B; \text{finite } B \rrbracket \implies Max A \leq Max B$
 ⟨proof⟩

lemma *Min-subset*: $\llbracket A \neq \{\}; A \subseteq B; \text{finite } B \rrbracket \implies Min B \leq Min A$
 ⟨proof⟩

lemma *iMin-Int-ge1*: $(A \cap B) \neq \{\} \implies iMin A \leq iMin (A \cap B)$
 ⟨proof⟩

lemma *iMin-Int-ge2*: $(A \cap B) \neq \{\} \implies iMin B \leq iMin (A \cap B)$
 ⟨proof⟩

lemma *iMin-Int-ge*: $(A \cap B) \neq \{\} \implies \max (iMin A) (iMin B) \leq iMin (A \cap B)$
 ⟨proof⟩

lemma *Max-Int-le1*: $\llbracket (A \cap B) \neq \{\}; \text{finite } A \rrbracket \implies Max (A \cap B) \leq Max A$
 ⟨proof⟩

lemma *Max-Int-le2*: $\llbracket (A \cap B) \neq \{\}; \text{finite } B \rrbracket \implies Max (A \cap B) \leq Max B$
 ⟨proof⟩

lemma *Max-Int-le*: $\llbracket (A \cap B) \neq \{\}; \text{finite } A; \text{finite } B \rrbracket \implies$
 $Max (A \cap B) \leq \min (Max A) (Max B)$
 ⟨proof⟩

lemma *iMin-Un*[*rule-format*]:
 $\llbracket A \neq \{\}; B \neq \{\} \rrbracket \implies$
 $iMin (A \cup B) = \min (iMin A) (iMin B)$
 ⟨proof⟩

lemma *iMin-singleton*[*simp*]: $iMin \{a\} = a$
 ⟨proof⟩

lemma *iMax-singleton*[*simp*]: $iMax \{a\} = enat a$

$\langle \text{proof} \rangle$

lemma *Max-le-Min-imp-singleton*:

$$\llbracket \text{finite } A; A \neq \{\}; \text{Max } A \leq \text{Min } A \rrbracket \implies A = \{\text{Min } A\}$$

$\langle \text{proof} \rangle$

lemma *Max-le-Min-conv-singleton*:

$$\llbracket \text{finite } A; A \neq \{\} \rrbracket \implies (\text{Max } A \leq \text{Min } A) = (\exists x. A = \{x\})$$

$\langle \text{proof} \rangle$

lemma *Max-le-iMin-imp-le*:

$$\llbracket \text{finite } A; \text{Max } A \leq \text{iMin } B; a \in A; b \in B \rrbracket \implies a \leq b$$

$\langle \text{proof} \rangle$

lemma *le-imp-Max-le-iMin*:

$$\llbracket \text{finite } A; A \neq \{\}; B \neq \{\}; \forall a \in A. \forall b \in B. a \leq b \rrbracket \implies \text{Max } A \leq \text{iMin } B$$

$\langle \text{proof} \rangle$

lemma *Max-le-iMin-conv-le*:

$$\llbracket \text{finite } A; A \neq \{\}; B \neq \{\} \rrbracket \implies (\text{Max } A \leq \text{iMin } B) = (\forall a \in A. \forall b \in B. a \leq b)$$

$\langle \text{proof} \rangle$

lemma *Max-less-iMin-imp-less*:

$$\llbracket \text{finite } A; \text{Max } A < \text{iMin } B; a \in A; b \in B \rrbracket \implies a < b$$

$\langle \text{proof} \rangle$

lemma *less-imp-Max-less-iMin*:

$$\llbracket \text{finite } A; A \neq \{\}; B \neq \{\}; \forall a \in A. \forall b \in B. a < b \rrbracket \implies \text{Max } A < \text{iMin } B$$

$\langle \text{proof} \rangle$

lemma *Max-less-iMin-conv-less*:

$$\llbracket \text{finite } A; A \neq \{\}; B \neq \{\} \rrbracket \implies (\text{Max } A < \text{iMin } B) = (\forall a \in A. \forall b \in B. a < b)$$

$\langle \text{proof} \rangle$

lemma *Max-less-iMin-imp-disjoint*:

$$\llbracket \text{finite } A; \text{Max } A < \text{iMin } B \rrbracket \implies A \cap B = \{\}$$

$\langle \text{proof} \rangle$

lemma *iMin-in-idem*: $n \in I \implies \text{min } n (\text{iMin } I) = \text{iMin } I$

$\langle \text{proof} \rangle$

lemma *iMin-insert*: $I \neq \{\} \implies \text{iMin } (\text{insert } n I) = \text{min } n (\text{iMin } I)$

$\langle \text{proof} \rangle$

lemma *iMin-insert-remove*:

$$\begin{aligned} & \text{iMin } (\text{insert } n I) = \\ & (\text{if } I - \{n\} = \{\} \text{ then } n \text{ else } \text{min } n (\text{iMin } (I - \{n\}))) \end{aligned}$$

<proof>

lemma *iMin-remove*: $n \in I \implies iMin\ I = (if\ I - \{n\} = \{\} \text{ then } n \text{ else } min\ n$
 $(iMin\ (I - \{n\})))$

<proof>

lemma *iMin-subset-idem*: $\llbracket B \neq \{\}; B \subseteq A \rrbracket \implies min\ (iMin\ B)\ (iMin\ A) = iMin\ A$

<proof>

lemma *iMin-union-inter*: $A \cap B \neq \{\} \implies min\ (iMin\ (A \cup B))\ (iMin\ (A \cap B))$
 $= min\ (iMin\ A)\ (iMin\ B)$

<proof>

lemma *iMin-ge-iff*: $I \neq \{\} \implies (n \leq iMin\ I) = (\forall a \in I. n \leq a)$

<proof>

lemma *iMin-gr-iff*: $I \neq \{\} \implies (n < iMin\ I) = (\forall a \in I. n < a)$

<proof>

lemma *iMin-le-iff*: $I \neq \{\} \implies (iMin\ I \leq n) = (\exists a \in I. a \leq n)$

<proof>

lemma *iMin-less-iff*: $I \neq \{\} \implies (iMin\ I < n) = (\exists a \in I. a < n)$

<proof>

lemma *hom-iMin-commute*: $\llbracket \bigwedge x\ y. h\ (min\ x\ y) = min\ (h\ x)\ (h\ y); I \neq \{\} \rrbracket \implies$
 $iMin\ (h\ ` I) = h\ (iMin\ I)$

<proof>

Synonyms for similarity with theorem names for *Min*"

lemmas *iMin-eqI = iMin-equality*

lemmas *iMin-in = iMinI-ex2*

6.3 Some auxiliary results for set operations

6.3.1 Some additional abbreviations for relations

Abbreviations for *refl*, *sym*, *equiv*, *refl*, *trans*

abbreviation *(input) reflP* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$ **where**
 $reflP\ r \equiv refl\ \{(x, y). r\ x\ y\}$

abbreviation *(input) symP* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$ **where**
 $symP\ r \equiv sym\ \{(x, y). r\ x\ y\}$

abbreviation *(input) transP* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$ **where**
 $transP\ r \equiv trans\ \{(x, y). r\ x\ y\}$

abbreviation (*input*) $\text{equivP} :: ('a \Rightarrow 'a \Rightarrow \text{bool}) \Rightarrow \text{bool}$ **where**
 $\text{equivP } r \equiv \text{reflP } r \wedge \text{symP } r \wedge \text{transp } r$

abbreviation (*input*) $\text{irreflP} :: ('a \Rightarrow 'a \Rightarrow \text{bool}) \Rightarrow \text{bool}$ **where**
 $\text{irreflP } r \equiv \text{irrefl } \{(x, y). r \ x \ y\}$

Example for reflP

lemma $\text{reflP } ((\leq)::('a::\text{preorder} \Rightarrow 'a \Rightarrow \text{bool}))$
 $\langle \text{proof} \rangle$

Example for symP

lemma $\text{symP } (=)$
 $\langle \text{proof} \rangle$

Example for equivP

lemma $\text{equivP } (=)$
 $\langle \text{proof} \rangle$

Example for irreflP

lemma $\text{irreflP } ((<)::('a::\text{preorder} \Rightarrow 'a \Rightarrow \text{bool}))$
 $\langle \text{proof} \rangle$

6.3.2 Auxiliary results for *singletons*

lemma *singleton-not-empty*: $\{a\} \neq \{\}$ $\langle \text{proof} \rangle$

lemma *singleton-finite*: $\text{finite } \{a\}$ $\langle \text{proof} \rangle$

lemma *singleton-ball*: $(\forall x \in \{a\}. P \ x) = P \ a$ $\langle \text{proof} \rangle$

lemma *singleton-bex*: $(\exists x \in \{a\}. P \ x) = P \ a$ $\langle \text{proof} \rangle$

lemma *subset-singleton-conv*: $(A \subseteq \{a\}) = (A = \{\} \vee A = \{a\})$ $\langle \text{proof} \rangle$

lemma *singleton-subset-conv*: $(\{a\} \subseteq A) = (a \in A)$ $\langle \text{proof} \rangle$

lemma *singleton-eq-conv*: $(\{a\} = \{b\}) = (a = b)$ $\langle \text{proof} \rangle$

lemma *singleton-subset-singleton-conv*: $(\{a\} \subseteq \{b\}) = (a = b)$ $\langle \text{proof} \rangle$

lemma *singleton-Int1-if*: $\{a\} \cap A = (\text{if } a \in A \text{ then } \{a\} \text{ else } \{\})$
 $\langle \text{proof} \rangle$

lemma *singleton-Int2-if*: $A \cap \{a\} = (\text{if } a \in A \text{ then } \{a\} \text{ else } \{\})$
 $\langle \text{proof} \rangle$

lemma *singleton-image*: $f \ ` \ \{a\} = \{f \ a\}$ $\langle \text{proof} \rangle$

lemma *inj-on-singleton*: $\text{inj-on } f \ \{a\}$ $\langle \text{proof} \rangle$

lemma *strict-mono-on-singleton*: $\text{strict-mono-on } f \ \{a\}$
 $\langle \text{proof} \rangle$

6.3.3 Auxiliary results for *finite* and *infinite* sets

lemma *infinite-imp-not-singleton*: $\text{infinite } A \implies \neg (\exists a. A = \{a\})$ *<proof>*

lemma *infinite-insert*: $\text{infinite } (\text{insert } a \ A) = \text{infinite } A$
<proof>

lemma *infinite-Diff-insert*: $\text{infinite } (A - \text{insert } a \ B) = \text{infinite } (A - B)$
<proof>

lemma *subset-finite-imp-finite*: $\llbracket \text{finite } A; B \subseteq A \rrbracket \implies \text{finite } B$
<proof>

lemma *infinite-not-subset-finite*: $\llbracket \text{infinite } A; \text{finite } B \rrbracket \implies \neg A \subseteq B$
<proof>

lemma *Un-infinite-right*: $\text{infinite } T \implies \text{infinite } (S \cup T)$ *<proof>*

lemma *Un-infinite-iff*: $\text{infinite } (S \cup T) = (\text{infinite } S \vee \text{infinite } T)$ *<proof>*

Give own name to the lemma about finiteness of the integer image of a nat set

corollary *finite-A-int-A-conv*: $\text{finite } A = \text{finite } (\text{int } 'A)$
<proof>

Corresponding fact fo infinite sets

corollary *infinite-A-int-A-conv*: $\text{infinite } A = \text{infinite } (\text{int } 'A)$
<proof>

lemma *cartesian-product-infiniteL-imp-infinite*: $\llbracket \text{infinite } A; B \neq \{\} \rrbracket \implies \text{infinite } (A \times B)$
<proof>

lemma *cartesian-product-infiniteR-imp-infinite*: $\llbracket \text{infinite } B; A \neq \{\} \rrbracket \implies \text{infinite } (A \times B)$
<proof>

lemma *finite-nat-iff-bounded2*:
 $\text{finite } S = (\exists (k::\text{nat}). \forall n \in S. n < k)$
<proof>

lemma *finite-nat-iff-bounded-le2*:
 $\text{finite } S = (\exists (k::\text{nat}). \forall n \in S. n \leq k)$
<proof>

lemma *nat-asc-chain-imp-unbounded*:
 $\llbracket S \neq \{\}; (\forall m \in S. \exists n \in S. m < (n::\text{nat})) \rrbracket \implies \forall m. \exists n \in S. m \leq n$
<proof>

lemma *infinite-nat-iff-asc-chain*:

$$S \neq \{\} \implies \text{infinite } S = (\forall m \in S. \exists n \in S. m < (n::\text{nat}))$$

<proof>

lemma *infinite-imp-asc-chain*: $\text{infinite } S \implies \forall m \in S. \exists n \in S. m < (n::\text{nat})$

<proof>

lemma *infinite-image-imp-infinite*: $\text{infinite } (f \text{ ' } A) \implies \text{infinite } A$

<proof>

lemma *inj-on-imp-infinite-image*: $\llbracket \text{infinite } A; \text{inj-on } f \ A \rrbracket \implies \text{infinite } (f \text{ ' } A)$

<proof>

lemma *inj-on-infinite-image-iff*: $\text{inj-on } f \ A \implies \text{infinite } (f \text{ ' } A) = \text{infinite } A$

<proof>

lemma *inj-on-finite-image-iff*: $\text{inj-on } f \ A \implies \text{finite } (f \text{ ' } A) = \text{finite } A$

<proof>

lemma *nat-ex-greater-finite-Max-conv*:

$$A \neq \{\} \implies (\exists x \in A. (n::\text{nat}) < x) = (\text{finite } A \longrightarrow n < \text{Max } A)$$

<proof>

corollary *nat-ex-greater-infinite-finite-Max-conv'*:

$$(\exists x \in A. (n::\text{nat}) < x) = (\text{finite } A \wedge A \neq \{\} \wedge n < \text{Max } A \vee \text{infinite } A)$$

<proof>

6.3.4 Some auxiliary results for disjoint sets

lemma *disjoint-iff-in-not-in1*: $(A \cap B = \{\}) = (\forall x \in A. x \notin B)$ *<proof>*

lemma *disjoint-iff-in-not-in2*: $(A \cap B = \{\}) = (\forall x \in B. x \notin A)$ *<proof>*

lemma *disjoint-in-Un*:

$$\llbracket A \cap B = \{\}; x \in A \cup B \rrbracket \implies x \notin A \vee x \notin B$$

<proof>

lemma *partition-Union*: $A \subseteq \bigcup C \implies (\bigcup c \in C. A \cap c) = A$ *<proof>*

lemma *disjoint-partition-Int*:

$$\begin{aligned} & \forall c1 \in C. \forall c2 \in C. c1 \neq c2 \longrightarrow c1 \cap c2 = \{\} \implies \\ & \forall a1 \in \{A \cap c \mid c. c \in C\}. \forall a2 \in \{A \cap c \mid c. c \in C\}. \\ & \quad a1 \neq a2 \longrightarrow a1 \cap a2 = \{\} \end{aligned}$$

<proof>

lemma $\{f \ x \mid x. x \in A\} = (\bigcup x \in A. \{f \ x\})$

<proof>

This lemma version drops the superfluous precondition $\text{finite } (\bigcup C)$ (and

turns the resulting equation to the sensible order $\text{card } .. = k * \text{card } ..$).

lemma *card-partition*:

$\llbracket \text{finite } C; \bigwedge c. c \in C \implies \text{card } c = k; \bigwedge c1\ c2. \llbracket c1 \in C; c2 \in C; c1 \neq c2 \rrbracket \implies$
 $c1 \cap c2 = \{\}$ $\rrbracket \implies$
 $\text{card } (\bigcup C) = k * \text{card } C$
 $\langle \text{proof} \rangle$

6.3.5 Some auxiliary results for subset relation

lemma *subset-image-Int*: $A \subseteq B \implies f' (A \cap B) = f' A \cap f' B$
 $\langle \text{proof} \rangle$

lemma *image-diff-disjoint-image-Int*:

$\llbracket f' (A - B) \cap f' B = \{\} \rrbracket \implies$
 $f' (A \cap B) = f' A \cap f' B$
 $\langle \text{proof} \rangle$

lemma *subset-imp-Int-subset1*: $A \subseteq C \implies A \cap B \subseteq C$
 $\langle \text{proof} \rangle$

lemma *subset-imp-Int-subset2*: $B \subseteq C \implies A \cap B \subseteq C$
 $\langle \text{proof} \rangle$

6.3.6 Auxiliary results for intervals from *SetInterval*

lemmas *set-interval-defs* =
lessThan-def atMost-def
greaterThan-def atLeast-def
greaterThanLessThan-def atLeastLessThan-def
greaterThanAtMost-def atLeastAtMost-def

lemma *image-add-atLeast*:
 $(\lambda n::\text{nat}. n+k)' \{i..\} = \{i+k..\}$ **(is ?A = ?B)**
 $\langle \text{proof} \rangle$

lemma *image-add-atMost*:
 $(\lambda n::\text{nat}. n+k)' \{..i\} = \{k..i+k\}$ **(is ?A = ?B)**
 $\langle \text{proof} \rangle$

corollary *image-Suc-atLeast*: $\text{Suc}' \{i..\} = \{\text{Suc } i..\}$
 $\langle \text{proof} \rangle$

corollary *image-Suc-atMost*: $\text{Suc}' \{..i\} = \{\text{Suc } 0..\text{Suc } i\}$
 $\langle \text{proof} \rangle$

lemmas *image-add-lemmas* =
image-add-atLeastAtMost
image-add-atLeast

image-add-atMost
lemmas *image-Suc-lemmas* =
image-Suc-atLeastAtMost
image-Suc-atLeast
image-Suc-atMost

lemma *atMost-atLeastAtMost-0-conv*: $\{..i::nat\} = \{0..i\}$
 ⟨*proof*⟩

lemma *atLeastAtMost-subset-atMost*: $(k::'a::order) \leq k' \implies \{l..k\} \subseteq \{..k'\}$
 ⟨*proof*⟩

lemma *lessThan-insert*: $insert (n::'a::order) \{..<n\} = \{..n\}$
 ⟨*proof*⟩

lemma *greaterThan-insert*: $insert (n::'a::order) \{n<..\} = \{n..\}$
 ⟨*proof*⟩

lemma *atMost-remove*: $\{..n\} - \{(n::'a::order)\} = \{..<n\}$
 ⟨*proof*⟩

lemma *atLeast-remove*: $\{n..\} - \{(n::'a::order)\} = \{n<..\}$
 ⟨*proof*⟩

lemma *atMost-lessThan-conv*: $\{..n\} = \{..<Suc\ n\}$
 ⟨*proof*⟩

lemma *atLeastAtMost-atLeastLessThan-conv*: $\{l..u\} = \{l..<Suc\ u\}$
 ⟨*proof*⟩

lemma *atLeast-greaterThan-conv*: $\{Suc\ n..\} = \{n<..\}$
 ⟨*proof*⟩

lemma *atLeastAtMost-greaterThanAtMost-conv*: $\{Suc\ l..u\} = \{l<..u\}$
 ⟨*proof*⟩

lemma *finite-subset-atLeastAtMost*: $finite\ A \implies A \subseteq \{Min\ A..Max\ A\}$
 ⟨*proof*⟩

lemma *Max-le-imp-subset-atMost*:
 $\llbracket finite\ A; Max\ A \leq n \rrbracket \implies A \subseteq \{..n\}$
 ⟨*proof*⟩

lemma *subset-atMost-imp-Max-le*:
 $\llbracket finite\ A; A \neq \{\}; A \subseteq \{..n\} \rrbracket \implies Max\ A \leq n$
 ⟨*proof*⟩

lemma *subset-atMost-Max-le-conv*:

$\llbracket \text{finite } A; A \neq \{\} \rrbracket \implies (A \subseteq \{..n\}) = (\text{Max } A \leq n)$
 ⟨proof⟩

lemma *iMin-atLeast*: $i\text{Min } \{n..\} = n$

⟨proof⟩

lemma *iMin-greaterThan*: $i\text{Min } \{n<..\} = \text{Suc } n$

⟨proof⟩

lemma *iMin-atMost*: $i\text{Min } \{..(n::\text{nat})\} = 0$

⟨proof⟩

lemma *iMin-lessThan*: $0 < n \implies i\text{Min } \{..<(n::\text{nat})\} = 0$

⟨proof⟩

lemma *Max-atMost*: $\text{Max } \{..(n::\text{nat})\} = n$

⟨proof⟩

lemma *Max-lessThan*: $0 < n \implies \text{Max } \{..<n\} = n - \text{Suc } 0$

⟨proof⟩

lemma *iMin-atLeastLessThan*: $m < n \implies i\text{Min } \{m..<n\} = m$

⟨proof⟩

lemma *Max-atLeastLessThan*: $m < n \implies \text{Max } \{m..<n\} = n - \text{Suc } 0$

⟨proof⟩

lemma *iMin-greaterThanLessThan*: $\text{Suc } m < n \implies i\text{Min } \{m<..<n\} = \text{Suc } m$

⟨proof⟩

lemma *Max-greaterThanLessThan*: $\text{Suc } m < n \implies \text{Max } \{m<..<n\} = n - \text{Suc } 0$

⟨proof⟩

lemma *iMin-greaterThanAtMost*: $m < n \implies i\text{Min } \{m<..n\} = \text{Suc } m$

⟨proof⟩

lemma *Max-greaterThanAtMost*: $m < n \implies \text{Max } \{m<..(n::\text{nat})\} = n$

⟨proof⟩

lemma *iMin-atLeastAtMost*: $m \leq n \implies i\text{Min } \{m..n\} = m$

⟨proof⟩

lemma *Max-atLeastAtMost*: $m \leq n \implies \text{Max } \{m..(n::\text{nat})\} = n$

⟨proof⟩

lemma *infinite-atLeast*: $\text{infinite } \{(n::\text{nat})..\}$

<proof>

lemma *infinite-greaterThan*: *infinite* $\{(n::nat)<..\}$
<proof>

lemma *infinite-atLeast-int*: *infinite* $\{(n::int)..\}$
<proof>

lemma *infinite-greaterThan-int*: *infinite* $\{(n::int)<..\}$
<proof>

lemma *infinite-atMost-int*: *infinite* $\{..(n::int)\}$
<proof>

lemma *infinite-lessThan-int*: *infinite* $\{..<(n::int)\}$
<proof>

6.3.7 Auxiliary results for *card*

lemma *sum-singleton*: $(\sum x \in \{a\}. f x) = f a$
<proof>

lemma *card-singleton*: $\text{card } \{a\} = \text{Suc } 0$
<proof>

lemma *card-cartesian-product-singleton-right*: $\text{card } (A \times \{x\}) = \text{card } A$
<proof>

lemma *card-1-imp-singleton*: $\text{card } A = \text{Suc } 0 \implies (\exists a. A = \{a\})$
<proof>

lemma *card-1-singleton-conv*: $(\text{card } A = \text{Suc } 0) = (\exists a. A = \{a\})$
<proof>

lemma *card-gr0-imp-finite*: $0 < \text{card } A \implies \text{finite } A$
<proof>

lemma *card-gr0-imp-not-empty*: $(0 < \text{card } A) \implies A \neq \{\}$
<proof>

lemma *not-empty-card-gr0-conv*: $\text{finite } A \implies (A \neq \{\}) = (0 < \text{card } A)$
<proof>

lemma *nat-card-le-Max*: $\text{card } (A::\text{nat set}) \leq \text{Suc } (\text{Max } A)$
<proof>

lemma *Int-card1*: $\text{finite } A \implies \text{card } (A \cap B) \leq \text{card } A$
<proof>

lemma *Int-card2*: $\text{finite } B \implies \text{card } (A \cap B) \leq \text{card } B$
<proof>

lemma *Un-card1*: $\llbracket \text{finite } A; \text{finite } B \rrbracket \implies \text{card } A \leq \text{card } (A \cup B)$

<proof>

lemma *Un-card2*: $\llbracket \text{finite } A; \text{finite } B \rrbracket \implies \text{card } B \leq \text{card } (A \cup B)$

<proof>

lemma *card-Un-conv*:

$\llbracket \text{finite } A; \text{finite } B \rrbracket \implies$

$\text{card } (A \cup B) = \text{card } A + \text{card } B - \text{card } (A \cap B)$

<proof>

lemma *card-Int-conv*:

$\llbracket \text{finite } A; \text{finite } B \rrbracket \implies$

$\text{card } (A \cap B) = \text{card } A + \text{card } B - \text{card } (A \cup B)$

<proof>

Pigeonhole principle, dirichlet's box principle

lemma *pigeonhole-principle**[rule-format]*:

$\text{card } (f \text{ ' } A) < \text{card } A \longrightarrow (\exists x \in A. \exists y \in A. x \neq y \wedge f x = f y)$

<proof>

corollary *pigeonhole-principle-linorder**[rule-format]*:

$\text{card } (f \text{ ' } A) < \text{card } (A::\text{'a}::\text{linorder set}) \implies (\exists x \in A. \exists y \in A. x < y \wedge f x = f y)$

<proof>

corollary *pigeonhole-mod*:

$\llbracket 0 < m; m < \text{card } A \rrbracket \implies \exists x \in A. \exists y \in A. x < y \wedge x \text{ mod } m = y \text{ mod } m$

<proof>

corollary *pigeonhole-mod2*:

$\llbracket (0::\text{nat}) < m; m \leq c; \text{inj-on } f \text{ \{..c\}} \rrbracket \implies \exists x \leq c. \exists y \leq c. x < y \wedge f x \text{ mod } m = f y \text{ mod } m$

<proof>

end

7 Cutting linearly ordered and natural sets

theory *SetIntervalCut*

imports *SetInterval2*

begin

7.1 Set restriction

A set to set function f is a *set restriction*, if there exists a predicate P , so that for every set s the function result $f s$ contains all its elements fulfilling P

definition *set-restriction* $:: ('a \text{ set} \Rightarrow 'a \text{ set}) \Rightarrow \text{bool}$

where *set-restriction* $f \equiv \exists P. \forall A. f A = \{x \in A. P x\}$

lemma *set-restrictionD*: *set-restriction* $f \implies \exists P. \forall A. f A = \{x \in A. P x\}$

<proof>

lemma *set-restrictionD-spec*: $set_restriction\ f \implies \exists P. f\ A = \{x \in A. P\ x\}$

<proof>

lemma *set-restrictionI*: $f = (\lambda A. \{x \in A. P\ x\}) \implies set_restriction\ f$

<proof>

lemma *set-restriction-comp*:

$\llbracket set_restriction\ f; set_restriction\ g \rrbracket \implies set_restriction\ (f \circ g)$

<proof>

lemma *set-restriction-commute*:

$\llbracket set_restriction\ f; set_restriction\ g \rrbracket \implies f\ (g\ I) = g\ (f\ I)$

<proof>

Constructs a set restriction function with the given restriction predicate

definition

$set_restriction_fun :: ('a \Rightarrow bool) \Rightarrow ('a\ set \Rightarrow 'a\ set)$

where

$set_restriction_fun\ P \equiv \lambda A. \{x \in A. P\ x\}$

lemma *set-restriction-fun-is-set-restriction*:

$set_restriction\ (set_restriction_fun\ P)$

<proof>

lemma *set-restriction-Int-conv*:

$set_restriction\ f = (\exists B. \forall A. f\ A = A \cap B)$

<proof>

lemma *set-restriction-Un*:

$set_restriction\ f \implies f\ (A \cup B) = f\ A \cup f\ B$

<proof>

lemma *set-restriction-Int*:

$set_restriction\ f \implies f\ (A \cap B) = f\ A \cap f\ B$

<proof>

lemma *set-restriction-Diff*:

$set_restriction\ f \implies f\ (A - B) = f\ A - f\ B$

<proof>

lemma *set-restriction-mono*:

$\llbracket set_restriction\ f; A \subseteq B \rrbracket \implies f\ A \subseteq f\ B$

<proof>

lemma *set-restriction-absorb*:

$set_restriction\ f \implies f\ (f\ A) = f\ A$

<proof>

lemma *set-restriction-empty*:

$set_restriction\ f \implies f\ \{\} = \{\}$

<proof>

lemma *set-restriction-non-empty-imp*:

$\llbracket set_restriction\ f; f\ A \neq \{\} \rrbracket \implies A \neq \{\}$

<proof>

lemma *set-restriction-subset*:

$set_restriction\ f \implies f\ A \subseteq A$
 <proof>
lemma *set-restriction-finite*:
 $\llbracket\ set_restriction\ f; finite\ A \rrbracket \implies finite\ (f\ A)$
 <proof>
lemma *set-restriction-card*:
 $\llbracket\ set_restriction\ f; finite\ A \rrbracket \implies$
 $card\ (f\ A) = card\ A - card\ \{a \in A. f\ \{a\} = \{\}\}$
 <proof>
lemma *set-restriction-card-le*:
 $\llbracket\ set_restriction\ f; finite\ A \rrbracket \implies card\ (f\ A) \leq card\ A$
 <proof>
lemma *set-restriction-not-in-imp*:
 $\llbracket\ set_restriction\ f; x \notin A \rrbracket \implies x \notin f\ A$
 <proof>
lemma *set-restriction-in-imp*:
 $\llbracket\ set_restriction\ f; x \in f\ A \rrbracket \implies x \in A$
 <proof>
lemma *set-restriction-fun-singleton*:
 $set_restriction_fun\ P\ \{a\} = (if\ P\ a\ then\ \{a\}\ else\ \{\})$
 <proof>
lemma *set-restriction-fun-all-conv*:
 $((set_restriction_fun\ P)\ A = A) = (\forall\ x \in A. P\ x)$
 <proof>
lemma *set-restriction-fun-empty-conv*:
 $((set_restriction_fun\ P)\ A = \{\}) = (\forall\ x \in A. \neg P\ x)$
 <proof>

7.2 Cut operators for sets/intervals

7.2.1 Definitions and basic lemmata for cut operators

definition *cut-le* :: 'a::linorder set \Rightarrow 'a \Rightarrow 'a set (infixl $\downarrow \leq$ 100)
 where $I \downarrow \leq t \equiv \{x \in I. x \leq t\}$

definition *cut-less* :: 'a::linorder set \Rightarrow 'a \Rightarrow 'a set (infixl $\downarrow <$ 100)
 where $I \downarrow < t \equiv \{x \in I. x < t\}$

definition *cut-ge* :: 'a::linorder set \Rightarrow 'a \Rightarrow 'a set (infixl $\downarrow \geq$ 100)
 where $I \downarrow \geq t \equiv \{x \in I. t \leq x\}$

definition *cut-greater* :: 'a::linorder set \Rightarrow 'a \Rightarrow 'a set (infixl $\downarrow >$ 100)
 where $I \downarrow > t \equiv \{x \in I. t < x\}$

lemmas *i-cut-defs* =
 cut-le-def cut-less-def

cut-ge-def cut-greater-def

lemma *cut-le-mem-iff*: $x \in I \downarrow \leq t = (x \in I \wedge x \leq t)$
 ⟨proof⟩

lemma *cut-less-mem-iff*: $x \in I \downarrow < t = (x \in I \wedge x < t)$
 ⟨proof⟩

lemma *cut-ge-mem-iff*: $x \in I \downarrow \geq t = (x \in I \wedge t \leq x)$
 ⟨proof⟩

lemma *cut-greater-mem-iff*: $x \in I \downarrow > t = (x \in I \wedge t < x)$
 ⟨proof⟩

lemmas *i-cut-mem-iff* =
cut-le-mem-iff cut-less-mem-iff
cut-ge-mem-iff cut-greater-mem-iff

lemma
cut-leI [intro!]: $x \in I \implies x \leq t \implies x \in I \downarrow \leq t$ **and**
cut-lessI [intro!]: $x \in I \implies x < t \implies x \in I \downarrow < t$ **and**
cut-geI [intro!]: $x \in I \implies x \geq t \implies x \in I \downarrow \geq t$ **and**
cut-greaterI [intro!]: $x \in I \implies x > t \implies x \in I \downarrow > t$
 ⟨proof⟩

lemma
cut-leE [elim!]: $x \in I \downarrow \leq t \implies (x \in I \implies x \leq t \implies P) \implies P$ **and**
cut-lessE [elim!]: $x \in I \downarrow < t \implies (x \in I \implies x < t \implies P) \implies P$ **and**
cut-geE [elim!]: $x \in I \downarrow \geq t \implies (x \in I \implies x \geq t \implies P) \implies P$ **and**
cut-greaterE [elim!]: $x \in I \downarrow > t \implies (x \in I \implies x > t \implies P) \implies P$
 ⟨proof⟩

lemma
cut-less-bound: $n \in I \downarrow < t \implies n < t$ **and**
cut-le-bound: $n \in I \downarrow \leq t \implies n \leq t$ **and**
cut-greater-bound: $n \in i \downarrow > t \implies t < n$ **and**
cut-ge-bound: $n \in i \downarrow \geq t \implies t \leq n$
 ⟨proof⟩

lemmas *i-cut-bound* =
cut-less-bound cut-le-bound
cut-greater-bound cut-ge-bound

lemma
cut-le-Int-conv: $I \downarrow \leq t = I \cap \{..t\}$ **and**
cut-less-Int-conv: $I \downarrow < t = I \cap \{..<t\}$ **and**
cut-ge-Int-conv: $I \downarrow \geq t = I \cap \{t..\}$ **and**
cut-greater-Int-conv: $I \downarrow > t = I \cap \{t<..\}$

<proof>

lemmas *i-cut-Int-conv* =
cut-le-Int-conv cut-less-Int-conv
cut-ge-Int-conv cut-greater-Int-conv

lemma
cut-le-Diff-conv: $I \downarrow \leq t = I - \{t < ..\}$ **and**
cut-less-Diff-conv: $I \downarrow < t = I - \{t ..\}$ **and**
cut-ge-Diff-conv: $I \downarrow \geq t = I - \{.. < t\}$ **and**
cut-greater-Diff-conv: $I \downarrow > t = I - \{..t\}$
<proof>

lemmas *i-cut-Diff-conv* =
cut-le-Diff-conv cut-less-Diff-conv
cut-ge-Diff-conv cut-greater-Diff-conv

7.2.2 Basic results for cut operators

lemma
cut-less-eq-set-restriction-fun': $(\lambda I. I \downarrow < t) = \text{set-restriction-fun } (\lambda x. x < t)$
and
cut-le-eq-set-restriction-fun': $(\lambda I. I \downarrow \leq t) = \text{set-restriction-fun } (\lambda x. x \leq t)$
and
cut-greater-eq-set-restriction-fun': $(\lambda I. I \downarrow > t) = \text{set-restriction-fun } (\lambda x. x > t)$
and
cut-ge-eq-set-restriction-fun': $(\lambda I. I \downarrow \geq t) = \text{set-restriction-fun } (\lambda x. x \geq t)$
<proof>

lemmas *i-cut-eq-set-restriction-fun'* =
cut-less-eq-set-restriction-fun' cut-le-eq-set-restriction-fun'
cut-greater-eq-set-restriction-fun' cut-ge-eq-set-restriction-fun'

lemma
cut-less-eq-set-restriction-fun: $I \downarrow < t = \text{set-restriction-fun } (\lambda x. x < t) I$ **and**
cut-le-eq-set-restriction-fun: $I \downarrow \leq t = \text{set-restriction-fun } (\lambda x. x \leq t) I$ **and**
cut-greater-eq-set-restriction-fun: $I \downarrow > t = \text{set-restriction-fun } (\lambda x. x > t) I$ **and**
cut-ge-eq-set-restriction-fun: $I \downarrow \geq t = \text{set-restriction-fun } (\lambda x. x \geq t) I$
<proof>

lemmas *i-cut-eq-set-restriction-fun* =
cut-less-eq-set-restriction-fun cut-le-eq-set-restriction-fun
cut-greater-eq-set-restriction-fun cut-ge-eq-set-restriction-fun

lemma *i-cut-set-restriction-disj*:
 $\llbracket \text{cut-op} = (\downarrow <) \vee \text{cut-op} = (\downarrow \leq) \vee$
 $\text{cut-op} = (\downarrow >) \vee \text{cut-op} = (\downarrow \geq);$
 $f = (\lambda I. \text{cut-op } I t) \rrbracket \implies \text{set-restriction } f$
<proof>

corollary
i-cut-less-set-restriction: $\text{set-restriction } (\lambda I. I \downarrow < t)$ **and**

i-cut-le-set-restriction: *set-restriction* ($\lambda I. I \downarrow \leq t$) **and**
i-cut-greater-set-restriction: *set-restriction* ($\lambda I. I \downarrow > t$) **and**
i-cut-ge-set-restriction: *set-restriction* ($\lambda I. I \downarrow \geq t$)
 <proof>

lemmas *i-cut-set-restriction* =
i-cut-le-set-restriction i-cut-less-set-restriction
i-cut-ge-set-restriction i-cut-greater-set-restriction

lemma *i-cut-commute-disj*: \llbracket
 $cut-op1 = (\downarrow <) \vee cut-op1 = (\downarrow \leq) \vee$
 $cut-op1 = (\downarrow >) \vee cut-op1 = (\downarrow \geq);$
 $cut-op2 = (\downarrow <) \vee cut-op2 = (\downarrow \leq) \vee$
 $cut-op2 = (\downarrow >) \vee cut-op2 = (\downarrow \geq) \rrbracket \implies$
 $cut-op2 (cut-op1 I t1) t2 = cut-op1 (cut-op2 I t2) t1$
 <proof>

lemma
cut-less-empty: $\{\} \downarrow < t = \{\}$ **and**
cut-le-empty: $\{\} \downarrow \leq t = \{\}$ **and**
cut-greater-empty: $\{\} \downarrow > t = \{\}$ **and**
cut-ge-empty: $\{\} \downarrow \geq t = \{\}$
 <proof>

lemmas *i-cut-empty* =
cut-less-empty cut-le-empty
cut-greater-empty cut-ge-empty

lemma
cut-less-not-empty-imp: $I \downarrow < t \neq \{\} \implies I \neq \{\}$ **and**
cut-le-not-empty-imp: $I \downarrow \leq t \neq \{\} \implies I \neq \{\}$ **and**
cut-greater-not-empty-imp: $I \downarrow > t \neq \{\} \implies I \neq \{\}$ **and**
cut-ge-not-empty-imp: $I \downarrow \geq t \neq \{\} \implies I \neq \{\}$
 <proof>

lemma
cut-less-singleton: $\{a\} \downarrow < t = (if\ a < t\ then\ \{a\}\ else\ \{\})$ **and**
cut-le-singleton: $\{a\} \downarrow \leq t = (if\ a \leq t\ then\ \{a\}\ else\ \{\})$ **and**
cut-greater-singleton: $\{a\} \downarrow > t = (if\ a > t\ then\ \{a\}\ else\ \{\})$ **and**
cut-ge-singleton: $\{a\} \downarrow \geq t = (if\ a \geq t\ then\ \{a\}\ else\ \{\})$
 <proof>

lemmas *i-cut-singleton* =
cut-le-singleton cut-less-singleton
cut-ge-singleton cut-greater-singleton

lemma

cut-less-subset: $I \downarrow < t \subseteq I$ **and**

cut-le-subset: $I \downarrow \leq t \subseteq I$ **and**

cut-greater-subset: $I \downarrow > t \subseteq I$ **and**

cut-ge-subset: $I \downarrow \geq t \subseteq I$

<proof>

lemmas *i-cut-subset* =

cut-less-subset cut-le-subset

cut-greater-subset cut-ge-subset

lemma *i-cut-Un-disj*:

$\llbracket \textit{cut-op} = (\downarrow <) \vee \textit{cut-op} = (\downarrow \leq) \vee$

$\textit{cut-op} = (\downarrow >) \vee \textit{cut-op} = (\downarrow \geq) \rrbracket$

$\implies \textit{cut-op} (A \cup B) t = \textit{cut-op} A t \cup \textit{cut-op} B t$

<proof>

corollary

cut-less-Un: $(A \cup B) \downarrow < t = A \downarrow < t \cup B \downarrow < t$ **and**

cut-le-Un: $(A \cup B) \downarrow \leq t = A \downarrow \leq t \cup B \downarrow \leq t$ **and**

cut-greater-Un: $(A \cup B) \downarrow > t = A \downarrow > t \cup B \downarrow > t$ **and**

cut-ge-Un: $(A \cup B) \downarrow \geq t = A \downarrow \geq t \cup B \downarrow \geq t$

<proof>

lemmas *i-cut-Un* =

cut-less-Un cut-le-Un

cut-greater-Un cut-ge-Un

lemma *i-cut-Int-disj*:

$\llbracket \textit{cut-op} = (\downarrow <) \vee \textit{cut-op} = (\downarrow \leq) \vee$

$\textit{cut-op} = (\downarrow >) \vee \textit{cut-op} = (\downarrow \geq) \rrbracket$

$\implies \textit{cut-op} (A \cap B) t = \textit{cut-op} A t \cap \textit{cut-op} B t$

<proof>

lemma

cut-less-Int: $(A \cap B) \downarrow < t = A \downarrow < t \cap B \downarrow < t$ **and**

cut-le-Int: $(A \cap B) \downarrow \leq t = A \downarrow \leq t \cap B \downarrow \leq t$ **and**

cut-greater-Int: $(A \cap B) \downarrow > t = A \downarrow > t \cap B \downarrow > t$ **and**

cut-ge-Int: $(A \cap B) \downarrow \geq t = A \downarrow \geq t \cap B \downarrow \geq t$

<proof>

lemmas *i-cut-Int* =

cut-less-Int cut-le-Int

cut-greater-Int cut-ge-Int

lemma

cut-less-Int-left: $(A \cap B) \downarrow < t = A \downarrow < t \cap B$ **and**

cut-le-Int-left: $(A \cap B) \downarrow_{\leq} t = A \downarrow_{\leq} t \cap B$ **and**
cut-greater-Int-left: $(A \cap B) \downarrow_{>} t = A \downarrow_{>} t \cap B$ **and**
cut-ge-Int-left: $(A \cap B) \downarrow_{\geq} t = A \downarrow_{\geq} t \cap B$

<proof>

lemmas *i-cut-Int-left* =

cut-less-Int-left cut-le-Int-left
cut-greater-Int-left cut-ge-Int-left

lemma

cut-less-Int-right: $(A \cap B) \downarrow_{<} t = A \cap B \downarrow_{<} t$ **and**
cut-le-Int-right: $(A \cap B) \downarrow_{\leq} t = A \cap B \downarrow_{\leq} t$ **and**
cut-greater-Int-right: $(A \cap B) \downarrow_{>} t = A \cap B \downarrow_{>} t$ **and**
cut-ge-Int-right: $(A \cap B) \downarrow_{\geq} t = A \cap B \downarrow_{\geq} t$

<proof>

lemmas *i-cut-Int-right* =

cut-less-Int-right cut-le-Int-right
cut-greater-Int-right cut-ge-Int-right

lemma *i-cut-Diff-disj*:

$\llbracket \text{cut-op} = (\downarrow_{<}) \vee \text{cut-op} = (\downarrow_{\leq}) \vee$
 $\text{cut-op} = (\downarrow_{>}) \vee \text{cut-op} = (\downarrow_{\geq}) \rrbracket$
 $\implies \text{cut-op} (A - B) t = \text{cut-op} A t - \text{cut-op} B t$

<proof>

corollary

cut-less-Diff: $(A - B) \downarrow_{<} t = A \downarrow_{<} t - B \downarrow_{<} t$ **and**
cut-le-Diff: $(A - B) \downarrow_{\leq} t = A \downarrow_{\leq} t - B \downarrow_{\leq} t$ **and**
cut-greater-Diff: $(A - B) \downarrow_{>} t = A \downarrow_{>} t - B \downarrow_{>} t$ **and**
cut-ge-Diff: $(A - B) \downarrow_{\geq} t = A \downarrow_{\geq} t - B \downarrow_{\geq} t$

<proof>

lemmas *i-cut-Diff* =

cut-less-Diff cut-le-Diff
cut-greater-Diff cut-ge-Diff

lemma *i-cut-subset-mono-disj*:

$\llbracket \text{cut-op} = (\downarrow_{<}) \vee \text{cut-op} = (\downarrow_{\leq}) \vee$
 $\text{cut-op} = (\downarrow_{>}) \vee \text{cut-op} = (\downarrow_{\geq}); A \subseteq B \rrbracket$
 $\implies \text{cut-op} A t \subseteq \text{cut-op} B t$

<proof>

corollary

cut-less-subset-mono: $A \subseteq B \implies A \downarrow_{<} t \subseteq B \downarrow_{<} t$ **and**
cut-le-subset-mono: $A \subseteq B \implies A \downarrow_{\leq} t \subseteq B \downarrow_{\leq} t$ **and**
cut-greater-subset-mono: $A \subseteq B \implies A \downarrow_{>} t \subseteq B \downarrow_{>} t$ **and**
cut-ge-subset-mono: $A \subseteq B \implies A \downarrow_{\geq} t \subseteq B \downarrow_{\geq} t$

<proof>

lemmas *i-cut-subset-mono* =

cut-less-subset-mono cut-le-subset-mono
cut-greater-subset-mono cut-ge-subset-mono

lemma

cut-less-mono: $t \leq t' \implies I \downarrow < t \subseteq I \downarrow < t'$ **and**
cut-greater-mono: $t' \leq t \implies I \downarrow > t \subseteq I \downarrow > t'$ **and**
cut-le-mono: $t \leq t' \implies I \downarrow \leq t \subseteq I \downarrow \leq t'$ **and**
cut-ge-mono: $t' \leq t \implies I \downarrow \geq t \subseteq I \downarrow \geq t'$

<proof>

lemmas *i-cut-mono* =

cut-le-mono cut-less-mono
cut-ge-mono cut-greater-mono

lemma

cut-cut-le: $i \downarrow \leq a \downarrow \leq b = i \downarrow \leq \min a b$ **and**
cut-cut-less: $i \downarrow < a \downarrow < b = i \downarrow < \min a b$ **and**
cut-cut-ge: $i \downarrow \geq a \downarrow \geq b = i \downarrow \geq \max a b$ **and**
cut-cut-greater: $i \downarrow > a \downarrow > b = i \downarrow > \max a b$

<proof>

lemmas *i-cut-cut* =

cut-cut-le cut-cut-less
cut-cut-ge cut-cut-greater

lemma *i-cut-absorb-disj*:

$\llbracket \text{cut-op} = (\downarrow <) \vee \text{cut-op} = (\downarrow \leq) \vee$
 $\text{cut-op} = (\downarrow >) \vee \text{cut-op} = (\downarrow \geq) \rrbracket$
 $\implies \text{cut-op} (\text{cut-op } I t) t = \text{cut-op } I t$

<proof>

corollary

cut-le-absorb: $I \downarrow \leq t \downarrow \leq t = I \downarrow \leq t$ **and**
cut-less-absorb: $I \downarrow < t \downarrow < t = I \downarrow < t$ **and**
cut-ge-absorb: $I \downarrow \geq t \downarrow \geq t = I \downarrow \geq t$ **and**
cut-greater-absorb: $I \downarrow > t \downarrow > t = I \downarrow > t$

<proof>

lemmas *i-cut-absorb* =

cut-le-absorb cut-less-absorb
cut-ge-absorb cut-greater-absorb

lemma

cut-less-0-empty: $I \downarrow < (0::\text{nat}) = \{\}$ **and**
cut-ge-0-all: $I \downarrow \geq (0::\text{nat}) = I$

<proof>

lemma

cut-le-all-iff: $(I \downarrow \leq t = I) = (\forall x \in I. x \leq t)$ **and**
cut-less-all-iff: $(I \downarrow < t = I) = (\forall x \in I. x < t)$ **and**
cut-ge-all-iff: $(I \downarrow \geq t = I) = (\forall x \in I. x \geq t)$ **and**
cut-greater-all-iff: $(I \downarrow > t = I) = (\forall x \in I. x > t)$

*<proof>***lemmas** *i-cut-all-iff* =

cut-le-all-iff *cut-less-all-iff*
cut-ge-all-iff *cut-greater-all-iff*

lemma

cut-le-empty-iff: $(I \downarrow \leq t = \{\}) = (\forall x \in I. t < x)$ **and**
cut-less-empty-iff: $(I \downarrow < t = \{\}) = (\forall x \in I. t \leq x)$ **and**
cut-ge-empty-iff: $(I \downarrow \geq t = \{\}) = (\forall x \in I. x < t)$ **and**
cut-greater-empty-iff: $(I \downarrow > t = \{\}) = (\forall x \in I. x \leq t)$

*<proof>***lemmas** *i-cut-empty-iff* =

cut-le-empty-iff *cut-less-empty-iff*
cut-ge-empty-iff *cut-greater-empty-iff*

lemma

cut-le-not-empty-iff: $(I \downarrow \leq t \neq \{\}) = (\exists x \in I. x \leq t)$ **and**
cut-less-not-empty-iff: $(I \downarrow < t \neq \{\}) = (\exists x \in I. x < t)$ **and**
cut-ge-not-empty-iff: $(I \downarrow \geq t \neq \{\}) = (\exists x \in I. t \leq x)$ **and**
cut-greater-not-empty-iff: $(I \downarrow > t \neq \{\}) = (\exists x \in I. t < x)$

*<proof>***lemmas** *i-cut-not-empty-iff* =

cut-le-not-empty-iff *cut-less-not-empty-iff*
cut-ge-not-empty-iff *cut-greater-not-empty-iff*

lemma *nat-cut-ge-infinite-not-empty*: *infinite* $I \implies I \downarrow \geq (t::nat) \neq \{\}$ *<proof>***lemma** *nat-cut-greater-infinite-not-empty*: *infinite* $I \implies I \downarrow > (t::nat) \neq \{\}$ *<proof>***corollary**

cut-le-not-in-imp: $x \notin I \implies x \notin I \downarrow \leq t$ **and**
cut-less-not-in-imp: $x \notin I \implies x \notin I \downarrow < t$ **and**
cut-ge-not-in-imp: $x \notin I \implies x \notin I \downarrow \geq t$ **and**
cut-greater-not-in-imp: $x \notin I \implies x \notin I \downarrow > t$

*<proof>***lemmas** *i-cut-not-in-imp* =

cut-le-not-in-imp *cut-less-not-in-imp*

cut-ge-not-in-imp cut-greater-not-in-imp

corollary

cut-le-in-imp: $x \in I \downarrow \leq t \implies x \in I$ **and**
cut-less-in-imp: $x \in I \downarrow < t \implies x \in I$ **and**
cut-ge-in-imp: $x \in I \downarrow \geq t \implies x \in I$ **and**
cut-greater-in-imp: $x \in I \downarrow > t \implies x \in I$

<proof>

lemmas *i-cut-in-imp* =

cut-le-in-imp cut-less-in-imp
cut-ge-in-imp cut-greater-in-imp

lemma *Collect-minI-cut:* $\llbracket k \in I; P (k::('a::wellorder)) \rrbracket \implies \exists x \in I. P x \wedge (\forall y \in (I \downarrow < x). \neg P y)$

<proof>

corollary *Collect-minI-ex-cut:* $\exists k \in I. P (k::('a::wellorder)) \implies \exists x \in I. P x \wedge (\forall y \in (I \downarrow < x). \neg P y)$

<proof>

corollary *Collect-minI-ex2-cut:* $\{k \in I. P (k::('a::wellorder))\} \neq \{\} \implies \exists x \in I. P x \wedge (\forall y \in (I \downarrow < x). \neg P y)$

<proof>

lemma *cut-le-cut-greater-ident:* $t2 \leq t1 \implies I \downarrow \leq t1 \cup I \downarrow > t2 = I$

<proof>

lemma *cut-less-cut-ge-ident:* $t2 \leq t1 \implies I \downarrow < t1 \cup I \downarrow \geq t2 = I$

<proof>

lemma *cut-le-cut-ge-ident:* $t2 \leq t1 \implies I \downarrow \leq t1 \cup I \downarrow \geq t2 = I$

<proof>

lemma *cut-less-cut-greater-ident:*

$\llbracket t2 \leq t1; I \cap \{t1..t2\} = \{\} \rrbracket \implies I \downarrow < t1 \cup I \downarrow > t2 = I$

<proof>

corollary *cut-less-cut-greater-ident':*

$t \notin I \implies I \downarrow < t \cup I \downarrow > t = I$

<proof>

lemma *insert-eq-cut-less-cut-greater:* $insert\ n\ I = I \downarrow < n \cup \{n\} \cup I \downarrow > n$

<proof>

7.2.3 Relations between cut operators

lemma *insert-Int-conv-if*: $A \cap (\text{insert } x B) =$
if $x \in A$ *then* $\text{insert } x (A \cap B)$ *else* $A \cap B$
 ⟨proof⟩

lemma *cut-le-less-conv-if*: $I \downarrow \leq t =$
if $t \in I$ *then* $\text{insert } t (I \downarrow < t)$ *else* $(I \downarrow < t)$
 ⟨proof⟩

lemma *cut-le-less-conv*: $I \downarrow \leq t = (\{t\} \cap I) \cup (I \downarrow < t)$
 ⟨proof⟩

lemma *cut-less-le-conv*: $I \downarrow < t = (I \downarrow \leq t) - \{t\}$
 ⟨proof⟩

lemma *cut-less-le-conv-if*: $I \downarrow < t =$
if $t \in I$ *then* $(I \downarrow \leq t) - \{t\}$ *else* $(I \downarrow \leq t)$
 ⟨proof⟩

lemma *cut-ge-greater-conv-if*: $I \downarrow \geq t =$
if $t \in I$ *then* $\text{insert } t (I \downarrow > t)$ *else* $(I \downarrow > t)$
 ⟨proof⟩

lemma *cut-ge-greater-conv*: $I \downarrow \geq t = (\{t\} \cap I) \cup (I \downarrow > t)$
 ⟨proof⟩

lemma *cut-greater-ge-conv*: $I \downarrow > t = (I \downarrow \geq t) - \{t\}$
 ⟨proof⟩

lemma *cut-greater-ge-conv-if*: $I \downarrow > t =$
if $t \in I$ *then* $(I \downarrow \geq t) - \{t\}$ *else* $(I \downarrow \geq t)$
 ⟨proof⟩

lemma *nat-cut-le-less-conv*: $I \downarrow \leq t = I \downarrow < \text{Suc } t$
 ⟨proof⟩

lemma *nat-cut-less-le-conv*: $0 < t \implies I \downarrow < t = I \downarrow \leq (t - \text{Suc } 0)$
 ⟨proof⟩

lemma *nat-cut-ge-greater-conv*: $I \downarrow \geq \text{Suc } t = I \downarrow > t$
 ⟨proof⟩

lemma *nat-cut-greater-ge-conv*: $0 < t \implies I \downarrow > (t - \text{Suc } 0) = I \downarrow \geq t$
 ⟨proof⟩

7.2.4 Function images with cut operators

lemma *cut-less-image*:

$$\llbracket \text{strict-mono-on } f \ A; I \subseteq A; n \in A \rrbracket \implies$$

$$(f \ ' \ I) \downarrow < f \ n = f \ ' \ (I \downarrow < n)$$
 <proof>

lemma *cut-le-image*:

$$\llbracket \text{strict-mono-on } f \ A; I \subseteq A; n \in A \rrbracket \implies$$

$$(f \ ' \ I) \downarrow \leq f \ n = f \ ' \ (I \downarrow \leq n)$$
 <proof>

lemma *cut-greater-image*:

$$\llbracket \text{strict-mono-on } f \ A; I \subseteq A; n \in A \rrbracket \implies$$

$$(f \ ' \ I) \downarrow > f \ n = f \ ' \ (I \downarrow > n)$$
 <proof>

lemma *cut-ge-image*:

$$\llbracket \text{strict-mono-on } f \ A; I \subseteq A; n \in A \rrbracket \implies$$

$$(f \ ' \ I) \downarrow \geq f \ n = f \ ' \ (I \downarrow \geq n)$$
 <proof>

lemmas *i-cut-image =*

cut-le-image cut-less-image
cut-ge-image cut-greater-image

7.2.5 Finiteness and cardinality with cut operators

lemma

cut-le-finite: finite I \implies finite (I $\downarrow \leq t$) and
cut-less-finite: finite I \implies finite (I $\downarrow < t$) and
cut-ge-finite: finite I \implies finite (I $\downarrow \geq t$) and
cut-greater-finite: finite I \implies finite (I $\downarrow > t$)
 <proof>

lemma *nat-cut-le-finite: finite (I $\downarrow \leq (t::nat)$)*

<proof>

lemma *nat-cut-less-finite: finite (I $\downarrow < (t::nat)$)*

<proof>

lemma *nat-cut-ge-finite-iff: finite (I $\downarrow \geq (t::nat)$) = finite I*

<proof>

lemma *nat-cut-greater-finite-iff: finite (I $\downarrow > (t::nat)$) = finite I*

<proof>

lemma

cut-le-card: finite I \implies card (I $\downarrow \leq t$) \leq card I and
cut-less-card: finite I \implies card (I $\downarrow < t$) \leq card I and
cut-ge-card: finite I \implies card (I $\downarrow \geq t$) \leq card I and
cut-greater-card: finite I \implies card (I $\downarrow > t$) \leq card I

<proof>

lemma *nat-cut-greater-card*: $\text{card } (I \downarrow > (t::\text{nat})) \leq \text{card } I$
<proof>

lemma *nat-cut-ge-card*: $\text{card } (I \downarrow \geq (t::\text{nat})) \leq \text{card } I$
<proof>

7.2.6 Cutting a set at *Min* or *Max* element

lemma *cut-greater-Min-eq-Diff*: $I \downarrow > (i\text{Min } I) = I - \{i\text{Min } I\}$
<proof>

lemma *cut-less-Max-eq-Diff*: $\text{finite } I \implies I \downarrow < (\text{Max } I) = I - \{\text{Max } I\}$
<proof>

lemma *cut-le-Min-empty*: $t < i\text{Min } I \implies I \downarrow \leq t = \{\}$
<proof>

lemma *cut-less-Min-empty*: $t \leq i\text{Min } I \implies I \downarrow < t = \{\}$
<proof>

lemma *cut-le-Min-not-empty*: $\llbracket I \neq \{\}; i\text{Min } I \leq t \rrbracket \implies I \downarrow \leq t \neq \{\}$
<proof>

lemma *cut-less-Min-not-empty*: $\llbracket I \neq \{\}; i\text{Min } I < t \rrbracket \implies I \downarrow < t \neq \{\}$
<proof>

lemma *cut-ge-Min-all*: $t \leq i\text{Min } I \implies I \downarrow \geq t = I$
<proof>

lemma *cut-greater-Min-all*: $t < i\text{Min } I \implies I \downarrow > t = I$
<proof>

lemmas *i-cut-min-empty* =
cut-le-Min-empty
cut-less-Min-empty
cut-le-Min-not-empty
cut-less-Min-not-empty

lemmas *i-cut-min-all* =
cut-ge-Min-all
cut-greater-Min-all

lemma *cut-ge-Max-empty*: $\text{finite } I \implies \text{Max } I < t \implies I \downarrow \geq t = \{\}$
<proof>

lemma *cut-greater-Max-empty*: $\text{finite } I \implies \text{Max } I \leq t \implies I \downarrow > t = \{\}$
<proof>

lemma *cut-ge-Max-not-empty*: $\llbracket I \neq \{\}; \text{finite } I; t \leq \text{Max } I \rrbracket \implies I \downarrow \geq t \neq \{\}$

<proof>

lemma *cut-greater-Max-not-empty*: $\llbracket I \neq \{\}; \text{finite } I; t < \text{Max } I \rrbracket \implies I \downarrow > t \neq \{\}$
<proof>

lemma *cut-le-Max-all*: $\text{finite } I \implies \text{Max } I \leq t \implies I \downarrow \leq t = I$
<proof>

lemma *cut-less-Max-all*: $\text{finite } I \implies \text{Max } I < t \implies I \downarrow < t = I$
<proof>

lemmas *i-cut-max-empty* =
cut-ge-Max-empty
cut-greater-Max-empty
cut-ge-Max-not-empty
cut-greater-Max-not-empty

lemmas *i-cut-max-all* =
cut-le-Max-all
cut-less-Max-all

lemma *cut-less-Max-less*:
 $\llbracket \text{finite } (I \downarrow < t); I \downarrow < t \neq \{\} \rrbracket \implies \text{Max } (I \downarrow < t) < t$
<proof>

lemma *cut-le-Max-le*:
 $\llbracket \text{finite } (I \downarrow \leq t); I \downarrow \leq t \neq \{\} \rrbracket \implies \text{Max } (I \downarrow \leq t) \leq t$
<proof>

lemma *nat-cut-less-Max-less*:
 $I \downarrow < t \neq \{\} \implies \text{Max } (I \downarrow < t) < (t::\text{nat})$
<proof>

lemma *nat-cut-le-Max-le*:
 $I \downarrow \leq t \neq \{\} \implies \text{Max } (I \downarrow \leq t) \leq (t::\text{nat})$
<proof>

lemma *cut-greater-Min-greater*:
 $I \downarrow > t \neq \{\} \implies i\text{Min } (I \downarrow > t) > t$
<proof>

lemma *cut-ge-Min-greater*:
 $I \downarrow \geq t \neq \{\} \implies i\text{Min } (I \downarrow \geq t) \geq t$
<proof>

lemma *cut-less-Min-eq*: $I \downarrow < t \neq \{\} \implies i\text{Min } (I \downarrow < t) = i\text{Min } I$
<proof>

lemma *cut-le-Min-eq*: $I \downarrow \leq t \neq \{\} \implies i\text{Min } (I \downarrow \leq t) = i\text{Min } I$
<proof>

lemma *cut-ge-Max-eq*: $\llbracket \text{finite } I; I \downarrow \geq t \neq \{\} \rrbracket \implies \text{Max } (I \downarrow \geq t) = \text{Max } I$
 ⟨proof⟩

lemma *cut-greater-Max-eq*: $\llbracket \text{finite } I; I \downarrow > t \neq \{\} \rrbracket \implies \text{Max } (I \downarrow > t) = \text{Max } I$
 ⟨proof⟩

7.2.7 Cut operators with intervals from SetInterval

lemma

UNIV-cut-le: $UNIV \downarrow \leq t = \{..t\}$ **and**
UNIV-cut-less: $UNIV \downarrow < t = \{..<t\}$ **and**
UNIV-cut-ge: $UNIV \downarrow \geq t = \{t.. \}$ **and**
UNIV-cut-greater: $UNIV \downarrow > t = \{t<.. \}$
 ⟨proof⟩

lemma

lessThan-cut-le: $\{..<n\} \downarrow \leq t = (\text{if } n \leq t \text{ then } \{..<n\} \text{ else } \{..t\})$ **and**
lessThan-cut-less: $\{..<n\} \downarrow < t = (\text{if } n \leq t \text{ then } \{..<n\} \text{ else } \{..<t\})$ **and**
lessThan-cut-ge: $\{..<n\} \downarrow \geq t = \{t..<n\}$ **and**
lessThan-cut-greater: $\{..<n\} \downarrow > t = \{t<..<n\}$ **and**
atMost-cut-le: $\{..n\} \downarrow \leq t = (\text{if } n \leq t \text{ then } \{..n\} \text{ else } \{..t\})$ **and**
atMost-cut-less: $\{..n\} \downarrow < t = (\text{if } n < t \text{ then } \{..n\} \text{ else } \{..<t\})$ **and**
atMost-cut-ge: $\{..n\} \downarrow \geq t = \{t..n\}$ **and**
atMost-cut-greager: $\{..n\} \downarrow > t = \{t<..n\}$ **and**
greaterThan-cut-le: $\{n<.. \} \downarrow \leq t = \{n<..t\}$ **and**
greaterThan-cut-less: $\{n<.. \} \downarrow < t = \{n<..<t\}$ **and**
greaterThan-cut-ge: $\{n<.. \} \downarrow \geq t = (\text{if } t \leq n \text{ then } \{n<.. \} \text{ else } \{t.. \})$ **and**
greaterThan-cut-greater: $\{n<.. \} \downarrow > t = (\text{if } t \leq n \text{ then } \{n<.. \} \text{ else } \{t<.. \})$ **and**
atLeast-cut-le: $\{n.. \} \downarrow \leq t = \{n..t\}$ **and**
atLeast-cut-less: $\{n.. \} \downarrow < t = \{n..<t\}$ **and**
atLeast-cut-ge: $\{n.. \} \downarrow \geq t = (\text{if } t \leq n \text{ then } \{n.. \} \text{ else } \{t.. \})$ **and**
atLeast-cut-greater: $\{n.. \} \downarrow > t = (\text{if } t \leq n \text{ then } \{n.. \} \text{ else } \{t.. \})$
 ⟨proof⟩

lemma

greaterThanLessThan-cut-le: $\{m<..<n\} \downarrow \leq t = (\text{if } n \leq t \text{ then } \{m<..<n\} \text{ else } \{m<..t\})$ **and**
greaterThanLessThan-cut-less: $\{m<..<n\} \downarrow < t = (\text{if } n \leq t \text{ then } \{m<..<n\} \text{ else } \{m<..<t\})$ **and**
greaterThanLessThan-cut-ge: $\{m<..<n\} \downarrow \geq t = (\text{if } t \leq m \text{ then } \{m<..<n\} \text{ else } \{t..<n\})$ **and**
greaterThanLessThan-cut-greater: $\{m<..<n\} \downarrow > t = (\text{if } t \leq m \text{ then } \{m<..<n\} \text{ else } \{t<..<n\})$ **and**
atLeastLessThan-cut-le: $\{m..<n\} \downarrow \leq t = (\text{if } n \leq t \text{ then } \{m..<n\} \text{ else } \{m..t\})$
and
atLeastLessThan-cut-less: $\{m..<n\} \downarrow < t = (\text{if } n \leq t \text{ then } \{m..<n\} \text{ else } \{m..<t\})$
and
atLeastLessThan-cut-ge: $\{m..<n\} \downarrow \geq t = (\text{if } t \leq m \text{ then } \{m..<n\} \text{ else } \{t..<n\})$

and

atLeastLessThan-cut-greater: $\{m..<n\} \downarrow > t = (\text{if } t < m \text{ then } \{m..<n\} \text{ else } \{t<..<n\})$ **and**

greaterThanAtMost-cut-le: $\{m<..n\} \downarrow \leq t = (\text{if } n \leq t \text{ then } \{m<..n\} \text{ else } \{m<..t\})$ **and**

greaterThanAtMost-cut-less: $\{m<..n\} \downarrow < t = (\text{if } n < t \text{ then } \{m<..n\} \text{ else } \{m<..<t\})$ **and**

greaterThanAtMost-cut-ge: $\{m<..n\} \downarrow \geq t = (\text{if } t \leq m \text{ then } \{m<..n\} \text{ else } \{t..n\})$ **and**

greaterThanAtMost-cut-greater: $\{m<..n\} \downarrow > t = (\text{if } t \leq m \text{ then } \{m<..n\} \text{ else } \{t<..n\})$ **and**

atLeastAtMost-cut-le: $\{m..n\} \downarrow \leq t = (\text{if } n \leq t \text{ then } \{m..n\} \text{ else } \{m..t\})$ **and**

atLeastAtMost-cut-less: $\{m..n\} \downarrow < t = (\text{if } n < t \text{ then } \{m..n\} \text{ else } \{m..<t\})$

and

atLeastAtMost-cut-ge: $\{m..n\} \downarrow \geq t = (\text{if } t \leq m \text{ then } \{m..n\} \text{ else } \{t..n\})$ **and**

atLeastAtMost-cut-greater: $\{m..n\} \downarrow > t = (\text{if } t < m \text{ then } \{m..n\} \text{ else } \{t<..n\})$

$\langle \text{proof} \rangle$

7.2.8 Mirroring finite natural sets between their *Min* and *Max* element

Mirroring a number at the middle of the interval $\text{min } l \text{ r}.. \text{max } l \text{ r}$

Mirroring a single element n between the interval boundaries l and r

definition *nat-mirror* :: $\text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat}$

where *nat-mirror* $n \ l \ r \equiv l + r - n$

lemma *nat-mirror-commute*: $\text{nat-mirror } n \ l \ r = \text{nat-mirror } n \ r \ l$

$\langle \text{proof} \rangle$

lemma *nat-mirror-inj-on*: $\text{inj-on } (\lambda x. \text{nat-mirror } x \ l \ r) \ \{..l + r\}$

$\langle \text{proof} \rangle$

lemma *nat-mirror-nat-mirror-ident*:

$n \leq l + r \implies \text{nat-mirror } (\text{nat-mirror } n \ l \ r) \ l \ r = n$

$\langle \text{proof} \rangle$

lemma *nat-mirror-add*:

$\text{nat-mirror } (n + k) \ l \ r = (\text{nat-mirror } n \ l \ r) - k$

$\langle \text{proof} \rangle$

lemma *nat-mirror-diff*:

$\llbracket k \leq n; n \leq l + r \rrbracket \implies$

$\text{nat-mirror } (n - k) \ l \ r = (\text{nat-mirror } n \ l \ r) + k$

$\langle \text{proof} \rangle$

lemma *nat-mirror-le*: $a \leq b \implies \text{nat-mirror } b \ l \ r \leq \text{nat-mirror } a \ l \ r$

$\langle \text{proof} \rangle$

lemma *nat-mirror-le-conv*:

$a \leq l + r \implies (\text{nat-mirror } b \ l \ r \leq \text{nat-mirror } a \ l \ r) = (a \leq b)$
 ⟨proof⟩

lemma *nat-mirror-less*:

$\llbracket a < b; a < l + r \rrbracket \implies$
 $\text{nat-mirror } b \ l \ r < \text{nat-mirror } a \ l \ r$
 ⟨proof⟩

lemma *nat-mirror-less-imp-less*:

$\text{nat-mirror } b \ l \ r < \text{nat-mirror } a \ l \ r \implies a < b$
 ⟨proof⟩

lemma *nat-mirror-less-conv*:

$a < l + r \implies (\text{nat-mirror } b \ l \ r < \text{nat-mirror } a \ l \ r) = (a < b)$
 ⟨proof⟩

lemma *nat-mirror-eq-conv*:

$\llbracket a \leq l + r; b \leq l + r \rrbracket \implies$
 $(\text{nat-mirror } a \ l \ r = \text{nat-mirror } b \ l \ r) = (a = b)$
 ⟨proof⟩

Mirroring a single element n between the interval boundaries of I

definition

$\text{mirror-elem} :: \text{nat} \Rightarrow \text{nat set} \Rightarrow \text{nat}$

where

$\text{mirror-elem } n \ I \equiv \text{nat-mirror } n \ (\text{iMin } I) \ (\text{Max } I)$

lemma *mirror-elem-inj-on*: $\text{finite } I \implies \text{inj-on } (\lambda x. \text{mirror-elem } x \ I) \ I$

⟨proof⟩

lemma *mirror-elem-add*:

$\text{finite } I \implies \text{mirror-elem } (n + k) \ I = \text{mirror-elem } n \ I - k$
 ⟨proof⟩

lemma *mirror-elem-diff*:

$\llbracket \text{finite } I; k \leq n; n \in I \rrbracket \implies \text{mirror-elem } (n - k) \ I = \text{mirror-elem } n \ I + k$
 ⟨proof⟩

lemma *mirror-elem-Min*:

$\llbracket \text{finite } I; I \neq \{\} \rrbracket \implies \text{mirror-elem } (\text{iMin } I) \ I = \text{Max } I$
 ⟨proof⟩

lemma *mirror-elem-Max*:

$\llbracket \text{finite } I; I \neq \{\} \rrbracket \implies \text{mirror-elem } (\text{Max } I) \ I = \text{iMin } I$
 ⟨proof⟩

lemma *mirror-elem-mirror-elem-ident*:

$\llbracket \text{finite } I; n \leq \text{iMin } I + \text{Max } I \rrbracket \implies \text{mirror-elem } (\text{mirror-elem } n \ I) \ I = n$
 ⟨proof⟩

lemma *mirror-elem-le-conv*:

$\llbracket \text{finite } I; a \in I; b \in I \rrbracket \implies$
 $(\text{mirror-elem } b \ I \leq \text{mirror-elem } a \ I) = (a \leq b)$
 ⟨proof⟩

lemma *mirror-elem-less-conv*:

$\llbracket \text{finite } I; a \in I; b \in I \rrbracket \implies$
 $(\text{mirror-elem } b \ I < \text{mirror-elem } a \ I) = (a < b)$
 <proof>

lemma *mirror-elem-eq-conv*:

$\llbracket a \leq iMin \ I + Max \ I; b \leq iMin \ I + Max \ I \rrbracket \implies$
 $(\text{mirror-elem } a \ I = \text{mirror-elem } b \ I) = (a = b)$
 <proof>

lemma *mirror-elem-eq-conv'*:

$\llbracket \text{finite } I; a \in I; b \in I \rrbracket \implies (\text{mirror-elem } a \ I = \text{mirror-elem } b \ I) = (a = b)$
 <proof>

definition

imirror-bounds :: *nat set* \Rightarrow *nat* \Rightarrow *nat* \Rightarrow *nat set*

where

imirror-bounds *I l r* $\equiv (\lambda x. \text{nat-mirror } x \ l \ r) \ ' I$

Mirroring all elements between the interval boundaries of I

definition

imirror :: *nat set* \Rightarrow *nat set*

where

imirror *I* $\equiv (\lambda x. iMin \ I + Max \ I - x) \ ' I$

lemma *imirror-eq-nat-mirror-image*:

imirror *I* = $(\lambda x. \text{nat-mirror } x \ (iMin \ I) \ (Max \ I)) \ ' I$
 <proof>

lemma *imirror-eq-mirror-elem-image*:

imirror *I* = $(\lambda x. \text{mirror-elem } x \ I) \ ' I$
 <proof>

lemma *imirror-eq-imirror-bounds*:

imirror *I* = *imirror-bounds* *I* $(iMin \ I) \ (Max \ I)$
 <proof>

lemma *imirror-empty*: *imirror* {} = {}

<proof>

lemma *imirror-is-empty*: (*imirror* *I* = {}) = (*I* = {})

<proof>

lemma *imirror-not-empty*: *I* \neq {} \implies *imirror* *I* \neq {}

<proof>

lemma *imirror-singleton*: *imirror* {*a*} = {*a*}

<proof>

lemma *imirror-finite*: $\text{finite } I \implies \text{finite } (\text{imirror } I)$
 ⟨proof⟩

lemma *imirror-bounds-iMin*:
 $\llbracket \text{finite } I; I \neq \{\}; i\text{Min } I \leq l + r \rrbracket \implies$
 $i\text{Min } (\text{imirror-bounds } I \text{ } l \text{ } r) = l + r - \text{Max } I$
 ⟨proof⟩

lemma *imirror-bounds-Max*:
 $\llbracket \text{finite } I; I \neq \{\}; \text{Max } I \leq l + r \rrbracket \implies$
 $\text{Max } (\text{imirror-bounds } I \text{ } l \text{ } r) = l + r - i\text{Min } I$
 ⟨proof⟩

lemma *imirror-iMin*: $\text{finite } I \implies i\text{Min } (\text{imirror } I) = i\text{Min } I$
 ⟨proof⟩

lemma *imirror-Max*: $\text{finite } I \implies \text{Max } (\text{imirror } I) = \text{Max } I$
 ⟨proof⟩

corollary *imirror-iMin-Max*: $\llbracket \text{finite } I; n \in \text{imirror } I \rrbracket \implies i\text{Min } I \leq n \wedge n \leq$
 $\text{Max } I$
 ⟨proof⟩

lemma *imirror-bounds-iff*:
 $(n \in \text{imirror-bounds } I \text{ } l \text{ } r) = (\exists x \in I. n = l + r - x)$
 ⟨proof⟩

lemma *imirror-iff*: $(n \in \text{imirror } I) = (\exists x \in I. n = i\text{Min } I + \text{Max } I - x)$
 ⟨proof⟩

lemma *imirror-bounds-imirror-bounds-ident*:
 $\llbracket \text{finite } I; \text{Max } I \leq l + r \rrbracket \implies$
 $\text{imirror-bounds } (\text{imirror-bounds } I \text{ } l \text{ } r) \text{ } l \text{ } r = I$
 ⟨proof⟩

lemma *imirror-imirror-ident*: $\text{finite } I \implies \text{imirror } (\text{imirror } I) = I$
 ⟨proof⟩

lemma *mirror-elem-imirror*:
 $\text{finite } I \implies \text{mirror-elem } t \text{ } (\text{imirror } I) = \text{mirror-elem } t \text{ } I$
 ⟨proof⟩

lemma *imirror-card*: $\text{finite } I \implies \text{card } (\text{imirror } I) = \text{card } I$
 ⟨proof⟩

lemma *imirror-bounds-elem-conv*:
 $\llbracket \text{finite } I; n \leq l + r; \text{Max } I \leq l + r \rrbracket \implies$

$((\text{nat-mirror } n \ l \ r) \in \text{imirror-bounds } I \ l \ r) = (n \in I)$
 <proof>

lemma *imirror-mem-conv*:

$\llbracket \text{finite } I; n \leq i\text{Min } I + \text{Max } I \rrbracket \implies ((\text{mirror-elem } n \ I) \in \text{imirror } I) = (n \in I)$
 <proof>

corollary *in-imp-mirror-elem-in*:

$\llbracket \text{finite } I; n \in I \rrbracket \implies (\text{mirror-elem } n \ I) \in \text{imirror } I$
 <proof>

lemma *imirror-cut-less*:

$\text{finite } I \implies$
 $\text{imirror } I \downarrow < (\text{mirror-elem } t \ I) =$
 $\text{imirror-bounds } (I \downarrow > t) (i\text{Min } I) (\text{Max } I)$
 <proof>

lemma *imirror-cut-le*:

$\llbracket \text{finite } I; t \leq i\text{Min } I + \text{Max } I \rrbracket \implies$
 $\text{imirror } I \downarrow \leq (\text{mirror-elem } t \ I) =$
 $\text{imirror-bounds } (I \downarrow \geq t) (i\text{Min } I) (\text{Max } I)$
 <proof>

lemma *imirror-cut-ge*:

$\text{finite } I \implies$
 $\text{imirror } I \downarrow \geq (\text{mirror-elem } t \ I) =$
 $\text{imirror-bounds } (I \downarrow \leq t) (i\text{Min } I) (\text{Max } I)$
 (is ?P \implies ?lhs I = ?rhs I t)
 <proof>

lemma *imirror-cut-greater*: $\llbracket \text{finite } I; t \leq i\text{Min } I + \text{Max } I \rrbracket \implies$

$\text{imirror } I \downarrow > (\text{mirror-elem } t \ I) =$
 $\text{imirror-bounds } (I \downarrow < t) (i\text{Min } I) (\text{Max } I)$
 <proof>

lemmas *imirror-cut =*

imirror-cut-less imirror-cut-ge
imirror-cut-le imirror-cut-greater

corollary *imirror-cut-le'*:

$\llbracket \text{finite } I; t \in I \rrbracket \implies$
 $\text{imirror } I \downarrow \leq \text{mirror-elem } t \ I =$
 $\text{imirror-bounds } (I \downarrow \geq t) (i\text{Min } I) (\text{Max } I)$
 <proof>

corollary *imirror-cut-greater'*:

$\llbracket \text{finite } I; t \in I \rrbracket \implies$
 $\text{imirror } I \downarrow > \text{mirror-elem } t \ I =$
 $\text{imirror-bounds } (I \downarrow < t) (i\text{Min } I) (\text{Max } I)$

<proof>

lemmas *imirror-cut'* =
imirror-cut-le' *imirror-cut-greater'*

lemma *imirror-bounds-Un*:
imirror-bounds ($A \cup B$) l r =
imirror-bounds A l $r \cup$ *imirror-bounds* B l r

<proof>

lemma *imirror-bounds-Int*:
 $\llbracket A \subseteq \{..l + r\}; B \subseteq \{..l + r\} \rrbracket \implies$
imirror-bounds ($A \cap B$) l r =
imirror-bounds A l $r \cap$ *imirror-bounds* B l r

<proof>

end

8 Stepping through sets of natural numbers

theory *SetIntervalStep*
imports *SetIntervalCut*
begin

8.1 Function *inext* and *iprev* for stepping through natural sets

definition *inext* :: $\text{nat} \Rightarrow \text{nat set} \Rightarrow \text{nat}$

where

inext n $I \equiv$ (
 if ($n \in I \wedge (I \downarrow > n \neq \{\})$)
 then *iMin* ($I \downarrow > n$)
 else n)

definition *iprev* :: $\text{nat} \Rightarrow \text{nat set} \Rightarrow \text{nat}$

where

iprev n $I \equiv$ (
 if ($n \in I \wedge (I \downarrow < n \neq \{\})$)
 then *Max* ($I \downarrow < n$)
 else n)

inext and *iprev* can be viewed as generalisations of *Suc* and *prev*

lemma *inext-UNIV*: *inext* n *UNIV* = *Suc* n

<proof>

lemma *iprev-UNIV*: *iprev* n *UNIV* = $n - \text{Suc } 0$

<proof>

lemma *inext-empty*: *inext* n $\{\}$ = n

<proof>

lemma *iprev-empty*: *iprev* n $\{\}$ = n

<proof>

lemma *not-in-inext-fix*: $n \notin I \implies \text{inext } n \ I = n$

<proof>

lemma *not-in-iprev-fix*: $n \notin I \implies \text{iprev } n \ I = n$

<proof>

lemma *inext-all-le-fix*: $\forall x \in I. x \leq n \implies \text{inext } n \ I = n$

<proof>

lemma *iprev-all-ge-fix*: $\forall x \in I. n \leq x \implies \text{iprev } n \ I = n$

<proof>

lemma *inext-Max*: $\text{finite } I \implies \text{inext } (\text{Max } I) \ I = \text{Max } I$

<proof>

lemma *iprev-iMin*: $\text{iprev } (i\text{Min } I) \ I = i\text{Min } I$

<proof>

lemma *inext-ge-Max*: $\llbracket \text{finite } I; \text{Max } I \leq n \rrbracket \implies \text{inext } n \ I = n$

<proof>

lemma *iprev-le-iMin*: $n \leq i\text{Min } I \implies \text{iprev } n \ I = n$

<proof>

lemma *inext-singleton*: $\text{inext } n \ \{a\} = n$

<proof>

lemma *iprev-singleton*: $\text{iprev } n \ \{a\} = n$

<proof>

lemma *inext-closed*: $n \in I \implies \text{inext } n \ I \in I$

<proof>

lemma *iprev-closed*: $n \in I \implies \text{iprev } n \ I \in I$

<proof>

lemma *inext-in-imp-in*: $\text{inext } n \ I \in I \implies n \in I$

<proof>

lemma *inext-in-iff*: $(\text{inext } n \ I \in I) = (n \in I)$

<proof>

lemma *subset-inext-closed*: $\llbracket n \in B; A \subseteq B \rrbracket \implies \text{inext } n \ A \in B$

<proof>

lemma *subset-inext-in-imp-in*: $\llbracket \text{inext } n \ A \in B; A \subseteq B \rrbracket \implies n \in B$

<proof>

lemma *subset-inext-in-iff*: $A \subseteq B \implies (\text{inext } n \ A \in B) = (n \in B)$

<proof>

lemma *iprev-in-imp-in*: $iprev\ n\ I \in I \implies n \in I$
 ⟨proof⟩

lemma *iprev-in-iff*: $(iprev\ n\ I \in I) = (n \in I)$
 ⟨proof⟩

lemma *subset-iprev-closed*: $\llbracket n \in B; A \subseteq B \rrbracket \implies iprev\ n\ A \in B$
 ⟨proof⟩

lemma *subset-iprev-in-imp-in*: $\llbracket iprev\ n\ A \in B; A \subseteq B \rrbracket \implies n \in B$
 ⟨proof⟩

lemma *subset-iprev-in-iff*: $A \subseteq B \implies (iprev\ n\ A \in B) = (n \in B)$
 ⟨proof⟩

lemma *inext-mono*: $n \leq inext\ n\ I$
 ⟨proof⟩

corollary *inext-neq-imp-less*: $n \neq inext\ n\ I \implies n < inext\ n\ I$
 ⟨proof⟩

lemma *inext-mono2*: $\llbracket n \in I; \exists x \in I. n < x \rrbracket \implies n < inext\ n\ I$
 ⟨proof⟩

lemma *inext-mono2-infin*: $\llbracket n \in I; infinite\ I \rrbracket \implies n < inext\ n\ I$
 ⟨proof⟩

lemma *inext-mono2-fin*: $\llbracket n \in I; finite\ I; n \neq Max\ I \rrbracket \implies n < inext\ n\ I$
 ⟨proof⟩

lemma *inext-mono2-infin-fin*:
 $\llbracket n \in I; n \neq Max\ I \vee infinite\ I \rrbracket \implies n < inext\ n\ I$
 ⟨proof⟩

lemma *inext-neq-iMin*: $\exists x \in I. n < x \implies inext\ n\ I \neq iMin\ I$
 ⟨proof⟩

lemma *inext-neq-iMin-infin*: $infinite\ I \implies inext\ n\ I \neq iMin\ I$
 ⟨proof⟩

lemma *Max-le-iMin-imp-singleton*: $\llbracket finite\ I; I \neq \{\}; Max\ I \leq iMin\ I \rrbracket \implies I = \{iMin\ I\}$
 ⟨proof⟩

lemma *inext-neq-iMin-not-singleton*:
 $\llbracket I \neq \{\}; \neg(\exists a. I = \{a\}) \rrbracket \implies inext\ n\ I \neq iMin\ I$
 ⟨proof⟩

corollary *inext-neq-iMin-not-card-1*:
 $\llbracket I \neq \{\}; card\ I \neq Suc\ 0 \rrbracket \implies inext\ n\ I \neq iMin\ I$

$\langle proof \rangle$

lemma *inext-neq-imp-Max*: $n \neq \text{inext } n \ I \implies n < \text{Max } I \vee \text{infinite } I$
 $\langle proof \rangle$

lemma *inext-less-conv*: $(n \in I \wedge (n < \text{Max } I \vee \text{infinite } I)) = (n < \text{inext } n \ I)$
 $\langle proof \rangle$

lemma *inext-min-step*: $\llbracket n < k; k < \text{inext } n \ I \rrbracket \implies k \notin I$
 $\langle proof \rangle$

corollary *inext-min-step2*: $\neg(\exists k \in I. n < k \wedge k < \text{inext } n \ I)$
 $\langle proof \rangle$

lemma *min-step-inext*[*rule-format*]:
 $\llbracket x < y; x \in I; y \in I; \bigwedge k. \llbracket x < k; k < y \rrbracket \implies k \notin I \rrbracket \implies$
 $\text{inext } x \ I = y$
 $\langle proof \rangle$

corollary *min-step-inext2*[*rule-format*]:
 $\llbracket x < y; x \in I; y \in I; \neg(\exists k \in I. x < k \wedge k < y) \rrbracket \implies$
 $\text{inext } x \ I = y$
 $\langle proof \rangle$

lemma *between-empty-imp-inext-eq*:
 $\llbracket n \in A; n < \text{inext } n \ A; n \in B; \text{inext } n \ A \in B; B \downarrow > n \downarrow < (\text{inext } n \ A) = \{\} \rrbracket$
 \implies
 $\text{inext } n \ B = \text{inext } n \ A$
 $\langle proof \rangle$

lemma *inext-le-mono*: $\llbracket a \leq b; a \in I; b \in I \rrbracket \implies \text{inext } a \ I \leq \text{inext } b \ I$
 $\langle proof \rangle$

lemma *inext-less-mono*:
 $\llbracket a < b; a \in I; b \in I; \exists x \in I. b < x \rrbracket \implies \text{inext } a \ I < \text{inext } b \ I$
 $\langle proof \rangle$

lemma *inext-less-mono-fin*:
 $\llbracket a < b; a \in I; b \in I; \text{finite } I; b \neq \text{Max } I \rrbracket \implies \text{inext } a \ I < \text{inext } b \ I$
 $\langle proof \rangle$

lemma *inext-less-mono-infin*:
 $\llbracket a < b; a \in I; b \in I; \text{infinite } I \rrbracket \implies \text{inext } a \ I < \text{inext } b \ I$
 $\langle proof \rangle$

lemma *inext-less-mono-infin-fin*:
 $\llbracket a < b; a \in I; b \in I; b \neq \text{Max } I \vee \text{infinite } I \rrbracket \implies \text{inext } a \ I < \text{inext } b \ I$

$\langle proof \rangle$

lemma *inext-le-mono-rev*:

$\llbracket inext\ a\ I \leq inext\ b\ I; a \in I; b \in I; \exists x \in I. inext\ a\ I < x \rrbracket \implies a \leq b$
 $\langle proof \rangle$

lemma *inext-le-mono-fin-rev*:

$\llbracket inext\ a\ I \leq inext\ b\ I; a \in I; b \in I; finite\ I; inext\ a\ I \neq Max\ I \rrbracket \implies a \leq b$
 $\langle proof \rangle$

lemma *inext-le-mono-infin-rev*:

$\llbracket inext\ a\ I \leq inext\ b\ I; a \in I; b \in I; infinite\ I \rrbracket \implies a \leq b$
 $\langle proof \rangle$

lemma *inext-le-mono-infin-fin-rev*:

$\llbracket inext\ a\ I \leq inext\ b\ I; a \in I; b \in I; inext\ a\ I \neq Max\ I \vee infinite\ I \rrbracket \implies a \leq b$
 $\langle proof \rangle$

lemma *inext-less-mono-rev*:

$\llbracket inext\ a\ I < inext\ b\ I; a \in I; b \in I \rrbracket \implies a < b$
 $\langle proof \rangle$

lemma *less-imp-inext-le*: $\llbracket a < b; a \in I; b \in I \rrbracket \implies inext\ a\ I \leq b$

$\langle proof \rangle$

lemma *iprev-mono*: $iprev\ n\ I \leq n$

$\langle proof \rangle$

corollary *iprev-neq-imp-greater*: $n \neq iprev\ n\ I \implies iprev\ n\ I < n$

$\langle proof \rangle$

lemma *iprev-mono2*: $\llbracket n \in I; \exists x \in I. x < n \rrbracket \implies iprev\ n\ I < n$

$\langle proof \rangle$

lemma *iprev-mono2-if-neq-iMin*: $\llbracket n \in I; iMin\ I \neq n \rrbracket \implies iprev\ n\ I < n$

$\langle proof \rangle$

lemma *iprev-neq-Max*: $\llbracket finite\ I; \exists x \in I. x < n \rrbracket \implies iprev\ n\ I \neq Max\ I$

$\langle proof \rangle$

lemma *iprev-neq-Max-not-singleton*:

$\llbracket finite\ I; I \neq \{\}; \neg(\exists a. I = \{a\}) \rrbracket \implies iprev\ n\ I \neq Max\ I$
 $\langle proof \rangle$

corollary *iprev-neq-Max-not-card-1*:

$\llbracket finite\ I; I \neq \{\}; card\ I \neq Suc\ 0 \rrbracket \implies iprev\ n\ I \neq Max\ I$
 $\langle proof \rangle$

lemma *iprev-neq-imp-iMin*: $iprev\ n\ I \neq n \implies iMin\ I < n$

$\langle proof \rangle$

lemma *iprev-greater-conv*: $(n \in I \wedge iMin\ I < n) = (iprev\ n\ I < n)$
 ⟨proof⟩

lemma *inext-fix-iff*: $(n \notin I \vee (finite\ I \wedge Max\ I = n)) = (inext\ n\ I = n)$
 ⟨proof⟩

lemma *iprev-fix-iff*: $(n \notin I \vee iMin\ I = n) = (iprev\ n\ I = n)$
 ⟨proof⟩

lemma *iprev-min-step*: $\llbracket iprev\ n\ I < k; k < n \rrbracket \implies k \notin I$
 ⟨proof⟩

corollary *iprev-min-step2*: $\neg(\exists x \in I. iprev\ n\ I < x \wedge x < n)$
 ⟨proof⟩

lemma *min-step-iprev*:
 $\llbracket x < y; x \in I; y \in I; \bigwedge k. \llbracket x < k; k < y \rrbracket \implies k \notin I \rrbracket \implies$
 $iprev\ y\ I = x$
 ⟨proof⟩

corollary *min-step-iprev2*[rule-format]:
 $\llbracket x < y; x \in I; y \in I; \neg(\exists k \in I. x < k \wedge k < y) \rrbracket \implies$
 $iprev\ y\ I = x$
 ⟨proof⟩

lemma *between-empty-imp-iprev-eq*:
 $\llbracket n \in A; iprev\ n\ A < n; n \in B; iprev\ n\ A \in B; B \downarrow > (iprev\ n\ A) \downarrow < n = \{\} \rrbracket$
 \implies
 $iprev\ n\ B = iprev\ n\ A$
 ⟨proof⟩

lemma *iprev-le-mono*: $\llbracket a \leq b; a \in I; b \in I \rrbracket \implies iprev\ a\ I \leq iprev\ b\ I$
 ⟨proof⟩

lemma *iprev-less-mono*:
 $\llbracket a < b; a \in I; b \in I; \exists x \in I. x < a \rrbracket \implies iprev\ a\ I < iprev\ b\ I$
 ⟨proof⟩

lemma *iprev-less-mono-if-neq-iMin*:
 $\llbracket a < b; a \in I; b \in I; iMin\ I \neq a \rrbracket \implies iprev\ a\ I < iprev\ b\ I$
 ⟨proof⟩

lemma *iprev-le-mono-rev*:
 $\llbracket iprev\ a\ I \leq iprev\ b\ I; a \in I; b \in I; iMin\ I \neq iprev\ b\ I \rrbracket \implies a \leq b$
 ⟨proof⟩

lemma *iprev-less-mono-rev*:

$\llbracket \text{iprev } a \ I < \text{iprev } b \ I; a \in I; b \in I \rrbracket \implies a < b$
 $\langle \text{proof} \rangle$

lemma *set-restriction-inext-eq*:

$\llbracket \text{set-restriction interval-fun}; n \in \text{interval-fun } I; \text{inext } n \ I \in \text{interval-fun } I \rrbracket \implies$
 $\text{inext } n \ (\text{interval-fun } I) = \text{inext } n \ I$
 $\langle \text{proof} \rangle$

lemma *set-restriction-inext-singleton-eq*:

$\llbracket \text{set-restriction interval-fun}; n \in \text{interval-fun } I; \text{inext } n \ I \in \text{interval-fun } I \rrbracket \implies$
 $\{\text{inext } n \ (\text{interval-fun } I)\} = \text{interval-fun } \{\text{inext } n \ I\}$
 $\langle \text{proof} \rangle$

lemma *inext-iprev*: $iMin \ I \neq n \implies \text{inext } (\text{iprev } n \ I) \ I = n$

$\langle \text{proof} \rangle$

lemma *iprev-inext-infin*: $\text{infinite } I \implies \text{iprev } (\text{inext } n \ I) \ I = n$

$\langle \text{proof} \rangle$

lemma *iprev-inext-fin*:

$\llbracket \text{finite } I; n \neq \text{Max } I \rrbracket \implies \text{iprev } (\text{inext } n \ I) \ I = n$
 $\langle \text{proof} \rangle$

lemma *iprev-inext*:

$n \neq \text{Max } I \vee \text{infinite } I \implies \text{iprev } (\text{inext } n \ I) \ I = n$
 $\langle \text{proof} \rangle$

lemma *inext-eq-infin*:

$\llbracket \text{inext } a \ I = \text{inext } b \ I; \text{infinite } I \rrbracket \implies a = b$
 $\langle \text{proof} \rangle$

lemma *inext-eq-fin*:

$\llbracket \text{inext } a \ I = \text{inext } b \ I; \text{finite } I; a \neq \text{Max } I; b \neq \text{Max } I \rrbracket \implies a = b$
 $\langle \text{proof} \rangle$

lemma *inext-eq-infin-fin*:

$\llbracket \text{inext } a \ I = \text{inext } b \ I; a \neq \text{Max } I \wedge b \neq \text{Max } I \vee \text{infinite } I \rrbracket \implies a = b$
 $\langle \text{proof} \rangle$

lemma *inext-eq*:

$\llbracket \text{inext } a \ I = \text{inext } b \ I; \exists x \in I. a < x; \exists x \in I. b < x \rrbracket \implies a = b$
 $\langle \text{proof} \rangle$

lemma *iprev-eq-if-neq-iMin*:

$\llbracket \text{iprev } a \ I = \text{iprev } b \ I; \text{iMin } I \neq a; \text{iMin } I \neq b \rrbracket \implies a = b$
 ⟨proof⟩

lemma *iprev-eq*:

$\llbracket \text{iprev } a \ I = \text{iprev } b \ I; \exists x \in I. x < a; \exists x \in I. x < b \rrbracket \implies a = b$
 ⟨proof⟩

lemma *greater-imp-iprev-ge*: $\llbracket b < a; a \in I; b \in I \rrbracket \implies b \leq \text{iprev } a \ I$
 ⟨proof⟩

lemma *inext-cut-less-conv*: $\text{inext } n \ I < t \implies \text{inext } n \ (I \downarrow < t) = \text{inext } n \ I$
 ⟨proof⟩

lemma *inext-cut-le-conv*: $\text{inext } n \ I \leq t \implies \text{inext } n \ (I \downarrow \leq t) = \text{inext } n \ I$
 ⟨proof⟩

lemma *inext-cut-greater-conv*: $t < n \implies \text{inext } n \ (I \downarrow > t) = \text{inext } n \ I$
 ⟨proof⟩

lemma *inext-cut-ge-conv*: $t \leq n \implies \text{inext } n \ (I \downarrow \geq t) = \text{inext } n \ I$
 ⟨proof⟩

lemmas *inext-cut-conv* =

inext-cut-less-conv inext-cut-le-conv

inext-cut-greater-conv inext-cut-ge-conv

lemma *iprev-cut-greater-conv*: $t < \text{iprev } n \ I \implies \text{iprev } n \ (I \downarrow > t) = \text{iprev } n \ I$
 ⟨proof⟩

lemma *iprev-cut-ge-conv*: $t \leq \text{iprev } n \ I \implies \text{iprev } n \ (I \downarrow \geq t) = \text{iprev } n \ I$
 ⟨proof⟩

lemma *iprev-cut-less-conv*: $n < t \implies \text{iprev } n \ (I \downarrow < t) = \text{iprev } n \ I$
 ⟨proof⟩

lemma *iprev-cut-le-conv*: $n \leq t \implies \text{iprev } n \ (I \downarrow \leq t) = \text{iprev } n \ I$
 ⟨proof⟩

lemmas *iprev-cut-conv* =

iprev-cut-less-conv iprev-cut-le-conv

iprev-cut-greater-conv iprev-cut-ge-conv

lemma *inext-cut-less-fix*: $t \leq \text{inext } n \ I \implies \text{inext } n \ (I \downarrow < t) = n$
 ⟨proof⟩

lemma *inext-cut-le-fix*: $t < \text{inext } n \ I \implies \text{inext } n \ (I \downarrow \leq t) = n$

<proof>

lemma *iprev-cut-greater-fix*: $iprev\ n\ I \leq t \implies iprev\ n\ (I \downarrow > t) = n$
<proof>

lemma *iprev-cut-ge-fix*: $iprev\ n\ I < t \implies iprev\ n\ (I \downarrow \geq t) = n$
<proof>

definition

CommuteWithIntervalCut4 :: $((\text{'a}::\text{linorder})\ \text{set} \Rightarrow \text{'a}\ \text{set}) \Rightarrow \text{bool}$

where

CommuteWithIntervalCut4 fun \equiv

$\forall t\ fun2\ I.$

$(fun2 = (\lambda I. I \downarrow < t) \vee fun2 = (\lambda I. I \downarrow \leq t) \vee fun2 = (\lambda I. I \downarrow > t) \vee fun2 =$
 $(\lambda I. I \downarrow \geq t)) \longrightarrow$

$fun\ (fun2\ I) = fun2\ (fun\ I)$

definition *CommuteWithIntervalCut2* :: $((\text{'a}::\text{linorder})\ \text{set} \Rightarrow \text{'a}\ \text{set}) \Rightarrow \text{bool}$

where

CommuteWithIntervalCut2 fun \equiv

$\forall t\ fun2\ I.$

$(fun2 = (\lambda I. I \downarrow < t) \vee fun2 = (\lambda I. I \downarrow > t)) \longrightarrow$

$fun\ (fun2\ I) = fun2\ (fun\ I)$

lemma *CommuteWithIntervalCut4-imp-2*: $CommuteWithIntervalCut4\ fun \implies Com-$
muteWithIntervalCut2 fun
<proof>

lemma *nat-CommuteWithIntervalCut2-4-eg*:

$CommuteWithIntervalCut4\ (fun::\text{nat}\ \text{set} \Rightarrow \text{nat}\ \text{set}) = CommuteWithInterval-$
Cut2 fun
<proof>

lemma

cut-less-CommuteWithIntervalCut4: $CommuteWithIntervalCut4\ (\lambda I. I \downarrow < t)$

and

cut-le-CommuteWithIntervalCut4: $CommuteWithIntervalCut4\ (\lambda I. I \downarrow \leq t)$

and

cut-greater-CommuteWithIntervalCut4: $CommuteWithIntervalCut4\ (\lambda I. I \downarrow > t)$

and

cut-ge-CommuteWithIntervalCut4: $CommuteWithIntervalCut4\ (\lambda I. I \downarrow \geq t)$

<proof>

lemmas *i-cut-CommuteWithIntervalCut4* =

cut-less-CommuteWithIntervalCut4 *cut-le-CommuteWithIntervalCut4*

cut-greater-CommuteWithIntervalCut4 *cut-ge-CommuteWithIntervalCut4*

lemma *inext-image*:

$\llbracket n \in I; \text{strict-mono-on } f\ I \rrbracket \implies inext\ (f\ n)\ (f\ 'I) = f\ (inext\ n\ I)$
<proof>

lemma *iprev-image*:

$\llbracket n \in I; \text{strict-mono-on } f \ I \rrbracket \implies \text{iprev } (f \ n) \ (f \ ' \ I) = f \ (\text{iprev } n \ I)$
 <proof>

lemma *inext-image2*:

$\text{strict-mono } f \implies \text{inext } (f \ n) \ (f \ ' \ I) = f \ (\text{inext } n \ I)$
 <proof>

lemma *iprev-image2*:

$\text{strict-mono } f \implies \text{iprev } (f \ n) \ (f \ ' \ I) = f \ (\text{iprev } n \ I)$
 <proof>

lemma *inext-imirror-iprev-conv*:

$\llbracket \text{finite } I; n \leq iMin \ I + Max \ I \rrbracket \implies$
 $\text{inext } (\text{mirror-elem } n \ I) \ (\text{imirror } I) = \text{mirror-elem } (\text{iprev } n \ I) \ I$
 <proof>

corollary *inext-imirror-iprev-conv'*:

$\llbracket \text{finite } I; n \in I \rrbracket \implies$
 $\text{inext } (\text{mirror-elem } n \ I) \ (\text{imirror } I) = \text{mirror-elem } (\text{iprev } n \ I) \ I$
 <proof>

lemma *iprev-imirror-inext-conv*:

$\llbracket \text{finite } I; n \leq iMin \ I + Max \ I \rrbracket \implies$
 $\text{iprev } (\text{mirror-elem } n \ I) \ (\text{imirror } I) = \text{mirror-elem } (\text{inext } n \ I) \ I$
 <proof>

corollary *iprev-imirror-inext-conv'*:

$\llbracket \text{finite } I; n \in I \rrbracket \implies$
 $\text{iprev } (\text{mirror-elem } n \ I) \ (\text{imirror } I) = \text{mirror-elem } (\text{inext } n \ I) \ I$
 <proof>

lemma *inext-insert-ge-Max*:

$\llbracket \text{finite } I; I \neq \{\}; Max \ I \leq a \rrbracket \implies \text{inext } (Max \ I) \ (\text{insert } a \ I) = a$
 <proof>

lemma *iprev-insert-le-iMin*:

$\llbracket \text{finite } I; I \neq \{\}; a \leq iMin \ I \rrbracket \implies \text{iprev } (iMin \ I) \ (\text{insert } a \ I) = a$
 <proof>

lemma *cut-less-le-iprev-conv*:

$\llbracket t \in I; t \neq iMin \ I \rrbracket \implies I \downarrow < t = I \downarrow \leq (\text{iprev } t \ I)$
 <proof>

lemma *neq-Max-imp-inext-neq-iMin*:

$\llbracket t \in I; t \neq Max \ I \vee \text{infinite } I \rrbracket \implies \text{inext } t \ I \neq iMin \ I$
 <proof>

corollary *neq-Max-imp-inext-gr-iMin:*

$\llbracket t \in I; t \neq \text{Max } I \vee \text{infinite } I \rrbracket \implies \text{iMin } I < \text{inext } t I$
 ⟨proof⟩

lemma *cut-le-less-inext-conv:*

$\llbracket t \in I; t \neq \text{Max } I \vee \text{infinite } I \rrbracket \implies I \downarrow \leq t = I \downarrow < (\text{inext } t I)$
 ⟨proof⟩

lemma *cut-ge-greater-iprev-conv:*

$\llbracket t \in I; t \neq \text{iMin } I \rrbracket \implies I \downarrow \geq t = I \downarrow > (\text{iprev } t I)$
 ⟨proof⟩

lemma *cut-greater-ge-inext-conv:*

$\llbracket t \in I; t \neq \text{Max } I \vee \text{infinite } I \rrbracket \implies I \downarrow > t = I \downarrow \geq (\text{inext } t I)$
 ⟨proof⟩

lemma *inext-append:*

$\llbracket \text{finite } A; A \neq \{\}; B \neq \{\}; \text{Max } A < \text{iMin } B \rrbracket \implies$
 $\text{inext } n (A \cup B) = (\text{if } n \in B \text{ then inext } n B \text{ else (if } n = \text{Max } A \text{ then iMin } B \text{ else inext } n A))$
 ⟨proof⟩

corollary *inext-append-eq1:*

$\llbracket \text{finite } A; A \neq \{\}; B \neq \{\}; \text{Max } A < \text{iMin } B; n \in A; n \neq \text{Max } A \rrbracket \implies$
 $\text{inext } n (A \cup B) = \text{inext } n A$
 ⟨proof⟩

corollary *inext-append-eq2:*

$\llbracket \text{finite } A; A \neq \{\}; B \neq \{\}; \text{Max } A < \text{iMin } B; n \in B \rrbracket \implies$
 $\text{inext } n (A \cup B) = \text{inext } n B$
 ⟨proof⟩

corollary *inext-append-eq3:*

$\llbracket \text{finite } A; A \neq \{\}; B \neq \{\}; \text{Max } A < \text{iMin } B \rrbracket \implies$
 $\text{inext } (\text{Max } A) (A \cup B) = \text{iMin } B$
 ⟨proof⟩

lemma *iprev-append:* $\llbracket \text{finite } A; A \neq \{\}; B \neq \{\}; \text{Max } A < \text{iMin } B \rrbracket \implies$

$\text{iprev } n (A \cup B) = (\text{if } n \in A \text{ then iprev } n A \text{ else (if } n = \text{iMin } B \text{ then Max } A \text{ else iprev } n B))$
 ⟨proof⟩

corollary *iprev-append-eq1:*

$\llbracket \text{finite } A; A \neq \{\}; B \neq \{\}; \text{Max } A < \text{iMin } B; n \in A \rrbracket \implies$
 $\text{iprev } n (A \cup B) = \text{iprev } n A$
 ⟨proof⟩

corollary *iprev-append-eq2:*

$\llbracket \text{finite } A; A \neq \{\}; B \neq \{\}; \text{Max } A < \text{iMin } B; n \in B; n \neq \text{iMin } B \rrbracket \implies$
 $\text{iprev } n (A \cup B) = \text{iprev } n B$
 ⟨proof⟩

corollary *iprev-append-eq3*:

$$\begin{aligned} & \llbracket \text{finite } A; A \neq \{\}; B \neq \{\}; \text{Max } A < \text{iMin } B \rrbracket \implies \\ & \text{iprev } (\text{iMin } B) (A \cup B) = \text{Max } A \\ & \langle \text{proof} \rangle \end{aligned}$$

lemma *inext-predicate-change-exists-aux*: $\bigwedge a.$

$$\begin{aligned} & \llbracket c = \text{card } (I \downarrow \geq a \downarrow < b); a < b; a \in I; b \in I; \neg P a; P b \rrbracket \implies \\ & \exists n \in (I \downarrow \geq a \downarrow < b). \neg P n \wedge P (\text{inext } n I) \\ & \langle \text{proof} \rangle \end{aligned}$$

lemma *inext-predicate-change-exists*:

$$\begin{aligned} & \llbracket a \leq b; a \in I; b \in I; \neg P a; P b \rrbracket \implies \\ & \exists n \in I. a \leq n \wedge n < b \wedge \neg P n \wedge P (\text{inext } n I) \\ & \langle \text{proof} \rangle \end{aligned}$$

lemma *iprev-predicate-change-exists*:

$$\begin{aligned} & \llbracket a \leq b; a \in I; b \in I; \neg P b; P a \rrbracket \implies \\ & \exists n \in I. a < n \wedge n \leq b \wedge \neg P n \wedge P (\text{iprev } n I) \\ & \langle \text{proof} \rangle \end{aligned}$$

corollary *nat-Suc-predicate-change-exists*:

$$\begin{aligned} & \llbracket a \leq b; \neg P a; P b \rrbracket \implies \exists n \geq a. n < b \wedge \neg P n \wedge P (\text{Suc } n) \\ & \langle \text{proof} \rangle \end{aligned}$$

corollary *nat-pred-predicate-change-exists*:

$$\begin{aligned} & \llbracket a \leq b; \neg P b; P a \rrbracket \implies \exists n \leq b. a < n \wedge \neg P n \wedge P (n - \text{Suc } 0) \\ & \langle \text{proof} \rangle \end{aligned}$$

lemma *inext-predicate-change-exists2-all*:

$$\begin{aligned} & \llbracket (a :: \text{nat}) \leq b; a \in I; b \in I; \neg P a; \forall k \in I \downarrow \geq b. P k \rrbracket \implies \\ & \exists n \in I. a \leq n \wedge n < b \wedge \neg P n \wedge (\forall k \in I \downarrow > n. P k) \\ & \langle \text{proof} \rangle \end{aligned}$$

corollary *inext-predicate-change-exists2*:

$$\begin{aligned} & \llbracket (a :: \text{nat}) \leq b; a \in I; b \in I; \neg P a; P b \rrbracket \implies \\ & \exists n \in I. a \leq n \wedge n < b \wedge \neg P n \wedge (\forall k \in I. n < k \wedge k \leq b \longrightarrow P k) \\ & \langle \text{proof} \rangle \end{aligned}$$

corollary *nat-Suc-predicate-change-exists2-all*:

$$\begin{aligned} & \llbracket (a :: \text{nat}) \leq b; \neg P a; \forall k \geq b. P k \rrbracket \implies \\ & \exists n \geq a. n < b \wedge \neg P n \wedge (\forall k > n. P k) \\ & \langle \text{proof} \rangle \end{aligned}$$

corollary *nat-Suc-predicate-change-exists2*:

$$\begin{aligned} & \llbracket (a::\text{nat}) \leq b; \neg P a; P b \rrbracket \implies \\ & \exists n \geq a. n < b \wedge \neg P n \wedge (\forall k \leq b. n < k \longrightarrow P k) \\ & \langle \text{proof} \rangle \end{aligned}$$

lemma *iprev-predicate-change-exists2-all*:

$$\begin{aligned} & \llbracket (a::\text{nat}) \leq b; a \in I; b \in I; \neg P b; \forall k \in I \downarrow \leq a. P k \rrbracket \implies \\ & \exists n \in I. a < n \wedge n \leq b \wedge \neg P n \wedge (\forall k \in I \downarrow < n. P k) \\ & \langle \text{proof} \rangle \end{aligned}$$

corollary *iprev-predicate-change-exists2*:

$$\begin{aligned} & \llbracket (a::\text{nat}) \leq b; a \in I; b \in I; \neg P b; P a \rrbracket \implies \\ & \exists n \in I. a < n \wedge n \leq b \wedge \neg P n \wedge (\forall k \in I. a \leq k \wedge k < n \longrightarrow P k) \\ & \langle \text{proof} \rangle \end{aligned}$$

corollary *nat-pred-predicate-change-exists2-all*:

$$\begin{aligned} & \llbracket (a::\text{nat}) \leq b; \neg P b; \forall k \leq a. P k \rrbracket \implies \\ & \exists n > a. n \leq b \wedge \neg P n \wedge (\forall k < n. P k) \\ & \langle \text{proof} \rangle \end{aligned}$$

corollary *nat-pred-predicate-change-exists2*:

$$\begin{aligned} & \llbracket (a::\text{nat}) \leq b; \neg P b; P a \rrbracket \implies \\ & \exists n > a. n \leq b \wedge \neg P n \wedge (\forall k \geq a. k < n \longrightarrow P k) \\ & \langle \text{proof} \rangle \end{aligned}$$

8.2 *inext-nth* and *iprev-nth* – nth element of a natural set

primrec *inext-nth* :: *nat set* \Rightarrow *nat* \Rightarrow *nat* $((- \rightarrow -) [100, 100] 60)$

where

$$\begin{aligned} & I \rightarrow 0 = iMin I \\ & | I \rightarrow Suc n = inext (inext-nth I n) I \end{aligned}$$

lemma *inext-nth-closed*: $I \neq \{\} \implies I \rightarrow n \in I$

$\langle \text{proof} \rangle$

lemma *inext-nth-image*:

$$\llbracket I \neq \{\}; \text{strict-mono-on } f I \rrbracket \implies (f ' I) \rightarrow n = f (I \rightarrow n)$$

$\langle \text{proof} \rangle$

lemma *inext-nth-Suc-mono*: $I \rightarrow n \leq I \rightarrow Suc n$

$\langle \text{proof} \rangle$

lemma *inext-nth-mono*: $a \leq b \implies I \rightarrow a \leq I \rightarrow b$

$\langle \text{proof} \rangle$

lemma *inext-nth-Suc-mono2*: $\exists x \in I. I \rightarrow n < x \implies I \rightarrow n < I \rightarrow Suc n$

$\langle \text{proof} \rangle$

lemma *inext-nth-mono2*: $\exists x \in I. I \rightarrow a < x \implies (I \rightarrow a < I \rightarrow b) = (a < b)$
 ⟨proof⟩

lemma *inext-nth-mono2-infin*:
 $\text{infinite } I \implies (I \rightarrow a < I \rightarrow b) = (a < b)$
 ⟨proof⟩

lemma *inext-nth-Max-fix*:
 $\llbracket \text{finite } I; I \neq \{\}; I \rightarrow a = \text{Max } I; a \leq b \rrbracket \implies I \rightarrow b = \text{Max } I$
 ⟨proof⟩

lemma *inext-nth-cut-less-conv*:
 $\bigwedge I. I \rightarrow n < t \implies (I \downarrow < t) \rightarrow n = I \rightarrow n$
 ⟨proof⟩

lemma *remove-Min-inext-nth-Suc-conv*: $\bigwedge I.$
 $\text{Suc } 0 < \text{card } I \vee \text{infinite } I \implies$
 $(I - \{\text{iMin } I\}) \rightarrow n = I \rightarrow \text{Suc } n$

⟨proof⟩

corollary *remove-Min-inext-nth-Suc-conv-finite*: $\text{Suc } 0 < \text{card } I \implies (I - \{\text{iMin } I\}) \rightarrow n = I \rightarrow \text{Suc } n$
 ⟨proof⟩

corollary *remove-Min-inext-nth-Suc-conv-infinite*: $\text{infinite } I \implies (I - \{\text{iMin } I\}) \rightarrow n = I \rightarrow \text{Suc } n$
 ⟨proof⟩

lemma *remove-Max-eq*: $\llbracket \text{finite } I; I \neq \{\}; n \neq \text{Max } I \rrbracket \implies \text{Max } (I - \{n\}) = \text{Max } I$
 ⟨proof⟩

lemma *remove-iMin-eq*: $\llbracket I \neq \{\}; n \neq \text{iMin } I \rrbracket \implies \text{iMin } (I - \{n\}) = \text{iMin } I$
 ⟨proof⟩

lemma *remove-Min-eq*: $\llbracket \text{finite } I; I \neq \{\}; n \neq \text{Min } I \rrbracket \implies \text{Min } (I - \{n\}) = \text{Min } I$
 ⟨proof⟩

lemma *Max-le-iMin-conv-singleton*: $\llbracket \text{finite } I; I \neq \{\} \rrbracket \implies (\text{Max } I \leq \text{iMin } I) = (\exists x. I = \{x\})$
 ⟨proof⟩

lemma *inext-nth-card-less-Max*:
 $\bigwedge I. \text{Suc } n < \text{card } I \implies I \rightarrow n < \text{Max } I$
 ⟨proof⟩

lemma *inext-nth-card-less-Max'*:
 $n < \text{card } I - \text{Suc } 0 \implies I \rightarrow n < \text{Max } I$

$\langle \text{proof} \rangle$

lemma *inext-nth-card-Max-aux*:

$\bigwedge I. \text{card } I = \text{Suc } n \implies I \rightarrow n = \text{Max } I$
 $\langle \text{proof} \rangle$

lemma *inext-nth-card-Max-aux'*:

$\bigwedge I. \llbracket \text{finite } I; I \neq \{\} \rrbracket \implies I \rightarrow (\text{card } I - \text{Suc } 0) = \text{Max } I$
 $\langle \text{proof} \rangle$

lemma *inext-nth-card-Max*:

$\llbracket \text{finite } I; I \neq \{\}; \text{card } I \leq \text{Suc } n \rrbracket \implies I \rightarrow n = \text{Max } I$
 $\langle \text{proof} \rangle$

lemma *inext-nth-card-Max'*:

$\llbracket \text{finite } I; I \neq \{\}; \text{card } I - \text{Suc } 0 \leq n \rrbracket \implies I \rightarrow n = \text{Max } I$
 $\langle \text{proof} \rangle$

lemma *inext-nth-singleton*: $\{a\} \rightarrow n = a$

$\langle \text{proof} \rangle$

lemma *inext-nth-eq-Min-conv*:

$I \neq \{\} \implies (I \rightarrow n = \text{iMin } I) = (n = 0 \vee (\exists a. I = \{a\}))$
 $\langle \text{proof} \rangle$

lemma *inext-nth-gr-Min-conv*:

$I \neq \{\} \implies (\text{iMin } I < I \rightarrow n) = (0 < n \wedge \neg(\exists a. I = \{a\}))$
 $\langle \text{proof} \rangle$

lemma *inext-nth-gr-Min-conv-infinite*:

$\text{infinite } I \implies (\text{iMin } I < I \rightarrow n) = (0 < n)$
 $\langle \text{proof} \rangle$

lemma *inext-nth-cut-ge-inext-nth*: $\bigwedge I b.$

$I \neq \{\} \implies I \downarrow \geq (I \rightarrow a) \rightarrow b = I \rightarrow (a + b)$
 $\langle \text{proof} \rangle$

lemma *inext-nth-append-eq1*:

$\llbracket \text{finite } A; A \neq \{\}; \text{Max } A < \text{iMin } B; A \rightarrow n \neq \text{Max } A \rrbracket \implies$
 $(A \cup B) \rightarrow n = A \rightarrow n$
 $\langle \text{proof} \rangle$

lemma *inext-nth-card-append-eq1*:

$\bigwedge A B. \llbracket \text{Max } A < \text{iMin } B; n < \text{card } A \rrbracket \implies$
 $(A \cup B) \rightarrow n = A \rightarrow n$
 $\langle \text{proof} \rangle$

lemma *inext-nth-card-append-eq3*:

$\llbracket \text{finite } A; B \neq \{\}; \text{Max } A < i\text{Min } B \rrbracket \implies$
 $(A \cup B) \rightarrow (\text{card } A) = i\text{Min } B$
 <proof>

lemma *inext-nth-card-append-eq2*:

$\llbracket \text{finite } A; A \neq \{\}; B \neq \{\}; \text{Max } A < i\text{Min } B; \text{card } A \leq n \rrbracket \implies$
 $(A \cup B) \rightarrow n = B \rightarrow (n - \text{card } A)$
 <proof>

lemma *inext-nth-card-append*:

$\llbracket \text{finite } A; A \neq \{\}; B \neq \{\}; \text{Max } A < i\text{Min } B \rrbracket \implies$
 $(A \cup B) \rightarrow n = (\text{if } n < \text{card } A \text{ then } A \rightarrow n \text{ else } B \rightarrow (n - \text{card } A))$
 <proof>

lemma *inext-nth-insert-Suc*:

$\llbracket I \neq \{\}; a < i\text{Min } I \rrbracket \implies (\text{insert } a \text{ } I) \rightarrow \text{Suc } n = I \rightarrow n$
 <proof>

lemma *inext-nth-cut-less-eq*:

$n < \text{card } (I \downarrow < t) \implies (I \downarrow < t) \rightarrow n = I \rightarrow n$
 <proof>

lemma *less-card-cut-less-imp-inext-nth-less*:

$n < \text{card } (I \downarrow < t) \implies I \rightarrow n < t$
 <proof>

lemma *inext-nth-less-less-card-conv*:

$I \downarrow \geq t \neq \{\} \implies (I \rightarrow n < t) = (n < \text{card } (I \downarrow < t))$
 <proof>

lemma *cut-less-inext-nth-card-eq1*:

$n < \text{card } I \vee \text{infinite } I \implies \text{card } (I \downarrow < (I \rightarrow n)) = n$
 <proof>

lemma *cut-less-inext-nth-card-eq2*:

$\llbracket \text{finite } I; \text{card } I \leq \text{Suc } n \rrbracket \implies \text{card } (I \downarrow < (I \rightarrow n)) = \text{card } I - \text{Suc } 0$
 <proof>

lemma *cut-less-inext-nth-card-if*:

$\text{card } (I \downarrow < (I \rightarrow n)) =$
 $\text{if } (n < \text{card } I \vee \text{infinite } I) \text{ then } n \text{ else } \text{card } I - \text{Suc } 0$
 <proof>

lemma *cut-le-inext-nth-card-eq1*:

$n < \text{card } I \vee \text{infinite } I \implies \text{card } (I \downarrow \leq (I \rightarrow n)) = \text{Suc } n$
 <proof>

lemma *cut-le-inext-nth-card-eq2*:

$\llbracket \text{finite } I; \text{card } I \leq \text{Suc } n \rrbracket \implies \text{card } (I \downarrow \leq (I \rightarrow n)) = \text{card } I$
 $\langle \text{proof} \rangle$

lemma *cut-le-inext-nth-card-if*:

$\text{card } (I \downarrow \leq (I \rightarrow n)) = ($
 $\text{if } (n < \text{card } I \vee \text{infinite } I) \text{ then Suc } n \text{ else card } I)$
 $\langle \text{proof} \rangle$

primrec *iprev-nth* :: $\text{nat set} \Rightarrow \text{nat} \Rightarrow \text{nat} \ ((- \leftarrow -) [100, 100] 60)$

where

$I \leftarrow 0 = \text{Max } I$
 $| I \leftarrow \text{Suc } n = \text{iprev } (\text{iprev-nth } I \ n) \ I$

lemma *iprev-nth-closed*: $\llbracket \text{finite } I; I \neq \{\} \rrbracket \implies I \leftarrow n \in I$

$\langle \text{proof} \rangle$

lemma *iprev-nth-image*:

$\llbracket \text{finite } I; I \neq \{\}; \text{strict-mono-on } f \ I \rrbracket \implies (f \ ' \ I) \leftarrow n = f \ (I \leftarrow n)$
 $\langle \text{proof} \rangle$

lemma *iprev-nth-Suc-mono*: $I \leftarrow (\text{Suc } n) \leq I \leftarrow n$

$\langle \text{proof} \rangle$

lemma *iprev-nth-mono*: $a \leq b \implies I \leftarrow b \leq I \leftarrow a$

$\langle \text{proof} \rangle$

lemma *iprev-nth-Suc-mono2*:

$\llbracket \text{finite } I; \exists x \in I. x < I \leftarrow n \rrbracket \implies I \leftarrow (\text{Suc } n) < I \leftarrow n$
 $\langle \text{proof} \rangle$

lemma *iprev-nth-mono2*:

$\llbracket \text{finite } I; \exists x \in I. x < I \leftarrow a \rrbracket \implies (I \leftarrow b < I \leftarrow a) = (a < b)$
 $\langle \text{proof} \rangle$

lemma *iprev-nth-iMin-fix*:

$\llbracket I \neq \{\}; I \leftarrow a = \text{iMin } I; a \leq b \rrbracket \implies I \leftarrow b = \text{iMin } I$
 $\langle \text{proof} \rangle$

lemma *iprev-nth-singleton*: $\{a\} \leftarrow n = a$

$\langle \text{proof} \rangle$

8.3 Induction over arbitrary natural sets using the functions

inext and *iprev*

lemma *inext-nth-surj-aux1*:

$\{x \in I. \neg(\exists n. I \rightarrow n = x)\} = \{\}$
 $(\text{is } ?S = \{\})$

is $\{ x \in I. ?P x \} = \{\}$
 ⟨proof⟩

lemma *inext-nth-surj-on:surj-on* $(\lambda n. I \rightarrow n)$ *UNIV I*
 ⟨proof⟩

corollary *in-imp-ex-inext-nth*: $x \in I \implies \exists n. x = I \rightarrow n$
 ⟨proof⟩

lemma *inext-induct*:
 $\llbracket P (iMin I); \bigwedge n. \llbracket n \in I; P n \rrbracket \implies P (inext n I); n \in I \rrbracket \implies P n$
 ⟨proof⟩

lemma *iprev-nth-surj-aux1*:
 $finite I \implies \{ x \in I. \neg(\exists n. I \leftarrow n = x) \} = \{\}$
 ⟨proof⟩

lemma *iprev-nth-surj-on*: $finite I \implies surj-on (\lambda n. I \leftarrow n)$ *UNIV I*
 ⟨proof⟩

corollary *in-imp-ex-iprev-nth*:
 $\llbracket finite I; x \in I \rrbracket \implies \exists n. x = I \leftarrow n$
 ⟨proof⟩

lemma *iprev-induct*:
 $\llbracket P (Max I); \bigwedge n. \llbracket n \in I; P n \rrbracket \implies P (iprev n I); finite I; n \in I \rrbracket \implies P n$
 ⟨proof⟩

8.4 Natural intervals with *inext* and *iprev*

lemma *inext-atLeast*: $n \leq t \implies inext t \{n..\} = Suc t$
 ⟨proof⟩

lemma *iprev-atLeast'*: $n \leq t \implies iprev (Suc t) \{n..\} = t$
 ⟨proof⟩

lemma *iprev-atLeast*: $n < t \implies iprev t \{n..\} = t - Suc 0$
 ⟨proof⟩

lemma *inext-atMost*: $t < n \implies inext t \{..n\} = Suc t$
 ⟨proof⟩

lemma *iprev-atMost*: $t \leq n \implies iprev t \{..n\} = t - Suc 0$
 ⟨proof⟩

lemma *inext-lessThan*: $Suc t < n \implies inext t \{..<n\} = Suc t$
 ⟨proof⟩

lemma *iprev-lessThan*: $t < n \implies iprev t \{..<n\} = t - Suc 0$
 ⟨proof⟩

lemma *inext-atLeastAtMost*: $\llbracket m \leq t; t < n \rrbracket \implies \text{inext } t \{m..n\} = \text{Suc } t$

<proof>

lemma *iprev-atLeastAtMost*: $\llbracket m < t; t \leq n \rrbracket \implies \text{iprev } t \{m..n\} = t - \text{Suc } 0$

<proof>

lemma *iprev-atLeastAtMost'*: $\llbracket m \leq t; t < n \rrbracket \implies \text{iprev } (\text{Suc } t) \{m..n\} = t$

<proof>

lemma *inext-nth-atLeast* : $\{n..\} \rightarrow a = n + a$

<proof>

lemma *inext-nth-atLeastAtMost*: $\llbracket a \leq n - m; m \leq n \rrbracket \implies \{m..n\} \rightarrow a = m + a$

<proof>

lemma *iprev-nth-atLeastAtMost*: $\llbracket a \leq n - m; m \leq n \rrbracket \implies \{m..n\} \leftarrow a = n - a$

<proof>

lemma *inext-nth-atMost*: $a \leq n \implies \{..n\} \rightarrow a = a$

<proof>

lemma *iprev-nth-atMost*: $a \leq n \implies \{..n\} \leftarrow a = n - a$

<proof>

lemma *inext-nth-lessThan* : $a < n \implies \{..<n\} \rightarrow a = a$

<proof>

lemma *iprev-nth-lessThan*: $a < n \implies \{..<n\} \leftarrow a = n - \text{Suc } a$

<proof>

lemma *inext-nth-UNIV*: $\text{UNIV} \rightarrow a = a$

<proof>

8.5 Further result for *inext-nth* and *iprev-nth*

lemma *inext-iprev-nth-Suc*:

$i\text{Min } I \neq I \leftarrow n \implies \text{inext } (I \leftarrow \text{Suc } n) I = I \leftarrow n$

<proof>

lemma *inext-iprev-nth-pred*:

$\llbracket \text{finite } I; i\text{Min } I \neq I \leftarrow (n - \text{Suc } 0) \rrbracket \implies$

$\text{inext } (I \leftarrow n) I = I \leftarrow (n - \text{Suc } 0)$

<proof>

lemma *iprev-inext-nth-Suc*:

$I \rightarrow n \neq \text{Max } I \vee \text{infinite } I \implies \text{iprev } (I \rightarrow \text{Suc } n) I = I \rightarrow n$

<proof>

lemma *iprev-inext-nth-pred*:

$I \rightarrow (n - \text{Suc } 0) \neq \text{Max } I \vee \text{infinite } I \implies$

$\text{iprev } (I \rightarrow n) I = I \rightarrow (n - \text{Suc } 0)$

<proof>

lemma *inext-nth-imirror-iprev-nth-conv*:

$\llbracket \text{finite } I; I \neq \{\} \rrbracket \implies$
 $(\text{imirror } I) \rightarrow n = \text{mirror-elem } (I \leftarrow n) I$
 <proof>

corollary *inext-nth-imirror-iprev-nth-conv2*:
 $\llbracket \text{finite } I; I \neq \{\} \rrbracket \implies$
 $\text{mirror-elem } ((\text{imirror } I) \leftarrow n) I = I \rightarrow n$
 <proof>

lemma *iprev-nth-imirror-inext-nth-conv*:
 $\llbracket \text{finite } I; I \neq \{\} \rrbracket \implies$
 $(\text{imirror } I) \leftarrow n = \text{mirror-elem } (I \rightarrow n) I$
 <proof>

corollary *iprev-nth-imirror-inext-nth-conv2*:
 $\llbracket \text{finite } I; I \neq \{\} \rrbracket \implies$
 $\text{mirror-elem } ((\text{imirror } I) \rightarrow n) I = (I \leftarrow n)$
 <proof>

lemma *iprev-nth-card-greater-iMin*: $\text{Suc } n < \text{card } I \implies iMin I < I \leftarrow n$
 <proof>

lemma *iprev-nth-card-iMin*:
 $\llbracket \text{finite } I; I \neq \{\}; \text{card } I \leq \text{Suc } n \rrbracket \implies I \leftarrow n = iMin I$
 <proof>

lemma *iprev-nth-card-iMin'*:
 $\llbracket \text{finite } I; I \neq \{\}; \text{card } I - \text{Suc } 0 \leq n \rrbracket \implies I \leftarrow n = iMin I$
 <proof>

end

9 Additional definitions and results for lists

theory *List2*
imports *../CommonSet/SetIntervalCut*
begin

9.1 Additional definitions and results for lists

Infix syntactical abbreviations for operators *take* and *drop*. The abbreviations resemble to the operator symbols used later for take and drop operators on infinite lists in ListInf.

abbreviation *f-take'* :: 'a list \Rightarrow nat \Rightarrow 'a list (infixl \downarrow 100)
where $xs \downarrow n \equiv \text{take } n \ xs$
abbreviation *f-drop'* :: 'a list \Rightarrow nat \Rightarrow 'a list (infixl \uparrow 100)
where $xs \uparrow n \equiv \text{drop } n \ xs$

lemma *append-eq-Cons*: $[x] @ xs = x \# xs$
 ⟨proof⟩

lemma *length-Cons*: $length (x \# xs) = Suc (length xs)$
 ⟨proof⟩

lemma *length-snoc*: $length (xs @ [x]) = Suc (length xs)$
 ⟨proof⟩

9.1.1 Additional lemmata about list emptiness

lemma *length-greater-imp-not-empty*: $n < length xs \implies xs \neq []$
 ⟨proof⟩

lemma *length-ge-Suc-imp-not-empty*: $Suc n \leq length xs \implies xs \neq []$
 ⟨proof⟩

lemma *length-take-le*: $length (xs \downarrow n) \leq length xs$
 ⟨proof⟩

lemma *take-not-empty-conv*: $(xs \downarrow n \neq []) = (0 < n \wedge xs \neq [])$
 ⟨proof⟩

lemma *drop-not-empty-conv*: $(xs \uparrow n \neq []) = (n < length xs)$
 ⟨proof⟩

lemma *zip-eq-Nil*: $(zip xs ys = []) = (xs = [] \vee ys = [])$
 ⟨proof⟩

lemma *zip-not-empty-conv*: $(zip xs ys \neq []) = (xs \neq [] \wedge ys \neq [])$
 ⟨proof⟩

9.1.2 Additional lemmata about take, drop, hd, last, nth and filter

lemma *nth-tl-eq-nth-Suc*:
 $Suc n \leq length xs \implies (tl xs) ! n = xs ! Suc n$
 ⟨proof⟩

corollary *nth-tl-eq-nth-Suc2*:
 $n < length xs \implies (tl xs) ! n = xs ! Suc n$
 ⟨proof⟩

lemma *hd-eq-first*: $xs \neq [] \implies xs ! 0 = hd xs$
 ⟨proof⟩

corollary *take-first*: $xs \neq [] \implies xs \downarrow (Suc 0) = [xs ! 0]$
 ⟨proof⟩

corollary *take-hd*: $xs \neq [] \implies xs \downarrow (Suc 0) = [hd xs]$
 ⟨proof⟩

theorem *last-nth*: $xs \neq [] \implies last xs = xs ! (length xs - Suc 0)$

$\langle proof \rangle$

lemma *last-take*: $n < \text{length } xs \implies \text{last } (xs \downarrow \text{Suc } n) = xs ! n$

$\langle proof \rangle$

corollary *last-take2*:

$\llbracket 0 < n; n \leq \text{length } xs \rrbracket \implies \text{last } (xs \downarrow n) = xs ! (n - \text{Suc } 0)$

$\langle proof \rangle$

corollary *nth-0-drop*: $n \leq \text{length } xs \implies (xs \uparrow n) ! 0 = xs ! n$

$\langle proof \rangle$

lemma *drop-eq-tl*: $xs \uparrow (\text{Suc } 0) = \text{tl } xs$

$\langle proof \rangle$

lemma *drop-take-1*:

$n < \text{length } xs \implies xs \uparrow n \downarrow (\text{Suc } 0) = [xs ! n]$

$\langle proof \rangle$

lemma *upt-append*: $m \leq n \implies [0..<m] @ [m..<n] = [0..<n]$

$\langle proof \rangle$

lemma *nth-append1*: $n < \text{length } xs \implies (xs @ ys) ! n = xs ! n$

$\langle proof \rangle$

lemma *nth-append2*: $\text{length } xs \leq n \implies (xs @ ys) ! n = ys ! (n - \text{length } xs)$

$\langle proof \rangle$

lemma *list-all-conv*: $\text{list-all } P \text{ } xs = (\forall i < \text{length } xs. P (xs ! i))$

$\langle proof \rangle$

lemma *expand-list-eq*:

$\bigwedge ys. (xs = ys) = (\text{length } xs = \text{length } ys \wedge (\forall i < \text{length } xs. xs ! i = ys ! i))$

$\langle proof \rangle$

lemmas *list-eq-iff* = *expand-list-eq*

lemma *list-take-drop-imp-eq*:

$\llbracket xs \downarrow n = ys \downarrow n; xs \uparrow n = ys \uparrow n \rrbracket \implies xs = ys$

$\langle proof \rangle$

lemma *list-take-drop-eq-conv*:

$(xs = ys) = (\exists n. (xs \downarrow n = ys \downarrow n \wedge xs \uparrow n = ys \uparrow n))$

$\langle proof \rangle$

lemma *list-take-eq-conv*: $(xs = ys) = (\forall n. xs \downarrow n = ys \downarrow n)$

$\langle proof \rangle$

lemma *list-drop-eq-conv*: $(xs = ys) = (\forall n. xs \uparrow n = ys \uparrow n)$

<proof>

abbreviation *replicate'* :: 'a ⇒ nat ⇒ 'a list (- [1000,65])
where $x^n \equiv \text{replicate } n \ x$

lemma *replicate-snoc*: $x^n @ [x] = x^{\text{Suc } n}$
<proof>

lemma *eq-replicate-conv*: $(\forall i < \text{length } xs. xs ! i = m) = (xs = m^{\text{length } xs})$
<proof>

lemma *replicate-Cons-length*: $\text{length } (x \# a^n) = \text{Suc } n$
<proof>

lemma *replicate-pred-Cons-length*: $0 < n \implies \text{length } (x \# a^n - \text{Suc } 0) = n$
<proof>

lemma *replicate-le-diff*: $m \leq n \implies x^m @ x^{n-m} = x^n$
<proof>

lemma *replicate-le-diff2*: $\llbracket k \leq m; m \leq n \rrbracket \implies x^{m-k} @ x^{n-m} = x^{n-k}$
<proof>

lemma *append-constant-length-induct-aux*: $\bigwedge xs.$
 $\llbracket \text{length } xs \text{ div } k = n; \bigwedge ys. k = 0 \vee \text{length } ys < k \implies P \ ys;$
 $\bigwedge xs \ ys. \llbracket \text{length } xs = k; P \ ys \rrbracket \implies P \ (xs @ ys) \rrbracket \implies P \ xs$
<proof>

lemma *append-constant-length-induct*:
 $\llbracket \bigwedge ys. k = 0 \vee \text{length } ys < k \implies P \ ys;$
 $\bigwedge xs \ ys. \llbracket \text{length } xs = k; P \ ys \rrbracket \implies P \ (xs @ ys) \rrbracket \implies P \ xs$
<proof>

lemma *zip-swap*: $\text{map } (\lambda(y,x). (x,y)) (\text{zip } ys \ xs) = (\text{zip } xs \ ys)$
<proof>

lemma *zip-takeL*: $(\text{zip } xs \ ys) \downarrow n = \text{zip } (xs \downarrow n) \ ys$
<proof>

lemma *zip-takeR*: $(\text{zip } xs \ ys) \downarrow n = \text{zip } xs \ (ys \downarrow n)$
<proof>

lemma *zip-take*: $(\text{zip } xs \ ys) \downarrow n = \text{zip } (xs \downarrow n) \ (ys \downarrow n)$
<proof>

lemma *hd-zip*: $\llbracket xs \neq []; ys \neq [] \rrbracket \implies \text{hd } (\text{zip } xs \ ys) = (\text{hd } xs, \text{hd } ys)$
<proof>

lemma *map-id*: $\text{map } id \ xs = xs$
<proof>

lemma *map-id-subst*: $P (\text{map id } xs) \implies P \text{ } xs$
 ⟨proof⟩

lemma *map-one*: $\text{map } f [x] = [f \ x]$
 ⟨proof⟩

lemma *map-last*: $xs \neq [] \implies \text{last } (\text{map } f \ xs) = f (\text{last } \ xs)$
 ⟨proof⟩

lemma *filter-list-all*: $\text{list-all } P \ xs \implies \text{filter } P \ xs = xs$
 ⟨proof⟩

lemma *filter-snoc*: $\text{filter } P (xs \ @ \ [x]) = (\text{if } P \ x \ \text{then } (\text{filter } P \ xs) \ @ \ [x] \ \text{else } \text{filter } P \ xs)$
 ⟨proof⟩

lemma *filter-filter-eq*: $\text{list-all } (\lambda x. P \ x = Q \ x) \ xs \implies \text{filter } P \ xs = \text{filter } Q \ xs$
 ⟨proof⟩

lemma *filter-nth*: $\bigwedge n.$
 $n < \text{length } (\text{filter } P \ xs) \implies$
 $(\text{filter } P \ xs) ! n =$
 $xs ! (\text{LEAST } k.$
 $k < \text{length } xs \wedge$
 $n < \text{card } \{i. i \leq k \wedge i < \text{length } xs \wedge P (xs ! i)\})$
 ⟨proof⟩

9.1.3 Ordered lists

fun *list-ord* :: $('a \Rightarrow 'a \Rightarrow \text{bool}) \Rightarrow ('a::\text{ord}) \text{ list} \Rightarrow \text{bool}$
where
 $\text{list-ord } \text{ord } (x1 \ \# \ x2 \ \# \ xs) = (\text{ord } x1 \ x2 \wedge \text{list-ord } \text{ord } (x2 \ \# \ xs))$
 $| \text{list-ord } \text{ord } xs = \text{True}$

definition *list-asc* :: $('a::\text{ord}) \text{ list} \Rightarrow \text{bool}$ **where**
 $\text{list-asc } xs \equiv \text{list-ord } (\leq) \ xs$

definition *list-strict-asc* :: $('a::\text{ord}) \text{ list} \Rightarrow \text{bool}$ **where**
 $\text{list-strict-asc } xs \equiv \text{list-ord } (<) \ xs$

value *list-asc* [1::nat, 2, 2]

value *list-strict-asc* [1::nat, 2, 2]

definition *list-desc* :: $('a::\text{ord}) \text{ list} \Rightarrow \text{bool}$ **where**
 $\text{list-desc } xs \equiv \text{list-ord } (\geq) \ xs$

definition *list-strict-desc* :: $('a::\text{ord}) \text{ list} \Rightarrow \text{bool}$ **where**
 $\text{list-strict-desc } xs \equiv \text{list-ord } (>) \ xs$

lemma *list-ord-Nil*: $\text{list-ord } \text{ord } []$
 ⟨proof⟩

lemma *list-ord-one*: $list\text{-}ord\ ord\ [x]$
 ⟨proof⟩

lemma *list-ord-Cons*:

$$list\text{-}ord\ ord\ (x\ \#\ xs) = \\ (xs = [] \vee (ord\ x\ (hd\ xs) \wedge list\text{-}ord\ ord\ xs))$$

⟨proof⟩

lemma *list-ord-Cons-imp*: $\llbracket list\text{-}ord\ ord\ xs; ord\ x\ (hd\ xs) \rrbracket \implies list\text{-}ord\ ord\ (x\ \#\ xs)$

⟨proof⟩

lemma *list-ord-append*: $\bigwedge ys.$

$$list\text{-}ord\ ord\ (xs\ @\ ys) = \\ (list\text{-}ord\ ord\ xs \wedge \\ (ys = [] \vee (list\text{-}ord\ ord\ ys \wedge (xs = [] \vee ord\ (last\ xs)\ (hd\ ys))))))$$

⟨proof⟩

lemma *list-ord-snoc*:

$$list\text{-}ord\ ord\ (xs\ @\ [x]) = \\ (xs = [] \vee (ord\ (last\ xs)\ x \wedge list\text{-}ord\ ord\ xs))$$

⟨proof⟩

lemma *list-ord-all-conv*:

$$(list\text{-}ord\ ord\ xs) = (\forall n < length\ xs - 1. ord\ (xs\ !\ n)\ (xs\ !\ Suc\ n))$$

⟨proof⟩

lemma *list-ord-imp*:

$$\llbracket \bigwedge x\ y. ord\ x\ y \implies ord'\ x\ y; list\text{-}ord\ ord\ xs \rrbracket \implies \\ list\text{-}ord\ ord'\ xs$$

⟨proof⟩

corollary *list-strict-asc-imp-list-asc*:

$$list\text{-}strict\text{-}asc\ (xs::'a::preorder\ list) \implies list\text{-}asc\ xs$$

⟨proof⟩

corollary *list-strict-desc-imp-list-desc*:

$$list\text{-}strict\text{-}desc\ (xs::'a::preorder\ list) \implies list\text{-}desc\ xs$$

⟨proof⟩

lemma *list-ord-trans-imp*: $\bigwedge i.$

$$\llbracket transP\ ord; list\text{-}ord\ ord\ xs; j < length\ xs; i < j \rrbracket \implies \\ ord\ (xs\ !\ i)\ (xs\ !\ j)$$

⟨proof⟩

lemma *list-ord-trans*:

$$transP\ ord \implies \\ (list\text{-}ord\ ord\ xs) = \\ (\forall j < length\ xs. \forall i < j. ord\ (xs\ !\ i)\ (xs\ !\ j))$$

⟨proof⟩

lemma *list-ord-trans-refl-le*:

$$\llbracket transP\ ord; reflP\ ord \rrbracket \implies \\ (list\text{-}ord\ ord\ xs) = \\ (\forall j < length\ xs. \forall i \leq j. ord\ (xs\ !\ i)\ (xs\ !\ j))$$

$\langle \text{proof} \rangle$

lemma *list-ord-trans-refl-le-imp*:

$\llbracket \text{transP } \text{ord}; \bigwedge x y. \text{ord } x y \implies \text{ord}' x y; \text{reflP } \text{ord}';$
 $\text{list-ord } \text{ord } xs \rrbracket \implies$
 $(\forall j < \text{length } xs. \forall i \leq j. \text{ord}' (xs ! i) (xs ! j))$
 $\langle \text{proof} \rangle$

corollary

list-asc-trans:
 $(\text{list-asc } (xs::'a::\text{preorder list})) =$
 $(\forall j < \text{length } xs. \forall i < j. xs ! i \leq xs ! j)$ **and**
list-strict-asc-trans:
 $(\text{list-strict-asc } (xs::'a::\text{preorder list})) =$
 $(\forall j < \text{length } xs. \forall i < j. xs ! i < xs ! j)$ **and**
list-desc-trans:
 $(\text{list-desc } (xs::'a::\text{preorder list})) =$
 $(\forall j < \text{length } xs. \forall i < j. xs ! j \leq xs ! i)$ **and**
list-strict-desc-trans:
 $(\text{list-strict-desc } (xs::'a::\text{preorder list})) =$
 $(\forall j < \text{length } xs. \forall i < j. xs ! j < xs ! i)$
 $\langle \text{proof} \rangle$

corollary

list-asc-trans-le:
 $(\text{list-asc } (xs::'a::\text{preorder list})) =$
 $(\forall j < \text{length } xs. \forall i \leq j. xs ! i \leq xs ! j)$ **and**
list-desc-trans-le:
 $(\text{list-desc } (xs::'a::\text{preorder list})) =$
 $(\forall j < \text{length } xs. \forall i \leq j. xs ! j \leq xs ! i)$
 $\langle \text{proof} \rangle$

corollary

list-strict-asc-trans-le:
 $(\text{list-strict-asc } (xs::'a::\text{preorder list})) \implies$
 $(\forall j < \text{length } xs. \forall i \leq j. xs ! i \leq xs ! j)$
 $\langle \text{proof} \rangle$

lemma *list-ord-le-sorted-eq*: $\text{list-asc } xs = \text{sorted } xs$

$\langle \text{proof} \rangle$

corollary *list-asc-upto*: $\text{list-asc } [m..n]$

$\langle \text{proof} \rangle$

lemma *list-strict-asc-upt*: $\text{list-strict-asc } [m..<n]$

$\langle \text{proof} \rangle$

lemma *list-ord-distinct-aux*:

$$\llbracket \text{irrefl } \{(a, b). \text{ord } a \ b\}; \text{transP } \text{ord}; \text{list-ord } \text{ord } xs; \\ i < \text{length } xs; j < \text{length } xs; i < j \rrbracket \implies \\ xs ! i \neq xs ! j$$
 <proof>

lemma *list-ord-distinct*:

$$\llbracket \text{irrefl } \{(a, b). \text{ord } a \ b\}; \text{transP } \text{ord}; \text{list-ord } \text{ord } xs \rrbracket \implies \\ \text{distinct } xs$$
 <proof>

lemma *list-strict-asc-distinct*: *list-strict-asc* (*xs*::'*a*::*preorder list*) \implies *distinct xs*
<proof>

lemma *list-strict-desc-distinct*: *list-strict-desc* (*xs*::'*a*::*preorder list*) \implies *distinct xs*
<proof>

9.1.4 Additional definitions and results for sublists

primrec *sublist-list* :: '*a list* \Rightarrow *nat list* \Rightarrow '*a list*

where

$$\text{sublist-list } xs \ [] = [] \\ | \text{sublist-list } xs \ (y \# \ ys) = (xs ! y) \# (\text{sublist-list } xs \ ys)$$

value *sublist-list* [0::int,10::int,20,30,40,50] [1::nat,2,3]
value *sublist-list* [0::int,10::int,20,30,40,50] [1::nat,1,2,3]
value [nbe] *sublist-list* [0::int,10::int,20,30,40,50] [1::nat,1,2,3,10]

lemma *sublist-list-length*: *length* (*sublist-list xs ys*) = *length ys*
<proof>

lemma *sublist-list-append*:

$$\bigwedge zs. \text{sublist-list } xs \ (ys \ @ \ zs) = \text{sublist-list } xs \ ys \ @ \ \text{sublist-list } xs \ zs$$
 <proof>

lemma *sublist-list-Nil*: *sublist-list xs []* = []
<proof>

lemma *sublist-list-is-Nil-conv*:

$$(\text{sublist-list } xs \ ys = []) = (ys = [])$$
 <proof>

lemma *sublist-list-eq-imp-length-eq*:

$$\text{sublist-list } xs \ ys = \text{sublist-list } xs \ zs \implies \text{length } ys = \text{length } zs$$
 <proof>

lemma *sublist-list-nth*:

$$\bigwedge n. n < \text{length } ys \implies \text{sublist-list } xs \ ys ! n = xs ! (ys ! n)$$
 <proof>

lemma *take-drop-eq-sublist-list*:

$m + n \leq \text{length } xs \implies xs \uparrow m \downarrow n = \text{sublist-list } xs [m..<m+n]$
 <proof>

primrec *sublist-list-if* :: 'a list \Rightarrow nat list \Rightarrow 'a list

where

$\text{sublist-list-if } xs [] = []$
 | $\text{sublist-list-if } xs (y \# ys) =$
 (if $y < \text{length } xs$ then $(xs ! y) \# (\text{sublist-list-if } xs ys)$
 else $(\text{sublist-list-if } xs ys)$)

value *sublist-list-if* [0::int,10::int,20,30,40,50] [1::nat,2,3]

value *sublist-list-if* [0::int,10::int,20,30,40,50] [1::nat,1,2,3]

value *sublist-list-if* [0::int,10::int,20,30,40,50] [1::nat,1,2,3,10]

lemma *sublist-list-if-sublist-list-filter-conv*: $\bigwedge xs.$

$\text{sublist-list-if } xs ys = \text{sublist-list } xs (\text{filter } (\lambda i. i < \text{length } xs) ys)$
 <proof>

corollary *sublist-list-if-sublist-list-eq*: $\bigwedge xs.$

$\text{list-all } (\lambda i. i < \text{length } xs) ys \implies$
 $\text{sublist-list-if } xs ys = \text{sublist-list } xs ys$
 <proof>

corollary *sublist-list-if-sublist-list-eq2*: $\bigwedge xs.$

$\forall n < \text{length } ys. ys ! n < \text{length } xs \implies$
 $\text{sublist-list-if } xs ys = \text{sublist-list } xs ys$
 <proof>

lemma *sublist-list-if-Nil-left*: $\text{sublist-list-if } [] ys = []$

<proof>

lemma *sublist-list-if-Nil-right*: $\text{sublist-list-if } xs [] = []$

<proof>

lemma *sublist-list-if-length*:

$\text{length } (\text{sublist-list-if } xs ys) = \text{length } (\text{filter } (\lambda i. i < \text{length } xs) ys)$
 <proof>

lemma *sublist-list-if-append*:

$\text{sublist-list-if } xs (ys @ zs) = \text{sublist-list-if } xs ys @ \text{sublist-list-if } xs zs$
 <proof>

lemma *sublist-list-if-snoc*:

$\text{sublist-list-if } xs (ys @ [y]) = \text{sublist-list-if } xs ys @ (\text{if } y < \text{length } xs \text{ then } [xs ! y]$
 else $[])$
 <proof>

lemma *sublist-list-if-is-Nil-conv*:

$(\text{sublist-list-if } xs ys = []) = (\text{list-all } (\lambda i. \text{length } xs \leq i) ys)$

<proof>

lemma *sublist-list-if-nth*:

$n < \text{length } ((\text{filter } (\lambda i. i < \text{length } xs) ys)) \implies$
 $\text{sublist-list-if } xs \text{ } ys ! n = xs ! ((\text{filter } (\lambda i. i < \text{length } xs) ys) ! n)$
<proof>

lemma *take-drop-eq-sublist-list-if*:

$m + n \leq \text{length } xs \implies xs \uparrow m \downarrow n = \text{sublist-list-if } xs [m..<m+n]$
<proof>

lemma *nths-empty-conv*: $(\text{nths } xs \ I = []) = (\forall i \in I. \text{length } xs \leq i)$

<proof>

lemma *nths-singleton2*: $\text{nths } xs \ \{y\} = (\text{if } y < \text{length } xs \text{ then } [xs ! y] \text{ else } [])$

<proof>

lemma *nths-take-eq*:

$[[\text{finite } I; \text{Max } I < n] \implies \text{nths } (xs \downarrow n) \ I = \text{nths } xs \ I$
<proof>

lemma *nths-drop-eq*:

$n \leq iMin \ I \implies \text{nths } (xs \uparrow n) \ \{j. j + n \in I\} = \text{nths } xs \ I$
<proof>

lemma *nths-cut-less-eq*:

$\text{length } xs \leq n \implies \text{nths } xs \ (I \downarrow < n) = \text{nths } xs \ I$
<proof>

lemma *nths-disjoint-Un*:

$[[\text{finite } A; \text{Max } A < iMin \ B] \implies \text{nths } xs \ (A \cup B) = \text{nths } xs \ A \ @ \ \text{nths } xs \ B$
<proof>

corollary *nths-disjoint-insert-left*:

$[[\text{finite } I; x < iMin \ I] \implies \text{nths } xs \ (\text{insert } x \ I) = \text{nths } xs \ \{x\} \ @ \ \text{nths } xs \ I$
<proof>

corollary *nths-disjoint-insert-right*:

$[[\text{finite } I; \text{Max } I < x] \implies \text{nths } xs \ (\text{insert } x \ I) = \text{nths } xs \ I \ @ \ \text{nths } xs \ \{x\}$
<proof>

lemma *nths-all*: $\{..<\text{length } xs\} \subseteq I \implies \text{nths } xs \ I = xs$

<proof>

corollary *nths-UNIV*: $\text{nths } xs \ UNIV = xs$

<proof>

lemma *sublist-list-nths-eq*: $\bigwedge xs.$

$\text{list-strict-asc } ys \implies \text{sublist-list-if } xs \ ys = \text{nths } xs \ (\text{set } ys)$

<proof>

lemma *set-sublist-list-if*: $\bigwedge xs. \text{set } (\text{sublist-list-if } xs \ ys) = \{xs \ ! \ i \mid i < \text{length } xs \wedge i \in \text{set } ys\}$

<proof>

lemma *set-sublist-list*:

list-all $(\lambda i. i < \text{length } xs) \ ys \implies$

$\text{set } (\text{sublist-list } xs \ ys) = \{xs \ ! \ i \mid i < \text{length } xs \wedge i \in \text{set } ys\}$

<proof>

lemma *set-sublist-list-if-eq-set-sublist*: $\text{set } (\text{sublist-list-if } xs \ ys) = \text{set } (\text{nths } xs \ (\text{set } ys))$

<proof>

lemma *set-sublist-list-eq-set-sublist*:

list-all $(\lambda i. i < \text{length } xs) \ ys \implies$

$\text{set } (\text{sublist-list } xs \ ys) = \text{set } (\text{nths } xs \ (\text{set } ys))$

<proof>

9.1.5 Natural set images with lists

definition *f-image* :: 'a list \Rightarrow nat set \Rightarrow 'a set (infixr ^f 90)

where $xs \ ^f A \equiv \{y. \exists n \in A. n < \text{length } xs \wedge y = xs \ ! \ n\}$

abbreviation *f-range* :: 'a list \Rightarrow 'a set

where $f\text{-range } xs \equiv f\text{-image } xs \ UNIV$

lemma *f-image-eqI*[*simp, intro*]:

$\llbracket x = xs \ ! \ n; n \in A; n < \text{length } xs \rrbracket \implies x \in xs \ ^f A$

<proof>

lemma *f-imageI*: $\llbracket n \in A; n < \text{length } xs \rrbracket \implies xs \ ! \ n \in xs \ ^f A$

<proof>

lemma *rev-f-imageI*: $\llbracket n \in A; n < \text{length } xs; x = xs \ ! \ n \rrbracket \implies x \in xs \ ^f A$

<proof>

lemma *f-imageE*[*elim!*]:

$\llbracket x \in xs \ ^f A; \bigwedge n. \llbracket x = xs \ ! \ n; n \in A; n < \text{length } xs \rrbracket \implies P \rrbracket \implies P$

<proof>

lemma *f-image-Un*: $xs \ ^f (A \cup B) = xs \ ^f A \cup xs \ ^f B$

<proof>

lemma *f-image-mono*: $A \subseteq B \implies xs \ ^f A \subseteq xs \ ^f B$

<proof>

lemma *f-image-iff*: $(x \in xs \ ^f A) = (\exists n \in A. n < \text{length } xs \wedge x = xs \ ! \ n)$

<proof>

lemma *f-image-subset-iff*:

$$(xs \text{ 'f } A \subseteq B) = (\forall n \in A. n < \text{length } xs \longrightarrow xs ! n \in B)$$

<proof>

lemma *subset-f-image-iff*: $(B \subseteq xs \text{ 'f } A) = (\exists A' \subseteq A. B = xs \text{ 'f } A')$

<proof>

lemma *f-image-subsetI*:

$$\llbracket \bigwedge n. n \in A \wedge n < \text{length } xs \implies xs ! n \in B \rrbracket \implies xs \text{ 'f } A \subseteq B$$

<proof>

lemma *f-image-empty*: $xs \text{ 'f } \{\} = \{\}$

<proof>

lemma *f-image-insert-if*:

$$xs \text{ 'f } (\text{insert } n \ A) = (\text{if } n < \text{length } xs \text{ then insert } (xs ! n) \ (xs \text{ 'f } A) \text{ else } (xs \text{ 'f } A))$$

<proof>

lemma *f-image-insert-eq1*:

$$n < \text{length } xs \implies xs \text{ 'f } (\text{insert } n \ A) = \text{insert } (xs ! n) \ (xs \text{ 'f } A)$$

<proof>

lemma *f-image-insert-eq2*:

$$\text{length } xs \leq n \implies xs \text{ 'f } (\text{insert } n \ A) = (xs \text{ 'f } A)$$

<proof>

lemma *insert-f-image*:

$$\llbracket n \in A; n < \text{length } xs \rrbracket \implies \text{insert } (xs ! n) \ (xs \text{ 'f } A) = (xs \text{ 'f } A)$$

<proof>

lemma *f-image-is-empty*: $(xs \text{ 'f } A = \{\}) = (\{x. x \in A \wedge x < \text{length } xs\} = \{\})$

<proof>

lemma *f-image-Collect*: $xs \text{ 'f } \{n. P \ n\} = \{xs ! n \mid n. P \ n \wedge n < \text{length } xs\}$

<proof>

lemma *f-image-eq-set*: $\forall n < \text{length } xs. n \in A \implies xs \text{ 'f } A = \text{set } xs$

<proof>

lemma *f-range-eq-set*: $f\text{-range } xs = \text{set } xs$

<proof>

lemma *f-image-eq-set-nths*: $xs \text{ 'f } A = \text{set } (\text{nths } xs \ A)$

<proof>

lemma *f-image-eq-set-sublist-list-if*: $xs \text{ 'f } (\text{set } ys) = \text{set } (\text{sublist-list-if } xs \ ys)$

<proof>

lemma *f-image-eq-set-sublist-list*:

list-all $(\lambda i. i < \text{length } xs) \text{ } ys \implies xs \text{ } ^f \text{ } (\text{set } ys) = \text{set } (\text{sublist-list } xs \text{ } ys)$
 ⟨proof⟩

lemma *f-range-eqI*: $\llbracket x = xs ! n; n < \text{length } xs \rrbracket \implies x \in \text{f-range } xs$
 ⟨proof⟩

lemma *f-rangeI*: $n < \text{length } xs \implies xs ! n \in \text{f-range } xs$
 ⟨proof⟩

lemma *f-rangeE[elim?]*:

$\llbracket x \in \text{f-range } xs; \bigwedge n. \llbracket n < \text{length } xs; x = xs ! n \rrbracket \implies P \rrbracket \implies P$
 ⟨proof⟩

9.1.6 Mapping lists of functions to lists

primrec *map-list* :: $('a \Rightarrow 'b) \text{ list} \Rightarrow 'a \text{ list} \Rightarrow 'b \text{ list}$

where

map-list [] *xs* = []
 | *map-list* (f # *fs*) *xs* = f (hd *xs*) # *map-list* *fs* (tl *xs*)

lemma *map-list-Nil*: *map-list* [] *xs* = []
 ⟨proof⟩

lemma *map-list-Cons-Cons*:

map-list (f # *fs*) (x # *xs*) =
 (f x) # *map-list* *fs* *xs*
 ⟨proof⟩

lemma *map-list-length*: $\bigwedge xs.$

$\text{length } (\text{map-list } fs \text{ } xs) = \text{length } fs$
 ⟨proof⟩

corollary *map-list-empty-conv*:

$(\text{map-list } fs \text{ } xs = []) = (fs = [])$
 ⟨proof⟩

corollary *map-list-not-empty-conv*:

$(\text{map-list } fs \text{ } xs \neq []) = (fs \neq [])$
 ⟨proof⟩

lemma *map-list-nth*: $\bigwedge n \text{ } xs.$

$\llbracket n < \text{length } fs; n < \text{length } xs \rrbracket \implies$
 $(\text{map-list } fs \text{ } xs ! n) =$
 $(fs ! n) (xs ! n)$
 ⟨proof⟩

lemma *map-list-xs-take*: $\bigwedge n \text{ } xs.$

$\text{length } fs \leq n \implies$
 $\text{map-list } fs (xs \downarrow n) =$

map-list fs xs
 ⟨proof⟩

lemma *map-list-take*: $\bigwedge n xs.$

$(\text{map-list } fs \ xs) \downarrow n =$
 $(\text{map-list } (fs \downarrow n) \ xs)$

⟨proof⟩

lemma *map-list-take-take*: $\bigwedge n xs.$

$(\text{map-list } fs \ xs) \downarrow n =$
 $(\text{map-list } (fs \downarrow n) \ (xs \downarrow n))$

⟨proof⟩

lemma *map-list-drop*: $\bigwedge n xs.$

$(\text{map-list } fs \ xs) \uparrow n =$
 $(\text{map-list } (fs \uparrow n) \ (xs \uparrow n))$

⟨proof⟩

lemma *map-list-append-append*: $\bigwedge xs1 .$

$\text{length } fs1 = \text{length } xs1 \implies$
 $\text{map-list } (fs1 \ @ \ fs2) \ (xs1 \ @ \ xs2) =$
 $\text{map-list } fs1 \ xs1 \ @$
 $\text{map-list } fs2 \ xs2$

⟨proof⟩

lemma *map-list-snoc-snoc*:

$\text{length } fs = \text{length } xs \implies$
 $\text{map-list } (fs \ @ \ [f]) \ (xs \ @ \ [x]) =$
 $\text{map-list } fs \ xs \ @ \ [f \ x]$

⟨proof⟩

lemma *map-list-snoc*: $\bigwedge xs.$

$\text{length } fs < \text{length } xs \implies$
 $\text{map-list } (fs \ @ \ [f]) \ xs =$
 $\text{map-list } fs \ xs \ @ \ [f \ (xs \ ! \ (\text{length } fs))]$

⟨proof⟩

lemma *map-list-Cons-if*:

$\text{map-list } fs \ (x \ \# \ xs) =$
 $(\text{if } (fs = []) \ \text{then } [] \ \text{else } ($
 $((\text{hd } fs) \ x) \ \# \ \text{map-list } (\text{tl } fs) \ xs))$

⟨proof⟩

lemma *map-list-Cons-not-empty*:

$fs \neq [] \implies$
 $\text{map-list } fs \ (x \ \# \ xs) =$
 $((\text{hd } fs) \ x) \ \# \ \text{map-list } (\text{tl } fs) \ xs$

⟨proof⟩

lemma *map-eq-map-list-take*: $\bigwedge xs.$

$\llbracket \text{length } fs \leq \text{length } xs; \text{list-all } (\lambda x. x = f) \ fs \rrbracket \implies$

$map\text{-}list\ f\ xs = map\ f\ (xs\ \downarrow\ length\ fs)$

$\langle proof \rangle$

lemma *map-eq-map-list-take2*:

$\llbracket length\ fs = length\ xs; list\text{-}all\ (\lambda x. x = f)\ fs \rrbracket \implies$

$map\text{-}list\ f\ xs = map\ f\ xs$

$\langle proof \rangle$

lemma *map-eq-map-list-replicate*:

$map\text{-}list\ (f^{length\ xs})\ xs = map\ f\ xs$

$\langle proof \rangle$

9.1.7 Mapping functions with two arguments to lists

primrec *map2* ::

— Function taking two parameters

$'a \Rightarrow 'b \Rightarrow 'c \Rightarrow$

— Lists of parameters

$'a\ list \Rightarrow 'b\ list \Rightarrow$

$'c\ list$

where

$map2\ f\ []\ ys = []$

$| map2\ f\ (x\ \#\ xs)\ ys = f\ x\ (hd\ ys)\ \# map2\ f\ xs\ (tl\ ys)$

lemma *map2-map-list-conv*: $\bigwedge ys. map2\ f\ xs\ ys = map\text{-}list\ (map\ f\ xs)\ ys$

$\langle proof \rangle$

lemma *map2-Nil*: $map2\ f\ []\ ys = []$

$\langle proof \rangle$

lemma *map2-Cons-Cons*:

$map2\ f\ (x\ \#\ xs)\ (y\ \#\ ys) =$

$(f\ x\ y)\ \# map2\ f\ xs\ ys$

$\langle proof \rangle$

lemma *map2-length*: $\bigwedge ys. length\ (map2\ f\ xs\ ys) = length\ xs$

$\langle proof \rangle$

corollary *map2-empty-conv*:

$(map2\ f\ xs\ ys = []) = (xs = [])$

$\langle proof \rangle$

corollary *map2-not-empty-conv*:

$(map2\ f\ xs\ ys \neq []) = (xs \neq [])$

$\langle proof \rangle$

lemma *map2-nth*: $\bigwedge n\ ys.$

$\llbracket n < length\ xs; n < length\ ys \rrbracket \implies$

$(map2\ f\ xs\ ys\ !\ n) =$

$f\ (xs\ !\ n)\ (ys\ !\ n)$

$\langle proof \rangle$

lemma *map2-ys-take*: $\bigwedge n\ ys.$

$length\ xs \leq n \implies$
 $map2\ f\ xs\ (ys\ \downarrow\ n) =$
 $map2\ f\ xs\ ys$
 <proof>

lemma *map2-take*: $\bigwedge n\ ys.$
 $(map2\ f\ xs\ ys)\ \downarrow\ n =$
 $(map2\ f\ (xs\ \downarrow\ n)\ ys)$
 <proof>

lemma *map2-take-take*: $\bigwedge n\ ys.$
 $(map2\ f\ xs\ ys)\ \downarrow\ n =$
 $(map2\ f\ (xs\ \downarrow\ n)\ (ys\ \downarrow\ n))$
 <proof>

lemma *map2-drop*: $\bigwedge n\ ys.$
 $(map2\ f\ xs\ ys)\ \uparrow\ n =$
 $(map2\ f\ (xs\ \uparrow\ n)\ (ys\ \uparrow\ n))$
 <proof>

lemma *map2-append-append*: $\bigwedge ys1 .$
 $length\ xs1 = length\ ys1 \implies$
 $map2\ f\ (xs1\ @\ xs2)\ (ys1\ @\ ys2) =$
 $map2\ f\ xs1\ ys1\ @$
 $map2\ f\ xs2\ ys2$
 <proof>

lemma *map2-snoc-snoc*:
 $length\ xs = length\ ys \implies$
 $map2\ f\ (xs\ @\ [x])\ (ys\ @\ [y]) =$
 $map2\ f\ xs\ ys\ @$
 $[f\ x\ y]$
 <proof>

lemma *map2-snoc*: $\bigwedge ys.$
 $length\ xs < length\ ys \implies$
 $map2\ f\ (xs\ @\ [x])\ ys =$
 $map2\ f\ xs\ ys\ @$
 $[f\ x\ (ys\ !\ (length\ xs))]$
 <proof>

lemma *map2-Cons-if*:
 $map2\ f\ xs\ (y\ \#\ ys) =$
 $(if\ (xs = [])\ then\ []\ else\ ($
 $(f\ (hd\ xs)\ y)\ \# map2\ f\ (tl\ xs)\ ys))$
 <proof>

lemma *map2-Cons-not-empty*:
 $xs \neq [] \implies$

$map2\ f\ xs\ (y\ \# \ ys) =$
 $(f\ (hd\ xs)\ y)\ \# \ map2\ f\ (tl\ xs)\ ys$
 <proof>

lemma *map2-append1-take-drop*:
 $length\ xs1 \leq length\ ys \implies$
 $map2\ f\ (xs1\ @\ xs2)\ ys =$
 $map2\ f\ xs1\ (ys\ \downarrow\ length\ xs1)\ @$
 $map2\ f\ xs2\ (ys\ \uparrow\ length\ xs1)$
 <proof>

lemma *map2-append2-take-drop*:
 $length\ ys1 \leq length\ xs \implies$
 $map2\ f\ xs\ (ys1\ @\ ys2) =$
 $map2\ f\ (xs\ \downarrow\ length\ ys1)\ ys1\ @$
 $map2\ f\ (xs\ \uparrow\ length\ ys1)\ ys2$
 <proof>

lemma *map2-cong*:
 $\llbracket xs1 = xs2; ys1 = ys2; length\ xs2 \leq length\ ys2;$
 $\bigwedge x\ y. \llbracket x \in set\ xs2; y \in set\ ys2 \rrbracket \implies f\ x\ y = g\ x\ y \rrbracket \implies$
 $map2\ f\ xs1\ ys1 = map2\ g\ xs2\ ys2$
 <proof>

lemma *map2-eq-conv*:
 $length\ xs \leq length\ ys \implies$
 $(map2\ f\ xs\ ys = map2\ g\ xs\ ys) = (\forall i < length\ xs. f\ (xs\ !\ i)\ (ys\ !\ i) = g\ (xs\ !\ i)$
 $(ys\ !\ i))$
 <proof>

lemma *map2-replicate*: $map2\ f\ x^n\ y^n = (f\ x\ y)^n$
 <proof>

lemma *map2-zip-conv*: $\bigwedge ys.$
 $length\ xs \leq length\ ys \implies$
 $map2\ f\ xs\ ys = map\ (\lambda(x,y). f\ x\ y)\ (zip\ xs\ ys)$
 <proof>

lemma *map2-rev*: $\bigwedge ys.$
 $length\ xs = length\ ys \implies$
 $rev\ (map2\ f\ xs\ ys) = map2\ f\ (rev\ xs)\ (rev\ ys)$
 <proof>

hide-const (open) *map2*

end

10 Set operations with results of type enat

```
theory InfiniteSet2
imports SetInterval2
begin
```

10.1 Set operations with enat

10.1.1 Basic definitions

definition *icard* :: 'a set \Rightarrow enat
where *icard* A \equiv if finite A then enat (card A) else ∞

10.2 Results for icard

lemma *icard-UNIV-nat*: *icard* (UNIV::nat set) = ∞
 \langle proof \rangle

lemma *icard-finite-conv*: (*icard* A = enat (card A)) = finite A
 \langle proof \rangle

lemma *icard-infinite-conv*: (*icard* A = ∞) = infinite A
 \langle proof \rangle

corollary *icard-finite*: finite A \Longrightarrow *icard* A = enat (card A)
 \langle proof \rangle

corollary *icard-infinite[simp]*: infinite A \Longrightarrow *icard* A = ∞
 \langle proof \rangle

lemma *icard-eq-enat-imp*: *icard* A = enat n \Longrightarrow finite A
 \langle proof \rangle

lemma *icard-eq-Infty-imp*: *icard* A = ∞ \Longrightarrow infinite A
 \langle proof \rangle

lemma *icard-the-enat*: finite A \Longrightarrow the-enat (*icard* A) = card A
 \langle proof \rangle

lemma *icard-eq-enat-imp-card*: *icard* A = enat n \Longrightarrow card A = n
 \langle proof \rangle

lemma *icard-eq-enat-card-conv*: $0 < n \Longrightarrow$ (*icard* A = enat n) = (card A = n)
 \langle proof \rangle

lemma *icard-empty[simp]*: *icard* {} = 0
 \langle proof \rangle

lemma *icard-empty-iff*: (*icard* A = 0) = (A = {})
 \langle proof \rangle

lemmas *icard-empty-iff-enat* = *icard-empty-iff*[unfolded zero-enat-def]

lemma *icard-not-empty-iff*: ($0 <$ *icard* A) = (A \neq {})
 \langle proof \rangle

lemmas *icard-not-empty-iff-enat* = *icard-not-empty-iff*[*unfolded zero-enat-def*]

lemma *icard-singleton*: $\text{icard } \{a\} = \text{eSuc } 0$

<proof>

lemmas *icard-singleton-enat*[*simp*] = *icard-singleton*[*unfolded zero-enat-def*]

lemma *icard-1-imp-singleton*: $\text{icard } A = \text{eSuc } 0 \implies \exists a. A = \{a\}$

<proof>

lemma *icard-1-singleton-conv*: $(\text{icard } A = \text{eSuc } 0) = (\exists a. A = \{a\})$

<proof>

lemma *icard-insert-disjoint*: $x \notin A \implies \text{icard } (\text{insert } x A) = \text{eSuc } (\text{icard } A)$

<proof>

lemma *icard-insert-if*: $\text{icard } (\text{insert } x A) = (\text{if } x \in A \text{ then } \text{icard } A \text{ else } \text{eSuc } (\text{icard } A))$

<proof>

lemmas *icard-0-eq* = *icard-empty-iff*

lemma *icard-Suc-Diff1*: $x \in A \implies \text{eSuc } (\text{icard } (A - \{x\})) = \text{icard } A$

<proof>

lemma *icard-Diff-singleton*: $x \in A \implies \text{icard } (A - \{x\}) = \text{icard } A - 1$

<proof>

lemma *icard-Diff-singleton-if*: $\text{icard } (A - \{x\}) = (\text{if } x \in A \text{ then } \text{icard } A - 1 \text{ else } \text{icard } A)$

<proof>

lemma *icard-insert*: $\text{icard } (\text{insert } x A) = \text{eSuc } (\text{icard } (A - \{x\}))$

<proof>

lemma *icard-insert-le*: $\text{icard } A \leq \text{icard } (\text{insert } x A)$

<proof>

lemma *icard-mono*: $A \subseteq B \implies \text{icard } A \leq \text{icard } B$

<proof>

lemma *not-icard-seteq*: $\exists (A::\text{nat set}) B. (A \subseteq B \wedge \text{icard } B \leq \text{icard } A \wedge \neg A = B)$

<proof>

lemma *not-psubset-icard-mono*: $\exists (A::\text{nat set}) B. A \subset B \wedge \neg \text{icard } A < \text{icard } B$

<proof>

lemma *icard-Un-Int*: $\text{icard } A + \text{icard } B = \text{icard } (A \cup B) + \text{icard } (A \cap B)$

<proof>

lemma *icard-Un-disjoint*: $A \cap B = \{\} \implies \text{icard } (A \cup B) = \text{icard } A + \text{icard } B$

<proof>

lemma *not-icard-Diff-subset*: $\exists (A::\text{nat set}) B. B \subseteq A \wedge \neg \text{icard } (A - B) = \text{icard } A - \text{icard } B$

<proof>

lemma *not-icard-Diff1-less*: $\exists (A::\text{nat set})x. x \in A \wedge \neg \text{icard } (A - \{x\}) < \text{icard } A$

<proof>

lemma *not-icard-Diff2-less*: $\exists (A::\text{nat set})x y. x \in A \wedge y \in A \wedge \neg \text{icard } (A - \{x\} - \{y\}) < \text{icard } A$

<proof>

lemma *icard-Diff1-le*: $\text{icard } (A - \{x\}) \leq \text{icard } A$

<proof>

lemma *icard-psubset*: $\llbracket A \subseteq B; \text{icard } A < \text{icard } B \rrbracket \implies A \subset B$

<proof>

lemma *icard-partition*:

$\llbracket \bigwedge c. c \in C \implies \text{icard } c = k; \bigwedge c1 c2. \llbracket c1 \in C; c2 \in C; c1 \neq c2 \rrbracket \implies c1 \cap c2 = \{\} \rrbracket \implies$

$\text{icard } (\bigcup C) = k * \text{icard } C$

<proof>

lemma *icard-image-le*: $\text{icard } (f \text{ ' } A) \leq \text{icard } A$

<proof>

lemma *icard-image*: $\text{inj-on } f A \implies \text{icard } (f \text{ ' } A) = \text{icard } A$

<proof>

lemma *not-eq-icard-imp-inj-on*: $\exists (f::\text{nat} \Rightarrow \text{nat}) (A::\text{nat set}). \text{icard } (f \text{ ' } A) = \text{icard } A \wedge \neg \text{inj-on } f A$

<proof>

lemma *not-inj-on-iff-eq-icard*: $\exists (f::\text{nat} \Rightarrow \text{nat}) (A::\text{nat set}). \neg (\text{inj-on } f A = (\text{icard } (f \text{ ' } A) = \text{icard } A))$

<proof>

lemma *icard-inj-on-le*: $\llbracket \text{inj-on } f A; f \text{ ' } A \subseteq B \rrbracket \implies \text{icard } A \leq \text{icard } B$

<proof>

lemma *icard-bij-eq*:

$\llbracket \text{inj-on } f A; f \text{ ' } A \subseteq B; \text{inj-on } g B; g \text{ ' } B \subseteq A \rrbracket \implies$

$\text{icard } A = \text{icard } B$

<proof>

lemma *icard-cartesian-product*: $\text{icard } (A \times B) = \text{icard } A * \text{icard } B$

<proof>

lemma *icard-cartesian-product-singleton*: $\text{icard } (\{x\} \times A) = \text{icard } A$
 <proof>

lemma *icard-cartesian-product-singleton-right*: $\text{icard } (A \times \{x\}) = \text{icard } A$
 <proof>

lemma
icard-lessThan: $\text{icard } \{..<u\} = \text{enat } u$ **and**
icard-atMost: $\text{icard } \{..u\} = \text{enat } (\text{Suc } u)$ **and**
icard-atLeastLessThan: $\text{icard } \{l..<u\} = \text{enat } (u - l)$ **and**
icard-atLeastAtMost: $\text{icard } \{l..u\} = \text{enat } (\text{Suc } u - l)$ **and**
icard-greaterThanAtMost: $\text{icard } \{l<..u\} = \text{enat } (u - l)$ **and**
icard-greaterThanLessThan: $\text{icard } \{l<..<u\} = \text{enat } (u - \text{Suc } l)$
 <proof>

lemma *icard-atLeast*: $\text{icard } \{(u::\text{nat})..\} = \infty$
 <proof>

lemma *icard-greaterThan*: $\text{icard } \{(u::\text{nat})<..\} = \infty$
 <proof>

lemma
icard-atLeastZeroLessThan-int: $\text{icard } \{0..<u\} = \text{enat } (\text{nat } u)$ **and**
icard-atLeastLessThan-int: $\text{icard } \{l..<u\} = \text{enat } (\text{nat } (u - l))$ **and**
icard-atLeastAtMost-int: $\text{icard } \{l..u\} = \text{enat } (\text{nat } (u - l + 1))$ **and**
icard-greaterThanAtMost-int: $\text{icard } \{l<..u\} = \text{enat } (\text{nat } (u - l))$
 <proof>

lemma *icard-atLeast-int*: $\text{icard } \{(u::\text{int})..\} = \infty$
 <proof>

lemma *icard-greaterThan-int*: $\text{icard } \{(u::\text{int})<..\} = \infty$
 <proof>

lemma *icard-atMost-int*: $\text{icard } \{..(u::\text{int})\} = \infty$
 <proof>

lemma *icard-lessThan-int*: $\text{icard } \{..<(u::\text{int})\} = \infty$
 <proof>

end

11 Additional definitions and results for lists

theory *ListInf*
imports *List2 ../CommonSet/InfiniteSet2*
begin

11.1 Infinite lists

We define infinite lists as functions over natural numbers, i. e., we use functions $\text{nat} \Rightarrow 'a$ as infinite lists over elements of $'a$. Mapping functions to intervals lists $[m..<n]$ yields common finite lists.

11.1.1 Appending a functions to a list

type-synonym $'a \text{ ilist} = \text{nat} \Rightarrow 'a$

definition $i\text{-append} :: 'a \text{ list} \Rightarrow 'a \text{ ilist} \Rightarrow 'a \text{ ilist}$ (**infixr** $\wedge 65$)
where $xs \wedge f \equiv \lambda n. \text{if } n < \text{length } xs \text{ then } xs ! n \text{ else } f (n - \text{length } xs)$

Synonym for the lemma fun-eq-iff from the HOL library to unify lemma names for finite and infinite lists, providing list-eq-iff for finite and ilist-eq-iff for infinite lists.

lemmas $\text{expand-ilist-eq} = \text{fun-eq-iff}$

lemmas $\text{ilist-eq-iff} = \text{expand-ilist-eq}$

lemma $i\text{-append-nth}: (xs \wedge f) n = (\text{if } n < \text{length } xs \text{ then } xs ! n \text{ else } f (n - \text{length } xs))$

$\langle \text{proof} \rangle$

lemma $i\text{-append-nth1}[\text{simp}]: n < \text{length } xs \Longrightarrow (xs \wedge f) n = xs ! n$

$\langle \text{proof} \rangle$

lemma $i\text{-append-nth2}[\text{simp}]: \text{length } xs \leq n \Longrightarrow (xs \wedge f) n = f (n - \text{length } xs)$

$\langle \text{proof} \rangle$

lemma $i\text{-append-Nil}[\text{simp}]: [] \wedge f = f$

$\langle \text{proof} \rangle$

lemma $i\text{-append-assoc}[\text{simp}]: xs \wedge (ys \wedge f) = (xs @ ys) \wedge f$

$\langle \text{proof} \rangle$

lemma $i\text{-append-Cons}: (x \# xs) \wedge f = [x] \wedge (xs \wedge f)$

$\langle \text{proof} \rangle$

lemma $i\text{-append-eq-i-append-conv}[\text{simp}]:$

$\text{length } xs = \text{length } ys \Longrightarrow$

$(xs \wedge f = ys \wedge g) = (xs = ys \wedge f = g)$

$\langle \text{proof} \rangle$

lemma $i\text{-append-eq-i-append-conv2-aux}:$

$\llbracket xs \wedge f = ys \wedge g; \text{length } xs \leq \text{length } ys \rrbracket \Longrightarrow$

$\exists zs. xs @ zs = ys \wedge f = zs \wedge g$

$\langle \text{proof} \rangle$

lemma $i\text{-append-eq-i-append-conv2}:$

$(xs \wedge f = ys \wedge g) =$

$(\exists zs. xs = ys @ zs \wedge zs \wedge f = g \vee xs @ zs = ys \wedge f = zs \wedge g)$

$\langle \text{proof} \rangle$

lemma *same-i-append-eq*[iff]: $(xs \frown f = xs \frown g) = (f = g)$
 ⟨proof⟩

lemma *NOT-i-append-same-eq*:
 $\neg(\forall xs\ ys\ f. (xs \frown (f::(nat \Rightarrow nat)) = ys \frown f) = (xs = ys))$
 ⟨proof⟩

lemma *i-append-hd*: $(xs \frown f)\ 0 = (\text{if } xs = [] \text{ then } f\ 0 \text{ else } hd\ xs)$
 ⟨proof⟩

lemma *i-append-hd2*[simp]: $xs \neq [] \implies (xs \frown f)\ 0 = hd\ xs$
 ⟨proof⟩

lemma *eq-Nil-i-appendI*: $f = g \implies f = [] \frown g$
 ⟨proof⟩

lemma *i-append-eq-i-appendI*:
 $\llbracket xs\ @\ xs' = ys; f = xs' \frown g \rrbracket \implies xs \frown f = ys \frown g$
 ⟨proof⟩

lemma *o-ext*:
 $(\forall x. (x \in \text{range } h \longrightarrow f\ x = g\ x)) \implies f \circ h = g \circ h$
 ⟨proof⟩

lemma *i-append-o*[simp]: $g \circ (xs \frown f) = (\text{map } g\ xs) \frown (g \circ f)$
 ⟨proof⟩

lemma *o-eq-conv*: $(f \circ h = g \circ h) = (\forall x \in \text{range } h. f\ x = g\ x)$
 ⟨proof⟩

lemma *o-cong*:
 $\llbracket h = i; \bigwedge x. x \in \text{range } i \implies f\ x = g\ x \rrbracket \implies f \circ h = f \circ i$
 ⟨proof⟩

lemma *ex-o-conv*: $(\exists h. g = f \circ h) = (\forall y \in \text{range } g. \exists x. y = f\ x)$
 ⟨proof⟩

lemma *o-inj-on*:
 $\llbracket f \circ g = f \circ h; \text{inj-on } f\ (\text{range } g \cup \text{range } h) \rrbracket \implies g = h$
 ⟨proof⟩

lemma *inj-on-o-eq-o*:
 $\text{inj-on } f\ (\text{range } g \cup \text{range } h) \implies$
 $(f \circ g = f \circ h) = (g = h)$
 ⟨proof⟩

lemma *o-injective*: $\llbracket f \circ g = f \circ h; \text{inj } f \rrbracket \implies g = h$
 ⟨proof⟩

lemma *inj-o-eq-o*: $\text{inj } f \implies (f \circ g = f \circ h) = (g = h)$
 ⟨proof⟩

lemma *inj-oI*: $\text{inj } f \implies \text{inj } (\lambda g. f \circ g)$
 ⟨proof⟩

lemma *inj-oD*: $\text{inj } (\lambda g. f \circ g) \implies \text{inj } f$
 ⟨proof⟩

lemma *inj-o[iff]*: $\text{inj } (\lambda g. f \circ g) = \text{inj } f$
 ⟨proof⟩

lemma *inj-on-oI*:
 $\text{inj-on } f (\bigcup ((\lambda f. \text{range } f) \text{ ` } A)) \implies \text{inj-on } (\lambda g. f \circ g) A$
 ⟨proof⟩

lemma *o-idI*: $\forall x. x \in \text{range } g \longrightarrow f x = x \implies f \circ g = g$
 ⟨proof⟩

lemma *o-fun-upd[simp]*: $y \notin \text{range } g \implies f (y := x) \circ g = f \circ g$
 ⟨proof⟩

lemma *range-i-append[simp]*: $\text{range } (xs \frown f) = \text{set } xs \cup \text{range } f$
 ⟨proof⟩

lemma *set-subset-i-append*: $\text{set } xs \subseteq \text{range } (xs \frown f)$
 ⟨proof⟩

lemma *range-subset-i-append*: $\text{range } f \subseteq \text{range } (xs \frown f)$
 ⟨proof⟩

lemma *range-ConsD*: $y \in \text{range } ([x] \frown f) \implies y = x \vee y \in \text{range } f$
 ⟨proof⟩

lemma *range-o [simp]*: $\text{range } (f \circ g) = f \text{ ` } \text{range } g$
 ⟨proof⟩

lemma *in-range-conv-decomp*:
 $(x \in \text{range } f) = (\exists xs g. f = xs \frown ([x] \frown g))$
 ⟨proof⟩

nth

lemma *i-append-nth-Cons-0[simp]*: $((x \# xs) \frown f) 0 = x$
 ⟨proof⟩

lemma *i-append-nth-Cons-Suc[simp]*:

$((x \# xs) \frown f) (Suc\ n) = (xs \frown f)\ n$
 <proof>

lemma *i-append-nth-Cons*:
 $([x] \frown f)\ n = (\text{case } n \text{ of } 0 \Rightarrow x \mid Suc\ k \Rightarrow f\ k)$
 <proof>

lemma *i-append-nth-Cons'*:
 $([x] \frown f)\ n = (\text{if } n = 0 \text{ then } x \text{ else } f\ (n - Suc\ 0))$
 <proof>

lemma *i-append-nth-length[simp]*: $(xs \frown f)\ (\text{length } xs) = f\ 0$
 <proof>

lemma *i-append-nth-length-plus[simp]*: $(xs \frown f)\ (\text{length } xs + n) = f\ n$
 <proof>

lemma *range-iff*: $(y \in \text{range } f) = (\exists x. y = f\ x)$
 <proof>

lemma *range-ball-nth*: $\forall y \in \text{range } f. P\ y \Longrightarrow P\ (f\ x)$
 <proof>

lemma *all-nth-imp-all-range*: $\llbracket \forall x. P\ (f\ x); y \in \text{range } f \rrbracket \Longrightarrow P\ y$
 <proof>

lemma *all-range-conv-all-nth*: $(\forall y \in \text{range } f. P\ y) = (\forall x. P\ (f\ x))$
 <proof>

lemma *i-append-update1*:
 $n < \text{length } xs \Longrightarrow (xs \frown f)\ (n := x) = xs[n := x] \frown f$
 <proof>

lemma *i-append-update2*:
 $\text{length } xs \leq n \Longrightarrow (xs \frown f)\ (n := x) = xs \frown (f(n - \text{length } xs := x))$
 <proof>

lemma *i-append-update*:
 $(xs \frown f)\ (n := x) =$
 $(\text{if } n < \text{length } xs \text{ then } xs[n := x] \frown f$
 $\text{else } xs \frown (f(n - \text{length } xs := x)))$
 <proof>

lemma *i-append-update-length[simp]*:
 $(xs \frown f)\ (\text{length } xs := y) = xs \frown (f(0 := y))$
 <proof>

lemma *range-update-subset-insert*:
 $\text{range } (f(n := x)) \subseteq \text{insert } x\ (\text{range } f)$

<proof>

lemma *range-update-subsetI*:

$\llbracket \text{range } f \subseteq A; x \in A \rrbracket \implies \text{range } (f(n := x)) \subseteq A$
<proof>

lemma *range-update-memI*: $x \in \text{range } (f(n := x))$

<proof>

11.1.2 *take* and *drop* for infinite lists

The *i-take* operator takes the first n elements of an infinite list, i.e. $i\text{-take } f \ n = [f \ 0, f \ 1, \dots, f \ (n-1)]$. The *i-drop* operator drops the first n elements of an infinite list, i.e. $(i\text{-take } f \ n) \ 0 = f \ n$, $(i\text{-take } f \ n) \ 1 = f \ (n + 1)$, \dots

definition $i\text{-take} :: \text{nat} \Rightarrow 'a \text{ ilist} \Rightarrow 'a \text{ list}$

where $i\text{-take } n \ f \equiv \text{map } f \ [0..<n]$

definition $i\text{-drop} :: \text{nat} \Rightarrow 'a \text{ ilist} \Rightarrow 'a \text{ ilist}$

where $i\text{-drop } n \ f \equiv (\lambda x. f \ (n + x))$

abbreviation $i\text{-take}' :: 'a \text{ ilist} \Rightarrow \text{nat} \Rightarrow 'a \text{ list}$ (**infixl** \Downarrow 100)

where $f \ \Downarrow \ n \equiv i\text{-take } n \ f$

abbreviation $i\text{-drop}' :: 'a \text{ ilist} \Rightarrow \text{nat} \Rightarrow 'a \text{ ilist}$ (**infixl** \Uparrow 100)

where $f \ \Uparrow \ n \equiv i\text{-drop } n \ f$

lemma $f \ \Downarrow \ n = \text{map } f \ [0..<n]$

<proof>

lemma $f \ \Uparrow \ n = (\lambda x. f \ (n + x))$

<proof>

Basic results for *i-take* and *i-drop*

lemma *i-take-first*: $f \ \Downarrow \ \text{Suc } 0 = [f \ 0]$

<proof>

lemma *i-drop-i-take-1*: $f \ \Uparrow \ n \ \Downarrow \ \text{Suc } 0 = [f \ n]$

<proof>

lemma *i-take-take-eq1*: $m \leq n \implies (f \ \Downarrow \ n) \ \Downarrow \ m = f \ \Downarrow \ m$

<proof>

lemma *i-take-take-eq2*: $n \leq m \implies (f \ \Downarrow \ n) \ \Downarrow \ m = f \ \Downarrow \ n$

<proof>

lemma *i-take-take[simp]*: $(f \ \Downarrow \ n) \ \Downarrow \ m = f \ \Downarrow \ \text{min } n \ m$

<proof>

lemma *i-drop-nth[simp]*: $(s \ \Uparrow \ n) \ x = s \ (n + x)$

<proof>

lemma *i-drop-nth-sub*: $n \leq x \implies (s \uparrow n) (x - n) = s x$
 ⟨proof⟩

theorem *i-take-nth[simp]*: $i < n \implies (f \Downarrow n) ! i = f i$
 ⟨proof⟩

lemma *i-take-length[simp]*: $\text{length} (f \Downarrow n) = n$
 ⟨proof⟩

lemma *i-take-0[simp]*: $f \Downarrow 0 = []$
 ⟨proof⟩

lemma *i-drop-0[simp]*: $f \uparrow 0 = f$
 ⟨proof⟩

lemma *i-take-eq-Nil[simp]*: $(f \Downarrow n = []) = (n = 0)$
 ⟨proof⟩

lemma *i-take-not-empty-conv*: $(f \Downarrow n \neq []) = (0 < n)$
 ⟨proof⟩

lemma *last-i-take*: $\text{last} (f \Downarrow \text{Suc } n) = f n$
 ⟨proof⟩

lemma *last-i-take2*: $0 < n \implies \text{last} (f \Downarrow n) = f (n - \text{Suc } 0)$
 ⟨proof⟩

lemma *nth-0-i-drop*: $(f \uparrow n) 0 = f n$
 ⟨proof⟩

lemma *i-take-const[simp]*: $(\lambda n. x) \Downarrow i = \text{replicate } i x$
 ⟨proof⟩

lemma *i-drop-const[simp]*: $(\lambda n. x) \uparrow i = (\lambda n. x)$
 ⟨proof⟩

lemma *i-append-i-take-eq1*:
 $n \leq \text{length } xs \implies (xs \frown f) \Downarrow n = xs \Downarrow n$
 ⟨proof⟩

lemma *i-append-i-take-eq2*:
 $\text{length } xs \leq n \implies (xs \frown f) \Downarrow n = xs @ (f \Downarrow (n - \text{length } xs))$
 ⟨proof⟩

lemma *i-append-i-take-if*:
 $(xs \frown f) \Downarrow n = (\text{if } n \leq \text{length } xs \text{ then } xs \Downarrow n \text{ else } xs @ (f \Downarrow (n - \text{length } xs)))$
 ⟨proof⟩

lemma *i-append-i-take[simp]*:

$(xs \frown f) \Downarrow n = (xs \Downarrow n) @ (f \Downarrow (n - \text{length } xs))$
 <proof>

lemma *i-append-i-drop-eq1*:

$n \leq \text{length } xs \implies (xs \frown f) \Uparrow n = (xs \Uparrow n) \frown f$
 <proof>

lemma *i-append-i-drop-eq2*:

$\text{length } xs \leq n \implies (xs \frown f) \Uparrow n = f \Uparrow (n - \text{length } xs)$
 <proof>

lemma *i-append-i-drop-if*:

$(xs \frown f) \Uparrow n = (\text{if } n < \text{length } xs \text{ then } (xs \Uparrow n) \frown f \text{ else } f \Uparrow (n - \text{length } xs))$
 <proof>

lemma *i-append-i-drop[simp]*: $(xs \frown f) \Uparrow n = (xs \Uparrow n) \frown (f \Uparrow (n - \text{length } xs))$

<proof>

lemma *i-append-i-take-i-drop-id[simp]*: $(f \Downarrow n) \frown (f \Uparrow n) = f$

<proof>

lemma *ilist-i-take-i-drop-imp-eq*:

$\llbracket f \Downarrow n = g \Downarrow n; f \Uparrow n = g \Uparrow n \rrbracket \implies f = g$
 <proof>

lemma *ilist-i-take-i-drop-eq-conv*:

$(f = g) = (\exists n. (f \Downarrow n = g \Downarrow n \wedge f \Uparrow n = g \Uparrow n))$
 <proof>

lemma *ilist-i-take-eq-conv*: $(f = g) = (\forall n. f \Downarrow n = g \Downarrow n)$

<proof>

lemma *ilist-i-drop-eq-conv*: $(f = g) = (\forall n. f \Uparrow n = g \Uparrow n)$

<proof>

lemma *i-take-the-conv*:

$f \Downarrow k = (\text{THE } xs. \text{length } xs = k \wedge (\exists g. xs \frown g = f))$
 <proof>

lemma *i-drop-the-conv*:

$f \Uparrow k = (\text{THE } g. (\exists xs. \text{length } xs = k \wedge xs \frown g = f))$
 <proof>

lemma *i-take-Suc-append[simp]*:

$((x \# xs) \frown f) \Downarrow \text{Suc } n = x \# ((xs \frown f) \Downarrow n)$
 <proof>

corollary *i-take-Suc-Cons*: $([x] \frown f) \Downarrow \text{Suc } n = x \# (f \Downarrow n)$

<proof>

lemma *i-drop-Suc-append[simp]*: $((x \# xs) \frown f) \uparrow \text{Suc } n = ((xs \frown f) \uparrow n)$
 ⟨proof⟩

corollary *i-drop-Suc-Cons*: $([x] \frown f) \uparrow \text{Suc } n = f \uparrow n$
 ⟨proof⟩

lemma *i-take-Suc*: $f \downarrow \text{Suc } n = f \ 0 \ \# (f \uparrow \text{Suc } 0 \ \downarrow n)$
 ⟨proof⟩

lemma *i-take-Suc-conv-app-nth*: $f \downarrow \text{Suc } n = (f \downarrow n) \ @ \ [f \ n]$
 ⟨proof⟩

lemma *i-drop-i-drop[simp]*: $s \uparrow a \uparrow b = s \uparrow (a + b)$
 ⟨proof⟩

corollary *i-drop-Suc*: $f \uparrow \text{Suc } 0 \uparrow n = f \uparrow \text{Suc } n$
 ⟨proof⟩

lemma *i-take-commute*: $s \downarrow a \downarrow b = s \downarrow b \downarrow a$
 ⟨proof⟩

lemma *i-drop-commute*: $s \uparrow a \uparrow b = s \uparrow b \uparrow a$
 ⟨proof⟩

corollary *i-drop-tl*: $f \uparrow \text{Suc } 0 \uparrow n = f \uparrow n \uparrow \text{Suc } 0$
 ⟨proof⟩

lemma *nth-via-i-drop*: $(f \uparrow n) \ 0 = x \implies f \ n = x$
 ⟨proof⟩

lemma *i-drop-Suc-conv-tl*: $[f \ n] \frown (f \uparrow \text{Suc } n) = f \uparrow n$
 ⟨proof⟩

lemma *i-drop-Suc-conv-tl'*: $([f \ n] \frown f) \uparrow \text{Suc } n = f \uparrow n$
 ⟨proof⟩

lemma *i-take-i-drop*: $f \uparrow m \downarrow n = f \downarrow (n + m) \uparrow m$
 ⟨proof⟩

Appending an interval of a function

lemma *i-take-int-append*:
 $m \leq n \implies (f \downarrow m) \ @ \ \text{map } f \ [m..<n] = f \downarrow n$
 ⟨proof⟩

lemma *i-take-drop-map-empty-iff*: $(f \downarrow n \uparrow m = []) = (n \leq m)$
 ⟨proof⟩

lemma *i-take-drop-map*: $f \downarrow n \uparrow m = \text{map } f \ [m..<n]$

$\langle proof \rangle$

corollary *i-take-drop-append[simp]*:

$$m \leq n \implies (f \Downarrow m) @ (f \Downarrow n \uparrow m) = f \Downarrow n$$

$\langle proof \rangle$

lemma *i-take-drop*: $f \Downarrow n \uparrow m = f \uparrow m \Downarrow (n - m)$
 $\langle proof \rangle$

lemma *i-take-o[simp]*: $(f \circ g) \Downarrow n = map\ f\ (g \Downarrow n)$
 $\langle proof \rangle$

lemma *i-drop-o[simp]*: $(f \circ g) \uparrow n = f \circ (g \uparrow n)$
 $\langle proof \rangle$

lemma *set-i-take-subset*: $set\ (f \Downarrow n) \subseteq range\ f$
 $\langle proof \rangle$

lemma *range-i-drop-subset*: $range\ (f \uparrow n) \subseteq range\ f$
 $\langle proof \rangle$

lemma *in-set-i-takeD*: $x \in set\ (f \Downarrow n) \implies x \in range\ f$
 $\langle proof \rangle$

lemma *in-range-i-takeD*: $x \in range\ (f \uparrow n) \implies x \in range\ f$
 $\langle proof \rangle$

lemma *i-append-eq-conv-conj*:
 $((xs \frown f) = g) = (xs = g \Downarrow length\ xs \wedge f = g \uparrow length\ xs)$
 $\langle proof \rangle$

lemma *i-take-add*: $f \Downarrow (i + j) = (f \Downarrow i) @ (f \uparrow i \Downarrow j)$
 $\langle proof \rangle$

lemma *i-append-eq-i-append-conv-if-aux*:
 $length\ xs \leq length\ ys \implies$
 $(xs \frown f = ys \frown g) = (xs = ys \Downarrow length\ xs \wedge f = (ys \uparrow length\ xs) \frown g)$
 $\langle proof \rangle$

lemma *i-append-eq-i-append-conv-if*:
 $(xs \frown f = ys \frown g) =$
 $(if\ length\ xs \leq length\ ys$
 $\quad then\ xs = ys \Downarrow length\ xs \wedge f = (ys \uparrow length\ xs) \frown g$
 $\quad else\ xs \Downarrow length\ ys = ys \wedge (xs \uparrow length\ ys) \frown f = g)$
 $\langle proof \rangle$

lemma *i-take-hd-i-drop*: $(f \Downarrow n) @ [(f \uparrow n)\ 0] = f \Downarrow Suc\ n$
 $\langle proof \rangle$

lemma *id-i-take-nth-i-drop*: $f = (f \Downarrow n) \frown ([f\ n] \frown f) \Uparrow \text{Suc } n$
 ⟨proof⟩

lemma *upd-conv-i-take-nth-i-drop*:
 $f (n := x) = (f \Downarrow n) \frown ([x] \frown (f \Uparrow \text{Suc } n))$
 ⟨proof⟩

theorem *i-take-induct*:
 $\llbracket P (f \Downarrow 0); \bigwedge n. P (f \Downarrow n) \rrbracket \Longrightarrow P (f \Downarrow \text{Suc } n) \rrbracket \Longrightarrow P (f \Downarrow n)$
 ⟨proof⟩

theorem *take-induct[rule-format]*:
 $\llbracket P (s \Downarrow 0);$
 $\bigwedge n. \llbracket \text{Suc } n < \text{length } s; P (s \Downarrow n) \rrbracket \Longrightarrow P (s \Downarrow \text{Suc } n);$
 $i < \text{length } s \rrbracket$
 $\Longrightarrow P (s \Downarrow i)$
 ⟨proof⟩

theorem *i-drop-induct*:
 $\llbracket P (f \Uparrow 0); \bigwedge n. P (f \Uparrow n) \rrbracket \Longrightarrow P (f \Uparrow \text{Suc } n) \rrbracket \Longrightarrow P (f \Uparrow n)$
 ⟨proof⟩

theorem *f-drop-induct[rule-format]*:
 $\llbracket P (s \Uparrow 0);$
 $\bigwedge n. \llbracket \text{Suc } n < \text{length } s; P (s \Uparrow n) \rrbracket \Longrightarrow P (s \Uparrow \text{Suc } n);$
 $i < \text{length } s \rrbracket$
 $\Longrightarrow P (s \Uparrow i)$
 ⟨proof⟩

lemma *i-take-drop-eq-map*: $f \Uparrow m \Downarrow n = \text{map } f [m..<m+n]$
 ⟨proof⟩

lemma *o-eq-i-append-imp*:
 $f \circ g = ys \frown i \Longrightarrow$
 $\exists xs\ h. g = xs \frown h \wedge \text{map } f\ xs = ys \wedge f \circ h = i$
 ⟨proof⟩

corollary *o-eq-i-append-conv*:
 $(f \circ g = ys \frown i) =$
 $(\exists xs\ h. g = xs \frown h \wedge \text{map } f\ xs = ys \wedge f \circ h = i)$
 ⟨proof⟩

corollary *i-append-eq-o-conv*:
 $(ys \frown i = f \circ g) =$
 $(\exists xs\ h. g = xs \frown h \wedge \text{map } f\ xs = ys \wedge f \circ h = i)$
 ⟨proof⟩

11.1.3 zip for infinite lists

definition $i\text{-zip} :: 'a\ \text{ilist} \Rightarrow 'b\ \text{ilist} \Rightarrow ('a \times 'b)\ \text{ilist}$
where $i\text{-zip}\ f\ g \equiv \lambda n. (f\ n, g\ n)$

lemma $i\text{-zip-nth}$: $(i\text{-zip}\ f\ g)\ n = (f\ n, g\ n)$
 $\langle\text{proof}\rangle$

lemma $i\text{-zip-swap}$: $(\lambda(y, x). (x, y)) \circ i\text{-zip}\ g\ f = i\text{-zip}\ f\ g$
 $\langle\text{proof}\rangle$

lemma $i\text{-zip-i-take}$: $(i\text{-zip}\ f\ g)\ \Downarrow\ n = \text{zip}\ (f\ \Downarrow\ n)\ (g\ \Downarrow\ n)$
 $\langle\text{proof}\rangle$

lemma $i\text{-zip-i-drop}$: $(i\text{-zip}\ f\ g)\ \Uparrow\ n = i\text{-zip}\ (f\ \Uparrow\ n)\ (g\ \Uparrow\ n)$
 $\langle\text{proof}\rangle$

lemma fst-o-izip : $\text{fst} \circ (i\text{-zip}\ f\ g) = f$
 $\langle\text{proof}\rangle$

lemma snd-o-i-izip : $\text{snd} \circ (i\text{-zip}\ f\ g) = g$
 $\langle\text{proof}\rangle$

lemma update-i-izip :
 $(i\text{-zip}\ f\ g)(n := xy) = i\text{-zip}\ (f(n := \text{fst}\ xy))\ (g(n := \text{snd}\ xy))$
 $\langle\text{proof}\rangle$

lemma $i\text{-zip-Cons-Cons}$:
 $i\text{-zip}\ ([x] \frown f)\ ([y] \frown g) = [(x, y)] \frown (i\text{-zip}\ f\ g)$
 $\langle\text{proof}\rangle$

lemma $i\text{-zip-i-append1}$:
 $i\text{-zip}\ (xs \frown f)\ g = \text{zip}\ xs\ (g\ \Downarrow\ \text{length}\ xs) \frown (i\text{-zip}\ f\ (g\ \Uparrow\ \text{length}\ xs))$
 $\langle\text{proof}\rangle$

lemma $i\text{-zip-i-append2}$:
 $i\text{-zip}\ f\ (ys \frown g) = \text{zip}\ (f\ \Downarrow\ \text{length}\ ys)\ ys \frown (i\text{-zip}\ (f\ \Uparrow\ \text{length}\ ys)\ g)$
 $\langle\text{proof}\rangle$

lemma $i\text{-zip-append}$:
 $\text{length}\ xs = \text{length}\ ys \implies$
 $i\text{-zip}\ (xs \frown f)\ (ys \frown g) = \text{zip}\ xs\ ys \frown (i\text{-zip}\ f\ g)$
 $\langle\text{proof}\rangle$

lemma $i\text{-zip-range}$: $\text{range}\ (i\text{-zip}\ f\ g) = \{ (f\ n, g\ n) \mid n. \text{True} \}$
 $\langle\text{proof}\rangle$

lemma $i\text{-zip-update}$:
 $i\text{-zip}\ (f(n := x))\ (g(n := y)) = (i\text{-zip}\ f\ g)(n := (x, y))$
 $\langle\text{proof}\rangle$

lemma *i-zip-const*: $i\text{-zip } (\lambda n. x) (\lambda n. y) = (\lambda n. (x, y))$
 ⟨proof⟩

11.1.4 Mapping functions with two arguments to infinite lists

definition *i-map2* ::
 — Function taking two parameters
 $('a \Rightarrow 'b \Rightarrow 'c) \Rightarrow$
 — Lists of parameters
 $'a \text{ ilist} \Rightarrow 'b \text{ ilist} \Rightarrow$
 $'c \text{ ilist}$

where
 $i\text{-map2 } f \text{ } xs \text{ } ys \equiv \lambda n. f (xs \ n) (ys \ n)$

lemma *i-map2-nth*: $(i\text{-map2 } f \text{ } xs \text{ } ys) \ n = f (xs \ n) (ys \ n)$
 ⟨proof⟩

lemma *i-map2-Cons-Cons*:
 $i\text{-map2 } f ([x] \frown xs) ([y] \frown ys) =$
 $[f \ x \ y] \frown (i\text{-map2 } f \text{ } xs \text{ } ys)$
 ⟨proof⟩

lemma *i-map2-take-ge*:
 $n \leq n1 \implies$
 $i\text{-map2 } f \text{ } xs \text{ } ys \ \Downarrow \ n =$
 $\text{map2 } f (xs \ \Downarrow \ n) (ys \ \Downarrow \ n1)$
 ⟨proof⟩

lemma *i-map2-take-take*:
 $i\text{-map2 } f \text{ } xs \text{ } ys \ \Downarrow \ n =$
 $\text{map2 } f (xs \ \Downarrow \ n) (ys \ \Downarrow \ n)$
 ⟨proof⟩

lemma *i-map2-drop*:
 $(i\text{-map2 } f \text{ } xs \text{ } ys) \ \Uparrow \ n =$
 $(i\text{-map2 } f (xs \ \Uparrow \ n) (ys \ \Uparrow \ n))$
 ⟨proof⟩

lemma *i-map2-append-append*:
 $\text{length } xs1 = \text{length } ys1 \implies$
 $i\text{-map2 } f (xs1 \frown xs) (ys1 \frown ys) =$
 $\text{map2 } f \text{ } xs1 \text{ } ys1 \frown i\text{-map2 } f \text{ } xs \text{ } ys$
 ⟨proof⟩

lemma *i-map2-Cons-left*:
 $i\text{-map2 } f ([x] \frown xs) \text{ } ys =$
 $[f \ x \ (ys \ 0)] \frown i\text{-map2 } f \text{ } xs \text{ } (ys \ \Uparrow \ \text{Suc } 0)$
 ⟨proof⟩

lemma *i-map2-Cons-right*:

$i\text{-map2 } f \text{ } xs \ ([y] \frown ys) =$
 $[f \ (xs \ 0) \ y] \frown i\text{-map2 } f \ (xs \ \uparrow \text{Suc } 0) \ ys$
 $\langle \text{proof} \rangle$

lemma *i-map2-append-take-drop-left*:

$i\text{-map2 } f \ (xs1 \frown xs) \ ys =$
 $\text{map2 } f \ xs1 \ (ys \ \downarrow \text{length } xs1) \frown$
 $i\text{-map2 } f \ xs \ (ys \ \uparrow \text{length } xs1)$
 $\langle \text{proof} \rangle$

lemma *i-map2-append-take-drop-right*:

$i\text{-map2 } f \ xs \ (ys1 \frown ys) =$
 $\text{map2 } f \ (xs \ \downarrow \text{length } ys1) \ ys1 \frown$
 $i\text{-map2 } f \ (xs \ \uparrow \text{length } ys1) \ ys$
 $\langle \text{proof} \rangle$

lemma *i-map2-cong*:

$\llbracket xs1 = xs2; ys1 = ys2;$
 $\bigwedge x \ y. \llbracket x \in \text{range } xs2; y \in \text{range } ys2 \rrbracket \implies f \ x \ y = g \ x \ y \rrbracket \implies$
 $i\text{-map2 } f \ xs1 \ ys1 = i\text{-map2 } g \ xs2 \ ys2$
 $\langle \text{proof} \rangle$

lemma *i-map2-eq-conv*:

$(i\text{-map2 } f \ xs \ ys = i\text{-map2 } g \ xs \ ys) = (\forall i. f \ (xs \ i) \ (ys \ i) = g \ (xs \ i) \ (ys \ i))$
 $\langle \text{proof} \rangle$

lemma *i-map2-replicate*: $i\text{-map2 } f \ (\lambda n. \ x) \ (\lambda n. \ y) = (\lambda n. \ f \ x \ y)$

$\langle \text{proof} \rangle$

lemma *i-map2-i-zip-conv*:

$i\text{-map2 } f \ xs \ ys = (\lambda(x,y). f \ x \ y) \circ (i\text{-zip } xs \ ys)$
 $\langle \text{proof} \rangle$

11.2 Generalised lists as combination of finite and infinite lists

11.2.1 Basic definitions

datatype (*gset*: 'a) *glist* = *FL* 'a *list* | *IL* 'a *ilist* **for** *map*: *gmap*

definition *glength* :: 'a *glist* \Rightarrow *enat*

where

$glength \ a \equiv \text{case } a \text{ of}$
 $\text{FL } xs \Rightarrow \text{enat } (\text{length } xs) \mid$
 $\text{IL } f \Rightarrow \infty$

definition *gCons* :: 'a \Rightarrow 'a *glist* \Rightarrow 'a *glist* (**infixr** #_g 65)

where

$x \ \#_g \ a \equiv \text{case } a \text{ of}$
 $\text{FL } xs \Rightarrow \text{FL } (x \ \# \ xs) \mid$

$$IL\ g \Rightarrow IL\ ([x] \frown g)$$

definition $gappend :: 'a\ glist \Rightarrow 'a\ glist \Rightarrow 'a\ glist$ (**infixr** $@_g$ 65)

where

$$\begin{aligned} gappend\ a\ b &\equiv \text{case } a \text{ of} \\ FL\ xs &\Rightarrow (\text{case } b \text{ of } FL\ ys \Rightarrow FL\ (xs @ ys) \mid IL\ f \Rightarrow IL\ (xs \frown f)) \mid \\ IL\ f &\Rightarrow IL\ f \end{aligned}$$

definition $gtake :: enat \Rightarrow 'a\ glist \Rightarrow 'a\ glist$

where

$$\begin{aligned} gtake\ n\ a &\equiv \text{case } n \text{ of} \\ enat\ m &\Rightarrow FL\ (\text{case } a \text{ of} \\ &FL\ xs \Rightarrow xs \downarrow m \mid \\ &IL\ f \Rightarrow f \Downarrow m) \mid \\ \infty &\Rightarrow a \end{aligned}$$

definition $gdrop :: enat \Rightarrow 'a\ glist \Rightarrow 'a\ glist$

where

$$\begin{aligned} gdrop\ n\ a &\equiv \text{case } n \text{ of} \\ enat\ m &\Rightarrow (\text{case } a \text{ of} \\ &FL\ xs \Rightarrow FL\ (xs \uparrow m) \mid \\ &IL\ f \Rightarrow IL\ (f \uparrow m)) \mid \\ \infty &\Rightarrow FL\ [] \end{aligned}$$

definition $gnth :: 'a\ glist \Rightarrow nat \Rightarrow 'a$ (**infixl** $!_g$ 100)

where

$$\begin{aligned} a\ !_g\ n &\equiv \text{case } a \text{ of} \\ FL\ xs &\Rightarrow xs\ !\ n \mid \\ IL\ f &\Rightarrow f\ n \end{aligned}$$

abbreviation $g\text{-take}' :: 'a\ glist \Rightarrow enat \Rightarrow 'a\ glist$ (**infixl** \downarrow_g 100)

where $a\ \downarrow_g\ n \equiv gtake\ n\ a$

abbreviation $g\text{-drop}' :: 'a\ glist \Rightarrow enat \Rightarrow 'a\ glist$ (**infixl** \uparrow_g 100)

where $a\ \uparrow_g\ n \equiv gdrop\ n\ a$

11.2.2 *glength*

lemma $glength\text{-fin}[simp]$: $glength\ (FL\ xs) = enat\ (length\ xs)$

<proof>

lemma $glength\text{-infin}[simp]$: $glength\ (IL\ f) = \infty$

<proof>

lemma $gappend\text{-glength}[simp]$: $glength\ (a\ @_g\ b) = glength\ a + glength\ b$

<proof>

lemma $gmap\text{-glength}[simp]$: $glength\ (gmap\ f\ a) = glength\ a$

<proof>

lemma *glength-0-conv[simp]*: $(\text{glength } a = 0) = (a = FL [])$
 ⟨proof⟩

lemma *glength-greater-0-conv[simp]*: $(0 < \text{glength } a) = (a \neq FL [])$
 ⟨proof⟩

lemma *glength-gSuc-conv*:
 $(\text{glength } a = eSuc\ n) =$
 $(\exists x\ b. a = x \#_g b \wedge \text{glength } b = n)$
 ⟨proof⟩

lemma *gSuc-glength-conv*:
 $(eSuc\ n = \text{glength } a) =$
 $(\exists x\ b. a = x \#_g b \wedge \text{glength } b = n)$
 ⟨proof⟩

11.2.3 $@_g$ – gappend

lemma *gappend-Nil[simp]*: $(FL []) @_g a = a$
 ⟨proof⟩

lemma *gappend-Nil2[simp]*: $a @_g (FL []) = a$
 ⟨proof⟩

lemma *gappend-is-Nil-conv[simp]*: $(a @_g b = FL []) = (a = FL [] \wedge b = FL [])$
 ⟨proof⟩

lemma *Nil-is-gappend-conv[simp]*: $(FL [] = a @_g b) = (a = FL [] \wedge b = FL [])$
 ⟨proof⟩

lemma *gappend-assoc[simp]*: $(a @_g b) @_g c = a @_g b @_g c$
 ⟨proof⟩

lemma *gappend-infin[simp]*: $IL\ f @_g b = IL\ f$
 ⟨proof⟩

lemma *same-gappend-eq-disj[simp]*: $(a @_g b = a @_g c) = (\text{glength } a = \infty \vee b = c)$
 ⟨proof⟩

lemma *same-gappend-eq*:
 $\text{glength } a < \infty \implies (a @_g b = a @_g c) = (b = c)$
 ⟨proof⟩

11.2.4 *gmap*

lemma *gmap-gappend[simp]*: $gmap\ f (a @_g b) = gmap\ f\ a @_g gmap\ f\ b$
 ⟨proof⟩

lemmas *gmap-gmap[simp]* = *glist.map-comp*

lemma *gmap-eq-conv*[simp]: $(gmap\ f\ a = gmap\ g\ a) = (\forall x \in gset\ a. f\ x = g\ x)$
 ⟨proof⟩

lemmas *gmap-cong* = *glist.map-cong*

lemma *gmap-is-Nil-conv*: $(gmap\ f\ a = FL\ []) = (a = FL\ [])$
 ⟨proof⟩

lemma *gmap-eq-imp-glength-eq*:
 $gmap\ f\ a = gmap\ f\ b \implies glength\ a = glength\ b$
 ⟨proof⟩

11.2.5 *gset*

lemma *gset-gappend*[simp]:
 $gset\ (a\ @_g\ b) =$
 $(case\ a\ of\ FL\ a' \Rightarrow set\ a' \cup gset\ b \mid IL\ a' \Rightarrow range\ a')$
 ⟨proof⟩

lemma *gset-gappend-if*:
 $gset\ (a\ @_g\ b) =$
 $(if\ glength\ a < \infty\ then\ gset\ a \cup gset\ b\ else\ gset\ a)$
 ⟨proof⟩

lemma *gset-empty*[simp]: $(gset\ a = \{\}) = (a = FL\ [])$
 ⟨proof⟩

lemmas *gset-gmap*[simp] = *glist.set-map*

lemma *icard-glength*: $icard\ (gset\ a) \leq glength\ a$
 ⟨proof⟩

11.2.6 $!_g$ – *gnth*

lemma *gnth-gCons-0*[simp]: $(x\ \#_g\ a)\ !_g\ 0 = x$
 ⟨proof⟩

lemma *gnth-gCons-Suc*[simp]: $(x\ \#_g\ a)\ !_g\ Suc\ n = a\ !_g\ n$
 ⟨proof⟩

lemma *gnth-gappend*:
 $(a\ @_g\ b)\ !_g\ n =$
 $(if\ enat\ n < glength\ a\ then\ a\ !_g\ n$
 $else\ b\ !_g\ (n - the-enat\ (glength\ a)))$
 ⟨proof⟩

lemma *gnth-gappend-length-plus*[simp]: $(FL\ xs\ @_g\ b)\ !_g\ (length\ xs + n) = b\ !_g\ n$
 ⟨proof⟩

lemma *gmap-gnth[simp]*: $\text{enat } n < \text{glength } a \implies \text{gmap } f \ a \ !_g \ n = f \ (a \ !_g \ n)$
 ⟨proof⟩

lemma *in-gset-cong-gnth*: $(x \in \text{gset } a) = (\exists i. \text{enat } i < \text{glength } a \wedge a \ !_g \ i = x)$
 ⟨proof⟩

11.2.7 *gtake and gdrop*

lemma *gtake-0[simp]*: $a \ \downarrow_g \ 0 = FL \ []$
 ⟨proof⟩

lemma *gdrop-0[simp]*: $a \ \uparrow_g \ 0 = a$
 ⟨proof⟩

lemma *gtake-Infty[simp]*: $a \ \downarrow_g \ \infty = a$
 ⟨proof⟩

lemma *gdrop-Infty[simp]*: $a \ \uparrow_g \ \infty = FL \ []$
 ⟨proof⟩

lemma *gtake-all[simp]*: $\text{glength } a \leq n \implies a \ \downarrow_g \ n = a$
 ⟨proof⟩

lemma *gdrop-all[simp]*: $\text{glength } a \leq n \implies a \ \uparrow_g \ n = FL \ []$
 ⟨proof⟩

lemma *gtake-eSuc-gCons[simp]*: $(x \ \#_g \ a) \ \downarrow_g \ (eSuc \ n) = x \ \#_g \ a \ \downarrow_g \ n$
 ⟨proof⟩

lemma *gdrop-eSuc-gCons[simp]*: $(x \ \#_g \ a) \ \uparrow_g \ (eSuc \ n) = a \ \uparrow_g \ n$
 ⟨proof⟩

lemma *gtake-eSuc*: $a \neq FL \ [] \implies a \ \downarrow_g \ (eSuc \ n) = a \ !_g \ 0 \ \#_g \ (a \ \uparrow_g \ (eSuc \ 0) \ \downarrow_g \ n)$
 ⟨proof⟩

lemma *gdrop-eSuc*: $a \ \uparrow_g \ (eSuc \ n) = a \ \uparrow_g \ (eSuc \ 0) \ \uparrow_g \ n$
 ⟨proof⟩

lemma *gnth-via-grop*: $a \ \uparrow_g \ (\text{enat } n) = x \ \#_g \ b \implies a \ !_g \ n = x$
 ⟨proof⟩

lemma *gtake-eSuc-conv-gapp-gnth*:
 $\text{enat } n < \text{glength } a \implies a \ \downarrow_g \ \text{enat } (Suc \ n) = a \ \downarrow_g \ (\text{enat } n) \ @_g \ FL \ [a \ !_g \ n]$
 ⟨proof⟩

lemma *gdrop-eSuc-conv-tl*:
 $\text{enat } n < \text{glength } a \implies a \ !_g \ n \ \#_g \ a \ \uparrow_g \ \text{enat } (Suc \ n) = a \ \uparrow_g \ \text{enat } n$
 ⟨proof⟩

lemma *glength-gtake[simp]*: $\text{glength } (a \downarrow_g n) = \min (\text{glength } a) n$
 ⟨proof⟩

lemma *glength-drop[simp]*: $\text{glength } (a \uparrow_g (\text{enat } n)) = \text{glength } a - (\text{enat } n)$
 ⟨proof⟩

end

12 Prefices on finite and infinite lists

theory *ListInf-Prefix*
imports *HOL-Library.Sublist ListInf*
begin

12.1 Additional list prefix results

lemma *prefix-eq-prefix-take-ex*: $\text{prefix } xs \ ys = (\exists n. \ ys \downarrow n = xs)$
 ⟨proof⟩

lemma *prefix-take-eq-prefix-take-ex*: $(\ ys \downarrow (\text{length } xs) = xs) = (\exists n. \ ys \downarrow n = xs)$
 ⟨proof⟩

lemma *prefix-eq-prefix-take*: $\text{prefix } xs \ ys = (\ ys \downarrow (\text{length } xs) = xs)$
 ⟨proof⟩

lemma *strict-prefix-take-eq-strict-prefix-take-ex*:
 $(\ ys \downarrow (\text{length } xs) = xs \wedge xs \neq ys) =$
 $((\exists n. \ ys \downarrow n = xs) \wedge xs \neq ys)$
 ⟨proof⟩

lemma *strict-prefix-eq-strict-prefix-take-ex*: $\text{strict-prefix } xs \ ys = ((\exists n. \ ys \downarrow n = xs) \wedge xs \neq ys)$
 ⟨proof⟩

lemma *strict-prefix-eq-strict-prefix-take*: $\text{strict-prefix } xs \ ys = (\ ys \downarrow (\text{length } xs) = xs \wedge xs \neq ys)$
 ⟨proof⟩

lemma *take-imp-prefix*: $\text{prefix } (xs \downarrow n) \ xs$
 ⟨proof⟩

lemma *eq-imp-prefix*: $xs = (ys::'a \text{ list}) \implies \text{prefix } xs \ ys$
 ⟨proof⟩

lemma *le-take-imp-prefix*: $a \leq b \implies \text{prefix } (xs \downarrow a) \ (xs \downarrow b)$
 ⟨proof⟩

lemma *take-prefix-imp-le*:

$\llbracket a \leq \text{length } xs; \text{prefix } (xs \downarrow a) (xs \downarrow b) \rrbracket \implies a \leq b$
 $\langle \text{proof} \rangle$

lemma *take-prefixeq-le-conv*:

$a \leq \text{length } xs \implies \text{prefix } (xs \downarrow a) (xs \downarrow b) = (a \leq b)$
 $\langle \text{proof} \rangle$

lemma *append-imp-prefix[simp, intro]*: $\text{prefix } a (a @ b)$

$\langle \text{proof} \rangle$

lemma *prefix-imp-take-eq*:

$\llbracket n \leq \text{length } xs; \text{prefix } xs \ ys \rrbracket \implies xs \downarrow n = ys \downarrow n$
 $\langle \text{proof} \rangle$

lemma *prefix-length-le-eq-conv*: $(\text{prefix } xs \ ys \wedge \text{length } ys \leq \text{length } xs) = (xs = ys)$

$\langle \text{proof} \rangle$

lemma *take-length-prefix-conv*:

$\text{length } xs \leq \text{length } ys \implies \text{prefix } (ys \downarrow \text{length } xs) \ xs = \text{prefix } xs \ ys$
 $\langle \text{proof} \rangle$

lemma *append-eq-imp-take*:

$\llbracket k \leq \text{length } xs; \text{length } r1 = k; r1 @ r2 = xs \rrbracket \implies r1 = xs \downarrow k$
 $\langle \text{proof} \rangle$

lemma *take-the-conv*:

$xs \downarrow k = (\text{if } \text{length } xs \leq k \text{ then } xs \text{ else } (\text{THE } r. \text{length } r = k \wedge (\exists r2. r @ r2 = xs)))$
 $\langle \text{proof} \rangle$

lemma *prefix-refl*: $\text{prefix } xs (xs::'a \text{ list})$

$\langle \text{proof} \rangle$

lemma *prefix-trans*: $\llbracket \text{prefix } xs \ ys; \text{prefix } (ys::'a \text{ list}) \ zs \rrbracket \implies \text{prefix } xs \ zs$

$\langle \text{proof} \rangle$

lemma *prefixeq-antisym*: $\llbracket \text{prefix } xs \ ys; \text{prefix } (ys::'a \text{ list}) \ xs \rrbracket \implies xs = ys$

$\langle \text{proof} \rangle$

12.2 Counting equal pairs

Counting number of equal elements in two lists

definition *mirror-pair* :: $('a \times 'b) \Rightarrow ('b \times 'a)$

where $\text{mirror-pair } p \equiv (\text{snd } p, \text{fst } p)$

lemma *zip-mirror[rule-format]*:

$\llbracket i < \min (\text{length } xs) (\text{length } ys);$
 $p1 = (\text{zip } xs \ ys) ! i; p2 = (\text{zip } ys \ xs) ! i \rrbracket \implies$

$mirror\text{-}pair\ p1 = p2$
 $\langle proof \rangle$

definition $equal\text{-}pair :: ('a \times 'a) \Rightarrow bool$
where $equal\text{-}pair\ p \equiv (fst\ p = snd\ p)$

lemma $mirror\text{-}pair\text{-}equal$: $equal\text{-}pair\ (mirror\text{-}pair\ p) = (equal\text{-}pair\ p)$
 $\langle proof \rangle$

primrec

$equal\text{-}pair\text{-}count :: ('a \times 'a)\ list \Rightarrow nat$
where
 $equal\text{-}pair\text{-}count\ [] = 0$
 $| equal\text{-}pair\text{-}count\ (p \# ps) =$
 $\quad if\ (fst\ p = snd\ p)$
 $\quad \quad then\ Suc\ (equal\text{-}pair\text{-}count\ ps)$
 $\quad \quad else\ 0)$

lemma $equal\text{-}pair\text{-}count\text{-}le$: $equal\text{-}pair\text{-}count\ xs \leq length\ xs$
 $\langle proof \rangle$

lemma $equal\text{-}pair\text{-}count\text{-}0$:
 $fst\ (hd\ ps) \neq snd\ (hd\ ps) \implies equal\text{-}pair\text{-}count\ ps = 0$
 $\langle proof \rangle$

lemma $equal\text{-}pair\text{-}count\text{-}Suc$:
 $equal\text{-}pair\text{-}count\ ((a, a) \# ps) = Suc\ (equal\text{-}pair\text{-}count\ ps)$
 $\langle proof \rangle$

lemma $equal\text{-}pair\text{-}count\text{-}eq\text{-}pairwise$ [*rule-format*]:
 $\llbracket length\ ps1 = length\ ps2;$
 $\quad \forall i < length\ ps2. equal\text{-}pair\ (ps1\ !\ i) = equal\text{-}pair\ (ps2\ !\ i) \rrbracket$
 $\implies equal\text{-}pair\text{-}count\ ps1 = equal\text{-}pair\text{-}count\ ps2$
 $\langle proof \rangle$

lemma $equal\text{-}pair\text{-}count\text{-}mirror\text{-}pairwise$ [*rule-format*]:
 $\llbracket length\ ps1 = length\ ps2;$
 $\quad \forall i < length\ ps2. ps1\ !\ i = mirror\text{-}pair\ (ps2\ !\ i) \rrbracket$
 $\implies equal\text{-}pair\text{-}count\ ps1 = equal\text{-}pair\text{-}count\ ps2$
 $\langle proof \rangle$

lemma $equal\text{-}pair\text{-}count\text{-}correct$:
 $\bigwedge i. i < equal\text{-}pair\text{-}count\ ps \implies equal\text{-}pair\ (ps\ !\ i)$
 $\langle proof \rangle$

lemma $equal\text{-}pair\text{-}count\text{-}maximality\text{-}aux$ [*rule-format*]: $\bigwedge i.$

$i = \text{equal-pair-count } ps \implies \text{length } ps = i \vee \neg \text{equal-pair } (ps ! i)$
 ⟨proof⟩

corollary *equal-pair-count-maximality1a*[rule-format]:
 $\text{equal-pair-count } ps = \text{length } ps \vee \neg \text{equal-pair } (ps ! \text{equal-pair-count } ps)$
 ⟨proof⟩

corollary *equal-pair-count-maximality1b*[rule-format]:
 $\text{equal-pair-count } ps \neq \text{length } ps \implies$
 $\neg \text{equal-pair } (ps ! \text{equal-pair-count } ps)$
 ⟨proof⟩

lemma *equal-pair-count-maximality2a*[rule-format]:
 $\text{equal-pair-count } ps = \text{length } ps \vee \text{— either all pairs are equal}$
 $(\forall i \geq \text{equal-pair-count } ps. (\exists j \leq i. \neg \text{equal-pair } (ps ! j)))$
 ⟨proof⟩

corollary *equal-pair-count-maximality2b*[rule-format]:
 $\text{equal-pair-count } ps \neq \text{length } ps \implies$
 $\forall i \geq \text{equal-pair-count } ps. (\exists j \leq i. \neg \text{equal-pair } (ps ! j))$
 ⟨proof⟩

lemmas *equal-pair-count-maximality* =
equal-pair-count-maximality1a *equal-pair-count-maximality1b*
equal-pair-count-maximality2a *equal-pair-count-maximality2b*

12.3 Prefix length

Length of the prefix infimum

definition *inf-prefix-length* :: 'a list ⇒ 'a list ⇒ nat
 where *inf-prefix-length* *xs ys* ≡ *equal-pair-count* (*zip xs ys*)

value *int* (*inf-prefix-length* [1::int,2,3,4,7,8,15] [1::int,2,3,4,7,15])

value *int* (*inf-prefix-length* [1::int,2,3,4] [1::int,2,3,4,7,15])

value *int* (*inf-prefix-length* [] [1::int,2,3,4,7,15])

value *int* (*inf-prefix-length* [1::int,2,3,4,5] [1::int,2,3,4,5])

lemma *inf-prefix-length-commute*[rule-format]:
 $\text{inf-prefix-length } xs \ ys = \text{inf-prefix-length } ys \ xs$
 ⟨proof⟩

lemma *inf-prefix-length-leL*[intro]:
 $\text{inf-prefix-length } xs \ ys \leq \text{length } xs$
 ⟨proof⟩

corollary *inf-prefix-length-leR*[intro]:
 $\text{inf-prefix-length } xs \ ys \leq \text{length } ys$
 ⟨proof⟩

lemmas *inf-prefix-length-le* =
inf-prefix-length-leL
inf-prefix-length-leR

lemma *inf-prefix-length-le-min*[*rule-format*]:
 $inf\text{-}prefix\text{-}length\ xs\ ys \leq \min (length\ xs)\ (length\ ys)$
 ⟨*proof*⟩

lemma *hd-inf-prefix-length-0*:
 $hd\ xs \neq hd\ ys \implies inf\text{-}prefix\text{-}length\ xs\ ys = 0$
 ⟨*proof*⟩

lemma *inf-prefix-length-NilL*[*simp*]: $inf\text{-}prefix\text{-}length\ []\ ys = 0$
 ⟨*proof*⟩

lemma *inf-prefix-length-NilR*[*simp*]: $inf\text{-}prefix\text{-}length\ xs\ [] = 0$
 ⟨*proof*⟩

lemma *inf-prefix-length-Suc*[*simp*]:
 $inf\text{-}prefix\text{-}length\ (a\ \#\ xs)\ (a\ \#\ ys) = Suc\ (inf\text{-}prefix\text{-}length\ xs\ ys)$
 ⟨*proof*⟩

lemma *inf-prefix-length-correct*:
 $i < inf\text{-}prefix\text{-}length\ xs\ ys \implies xs\ !\ i = ys\ !\ i$
 ⟨*proof*⟩

corollary *nth-neq-imp-inf-prefix-length-le*:
 $xs\ !\ i \neq ys\ !\ i \implies inf\text{-}prefix\text{-}length\ xs\ ys \leq i$
 ⟨*proof*⟩

lemma *inf-prefix-length-maximality1*[*rule-format*]:
 $inf\text{-}prefix\text{-}length\ xs\ ys \neq \min (length\ xs)\ (length\ ys) \implies$
 $xs\ !\ (inf\text{-}prefix\text{-}length\ xs\ ys) \neq ys\ !\ (inf\text{-}prefix\text{-}length\ xs\ ys)$
 ⟨*proof*⟩

corollary *inf-prefix-length-maximality2*[*rule-format*]:
 $\llbracket inf\text{-}prefix\text{-}length\ xs\ ys \neq \min (length\ xs)\ (length\ ys);$
 $inf\text{-}prefix\text{-}length\ xs\ ys \leq i \rrbracket \implies$
 $\exists j \leq i. xs\ !\ j \neq ys\ !\ j$
 ⟨*proof*⟩

lemmas *inf-prefix-length-maximality* =
inf-prefix-length-maximality1 inf-prefix-length-maximality2

lemma *inf-prefix-length-append*[*simp*]:
 $inf\text{-}prefix\text{-}length\ (zs\ @\ xs)\ (zs\ @\ ys) =$
 $length\ zs + inf\text{-}prefix\text{-}length\ xs\ ys$
 ⟨*proof*⟩

lemma *inf-prefix-length-take-correct*:

$n \leq \text{inf-prefix-length } xs \ ys \implies xs \downarrow n = ys \downarrow n$
 ⟨proof⟩

lemma *inf-prefix-length-0-imp-hd-neq*:

$\llbracket xs \neq []; ys \neq []; \text{inf-prefix-length } xs \ ys = 0 \rrbracket \implies \text{hd } xs \neq \text{hd } ys$
 ⟨proof⟩

12.4 Prefix infimum

definition *inf-prefix* :: 'a list \Rightarrow 'a list \Rightarrow 'a list (**infixl** \square 70)

where $xs \square ys \equiv xs \downarrow (\text{inf-prefix-length } xs \ ys)$

lemma *length-inf-prefix*: $\text{length } (xs \square ys) = \text{inf-prefix-length } xs \ ys$
 ⟨proof⟩

lemma *inf-prefix-commute*: $xs \square ys = ys \square xs$
 ⟨proof⟩

lemma *inf-prefix-takeL*: $xs \square ys = xs \downarrow (\text{inf-prefix-length } xs \ ys)$
 ⟨proof⟩

lemma *inf-prefix-takeR*: $xs \square ys = ys \downarrow (\text{inf-prefix-length } xs \ ys)$
 ⟨proof⟩

lemma *inf-prefix-correct*: $i < \text{length } (xs \square ys) \implies xs ! i = ys ! i$
 ⟨proof⟩

corollary *inf-prefix-correctL*:

$i < \text{length } (xs \square ys) \implies (xs \square ys) ! i = xs ! i$
 ⟨proof⟩

corollary *inf-prefix-correctR*:

$i < \text{length } (xs \square ys) \implies (xs \square ys) ! i = ys ! i$
 ⟨proof⟩

lemma *inf-prefix-take-correct*:

$n \leq \text{length } (xs \square ys) \implies xs \downarrow n = ys \downarrow n$
 ⟨proof⟩

lemma *is-inf-prefix*[rule-format]:

$\llbracket \text{length } zs = \text{length } (xs \square ys);$
 $\bigwedge i. i < \text{length } (xs \square ys) \implies zs ! i = xs ! i \wedge zs ! i = ys ! i \rrbracket \implies$
 $zs = xs \square ys$
 ⟨proof⟩

lemma *hd-inf-prefix-Nil*: $\text{hd } xs \neq \text{hd } ys \implies xs \square ys = []$
 ⟨proof⟩

lemma *inf-prefix-Nil-imp-hd-neq*:

$\llbracket xs \neq []; ys \neq []; xs \sqcap ys = [] \rrbracket \implies hd\ xs \neq hd\ ys$
 <proof>

lemma *length-inf-prefix-append*[simp]:
 $length\ ((zs\ @\ xs)\ \sqcap\ (zs\ @\ ys)) =$
 $length\ zs + length\ (xs\ \sqcap\ ys)$
 <proof>

lemma *inf-prefix-append*[simp]: $(zs\ @\ xs)\ \sqcap\ (zs\ @\ ys) = zs\ @\ (xs\ \sqcap\ ys)$
 <proof>

lemma *hd-neq-inf-prefix-append*:
 $hd\ xs \neq hd\ ys \implies (zs\ @\ xs)\ \sqcap\ (zs\ @\ ys) = zs$
 <proof>

lemma *inf-prefix-NilL*[simp]: $[]\ \sqcap\ ys = []$
 <proof>

corollary *inf-prefix-NilR*[simp]: $xs\ \sqcap\ [] = []$
 <proof>

lemmas *inf-prefix-Nil = inf-prefix-NilL inf-prefix-NilR*

lemma *inf-prefix-Cons*[simp]: $(a\ \#\ xs)\ \sqcap\ (a\ \#\ ys) = a\ \#\ xs\ \sqcap\ ys$
 <proof>

corollary *inf-prefix-hd*[simp]: $hd\ ((a\ \#\ xs)\ \sqcap\ (a\ \#\ ys)) = a$
 <proof>

lemma *inf-prefix-le1*: $prefix\ (xs\ \sqcap\ ys)\ xs$
 <proof>

lemma *inf-prefix-le2*: $prefix\ (xs\ \sqcap\ ys)\ ys$
 <proof>

lemma *le-inf-prefix-iff*: $prefix\ x\ (y\ \sqcap\ z) = (prefix\ x\ y \wedge prefix\ x\ z)$
 <proof>

lemma *le-imp-le-inf-prefix*: $\llbracket prefix\ x\ y; prefix\ x\ z \rrbracket \implies prefix\ x\ (y\ \sqcap\ z)$
 <proof>

interpretation *prefix*:

semilattice-inf

$(\sqcap) :: 'a\ list \Rightarrow 'a\ list \Rightarrow 'a\ list$

prefix

strict-prefix

<proof>

12.5 Prefices for infinite lists

definition $iprefix :: 'a list \Rightarrow 'a\ i\ list \Rightarrow bool$ (**infixl** \sqsubseteq 50)

where $xs \sqsubseteq f \equiv \exists g. f = xs \frown g$

lemma $iprefix\text{-}eq\text{-}iprefix\text{-}take$: $(xs \sqsubseteq f) = (f \Downarrow \text{length } xs = xs)$
 $\langle proof \rangle$

lemma $iprefix\text{-}take\text{-}eq\text{-}iprefix\text{-}take\text{-}ex$:
 $(f \Downarrow \text{length } xs = xs) = (\exists n. f \Downarrow n = xs)$
 $\langle proof \rangle$

lemma $iprefix\text{-}eq\text{-}iprefix\text{-}take\text{-}ex$: $(xs \sqsubseteq f) = (\exists n. f \Downarrow n = xs)$
 $\langle proof \rangle$

lemma $i\text{-}take\text{-}imp\text{-}iprefix$ [*intro*]: $f \Downarrow n \sqsubseteq f$
 $\langle proof \rangle$

lemma $i\text{-}take\text{-}prefix\text{-}le\text{-}conv$: $prefix (f \Downarrow a) (f \Downarrow b) = (a \leq b)$
 $\langle proof \rangle$

lemma $i\text{-}append\text{-}imp\text{-}iprefix$ [*simp, intro*]: $xs \sqsubseteq xs \frown f$
 $\langle proof \rangle$

lemma $iprefix\text{-}imp\text{-}take\text{-}eq$:
 $\llbracket n \leq \text{length } xs; xs \sqsubseteq f \rrbracket \Longrightarrow xs \Downarrow n = f \Downarrow n$
 $\langle proof \rangle$

lemma $prefix\text{-}iprefix\text{-}trans$: $\llbracket prefix\ xs\ ys; ys \sqsubseteq f \rrbracket \Longrightarrow xs \sqsubseteq f$
 $\langle proof \rangle$

lemma $i\text{-}take\text{-}length\text{-}prefix\text{-}conv$: $prefix (f \Downarrow \text{length } xs) xs = (xs \sqsubseteq f)$
 $\langle proof \rangle$

lemma $iprefixI$ [*intro?*]: $f = xs \frown g \Longrightarrow xs \sqsubseteq f$
 $\langle proof \rangle$

lemma $iprefixE$ [*elim?*]: $\llbracket xs \sqsubseteq f; \bigwedge g. f = xs \frown g \Longrightarrow C \rrbracket \Longrightarrow C$
 $\langle proof \rangle$

lemma $Nil\text{-}iprefix$ [*iff*]: $[] \sqsubseteq f$
 $\langle proof \rangle$

lemma $same\text{-}prefix\text{-}iprefix$ [*simp*]: $(xs @ ys \sqsubseteq xs \frown f) = (ys \sqsubseteq f)$
 $\langle proof \rangle$

lemma $prefix\text{-}iprefix$ [*simp*]: $prefix\ xs\ ys \Longrightarrow xs \sqsubseteq ys \frown f$
 $\langle proof \rangle$

lemma $append\text{-}iprefixD$: $xs @ ys \sqsubseteq f \Longrightarrow xs \sqsubseteq f$

<proof>

lemma *iprefix-length-le-imp-prefix:*

$\llbracket xs \sqsubseteq ys \frown f; \text{length } xs \leq \text{length } ys \rrbracket \implies \text{prefix } xs \text{ } ys$
<proof>

lemma *iprefix-i-append:*

$(xs \sqsubseteq ys \frown f) = (\text{prefix } xs \text{ } ys \vee (\exists zs. xs = ys @ zs \wedge zs \sqsubseteq f))$
<proof>

lemma *i-append-one-iprefix:*

$xs \sqsubseteq f \implies xs @ [f (\text{length } xs)] \sqsubseteq f$
<proof>

lemma *iprefix-same-length-le:*

$\llbracket xs \sqsubseteq f; ys \sqsubseteq f; \text{length } xs \leq \text{length } ys \rrbracket \implies \text{prefix } xs \text{ } ys$
<proof>

lemma *iprefix-same-cases:*

$\llbracket xs \sqsubseteq f; ys \sqsubseteq f \rrbracket \implies \text{prefix } xs \text{ } ys \vee \text{prefix } ys \text{ } xs$
<proof>

lemma *set-mono-iprefix:* $xs \sqsubseteq f \implies \text{set } xs \subseteq \text{range } f$

<proof>

end

theory *ListInfinite*

imports

CommonSet/SetIntervalStep

ListInf/ListInf-Prefix

begin

end