Infinite Lists

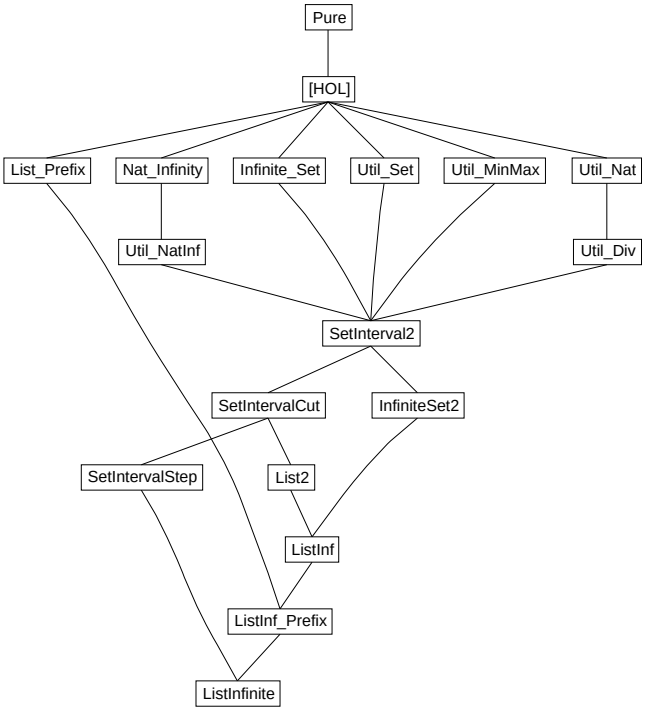
David Trachtenherz

February 24, 2011

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 Util-Set: Convenience results for set quantifiers

theory *Util-Set*
imports *Fun Set*
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*: $f\ (\mathit{let}\ x{=}a\ \mathit{in}\ g\ x) = (\mathit{let}\ x{=}a\ \mathit{in}\ f\ (g\ x))$ ⟨*proof*⟩

1.1.2 Some auxiliary *if*-rules

thm *if-P*
lemma *if-P'*: $[\![\ P;\ x = z\ ]\!] \Longrightarrow (\mathit{if}\ P\ \mathit{then}\ x\ \mathit{else}\ y) = z$ ⟨*proof*⟩
thm *if-not-P*
lemma *if-not-P'*: $[\![\ \neg\ P;\ y = z\ ]\!] \Longrightarrow (\mathit{if}\ P\ \mathit{then}\ x\ \mathit{else}\ y) = z$ ⟨*proof*⟩

lemma *if-P-both*: $[\![\ Q\ x;\ Q\ y\ ]\!] \Longrightarrow Q\ (\mathit{if}\ P\ \mathit{then}\ x\ \mathit{else}\ y)$ ⟨*proof*⟩
lemma *if-P-both-in-set*: $[\![\ x \in s;\ y \in s\ ]\!] \Longrightarrow (\mathit{if}\ P\ \mathit{then}\ x\ \mathit{else}\ y) \in s$ ⟨*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*:
$[\![\ I \subseteq A;\ \forall x{\in}A.\ f\ x = g\ x\ ]\!] \Longrightarrow (\forall x{\in}I.\ P\ (f\ x)) = (\forall x{\in}I.\ P\ (g\ x))$ ⟨*proof*⟩
lemma *bex-cong2*:
$[\![\ I \subseteq A;\ \forall x{\in}I.\ f\ x = g\ x\ ]\!] \Longrightarrow (\exists x{\in}I.\ P\ (f\ x)) = (\exists x{\in}I.\ P\ (g\ x))$ ⟨*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*:
$[\![P = P'; \bigwedge x.\ P\ x \Longrightarrow Q\ x = Q'\ x]\!] \Longrightarrow$
$(\forall x.\ P\ x \longrightarrow Q\ x) = (\forall x.\ P'\ x \longrightarrow Q'\ x)$
⟨*proof*⟩
lemma *ex-imp-eqI*:
$[\![P = P'; \bigwedge x.\ P\ x \Longrightarrow Q\ x = Q'\ x]\!] \Longrightarrow$
$(\exists x.\ P\ x \wedge Q\ x) = (\exists x.\ P'\ x \wedge Q'\ x)$
⟨*proof*⟩

1.2.2 Auxiliary results for *empty* sets

lemma *empty-imp-not-in*: $x \notin \{\}$ ⟨*proof*⟩
lemma *ex-imp-not-empty*: $\exists x.\ x \in A \Longrightarrow A \neq \{\}$ ⟨*proof*⟩
lemma *in-imp-not-empty*: $x \in A \Longrightarrow A \neq \{\}$ ⟨*proof*⟩
lemma *not-empty-imp-ex*: $A \neq \{\} \Longrightarrow \exists x.\ x \in A$ ⟨*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*: $[\![\exists x{\in}A.\ P\ x;\ A \subseteq B]\!] \Longrightarrow \exists x{\in}B.\ P\ x$ ⟨*proof*⟩
lemma *bex-imp-ex*: $\exists x{\in}A.\ P\ x \Longrightarrow \exists x.\ P\ x$ ⟨*proof*⟩
lemma *ball-subset-imp-ball*: $[\![\forall x{\in}B.\ P\ x;\ A \subseteq B]\!] \Longrightarrow \forall x{\in}A.\ P\ x$ ⟨*proof*⟩
thm
ball-subset-imp-ball
ball-subset-imp-ball[*rule-format*]
lemma *all-imp-ball*: $\forall x.\ P\ x \Longrightarrow \forall x{\in}A.\ P\ x$ ⟨*proof*⟩

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

end

2 Util-MinMax: Order and linear order: min and max

theory *Util-MinMax*
imports *Lattices*
begin

2.1 Additional lemmata about *min* and *max*

thm *min-less-iff-conj*

lemma *min-less-imp-conj*: $(z::'a::linorder) < min\ x\ y \Longrightarrow z < x \wedge z < y$ ⟨*proof*⟩
lemma *conj-less-imp-min*: $[\![z < x; z < y]\!] \Longrightarrow (z::'a::linorder) < min\ x\ y$ ⟨*proof*⟩

lemmas *min-le-iff-conj* = *min-max.le-inf-iff*
lemma *min-le-imp-conj*: $(z::'a::linorder) \leq min\ x\ y \Longrightarrow z \leq x \wedge z \leq y$ ⟨*proof*⟩

lemmas *conj-le-imp-min* = *min-max.le-infI*

thm *min-max.inf-absorb1*

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

lemmas *min-eqR* = *min-max.inf-absorb2*
lemmas *min-eq* = *min-eqL min-eqR*
thm *min-eq*

thm *max-less-iff-conj*
lemma *max-less-imp-conj*:$max\ x\ y < b \Longrightarrow x < (b::('a::linorder)) \wedge y < b$ ⟨*proof*⟩
lemma *conj-less-imp-max*:$[\![x < (b::('a::linorder));\ y < b]\!] \Longrightarrow max\ x\ y < b$ ⟨*proof*⟩

lemmas *max-le-iff-conj* = *min-max.le-sup-iff*
lemma *max-le-imp-conj*:$max\ x\ y \leq b \Longrightarrow x \leq (b::('a::linorder)) \wedge y \leq b$ ⟨*proof*⟩

lemmas *conj-le-imp-max* = *min-max.le-supI*

thm *min-max.sup-absorb1*

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

lemmas *max-eqR* = *min-max.sup-absorb2*
lemmas *max-eq* = *max-eqL max-eqR*
thm *max-eq*

thm
le-maxI1
le-maxI2

lemmas *le-minI1* = *min-max.inf-le1*

lemmas *le-minI2* = *min-max.inf-le2*

lemma

min-le-monoR: $(a::'a::linorder) \leq b \Longrightarrow min\ x\ a \leq min\ x\ b$ **and**
min-le-monoL: $(a::'a::linorder) \leq b \Longrightarrow min\ a\ x \leq min\ b\ x$
⟨*proof*⟩
lemma
max-le-monoR: $(a::'a::linorder) \leq b \Longrightarrow max\ x\ a \leq max\ x\ b$ **and**
max-le-monoL: $(a::'a::linorder) \leq b \Longrightarrow max\ a\ x \leq max\ b\ x$
⟨*proof*⟩

end

3 Util-NatInf: Results for natural arithmetics with infinity

theory *Util-NatInf*
imports $ISABELLE-HOME/src/HOL/Library/Nat-Infinity
begin

3.1 Arithmetic operations with *inat*

3.1.1 Additional definitions

thm *one-inat-def*
thm *plus-inat-def*
thm *times-inat-def*

instantiation *inat* :: {*minus*, *Divides.div*}
begin

definition
diff-inat-def [*code del*]:
$a - b \equiv (case\ a\ of$
$(Fin\ x) \Rightarrow (case\ b\ of\ (Fin\ y) \Rightarrow Fin\ (x - y) \mid \infty \Rightarrow 0) \mid$
$\infty \Rightarrow \infty)$

definition
div-inat-def [*code del*]:
$a\ div\ b \equiv (case\ a\ of$
$(Fin\ x) \Rightarrow (case\ b\ of\ (Fin\ y) \Rightarrow Fin\ (x\ div\ y) \mid \infty \Rightarrow 0) \mid$
$\infty \Rightarrow (case\ b\ of\ (Fin\ y) \Rightarrow ((case\ y\ of\ 0 \Rightarrow 0 \mid Suc\ n \Rightarrow \infty)) \mid \infty \Rightarrow \infty\ ))$
definition
mod-inat-def [*code del*]:
$a\ mod\ b \equiv (case\ a\ of$
$(Fin\ x) \Rightarrow (case\ b\ of\ (Fin\ y) \Rightarrow Fin\ (x\ mod\ y) \mid \infty \Rightarrow a) \mid$
$\infty \Rightarrow \infty)$

instance ⟨*proof*⟩

end

lemmas *inat-arith-defs* =
zero-inat-def one-inat-def
plus-inat-def diff-inat-def times-inat-def div-inat-def mod-inat-def
declare *zero-inat-def*[*simp*]

primrec *the-Fin* :: *inat* ⇒ *nat* **where**
the-Fin (*Fin n*) = *n*

lemma *Fin-the-Fin*: $n \neq \infty \Longrightarrow Fin\ (the\text{-}Fin\ n) = n$ ⟨*proof*⟩

3.1.2 Results for comparison operators

lemma *Fin-ile-mono*: $(Fin\ m \leq Fin\ n) = (m \leq n)$ ⟨*proof*⟩
lemma *Fin-iless-mono*: $(Fin\ m < Fin\ n) = (m < n)$ ⟨*proof*⟩
lemma *Infty-ub*: $n \leq \infty$ ⟨*proof*⟩

thm *less-not-sym*
lemma *iless-not-sym*: $(x::inat) < y \Longrightarrow \neg\ y < x$ ⟨*proof*⟩
thm *less-not-refl*
lemma *iless-not-refl*: $\neg\ (n::inat) < n$ ⟨*proof*⟩
thm *le-refl*
lemma *ile-refl*: $(n::inat) \leq n$ ⟨*proof*⟩
lemma *eq-imp-ile*: $(m::inat) = n \Longrightarrow m \leq n$ ⟨*proof*⟩

lemma *not-Infty-iless*: $\neg\ \infty < n$ ⟨*proof*⟩
lemma *Fin-iless-Infty*: $Fin\ n < \infty$ ⟨*proof*⟩
lemma *not-Infty-ile-Fin*: $\neg\ \infty \leq Fin\ n$ ⟨*proof*⟩

lemmas *Fin-ile-Infty* = *Infty-ub*

lemmas *neq-Infty-Fin-conv* = *not-Infty-eq*

lemma *Infty-le-eq*: $(\infty \leq n) = (n = \infty)$ ⟨*proof*⟩

lemma *ile-trans*: $[\![\ (i::inat) \leq j;\ j \leq k\ ]\!] \Longrightarrow i \leq k$ ⟨*proof*⟩
lemma *ile-anti-sym*: $[\![\ (m::inat) \leq n;\ n \leq m\ ]\!] \Longrightarrow m = n$ ⟨*proof*⟩
lemma *iless-ile*: $((m::inat) < n) = (m \leq n \wedge m \neq n)$ ⟨*proof*⟩
lemma *ile-linear*: $(m::inat) \leq n \vee n \leq m$ ⟨*proof*⟩

thm *neq0-conv*
lemma *ineq0-conv*: $(n \neq (0::inat)) = (0 < n)$
⟨*proof*⟩
lemmas *ineq0-conv-Fin*[*simp*] = *ineq0-conv*[*unfolded zero-inat-def*]

lemmas *not-iless0* = *not-ilessi0*
lemma *iless-iSuc0*: $(n < iSuc\ 0) = (n = 0)$
⟨*proof*⟩
lemmas *iless-iSuc0-Fin*[*simp*] = *iless-iSuc0*[*unfolded zero-inat-def*]

lemmas *ile-0-eq* = *i0-neq*

lemma *neq-Infty-imp-ex-Fin*: $n \neq \infty \Longrightarrow \exists\, nat.\ n = Fin\ nat$
⟨*proof*⟩
corollary *less-Infty-imp-ex-Fin*: $n < \infty \Longrightarrow \exists\, nat.\ n = Fin\ nat$
thm *neq-Infty-imp-ex-Fin*[*OF less-imp-neq*]
⟨*proof*⟩

3.1.3 Addition and difference

lemma *iadd-Fin-Fin*: $Fin\ a + Fin\ b = Fin\ (a + b)$ ⟨*proof*⟩
lemma *iadd-Infty*: $\infty + n = \infty$ ⟨*proof*⟩
lemma *iadd-Infty-right*: $n + \infty = \infty$ ⟨*proof*⟩

lemma *idiff-Fin-Fin*[*simp*,*code*]: $Fin\ a - Fin\ b = Fin\ (a - b)$
⟨*proof*⟩
lemma *idiff-Infty*[*simp*,*code*]: $\infty - n = \infty$
⟨*proof*⟩
lemma *idiff-Infty-right*[*simp*,*code*]: $Fin\ a - \infty = 0$
⟨*proof*⟩

lemma *idiff-0*: $(n::inat) - 0 = n$
⟨*proof*⟩
lemmas *idiff-0-Fin*[*simp*,*code*] = *idiff-0*[*unfolded zero-inat-def*]
lemma *idiff-0-eq-0*: $(0::inat) - n = 0$
⟨*proof*⟩
lemmas *idiff-0-eq-0-Fin*[*simp*,*code*] = *idiff-0-eq-0*[*unfolded zero-inat-def*]

lemma *diff-eq-conv-nat*: $(x - y = (z::nat)) = (if\ y < x\ then\ x = y + z\ else\ z = 0)$
⟨*proof*⟩
lemma *idiff-eq-conv*:
$(x - y = (z::inat)) =$
$(if\ y < x\ then\ x = y + z\ else\ if\ x \neq \infty\ then\ z = 0\ else\ z = \infty)$

⟨*proof*⟩
lemmas *idiff-eq-conv-Fin* = *idiff-eq-conv*[*unfolded zero-inat-def*]

lemma *idiff-self-eq-0*: $n \neq \infty \Longrightarrow (n::inat) - n = 0$
⟨*proof*⟩
lemmas *idiff-self-eq-0-Fin* = *idiff-self-eq-0*[*unfolded zero-inat-def*]

lemma *less-eq-idiff-eq-sum*: $y \leq (x::inat) \Longrightarrow (z \leq x - y) = (z + y \leq x)$
⟨*proof*⟩

lemma *iSuc-pred*: $0 < n \Longrightarrow iSuc\ (n - iSuc\ 0) = n$
⟨*proof*⟩
lemmas *iSuc-pred-Fin* = *iSuc-pred*[*unfolded zero-inat-def*]
lemma *iadd-0*: $(0::inat) + n = n$
⟨*proof*⟩
lemmas *iadd-0-Fin*[*simp*, *code*] = *iadd-0*[*unfolded zero-inat-def*]
lemma *iadd-0-right*: $n + (0::inat) = n$
⟨*proof*⟩
lemmas *iadd-0-right-Fin*[*simp*, *code*] = *iadd-0-right*[*unfolded zero-inat-def*]

lemma *iadd-commute*: $(a::inat) + b = b + a$
⟨*proof*⟩
thm *add-assoc*
lemma *iadd-assoc*: $(a::inat) + b + c = a + (b + c)$
⟨*proof*⟩

thm *add-Suc*
lemma *iadd-Suc*: $iSuc\ m + n = iSuc\ (m + n)$
⟨*proof*⟩

thm *add-Suc-right*
lemma *iadd-Suc-right*: $m + iSuc\ n = iSuc\ (m + n)$
⟨*proof*⟩

lemma *mono-iSuc*: *mono iSuc*
⟨*proof*⟩

thm *add-is-0*[*no-vars*]
lemma *iadd-is-0*: $(m + n = (0::inat)) = (m = 0 \wedge n = 0)$
⟨*proof*⟩
lemmas *iadd-is-0-Fin* = *iadd-is-0*[*unfolded zero-inat-def*]

lemma *ile-add1*: $(n::inat) \leq n + m$
⟨*proof*⟩
lemma *ile-add2*: $(n::inat) \leq m + n$
⟨*proof*⟩

lemma *iadd-ile-mono*: $[\![(i::inat) \leq j; k \leq l]\!] \Longrightarrow i + k \leq j + l$
⟨*proof*⟩
lemma *iadd-ile-mono1*: $(i::inat) \leq j \Longrightarrow i + k \leq j + k$
⟨*proof*⟩
lemma *iadd-ile-mono2*: $(i::inat) \leq j \Longrightarrow k + i \leq k + j$
⟨*proof*⟩

lemma *iadd-iless-mono*: $[\![(i::inat) < j; k < l]\!] \Longrightarrow i + k < j + l$
⟨*proof*⟩

lemma *trans-ile-iadd1*: $i \leq (j::inat) \Longrightarrow i \leq j + m$
⟨*proof*⟩
lemma *trans-ile-iadd2*: $i \leq (j::inat) \Longrightarrow i \leq m + j$
⟨*proof*⟩

lemma *trans-iless-iadd1*: $i < (j::inat) \Longrightarrow i < j + m$
⟨*proof*⟩
lemma *trans-iless-iadd2*: $i < (j::inat) \Longrightarrow i < m + j$
⟨*proof*⟩

thm *add-leD1*[*no-vars*]
lemma *iadd-ileD1*: $m + k \leq (n::inat) \Longrightarrow m \leq n$
⟨*proof*⟩
lemma *iadd-ileD2*: $m + k \leq (n::inat) \Longrightarrow k \leq n$
⟨*proof*⟩

thm *diff-le-mono*
lemma *idiff-ile-mono*: $m \leq (n::inat) \Longrightarrow m - l \leq n - l$
⟨*proof*⟩
thm *diff-le-mono2*
lemma *idiff-ile-mono2*: $m \leq (n::inat) \Longrightarrow l - n \leq l - m$
⟨*proof*⟩

thm *diff-less-mono*
lemma *idiff-iless-mono*: $[\![m < (n::inat); l \leq m]\!] \Longrightarrow m - l < n - l$
⟨*proof*⟩
thm *diff-less-mono2*
lemma *idiff-iless-mono2*: $[\![m < (n::inat); m < l]\!] \Longrightarrow l - n \leq l - m$
⟨*proof*⟩

3.1.4 Multiplication and division

lemma *imult-Fin-Fin*: *Fin a* $*$ *Fin b* $=$ *Fin* $(a * b)$ ⟨*proof*⟩
lemma *imult-Infty*: $(0::inat) < n \Longrightarrow \infty * n = \infty$
⟨*proof*⟩
lemmas *imult-Infty-Fin*[*simp*] $=$ *imult-Infty*[*unfolded zero-inat-def*]
lemma *imult-Infty-right*: $(0::inat) < n \Longrightarrow n * \infty = \infty$

⟨*proof*⟩
lemmas *imult-Infty-right-Fin*[*simp*] = *imult-Infty-right*[*unfolded zero-inat-def*]

lemma *idiv-Fin-Fin*[*simp*, *code*]: *Fin a div Fin b* = *Fin* (*a div b*)
⟨*proof*⟩
lemma *idiv-Infty*: *0* < *n* ⟹ ∞ *div n* = ∞
⟨*proof*⟩
lemmas *idiv-Infty-Fin*[*simp*] = *idiv-Infty*[*unfolded zero-inat-def*]

lemma *idiv-Infty-right*[*simp*]: *n* ≠ ∞ ⟹ *n div* ∞ = *0*
⟨*proof*⟩

lemma *idiv-Infty-if*: *n div* ∞ = (*if n* = ∞ *then* ∞ *else 0*)
⟨*proof*⟩

lemmas *idiv-Infty-if-Fin* = *idiv-Infty-if*[*unfolded zero-inat-def*]

lemma *imult-0*: *0* ∗ (*m*::*inat*) = *0*
⟨*proof*⟩
lemmas *imult-0-Fin*[*simp*, *code*] = *imult-0*[*unfolded zero-inat-def*]
lemma *imult-0-right*: (*m*::*inat*) ∗ *0* = *0*
⟨*proof*⟩
lemmas *imult-0-right-Fin*[*simp*, *code*] = *imult-0-right*[*unfolded zero-inat-def*]

lemma *imult-is-0*: ((*m*::*inat*) ∗ *n* = *0*) = (*m* = *0* ∨ *n* = *0*)
⟨*proof*⟩
lemma *inat-0-less-mult-iff*: (*0* < (*m*::*inat*) ∗ *n*) = (*0* < *m* ∧ *0* < *n*)
⟨*proof*⟩
lemmas *imult-is-0-Fin* = *imult-is-0*[*unfolded zero-inat-def*]
lemmas *inat-0-less-mult-iff-Fin* = *inat-0-less-mult-iff*[*unfolded zero-inat-def*]

lemma *imult-commute*: (*a*::*inat*) ∗ *b* = *b* ∗ *a*
⟨*proof*⟩

lemma *imult-Infty-if*: ∞ ∗ *n* = (*if n* = *0 then 0 else* ∞)
⟨*proof*⟩
lemma *imult-Infty-right-if*: *n* ∗ ∞ = (*if n* = *0 then 0 else* ∞)
⟨*proof*⟩
lemmas *imult-Infty-if-Fin* = *imult-Infty-if*[*unfolded zero-inat-def*]
lemmas *imult-Infty-right-if-Fin* = *imult-Infty-right-if*[*unfolded zero-inat-def*]

lemma *imult-is-Infty*: ((*a*::*inat*) ∗ *b* = ∞) = (*a* = ∞ ∧ *b* ≠ *0* ∨ *b* = ∞ ∧ *a* ≠ *0*)
⟨*proof*⟩
lemmas *imult-is-Infty-Fin* = *imult-is-Infty*[*unfolded zero-inat-def*]

lemma *imult-assoc*: (*a*::*inat*) ∗ *b* ∗ *c* = *a* ∗ (*b* ∗ *c*)
⟨*proof*⟩

lemma *idiv-by-0*: $(a::inat)\ div\ 0 = 0$
⟨*proof*⟩
lemmas *idiv-by-0-Fin*[*simp*, *code*] = *idiv-by-0*[*unfolded zero-inat-def*]

lemma *idiv-0*: $0\ div\ (a::inat) = 0$
⟨*proof*⟩
lemmas *idiv-0-Fin*[*simp*, *code*] = *idiv-0*[*unfolded zero-inat-def*]

thm *mod-by-0*
lemma *imod-by-0*: $(a::inat)\ mod\ 0 = a$
⟨*proof*⟩
lemmas *imod-by-0-Fin*[*simp*, *code*] = *imod-by-0*[*unfolded zero-inat-def*]

lemma *imod-0*: $0\ mod\ (a::inat) = 0$
⟨*proof*⟩
lemmas *imod-0-Fin*[*simp*, *code*] = *imod-0*[*unfolded zero-inat-def*]

lemma *imod-Fin-Fin*[*simp*, *code*]: $Fin\ a\ mod\ Fin\ b = Fin\ (a\ mod\ b)$
⟨*proof*⟩
lemma *imod-Infty*[*simp*, *code*]: $\infty\ mod\ n = \infty$
⟨*proof*⟩
lemma *imod-Infty-right*[*simp*, *code*]: $n\ mod\ \infty = n$
⟨*proof*⟩
lemma *idiv-self*: $[\![\ 0 < (n::inat);\ n \neq \infty\ ]\!] \Longrightarrow n\ div\ n = 1$
⟨*proof*⟩
lemma *imod-self*: $n \neq \infty \Longrightarrow (n::inat)\ mod\ n = 0$
⟨*proof*⟩

lemma *idiv-iless*: $m < (n::inat) \Longrightarrow m\ div\ n = 0$
⟨*proof*⟩
lemma *imod-iless*: $m < (n::inat) \Longrightarrow m\ mod\ n = m$
⟨*proof*⟩

lemma *imod-iless-divisor*: $[\![\ 0 < (n::inat);\ m \neq \infty\ ]\!] \Longrightarrow m\ mod\ n < n$
⟨*proof*⟩
lemma *imod-ile-dividend*: $(m::inat)\ mod\ n \leq m$
⟨*proof*⟩
lemma *idiv-ile-dividend*: $(m::inat)\ div\ n \leq m$
⟨*proof*⟩

thm *div-mult2-eq*
lemma *idiv-imult2-eq*: $(a::inat)\ div\ (b * c) = a\ div\ b\ div\ c$
⟨*proof*⟩
thm *idiv-Infty*
thm *idiv-Infty*[*OF inat-0-less-mult-iff*[*THEN iffD2*]]
⟨*proof*⟩

thm *add-mult-distrib*

lemma *iadd-imult-distrib*: $((m::inat) + n) * k = m * k + n * k$
⟨*proof*⟩

thm *add-mult-distrib2*
lemma *iadd-imult-distrib2*: $k * ((m::inat) + n) = k * m + k * n$
⟨*proof*⟩

thm *mult-le-mono*
lemma *imult-ile-mono*: $[\![(i::inat) \leq j; k \leq l]\!] \Longrightarrow i * k \leq j * l$
⟨*proof*⟩

lemma *imult-ile-mono1*: $(i::inat) \leq j \Longrightarrow i * k \leq j * k$
⟨*proof*⟩
thm *mult-le-mono2*
lemma *imult-ile-mono2*: $(i::inat) \leq j \Longrightarrow k * i \leq k * j$
⟨*proof*⟩

lemma *imult-iless-mono1*: $[\![(i::inat) < j; 0 < k; k \neq \infty]\!] \Longrightarrow i * k \leq j * k$
⟨*proof*⟩
lemma *imult-iless-mono2*: $[\![(i::inat) < j; 0 < k; k \neq \infty]\!] \Longrightarrow k * i \leq k * j$
⟨*proof*⟩

lemma *imod-1*: $(Fin\ m)\ mod\ iSuc\ 0 = 0$
⟨*proof*⟩
lemmas *imod-1-Fin*[*simp*, *code*] = *imod-1*[*unfolded zero-inat-def*]

lemma *imod-iadd-self2*: $(m + Fin\ n)\ mod\ (Fin\ n) = m\ mod\ (Fin\ n)$
⟨*proof*⟩

lemma *imod-iadd-self1*: $(Fin\ n + m)\ mod\ (Fin\ n) = m\ mod\ (Fin\ n)$
⟨*proof*⟩

lemma *idiv-imod-equality*: $(m::inat)\ div\ n * n + m\ mod\ n + k = m + k$
⟨*proof*⟩
lemma *imod-idiv-equality*: $(m::inat)\ div\ n * n + m\ mod\ n = m$
⟨*proof*⟩

lemma *idiv-ile-mono*: $m \leq (n::inat) \Longrightarrow m\ div\ k \leq n\ div\ k$
⟨*proof*⟩
lemma *idiv-ile-mono2*: $[\![0 < m; m \leq (n::inat)]\!] \Longrightarrow k\ div\ n \leq k\ div\ m$
⟨*proof*⟩

end

4 Util-Nat: Results for natural arithmetics

theory *Util-Nat*
imports *Nat*
begin

4.1 Some convenience arithmetic lemmata

thm *Nat.add-Suc-right*
lemma *add-1-Suc-conv*: $m + 1 = Suc\ m$ ⟨*proof*⟩
lemma *sub-Suc0-sub-Suc-conv*: $b - a - Suc\ 0 = b - Suc\ a$ ⟨*proof*⟩
thm *Nat.Suc-pred*
lemma *Suc-diff-Suc*: $m < n \Longrightarrow Suc\ (n - Suc\ m) = n - m$
⟨*proof*⟩

lemma *nat-grSuc0-conv*: $(Suc\ 0 < n) = (n \neq 0 \wedge n \neq Suc\ 0)$
⟨*proof*⟩
lemma *nat-geSucSuc0-conv*: $(Suc\ (Suc\ 0) \leq n) = (n \neq 0 \wedge n \neq Suc\ 0)$
⟨*proof*⟩

lemma *nat-lessSucSuc0-conv*: $(n < Suc\ (Suc\ 0)) = (n = 0 \vee n = Suc\ 0)$
⟨*proof*⟩
lemma *nat-leSuc0-conv*: $(n \leq Suc\ 0) = (n = 0 \vee n = Suc\ 0)$
⟨*proof*⟩

thm *Nat.mult-Suc*
lemma *mult-pred*: $(m - Suc\ 0) * n = m * n - n$
⟨*proof*⟩
thm *Nat.mult-Suc-right*
lemma *mult-pred-right*: $m * (n - Suc\ 0) = m * n - m$
⟨*proof*⟩

lemma *gr-implies-gr0*: $m < (n::nat) \Longrightarrow 0 < n$ ⟨*proof*⟩

thm
Nat.mult-cancel1
Nat.mult-cancel1
corollary *mult-cancel1-gr0*:
$(0::nat) < k \Longrightarrow (k * m = k * n) = (m = n)$ ⟨*proof*⟩
corollary *mult-cancel2-gr0*:
$(0::nat) < k \Longrightarrow (m * k = n * k) = (m = n)$ ⟨*proof*⟩

thm
Nat.mult-le-cancel1
Nat.mult-le-cancel2
corollary *mult-le-cancel1-gr0*:
$(0::nat) < k \Longrightarrow (k * m \leq k * n) = (m \leq n)$ ⟨*proof*⟩

corollary *mult-le-cancel2-gr0*:
$(0::nat) < k \Longrightarrow (m * k \leq n * k) = (m \leq n)$ ⟨*proof*⟩

thm *mult-le-mono*
lemma *gr0-imp-self-le-mult1*: $0 < (k::nat) \Longrightarrow m \leq m * k$
⟨*proof*⟩

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

lemma *less-imp-Suc-mult-le*: $m < n \Longrightarrow Suc\ m * k \leq n * k$
⟨*proof*⟩

lemma *less-imp-Suc-mult-pred-less*: $[\![\ m < n;\ 0 < k\ ]\!] \Longrightarrow Suc\ m * k - Suc\ 0 < n * k$
⟨*proof*⟩

thm *Nat.zero-less-diff*
lemma *ord-zero-less-diff*: $(0 < (b::'a::ordered\text{-}ab\text{-}group\text{-}add) - a) = (a < b)$
⟨*proof*⟩

lemma *ord-zero-le-diff*: $(0 \leq (b::'a::ordered\text{-}ab\text{-}group\text{-}add) - a) = (a \leq b)$
⟨*proof*⟩

diff-diff-right in rule format

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

thm *Nat.le-add1 Nat.le-add2*
lemma *less-add1*: $(0::nat) < j \Longrightarrow i < i + j$ ⟨*proof*⟩
lemma *less-add2*: $(0::nat) < j \Longrightarrow i < j + i$ ⟨*proof*⟩

thm *Nat.add-leD1 Nat.add-leD2*
thm *Nat.add-lessD1*
lemma *add-lessD2*: $i + j < (k::nat) \Longrightarrow j < k$ ⟨*proof*⟩

thm *Nat.add-le-mono1*
lemma *add-le-mono2*: $i \leq (j::nat) \Longrightarrow k + i \leq k + j$ ⟨*proof*⟩
thm *Nat.add-less-mono1*
lemma *add-less-mono2*: $i < (j::nat) \Longrightarrow k + i < k + j$ ⟨*proof*⟩

thm *Nat.diff-le-self*
lemma *diff-less-self*: $[\![\ (0::nat) < i;\ \ 0 < j\ ]\!] \Longrightarrow i - j < i$ ⟨*proof*⟩

lemma

ge-less-neq-conv: $((a::'a::linorder) \leq n) = (\forall x.\ x < a \longrightarrow n \neq x)$ **and**
le-greater-neq-conv: $(n \leq (a::'a::linorder)) = (\forall x.\ a < x \longrightarrow n \neq x)$
⟨*proof*⟩
lemma
greater-le-neq-conv: $((a::'a::linorder) < n) = (\forall x.\ x \leq a \longrightarrow n \neq x)$ **and**
less-ge-neq-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*: $[\![\ 0 \leq (a::'a::ordered\text{-}ab\text{-}group\text{-}add\text{-}abs);\ a \leq b\ ]\!] \Longrightarrow |a| \leq |b|$
⟨*proof*⟩
lemma *leq-neg-imp-abs-geq*: $[\![\ (a::'a::ordered\text{-}ab\text{-}group\text{-}add\text{-}abs) \leq 0;\ b \leq a\ ]\!] \Longrightarrow |a| \leq |b|$
⟨*proof*⟩
lemma *abs-range*: $[\![\ 0 \leq (a::'a::\{ordered\text{-}ab\text{-}group\text{-}add\text{-}abs,abs\text{-}if\});\ -a \leq x;\ x \leq a\ ]\!] \Longrightarrow |x| \leq a$
⟨*proof*⟩
thm *neg-le-iff-le*[*THEN iffD1*]
⟨*proof*⟩

Lemmas for @termsgn function

lemma *sgn-abs*:$(x::'a::linordered\text{-}idom) \neq 0 \Longrightarrow |sgn\ x| = 1$
⟨*proof*⟩
lemma *sgn-mult-abs*:$|x| * |sgn\ (a::'a::linordered\text{-}idom)| = |x * sgn\ a|$
⟨*proof*⟩
lemma *abs-imp-sgn-abs*: $|a| = |b| \Longrightarrow |sgn\ (a::'a::linordered\text{-}idom)| = |sgn\ b|$
⟨*proof*⟩
lemma *sgn-mono*: $a \leq b \Longrightarrow sgn\ (a::'a::\{linordered\text{-}idom,linordered\text{-}semidom\}) \leq sgn\ b$
⟨*proof*⟩

4.2 Additional facts about inequalities

thm *Nat.le-add-diff*
lemma *add-diff-le*: $k \leq n \Longrightarrow m + k - n \leq (m::nat)$
⟨*proof*⟩
thm
Nat.le-add-diff
add-diff-le

lemma *less-add-diff*: $k < (n::nat) \Longrightarrow m < n + m - k$
thm *add-less-imp-less-right*[*of* - *k*]
⟨*proof*⟩
thm *add-diff-le*
lemma *add-diff-less*: $[\![\ k < n;\ 0 < m\ ]\!] \Longrightarrow m + k - n < (m::nat)$
⟨*proof*⟩
thm
Nat.le-add-diff

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

thm *Nat.less-diff-conv*
lemma *add-le-imp-le-diff1*: $i + k \leq j \Longrightarrow i \leq j - (k::nat)$
⟨*proof*⟩
lemma *add-le-imp-le-diff2*: $k + i \leq j \Longrightarrow i \leq j - (k::nat)$ ⟨*proof*⟩
thm
Nat.less-diff-conv[*symmetric*]
Nat.le-diff-conv2[*symmetric*]
add-le-imp-le-diff1
add-le-imp-le-diff2

thm
Nat.le-diff-conv Nat.le-diff-conv2
Nat.less-diff-conv
lemma *diff-less-imp-less-add*: $j - (k::nat) < i \Longrightarrow j < i + k$ ⟨*proof*⟩
thm *Nat.le-diff-conv*
lemma *diff-less-conv*: $0 < i \Longrightarrow (j - (k::nat) < i) = (j < i + k)$
⟨*proof*⟩

lemma *le-diff-swap*: $[\![i \leq (k::nat); j \leq k]\!] \Longrightarrow (k - j \leq i) = (k - i \leq j)$
⟨*proof*⟩

lemma *diff-less-imp-swap*: $[\![0 < (i::nat); k - i < j]\!] \Longrightarrow (k - j < i)$ ⟨*proof*⟩
lemma *diff-less-swap*: $[\![0 < (i::nat); 0 < j]\!] \Longrightarrow (k - j < i) = (k - i < j)$
⟨*proof*⟩

lemma *less-diff-imp-less*: $(i::nat) < j - m \Longrightarrow i < j$ ⟨*proof*⟩
lemma *le-diff-imp-le*: $(i::nat) \leq j - m \Longrightarrow i \leq j$ ⟨*proof*⟩

lemma *less-diff-le-imp-less*: $[\![(i::nat) < j - m; n \leq m]\!] \Longrightarrow i < j - n$ ⟨*proof*⟩
lemma *le-diff-le-imp-le*: $[\![(i::nat) \leq j - m; n \leq m]\!] \Longrightarrow i \leq j - n$ ⟨*proof*⟩

thm *Nat.less-imp-diff-less*
lemma *le-imp-diff-le*: $(j::nat) \leq k \Longrightarrow j - n \leq k$ ⟨*proof*⟩

4.3 Inequalities for Suc and pred

thm *Nat.less-Suc-eq-le*
corollary *less-eq-le-pred*: $0 < (n::nat) \Longrightarrow (m < n) = (m \leq n - Suc\ 0)$
⟨*proof*⟩
corollary *less-imp-le-pred*: $m < n \Longrightarrow m \leq n - Suc\ 0$ ⟨*proof*⟩
corollary *le-pred-imp-less*: $[\![0 < n; m \leq n - Suc\ 0]\!] \Longrightarrow m < n$ ⟨*proof*⟩

thm *Nat.Suc-le-eq*
corollary *pred-less-eq-le*: $0 < m \Longrightarrow (m - Suc\ 0 < n) = (m \leq n)$ ⟨*proof*⟩
corollary *pred-less-imp-le*: $m - Suc\ 0 < n \Longrightarrow m \leq n$ ⟨*proof*⟩
corollary *le-imp-pred-less*: $[\![\ 0 < m;\ m \leq n\ ]\!] \Longrightarrow m - Suc\ 0 < n$ ⟨*proof*⟩

thm *Nat.diff-add-inverse*
lemma *diff-add-inverse-Suc*: $n < m \Longrightarrow n + (m - Suc\ n) = m - Suc\ 0$ ⟨*proof*⟩

thm *Nat.Suc-mono*
lemma *pred-mono*: $[\![\ m < n;\ 0 < m\ ]\!] \Longrightarrow m - Suc\ 0 < n - Suc\ 0$ ⟨*proof*⟩
corollary *pred-Suc-mono*: $[\![\ m < Suc\ n;\ 0 < m\ ]\!] \Longrightarrow m - Suc\ 0 < n$ ⟨*proof*⟩

lemma *Suc-less-pred-conv*: $(Suc\ m < n) = (m < n - Suc\ 0)$ ⟨*proof*⟩
lemma *Suc-le-pred-conv*: $0 < n \Longrightarrow (Suc\ m \leq n) = (m \leq n - Suc\ 0)$ ⟨*proof*⟩
lemma *Suc-le-imp-le-pred*: $Suc\ m \leq n \Longrightarrow m \leq n - Suc\ 0$ ⟨*proof*⟩

4.4 Additional facts about cancellation in (in-)equalities

lemma *diff-cancel-imp-eq*: $[\![\ 0 < (n::nat);\ \ n + i - j = n\ ]\!] \Longrightarrow i = j$ ⟨*proof*⟩

thm
Nat.nat-add-left-cancel-less
Nat.nat-add-left-cancel-le
Nat.nat-add-right-cancel
Nat.nat-add-left-cancel
Nat.diff-diff-eq
Nat.eq-diff-iff
Nat.less-diff-iff
Nat.le-diff-iff
lemma *nat-diff-left-cancel-less*: $k - m < k - (n::nat) \Longrightarrow n < m$ ⟨*proof*⟩
lemma *nat-diff-right-cancel-less*: $n - k < (m::nat) - k \Longrightarrow n < m$ ⟨*proof*⟩

lemma *nat-diff-left-cancel-le1*: $[\![\ k - m \leq k - (n::nat);\ m < k\ ]\!] \Longrightarrow n \leq m$ ⟨*proof*⟩
lemma *nat-diff-left-cancel-le2*: $[\![\ k - m \leq k - (n::nat);\ n \leq k\ ]\!] \Longrightarrow n \leq m$ ⟨*proof*⟩

lemma *nat-diff-right-cancel-le1*: $[\![\ m - k \leq n - (k::nat);\ k < m\ ]\!] \Longrightarrow m \leq n$ ⟨*proof*⟩
lemma *nat-diff-right-cancel-le2*: $[\![\ m - k \leq n - (k::nat);\ k \leq n\ ]\!] \Longrightarrow m \leq n$ ⟨*proof*⟩

lemma *nat-diff-left-cancel-eq1*: $[\![\ k - m = k - (n::nat);\ m < k\ ]\!] \Longrightarrow m = n$ ⟨*proof*⟩
lemma *nat-diff-left-cancel-eq2*: $[\![\ k - m = k - (n::nat);\ n < k\ ]\!] \Longrightarrow m = n$

⟨*proof*⟩

lemma *nat-diff-right-cancel-eq1*: $[\![\ m - k = n - (k::nat);\ k < m\ ]\!] \Longrightarrow m = n$
⟨*proof*⟩
lemma *nat-diff-right-cancel-eq2*: $[\![\ m - k = n - (k::nat);\ k < n\ ]\!] \Longrightarrow m = n$
⟨*proof*⟩

thm *eq-diff-iff*
lemma *eq-diff-left-iff*: $[\![\ (m::nat) \leq k;\ n \leq k]\!] \Longrightarrow (k - m = k - n) = (m = n)$
⟨*proof*⟩

thm *Nat.nat-add-right-cancel Nat.nat-add-left-cancel*
thm *Nat.diff-le-mono*
lemma *eq-imp-diff-eq*: $m = (n::nat) \Longrightarrow m - k = n - k$ ⟨*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
Nat.le-add-diff
add-diff-le
less-add-diff
add-diff-less

thm
Nat.le-diff-conv Nat.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
Nat.nat-add-right-cancel
Nat.nat-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
Nat.nat-add-right-cancel Nat.nat-add-left-cancel
Nat.diff-le-mono
eq-imp-diff-eq

end

5 Util-Div: Results for division and modulo operators on integers

theory *Util-Div*
imports *Util-Nat Presburger Sledgehammer*
begin

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

thm *Divides.mod-less-divisor*
corollary *Suc-mod-le-divisor*: $0 < m \Longrightarrow Suc\ (n\ mod\ m) \leq m$
⟨*proof*⟩

thm *Divides.mod-less*
thm *Divides.mod-less-divisor*
lemma *mod-less-dividend*: $[\![\ 0 < m;\ m \leq n\ ]\!] \Longrightarrow n\ mod\ m < (n::nat)$
thm *mod-less-divisor*[*of m n*]
thm *less-le-trans*[*OF mod-less-divisor*[*of m n*]]
⟨*proof*⟩
thm *Divides.mod-le-divisor*

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

lemma *diff-mod-le*: $(t - r)\ mod\ m \leq (t::nat)$
thm *le-trans*[*OF mod-le-dividend*, *OF diff-le-self*]
⟨*proof*⟩

thm *Divides.mult-div-cancel*

lemmas *div-mult-cancel* = *div-mod-equality'*

lemma *mod-0-div-mult-cancel*: $(n \ mod \ (m::nat) = 0) = (n \ div \ m * m = n)$
⟨*proof*⟩

thm *Divides.mult-div-cancel*
lemma *div-mult-le*: $(n::nat) \ div \ m * m \leq n$
⟨*proof*⟩
lemma *less-div-Suc-mult*: $0 < (m::nat) \Longrightarrow n < Suc \ (n \ div \ m) * m$
⟨*proof*⟩

lemma *nat-ge2-conv*: $((2::nat) \leq n) = (n \neq 0 \wedge n \neq 1)$
⟨*proof*⟩

lemma *Suc0-mod*: $m \neq Suc \ 0 \Longrightarrow Suc \ 0 \ mod \ m = Suc \ 0$
⟨*proof*⟩
corollary *Suc0-mod-subst*:
$[\![\ m \neq Suc \ 0; \ P \ (Suc \ 0) \ ]\!] \Longrightarrow P \ (Suc \ 0 \ mod \ m)$
thm *subst*[*OF Suc0-mod*[*symmetric*]]
⟨*proof*⟩
corollary *Suc0-mod-cong*:
$m \neq Suc \ 0 \Longrightarrow f \ (Suc \ 0 \ mod \ m) = f \ (Suc \ 0)$
thm *arg-cong*[*OF Suc0-mod*]
⟨*proof*⟩

5.2 Additional results for addition and subtraction with *mod*

thm *Divides.mod-Suc*
lemma *mod-Suc-conv*:
$((Suc \ a) \ mod \ m = (Suc \ b) \ mod \ m) = (a \ mod \ m = b \ mod \ m)$
⟨*proof*⟩
thm *mod-Suc-conv*

thm *mod-Suc*
lemma *mod-Suc'*:
$0 < n \Longrightarrow Suc \ m \ mod \ n = (if \ m \ mod \ n < n - Suc \ 0 \ then \ Suc \ (m \ mod \ n) \ else \ 0)$
⟨*proof*⟩

lemma *mod-add*:
$((a + k) \ mod \ m = (b + k) \ mod \ m) =$
$((a::nat) \ mod \ m = b \ mod \ m)$
⟨*proof*⟩

corollary *mod-sub-add*:
$k \leq (a::nat) \Longrightarrow$
$((a - k) \ mod \ m = b \ mod \ m) = (a \ mod \ m = (b + k) \ mod \ m)$
thm *mod-add*[**where** $m{=}m$ **and** $a{=}a{-}k$ **and** $b{=}b$ **and** $k{=}k$, *symmetric*]
⟨*proof*⟩

thm
mod-Suc-conv
mod-add
mod-sub-add

lemma *mod-sub-eq-mod-0-conv*:
$a + b \leq (n::nat) \Longrightarrow$
$((n - a)\ mod\ m = b\ mod\ m) = ((n - (a + b))\ mod\ m = 0)$
thm *mod-add*[*of* $n-(a+b)$ b m 0]
⟨*proof*⟩
lemma *mod-sub-eq-mod-swap*:
$[\![\ a \leq (n::nat);\ b \leq n\ ]\!] \Longrightarrow$
$((n - a)\ mod\ m = b\ mod\ m) = ((n - b)\ mod\ m = a\ mod\ m)$
thm *mod-sub-add*
⟨*proof*⟩

lemma *le-mod-greater-imp-div-less*:
$[\![\ a \leq (b::nat);\ a\ mod\ m > b\ mod\ m\ ]\!] \Longrightarrow a\ div\ m < b\ div\ m$
⟨*proof*⟩
thm *mult-le-mono1*[*of* $b\ div\ m\ a\ div\ m\ m$]
⟨*proof*⟩
thm *add-less-le-mono*[*of* $b\ mod\ m\ a\ mod\ m\ b\ div\ m * m\ a\ div\ m * m$]
⟨*proof*⟩

lemma *less-mod-ge-imp-div-less*: $[\![\ a < (b::nat);\ a\ mod\ m \geq b\ mod\ m\ ]\!] \Longrightarrow a\ div\ m < b\ div\ m$
⟨*proof*⟩
thm *mult-less-cancel2*[*THEN iffD1*, *THEN conjunct2*]
⟨*proof*⟩
corollary *less-mod-0-imp-div-less*: $[\![\ a < (b::nat);\ b\ mod\ m = 0\ ]\!] \Longrightarrow a\ div\ m < b\ div\ m$
⟨*proof*⟩

lemma *mod-diff-right-eq*:
$(a::nat) \leq b \Longrightarrow (b - a)\ mod\ m = (b - a\ mod\ m)\ mod\ m$
⟨*proof*⟩
thm *diff-diff-left*[*symmetric*]
⟨*proof*⟩
corollary *mod-eq-imp-diff-mod-eq*:
$[\![\ x\ mod\ m = y\ mod\ m;\ x \leq (t::nat);\ y \leq t\ ]\!] \Longrightarrow$
$(t - x)\ mod\ m = (t - y)\ mod\ m$
thm *mod-diff-right-eq*
⟨*proof*⟩
lemma *mod-eq-imp-diff-mod-eq2*:
$[\![\ x\ mod\ m = y\ mod\ m;\ (t::nat) \leq x;\ t \leq y\ ]\!] \Longrightarrow$
$(x - t)\ mod\ m = (y - t)\ mod\ m$
⟨*proof*⟩

thm *mod-mult-self2*[*of* $x - t\ m\ t$]
⟨*proof*⟩
thm
mod-diff-right-eq
mod-eq-imp-diff-mod-eq
mod-eq-imp-diff-mod-eq2

lemma *divisor-add-diff-mod-if*:
$(m + b\ mod\ m - a\ mod\ m)\ mod\ (m::nat) = ($
if $a\ mod\ m \leq b\ mod\ m$
then $(b\ mod\ m - a\ mod\ m)$
else $(m + b\ mod\ m - a\ mod\ m))$
⟨*proof*⟩
thm *less-imp-diff-less*
⟨*proof*⟩
corollary *divisor-add-diff-mod-eq1*:
$a\ mod\ m \leq b\ mod\ m \Longrightarrow$
$(m + b\ mod\ m - a\ mod\ m)\ mod\ (m::nat) = b\ mod\ m - a\ mod\ m$
⟨*proof*⟩
corollary *divisor-add-diff-mod-eq2*:
$b\ mod\ m < a\ mod\ m \Longrightarrow$
$(m + b\ mod\ m - a\ mod\ m)\ mod\ (m::nat) = m + b\ mod\ m - a\ mod\ m$
⟨*proof*⟩

lemma *mod-add-mod-if*:
$(a\ mod\ m + b\ mod\ m)\ mod\ (m::nat) = ($
if $a\ mod\ m + b\ mod\ m < m$
then $a\ mod\ m + b\ mod\ m$
else $a\ mod\ m + b\ mod\ m - m)$
⟨*proof*⟩
thm *diff-less-conv*[*THEN iffD2*]
⟨*proof*⟩
corollary *mod-add-mod-eq1*:
$a\ mod\ m + b\ mod\ m < m \Longrightarrow$
$(a\ mod\ m + b\ mod\ m)\ mod\ (m::nat) = a\ mod\ m + b\ mod\ m$
⟨*proof*⟩
corollary *mod-add-mod-eq2*:
$m \leq a\ mod\ m + b\ mod\ m \Longrightarrow$
$(a\ mod\ m + b\ mod\ m)\ mod\ (m::nat) = a\ mod\ m + b\ mod\ m - m$
⟨*proof*⟩

thm *Divides.mod-add-eq*
lemma *mod-add1-eq-if*:
$(a + b)\ mod\ (m::nat) = ($
if $(a\ mod\ m + b\ mod\ m < m)$ *then* $a\ mod\ m + b\ mod\ m$
else $a\ mod\ m + b\ mod\ m - m)$
⟨*proof*⟩

lemma *mod-add-eq-mod-conv*: $0 < (m::nat) \Longrightarrow$
$((x + a)\ mod\ m = b\ mod\ m\ ) =$
$(x\ mod\ m = (m + b\ mod\ m - a\ mod\ m)\ mod\ m)$
⟨*proof*⟩
thm *mod-add-mod-if*[*of x m a*]
⟨*proof*⟩
thm *mod-add-left-eq*[*symmetric*]
thm *le-add-diff-inverse2*[*OF trans-le-add1*[*OF mod-le-divisor*], *of m*]
⟨*proof*⟩

lemma *mod-diff1-eq*:
$(a::nat) \leq b \Longrightarrow (b - a)\ mod\ m = (m + b\ mod\ m - a\ mod\ m)\ mod\ m$
⟨*proof*⟩
thm *diff-add-assoc2*[*of a mod m b m*, *symmetric*]
⟨*proof*⟩
thm *mod-div-equality*
⟨*proof*⟩
thm *diff-add-assoc*[*of a mod m b mod m + m*]
⟨*proof*⟩
thm *diff-add-assoc*[*OF mod-le-divisor*, *OF m-as*]
⟨*proof*⟩
thm *divisor-add-diff-mod-if*
corollary *mod-diff1-eq-if*:
$(a::nat) \leq b \Longrightarrow (b - a)\ mod\ m = ($
if a mod m $\leq$ *b mod m then b mod m - a mod m*
else m + b mod m - a mod m)
⟨*proof*⟩
corollary *mod-diff1-eq1*:
$[\![\ (a::nat) \leq b;\ a\ mod\ m \leq b\ mod\ m\ ]\!]$
$\Longrightarrow (b - a)\ mod\ m = b\ mod\ m - a\ mod\ m$
⟨*proof*⟩
corollary *mod-diff1-eq2*:
$[\![\ (a::nat) \leq b;\ b\ mod\ m < a\ mod\ m]\!]$
$\Longrightarrow (b - a)\ mod\ m = m + b\ mod\ m - a\ mod\ m$
⟨*proof*⟩

5.2.1 Divisor subtraction with *div* and *mod*

thm
Divides.mod-add-self1
Divides.mod-add-self2
Divides.mod-mult-self1
Divides.mod-mult-self2
thm *mod-diff1-eq2*
lemma *mod-diff-self1*:
$0 < (n::nat) \Longrightarrow (m - n)\ mod\ m = m - n$

⟨*proof*⟩
lemma *mod-diff-self2*:
$m \leq (n::nat) \Longrightarrow (n - m)\ mod\ m = n\ mod\ m$
thm *mod-diff-right-eq*[*of m n m*]
⟨*proof*⟩
lemma *mod-diff-mult-self1*:
$k * m \leq (n::nat) \Longrightarrow (n - k * m)\ mod\ m = n\ mod\ m$
thm *mod-diff-right-eq*[*of m n m*]
⟨*proof*⟩
lemma *mod-diff-mult-self2*:
$m * k \leq (n::nat) \Longrightarrow (n - m * k)\ mod\ m = n\ mod\ m$
⟨*proof*⟩

thm
Divides.div-add-self1
Divides.div-add-self2
Divides.div-mult-self1
Divides.div-mult-self2
thm *div-0*
lemma *div-diff-self1*: $0 < (n::nat) \Longrightarrow (m - n)\ div\ m = 0$
⟨*proof*⟩
lemma *div-diff-self2*: $(n - m)\ div\ m = n\ div\ m - Suc\ 0$
⟨*proof*⟩
thm *div-if*
⟨*proof*⟩

lemma *div-diff-mult-self1*:
$(n - k * m)\ div\ m = n\ div\ m - (k::nat)$
⟨*proof*⟩
thm *div-le-mono*[*OF less-imp-le*]
⟨*proof*⟩
thm *iffD1*[*OF mult-cancel1-gr0*[**where** *k=m*]]
⟨*proof*⟩
thm *diff-mult-distrib2*
⟨*proof*⟩
thm *mult-div-cancel*[*of m n − k * m*]
⟨*proof*⟩
thm *mod-diff-mult-self1*
⟨*proof*⟩
lemma *div-diff-mult-self2*:
$(n - m * k)\ div\ m = n\ div\ m - (k::nat)$
⟨*proof*⟩

5.2.2 Modulo equality and modulo of difference

lemma *mod-eq-imp-diff-mod-0*:
$(a::nat)\ mod\ m = b\ mod\ m \Longrightarrow (b - a)\ mod\ m = 0$
(**is** *?P* $\Longrightarrow$ *?Q*)
⟨*proof*⟩

thm *mod-div-equality*
⟨*proof*⟩
corollary *mod-eq-imp-diff-dvd*:
$(a::nat)\ mod\ m = b\ mod\ m \Longrightarrow m\ dvd\ b - a$
⟨*proof*⟩

thm *mod-diff1-eq*
lemma *mod-neq-imp-diff-mod-neq0*:
$[\![\ (a::nat)\ mod\ m \neq b\ mod\ m;\ a \leq b\ ]\!] \Longrightarrow 0 < (b - a)\ mod\ m$
⟨*proof*⟩
thm *mod-diff1-eq1*
⟨*proof*⟩
thm *mod-diff1-eq2*[*OF less-imp-le*]
thm *trans-less-add1*[*OF mod-less-divisor*]
⟨*proof*⟩
corollary *mod-neq-imp-diff-not-dvd*:
$[\![\ (a::nat)\ mod\ m \neq b\ mod\ m;\ a \leq b\ ]\!] \Longrightarrow \neg\ m\ dvd\ b - a$
thm *dvd-eq-mod-eq-0*
⟨*proof*⟩

lemma *diff-mod-0-imp-mod-eq*:
$[\![\ (b - a)\ mod\ m = 0;\ a \leq b\ ]\!] \Longrightarrow (a::nat)\ mod\ m = b\ mod\ m$
⟨*proof*⟩
thm *mod-neq-imp-diff-mod-neq0*
⟨*proof*⟩
corollary *diff-dvd-imp-mod-eq*:
$[\![\ m\ dvd\ b - a;\ a \leq b\ ]\!] \Longrightarrow (a::nat)\ mod\ m = b\ mod\ m$
thm *dvd-eq-mod-eq-0*[*THEN iffD1*, *THEN diff-mod-0-imp-mod-eq*]
⟨*proof*⟩

thm
mod-eq-imp-diff-mod-0
diff-mod-0-imp-mod-eq
lemma *mod-eq-diff-mod-0-conv*:
$a \leq (b::nat) \Longrightarrow (a\ mod\ m = b\ mod\ m) = ((b - a)\ mod\ m = 0)$
⟨*proof*⟩
corollary *mod-eq-diff-dvd-conv*:
$a \leq (b::nat) \Longrightarrow (a\ mod\ m = b\ mod\ m) = (m\ dvd\ b - a)$
thm *dvd-eq-mod-eq-0*[*symmetric*, *THEN subst*]
⟨*proof*⟩

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

lemma *zmod-eq-imp-diff-mod-0*:
$(a::int)\ mod\ m = b\ mod\ m \Longrightarrow (b - a)\ mod\ m = 0$
⟨*proof*⟩
thm *mod-add-eq*[*of b −a m*]

⟨*proof*⟩
thm *zmod-zminus1-eq-if*
⟨*proof*⟩

thm
zmod-eq-imp-diff-mod-0[*of* a::*int* *m* *b*]
mod-eq-imp-diff-mod-0[*of* a::*nat* *m* *b*]

thm *Divides.nat-mod-distrib*
lemma ⟦ $0 \leq (n::int)$; $0 \leq m$ ⟧ ⟹ $nat\ (n\ mod\ m) = nat\ n\ mod\ nat\ m$
⟨*proof*⟩

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

lemma *zdiff-mod-0-imp-mod-eq--pos*:
⟦ $(b - a)\ mod\ m = 0$; $0 < (m::int)$ ⟧ ⟹ $a\ mod\ m = b\ mod\ m$
(**is** ⟦ *?P*; *?Pm* ⟧ ⟹ *?Q*)
⟨*proof*⟩

thm *divmod-int-rel-div-mod*[*of* *m* *a*]
thm *Divides.divmod-int-rel-def*

⟨*proof*⟩
thm *notI*[*of* *r1* ≠ *r2*, *simplified*]
⟨*proof*⟩
thm *pos-mod-conj*
⟨*proof*⟩
thm *minus-less-iff*[*of* *m*]
thm *diff-le-s*
⟨*proof*⟩
thm *mod-pos-pos-trivial*[*of* *r2*−*r1* *m*]
⟨*proof*⟩
thm *zmod-zminus2-eq-if*[*of* *r2*−*r1* *m*, *simplified*]
⟨*proof*⟩
thm *mod-neg-neg-trivial*[*of* *r2*−*r1* −*m*]
⟨*proof*⟩

lemma *zmod-zminus-eq-conv-pos*:
$0 < (m::int) \Longrightarrow (a\ mod\ -\ m = b\ mod\ -\ m) = (a\ mod\ m = b\ mod\ m)$
⟨*proof*⟩
thm *zmod-zminus1-eq-if*[*of* - *m*]
⟨*proof*⟩
thm *pos-mod-bound*[*of* *m*]
⟨*proof*⟩
lemma *zmod-zminus-eq-conv*:
$((a::int)\ mod\ -\ m = b\ mod\ -\ m) = (a\ mod\ m = b\ mod\ m)$
⟨*proof*⟩
thm *zmod-zminus-eq-conv-pos*[*of* −*m*, *symmetric*]

⟨*proof*⟩

lemma *zdiff-mod-0-imp-mod-eq*:
(*b* − *a*) *mod m* = *0* ⟹ (*a*::*int*) *mod m* = *b mod m*
⟨*proof*⟩
thm *zdiff-mod-0-imp-mod-eq*

thm
zmod-eq-imp-diff-mod-0
zdiff-mod-0-imp-mod-eq
lemma *zmod-eq-diff-mod-0-conv*:
((*a*::*int*) *mod m* = *b mod m*) = ((*b* − *a*) *mod m* = *0*)
⟨*proof*⟩

lemma ¬(∃(*a*::*int*) *b m*. (*b* − *a*) *mod m* = *0* ∧ *a mod m* ≠ *b mod m*)
⟨*proof*⟩
lemma ∃(*a*::*nat*) *b m*. (*b* − *a*) *mod m* = *0* ∧ *a mod m* ≠ *b mod m*
⟨*proof*⟩

thm
zdiff-mod-0-imp-mod-eq[*of b*::*int a m*]
diff-mod-0-imp-mod-eq[*of b*::*nat a m*]

lemma *zmult-div-leq-mono*:
⟦ (*0*::*int*) ≤ *x*; *a* ≤ *b*; *0* < *d* ⟧ ⟹ *x* ∗ *a div d* ≤ *x* ∗ *b div d*
⟨*proof*⟩

lemma *zmult-div-leq-mono-neg*:
⟦ *x* ≤ (*0*::*int*); *a* ≤ *b*; *0* < *d* ⟧ ⟹ *x* ∗ *b div d* ≤ *x* ∗ *a div d*
⟨*proof*⟩

lemma *zmult-div-pos-le*:
⟦ (*0*::*int*) ≤ *a*; *0* ≤ *b*; *b* ≤ *c* ⟧ ⟹ *a* ∗ *b div c* ≤ *a*
⟨*proof*⟩
thm *zdiv-mono1*
⟨*proof*⟩

lemma *zmult-div-neg-le*:
⟦ *a* ≤ (*0*::*int*); *0* < *c*; *c* ≤ *b* ⟧ ⟹ *a* ∗ *b div c* ≤ *a*
⟨*proof*⟩
thm *zdiv-mono1*
⟨*proof*⟩

lemma *zmult-div-ge-0*:⟦ (*0*::*int*) ≤ *x*; *0* ≤ *a*; *0* < *c* ⟧ ⟹ *0* ≤ *a* ∗ *x div c*
⟨*proof*⟩

corollary *zmult-div-plus-ge-0*:
$[\![(0::int) \leq x;\ 0 \leq a;\ 0 \leq b;\ 0 < c]\!] \Longrightarrow 0 \leq a * x\ div\ c + b$
⟨*proof*⟩

lemma *zmult-div-abs-ge*:
$[\![(0::int) \leq b;\ b \leq b';\ 0 \leq a;\ 0 < c]\!] \Longrightarrow$
$|a * b\ div\ c| \leq |a * b'\ div\ c|$
thm *zmult-div-ge-0*[*of b a c*]
⟨*proof*⟩

lemma *zmult-div-plus-abs-ge*:
$[\![(0::int) \leq b;\ b \leq b';\ 0 \leq a;\ 0 < c\ ]\!] \Longrightarrow$
$|a * b\ div\ c + a| \leq |a * b'\ div\ c + a|$
thm *zmult-div-plus-ge-0*
⟨*proof*⟩
thm *zmult-div-plus-abs-ge*

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

lemma *less-mod-eq-imp-add-divisor-le*:
$[\![(x::nat) < y;\ x\ mod\ m = y\ mod\ m\ ]\!] \Longrightarrow x + m \leq y$
⟨*proof*⟩
thm *contrapos-pp*[*of x mod m = y mod m*]
⟨*proof*⟩
thm *mod-eq-imp-diff-mod-0*
⟨*proof*⟩
thm *y-x-greater-0 y-x-less-m*
thm *mod-less*[*of y − x m*]
⟨*proof*⟩

lemma *less-div-imp-mult-add-divisor-le*:
$(x::nat) < n\ div\ m \Longrightarrow x * m + m \leq n$
⟨*proof*⟩
thm *le-diff-conv2*[*of m*]
⟨*proof*⟩
thm *le-diff-imp-le*[*of m * x + m*]
⟨*proof*⟩

lemma *mod-add-eq-imp-mod-0*:
$((n + k)\ mod\ (m::nat) = n\ mod\ m) = (k\ mod\ m = 0)$
⟨*proof*⟩

lemma *between-imp-mod-between*:
$[\![b < (m::nat);\ m * k + a \leq n;\ n \leq m * k + b\ ]\!] \Longrightarrow$
$a \leq n\ mod\ m \wedge n\ mod\ m \leq b$

⟨*proof*⟩

corollary *between-imp-mod-le*:
⟦ $b < (m::nat)$; $m * k \leq n$; $n \leq m * k + b$ ⟧ ⟹ $n \ mod \ m \leq b$
⟨*proof*⟩
corollary *between-imp-mod-gr0*:
⟦ $(m::nat) * k < n$; $n < m * k + m$ ⟧ ⟹ $0 < n \ mod \ m$
⟨*proof*⟩

Some variations of *split-div-lemma*

corollary *le-less-div-conv*:
$0 < m \Longrightarrow (k * m \leq n \wedge n < Suc \ k * m) = (n \ div \ m = k)$
⟨*proof*⟩
lemma *le-less-imp-div*:
⟦ $k * m \leq n$; $n < Suc \ k * m$ ⟧ ⟹ $n \ div \ m = k$
⟨*proof*⟩
lemma *div-imp-le-less*:
⟦ $n \ div \ m = k$; $0 < m$ ⟧ ⟹ $k * m \leq n \wedge n < Suc \ k * m$
thm *le-less-div-conv*[*THEN iffD2*]
⟨*proof*⟩

lemma *div-le-mod-le-imp-le*:
⟦ $(a::nat) \ div \ m \leq b \ div \ m$; $a \ mod \ m \leq b \ mod \ m$ ⟧ ⟹ $a \leq b$
⟨*proof*⟩

lemma *le-mod-add-eq-imp-add-mod-le*:
⟦ $a \leq b$; $(a + k) \ mod \ m = (b::nat) \ mod \ m$ ⟧ ⟹ $a + k \ mod \ m \leq b$
⟨*proof*⟩

corollary *mult-divisor-le-mod-ge-imp-ge*:
⟦ $(m::nat) * k \leq n$; $r \leq n \ mod \ m$ ⟧ ⟹ $m * k + r \leq n$
⟨*proof*⟩

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

lemma *mod-0-imp-mod-mult-right-0*:
$n \ mod \ m = (0::nat) \Longrightarrow n * k \ mod \ m = 0$
⟨*proof*⟩
lemma *mod-0-imp-mod-mult-left-0*:
$n \ mod \ m = (0::nat) \Longrightarrow k * n \ mod \ m = 0$
⟨*proof*⟩

lemma *mod-0-imp-div-mult-left-eq*:
$n \ mod \ m = (0::nat) \Longrightarrow k * n \ div \ m = k * (n \ div \ m)$
⟨*proof*⟩
lemma *mod-0-imp-div-mult-right-eq*:

$n\ mod\ m = (0::nat) \Longrightarrow n * k\ div\ m = k * (n\ div\ m)$
⟨*proof*⟩

thm *Rings.dvd-mult-left*
lemma *mod-0-imp-mod-factor-0-left*:
$n\ mod\ (m * m') = (0::nat) \Longrightarrow n\ mod\ m = 0$
⟨*proof*⟩
lemma *mod-0-imp-mod-factor-0-right*:
$n\ mod\ (m * m') = (0::nat) \Longrightarrow n\ mod\ m' = 0$
⟨*proof*⟩

5.6 Some factor distribution facts for *mod*

lemma *mod-eq-mult-distrib*:
$(a::nat)\ mod\ m = b\ mod\ m \Longrightarrow$
$a * k\ mod\ (m * k) = b * k\ mod\ (m * k)$
⟨*proof*⟩

lemma *mod-mult-eq-imp-mod-eq*:
$(a::nat)\ mod\ (m * k) = b\ mod\ (m * k) \Longrightarrow a\ mod\ m = b\ mod\ m$
thm *mod-mult2-eq*[*of a m k*]
⟨*proof*⟩
thm *arg-cong*[**where** $f=\lambda x.\ x\ mod\ m$]
⟨*proof*⟩
corollary *mod-eq-mod-0-imp-mod-eq*:
$[\![\ (a::nat)\ mod\ m' = b\ mod\ m';\ m'\ mod\ m = 0\ ]\!]$
$\Longrightarrow a\ mod\ m = b\ mod\ m$
⟨*proof*⟩

lemma *mod-factor-imp-mod-0*:
$[\![(x::nat)\ mod\ (m * k) = y * k\ mod\ (m * k)]\!] \Longrightarrow x\ mod\ k = 0$
(**is** $[\![\ ?P1\ ]\!] \Longrightarrow ?Q$)
⟨*proof*⟩
thm *mod-mult-distrib*[**where** $m=y$ **and** $n=m$ **and** $k=k$]
⟨*proof*⟩
thm *mod-add-eq*
⟨*proof*⟩
corollary *mod-factor-div*:
$[\![(x::nat)\ mod\ (m * k) = y * k\ mod\ (m * k)]\!] \Longrightarrow x\ div\ k * k = x$
thm *mod-factor-imp-mod-0*[*THEN mod-0-div-mult-cancel*[*THEN iffD1*]]
⟨*proof*⟩

lemma *mod-factor-div-mod*:
$[\![\ (x::nat)\ mod\ (m * k) = y * k\ mod\ (m * k);\ 0 < k\ ]\!]$
$\Longrightarrow x\ div\ k\ mod\ m = y\ mod\ m$
(**is** $[\![\ ?P1;\ ?P2\ ]\!] \Longrightarrow ?L = ?R$)
⟨*proof*⟩
thm *mod-mult2-eq*[**where** $a=x$ **and** $b=k$ **and** $c=m$]

⟨*proof*⟩
thm *mod-mult-distrib*
⟨*proof*⟩

thm
Divides.mod-mult-distrib
Divides.mod-mult-distrib2
thm
mod-eq-mult-distrib
thm
mod-factor-imp-mod-0
mod-factor-div
mod-factor-div-mod

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

thm *Divides.div-add1-eq*
thm *mod-add1-eq-if*
lemma *div-add1-eq-if*: $0 < m \Longrightarrow$
$(a + b)\ div\ (m::nat) = a\ div\ m + b\ div\ m + ($
if $a\ mod\ m + b\ mod\ m < m$ *then* 0 *else* $Suc\ 0)$
⟨*proof*⟩
thm *arg-cong*[*of* $(a\ mod\ m + b\ mod\ m)\ div\ m$]
⟨*proof*⟩
thm *le-less-imp-div*[*of* $Suc\ 0\ m\ a\ mod\ m + b\ mod\ m$]
⟨*proof*⟩
corollary *div-add1-eq1*:
$a\ mod\ m + b\ mod\ m < (m::nat) \Longrightarrow$
$(a + b)\ div\ (m::nat) = a\ div\ m + b\ div\ m$
⟨*proof*⟩
corollary *div-add1-eq1-mod-0-left*:
$a\ mod\ m = 0 \Longrightarrow (a + b)\ div\ (m::nat) = a\ div\ m + b\ div\ m$
⟨*proof*⟩
corollary *div-add1-eq1-mod-0-right*:
$b\ mod\ m = 0 \Longrightarrow (a + b)\ div\ (m::nat) = a\ div\ m + b\ div\ m$
⟨*proof*⟩
corollary *div-add1-eq2*:
$[\![\ 0 < m; (m::nat) \leq a\ mod\ m + b\ mod\ m\ ]\!] \Longrightarrow$
$(a + b)\ div\ (m::nat) = Suc\ (a\ div\ m + b\ div\ m)$
⟨*proof*⟩

lemma *div-Suc*:
$0 < n \Longrightarrow Suc\ m\ div\ n = (if\ Suc\ (m\ mod\ n) = n\ then\ Suc\ (m\ div\ n)\ else\ m\ div\ n)$
⟨*proof*⟩
thm *le-neq-trans*[*OF mod-less-divisor*[*THEN Suc-leI*]]
⟨*proof*⟩
lemma *div-Suc'*:

$0 < n \Longrightarrow Suc\ m\ div\ n = (if\ m\ mod\ n < n - Suc\ 0\ then\ m\ div\ n\ else\ Suc\ (m\ div\ n))$
⟨*proof*⟩

lemma *div-diff1-eq-if*:
$(b - a)\ div\ (m::nat) =$
$b\ div\ m - a\ div\ m - (if\ a\ mod\ m \leq b\ mod\ m\ then\ 0\ else\ Suc\ 0)$
⟨*proof*⟩
thm *mod-div-equality*[*of b m*]
⟨*proof*⟩
thm *mult-le-mono1*[*OF div-le*]
thm *diff-add-assoc2*[*OF mult-le-mono1*[*OF div-le*]]
⟨*proof*⟩
thm *b-a-s1*
⟨*proof*⟩
thm *div-add1-eq*
⟨*proof*⟩
thm *div-mult-self1*
⟨*proof*⟩
thm *less-imp-neq*[*OF m-as*, *THEN not-sym*]
thm *div-mult-self1*[*OF less-imp-neq*[*OF m-as*, *THEN not-sym*]]
⟨*proof*⟩
thm *le-mod-greater-imp-div-less*
⟨*proof*⟩
thm *a-div-less*
thm *a-div-less*[*THEN zero-less-diff*[*THEN iffD2*]]
thm *a-div-less*[*THEN zero-less-diff*[*THEN iffD2*],
THEN Suc-le-eq[*THEN iffD2*]]
thm *diff-add-assoc2*
⟨*proof*⟩
thm *diff-add-assoc*[*of a mod m m + b mod m*]
thm *trans-le-add1*[*of a mod m m*, *OF mod-le-divisor*]
⟨*proof*⟩
thm *div-mult-self1*
thm *div-mult-self1*[*OF less-imp-neq*[*OF m-as*, *THEN not-sym*]]
⟨*proof*⟩
thm *add-diff-less*
⟨*proof*⟩

thm *div-diff1-eq-if*
corollary *div-diff1-eq*:
$(b - a)\ div\ (m::nat) =$
$b\ div\ m - a\ div\ m - (m + a\ mod\ m - Suc\ (b\ mod\ m))\ div\ m$
⟨*proof*⟩
thm *div-diff1-eq-if*
thm *subst*[*of*
if a mod m ≤ b mod m then 0 else Suc 0
$(m + a\ mod\ m - Suc(b\ mod\ m))\ div\ m$]
⟨*proof*⟩

thm *Suc-le-eq*[*of b mod m, THEN iffD2*]
⟨*proof*⟩

corollary *div-diff1-eq1*:
a mod m ≤ *b mod m* ⟹
(*b* − *a*) *div* (*m*::*nat*) = *b div m* − *a div m*
⟨*proof*⟩
corollary *div-diff1-eq1-mod-0*:
a mod m = *0* ⟹
(*b* − *a*) *div* (*m*::*nat*) = *b div m* − *a div m*
⟨*proof*⟩
corollary *div-diff1-eq2*:
b mod m < *a mod m* ⟹
(*b* − *a*) *div* (*m*::*nat*) = *b div m* − *Suc* (*a 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*:
⟦ (*x*::*nat*) ≤ *y*; *y* − *x* < *m* ⟧ ⟹ *x* = *y* − (*y* − *x*) *mod m*
⟨*proof*⟩
lemma *diff-ge-divisor-imp-sub-mod-less*:
⟦ (*x*::*nat*) ≤ *y*; *m* ≤ *y* − *x*; *0* < *m* ⟧ ⟹ *x* < *y* − (*y* − *x*) *mod m*
⟨*proof*⟩
thm
diff-less-divisor-imp-sub-mod-eq
diff-ge-divisor-imp-sub-mod-less

lemma *le-imp-sub-mod-le*:
(*x*::*nat*) ≤ *y* ⟹ *x* ≤ *y* − (*y* − *x*) *mod m*
⟨*proof*⟩
thm
diff-less-divisor-imp-sub-mod-eq
diff-ge-divisor-imp-sub-mod-less
⟨*proof*⟩
thm *diff-ge-divisor-imp-sub-mod-less*
⟨*proof*⟩

thm *mod-diff1-eq*
lemma *mod-less-diff-mod*:
⟦ *n mod m* < *r*; *r* ≤ *m*; *r* ≤ (*n*::*nat*) ⟧ ⟹
(*n* − *r*) *mod m* = *m* + *n mod m* − *r*
⟨*proof*⟩
thm *mod-diff-self2*
⟨*proof*⟩

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

$[\![\ 0 < (n::nat);\ n\ mod\ m = 0\ ]\!] \Longrightarrow$
$(n - Suc\ 0)\ mod\ m = m - Suc\ 0$
⟨*proof*⟩
thm *mod-less-diff-mod*[*of n m Suc 0*]
⟨*proof*⟩
thm *mod-diff1-eq*[*of Suc 0 n m*]
⟨*proof*⟩

lemma *mod-pred*:
$0 < n \Longrightarrow$
$(n - Suc\ 0)\ mod\ m = ($
if n mod m = 0 then m − Suc 0 else n mod m − Suc 0)
⟨*proof*⟩
thm *subst*[*OF Suc0-mod*[*symmetric*], **where** $P{=}\lambda x.\ x \leq n\ mod\ m$]
⟨*proof*⟩
thm *mod-diff1-eq1*[*of Suc 0 n m*]
⟨*proof*⟩
thm *Suc0-mod*
⟨*proof*⟩
corollary *mod-pred-Suc-mod*:
$0 < n \Longrightarrow Suc\ ((n - Suc\ 0)\ mod\ m)\ mod\ m = n\ mod\ m$
⟨*proof*⟩
corollary *diff-mod-pred*:
$a < b \Longrightarrow$
$(b - Suc\ a)\ mod\ m = ($
if a mod m = b mod m then m − Suc 0 else $(b - a)\ mod\ m - Suc\ 0$)
⟨*proof*⟩
corollary *diff-mod-pred-Suc-mod*:
$a < b \Longrightarrow Suc\ ((b - Suc\ a)\ mod\ m)\ mod\ m = (b - a)\ mod\ m$
⟨*proof*⟩

lemma *mod-eq-imp-diff-mod-eq-divisor*:
$[\![\ a < b;\ 0 < m;\ a\ mod\ m = b\ mod\ m\ ]\!] \Longrightarrow$
$Suc\ ((b - Suc\ a)\ mod\ m) = m$
thm *mod-eq-imp-diff-mod-0*
⟨*proof*⟩
thm *iffD2*[*OF zero-less-diff*]
⟨*proof*⟩
thm *mod-0-imp-mod-pred*[*of b − a m*]
⟨*proof*⟩

lemma *sub-diff-mod-eq*:
$r \leq t \Longrightarrow (t - (t - r)\ mod\ m)\ mod\ (m::nat) = r\ mod\ m$
⟨*proof*⟩

thm *diff-mod-le*
lemma *sub-diff-mod-eq′*:
$r \leq t \Longrightarrow (k * m + t - (t - r)\ mod\ m)\ mod\ (m::nat) = r\ mod\ m$

thm *diff-mod-le*[*of t r m*]
⟨*proof*⟩

lemma *mod-eq-Suc-0-conv*: $Suc\ 0 < k \Longrightarrow ((x + k - Suc\ 0)\ mod\ k = 0) = (x\ mod\ k = Suc\ 0)$
⟨*proof*⟩

lemma *mod-eq-divisor-minus-Suc-0-conv*: $Suc\ 0 < k \Longrightarrow (x\ mod\ k = k - Suc\ 0) = (Suc\ x\ mod\ k = 0)$
⟨*proof*⟩

5.8.2 Some auxiliary facts about *div*

lemma *sub-mod-div-eq-div*: $((n::nat) - n\ mod\ m)\ div\ m = n\ div\ m$
⟨*proof*⟩
thm *mult-div-cancel*[*of m n*, *symmetric*]
⟨*proof*⟩

thm *split-div-lemma*
lemma *mod-less-imp-diff-div-conv*:
$[\![\ n\ mod\ m < r;\ r \le m + n\ mod\ m]\!] \Longrightarrow (n - r)\ div\ m = n\ div\ m - Suc\ 0$
⟨*proof*⟩
thm *split-div-lemma*[*of m n div m − Suc 0 n−r*]
thm *iffD1*[*OF split-div-lemma*[*of m n div m − Suc 0 n−r*], *symmetric*]
⟨*proof*⟩

corollary *mod-0-le-imp-diff-div-conv*:
$[\![\ n\ mod\ m = 0;\ 0 < r;\ r \le m\ ]\!] \Longrightarrow (n - r)\ div\ m = n\ div\ m - Suc\ 0$
thm *mod-less-imp-diff-div-conv*[*of n m r*]
⟨*proof*⟩
corollary *mod-0-less-imp-diff-Suc-div-conv*:
$[\![\ n\ mod\ m = 0;\ r < m\ ]\!] \Longrightarrow (n - Suc\ r)\ div\ m = n\ div\ m - Suc\ 0$
⟨*proof*⟩
corollary *mod-0-imp-diff-Suc-div-conv*:
$(n - r)\ mod\ m = 0 \Longrightarrow (n - Suc\ r)\ div\ m = (n - r)\ div\ m - Suc\ 0$
⟨*proof*⟩
corollary *mod-0-imp-sub-1-div-conv*:
$n\ mod\ m = 0 \Longrightarrow (n - Suc\ 0)\ div\ m = n\ div\ m - Suc\ 0$
thm *mod-0-less-imp-diff-Suc-div-conv*[**where** *r=0*]
⟨*proof*⟩
corollary *sub-Suc-mod-div-conv*:
$(n - Suc\ (n\ mod\ m))\ div\ m = n\ div\ m - Suc\ 0$
⟨*proof*⟩

thm *Divides.div-le-mono*

lemma *div-le-conv*: $0 < m \Longrightarrow n\ div\ m \leq k = (n \leq Suc\ k * m - Suc\ 0)$
⟨*proof*⟩
thm *diff-add-assoc*[*OF Suc-leI*]
⟨*proof*⟩
thm *add-le-mono*[*OF le-refl*]
⟨*proof*⟩
thm *mod-0-imp-sub-1-div-conv*
⟨*proof*⟩

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

lemma *less-mult-imp-div-less*: $n < k * m \Longrightarrow n\ div\ m < (k::nat)$
⟨*proof*⟩
thm *mod-0-imp-sub-1-div-conv*
⟨*proof*⟩

lemma *div-less-imp-less-mult*: $[\![\ 0 < (m::nat);\ n\ div\ m < k\ ]\!] \Longrightarrow n < k * m$
⟨*proof*⟩

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

lemma *div-eq-0-conv*: $(n\ div\ (m::nat) = 0) = (m = 0 \vee n < m)$
⟨*proof*⟩
lemma *div-eq-0-conv'*: $0 < m \Longrightarrow (n\ div\ (m::nat) = 0) = (n < m)$
⟨*proof*⟩
corollary *div-gr-imp-gr-divisor*: $x < n\ div\ (m::nat) \Longrightarrow m \leq n$
⟨*proof*⟩

lemma *mod-0-less-div-conv*:
$n\ mod\ (m::nat) = 0 \Longrightarrow (k * m < n) = (k < n\ div\ m)$
⟨*proof*⟩

lemma *add-le-divisor-imp-le-Suc-div*:
$[\![\ x\ div\ m \leq n;\ y \leq m\ ]\!] \Longrightarrow (x + y)\ div\ m \leq Suc\ n$
⟨*proof*⟩

List of definitions and lemmas

thm
Divides.mod-less
Divides.mod-less-divisor
Divides.mod-le-divisor
mod-less-dividend
mod-le-dividend

thm
Divides.mult-div-cancel
mod-0-div-mult-cancel

div-mult-le
less-div-Suc-mult
thm
Suc0-mod
Suc0-mod-subst
Suc0-mod-cong

thm
Divides.mod-Suc
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
Divides.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
Divides.mod-add-self1
Divides.mod-add-self2
Divides.mod-mult-self1
Divides.mod-mult-self2

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

thm
Divides.div-add-self1
Divides.div-add-self2
Divides.div-mult-self1
Divides.div-mult-self2

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
Divides.div-add1-eq
div-add1-eq-if
div-add1-eq1
div-add1-eq2
thm
div-diff1-eq-if
div-diff1-eq
div-diff1-eq1
div-diff1-eq2

thm
div-le-conv

end

6 SetInterval2: Sets of natural numbers

theory *SetInterval2*
imports
$*ISABELLE-HOME*/*src*/*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

thm *Orderings.strict-mono-def*

thm *mono-def*
definition *mono-on* :: ('*a*::*order* ⇒ '*b*::*order*) ⇒ '*a set* ⇒ *bool* **where**
mono-on f A ≡ ∀ *a*∈*A*. ∀ *b*∈*A*. *a* ≤ *b* ⟶ *f a* ≤ *f b*
definition *strict-mono-on* :: ('*a*::*order* ⇒ '*b*::*order*) ⇒ '*a set* ⇒ *bool* **where**
strict-mono-on f A ≡ ∀ *a*∈*A*. ∀ *b*∈*A*. *a* < *b* ⟶ *f a* < *f b*

lemma *mono-on-subset*: ⟦ *mono-on f A* ; *B* ⊆ *A* ⟧ ⟹ *mono-on f B*
⟨*proof*⟩
lemma *strict-mono-on-subset*: ⟦ *strict-mono-on f A* ; *B* ⊆ *A* ⟧ ⟹ *strict-mono-on f B*
⟨*proof*⟩

lemma *mono-on-UNIV-mono-conv*: *mono-on f UNIV* = *mono f*
⟨*proof*⟩
lemma *strict-mono-on-UNIV-strict-mono-conv*:
strict-mono-on f UNIV = *strict-mono f*
⟨*proof*⟩

lemma *mono-imp-mono-on*: *mono f* ⟹ *mono-on f A*
⟨*proof*⟩
lemma *strict-mono-imp-strict-mono-on*: *strict-mono f* ⟹ *strict-mono-on f A*
⟨*proof*⟩

lemma *strict-mono-on-imp-mono-on*: *strict-mono-on f A* ⟹ *mono-on f A*
⟨*proof*⟩

6.1.2 Injectivity

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

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

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

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

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

thm *inj-image-mem-iff*
lemma *inj-on-image-mem-iff*:
$[\![\ inj\text{-}on\ f\ A;\ B \subseteq A;\ a \in A\ ]\!] \Longrightarrow (f\ a \in f\ `\ B) = (a \in B)$
⟨*proof*⟩

thm *Set.image-Un*
thm *Fun.inj-on-def*

thm *image-Int*
thm *inj-on-image-Int*
corollary *inj-on-union-image-Int*:
$inj\text{-}on\ f\ (A \cup B) \Longrightarrow f\ `\ (A \cap B) = f\ `\ A \cap f\ `\ B$
thm *inj-on-image-Int*[*OF - Un-upper1 Un-upper2*]
⟨*proof*⟩

6.1.3 Surjectivity

thm *surj-def*

definition
$surj\text{-}on :: ('a \Rightarrow 'b) \Rightarrow 'a\ set \Rightarrow 'b\ set \Rightarrow bool$
where

surj-on f A B ≡ ∀ *b*∈*B*. ∃ *a*∈*A*. *b* = *f a*
thm *surj-on-def*

lemma *surj-on-conv*: (*surj-on f A B*) = (∀ *b*∈*B*. ∃ *a*∈*A*. *b* = *f a*)
⟨*proof*⟩

lemma *surj-on-image-conv*: (*surj-on f A B*) = (*B* ⊆ *f* ' *A*)
⟨*proof*⟩

lemma *surj-on-id*: *surj-on id A A*
⟨*proof*⟩

lemma
surj-onI: ⟦ ∀ *b* ∈ *B*. ∃ *a*∈*A*. *b* = *f a* ⟧ ⟹ *surj-on f A B* **and**
surj-onD2: *surj-on f A B* ⟹ ∀ *b* ∈ *B*. ∃ *a*∈*A*. *b* = *f a* **and**
surj-onD: ⟦ *surj-on f A B*; *b* ∈ *B* ⟧ ⟹ ∃ *a*∈*A*. *b* = *f a*
⟨*proof*⟩

thm *comp-surj*
lemma *comp-surj-on*:
⟦ *surj-on f A B*; *surj-on g B C* ⟧ ⟹ *surj-on* (*g* ∘ *f*) *A C*
⟨*proof*⟩

thm
inj-on-Un
inj-on-diff
inj-on-empty
inj-on-imageI
inj-on-insert
subset-inj-on
lemma *surj-on-Un-right*: *surj-on f A* (*B1* ∪ *B2*) = (*surj-on f A B1* ∧ *surj-on f A B2*)
⟨*proof*⟩
lemma *surj-on-Un-left*:
surj-on f (*A1* ∪ *A2*) *B* =
(∃ *B1*. ∃ *B2*. *B* ⊆ *B1* ∪ *B2* ∧ *surj-on f A1 B1* ∧ *surj-on f A2 B2*)
⟨*proof*⟩
lemma *surj-on-diff-right*: *surj-on f A B* ⟹ *surj-on f A* (*B* − *B'*)
⟨*proof*⟩
lemma *surj-on-empty-right*: *surj-on f A* {}
⟨*proof*⟩
lemma *surj-on-empty-left*: *surj-on f* {} *B* = (*B* = {})
⟨*proof*⟩
lemma *surj-on-imageI*: *surj-on* (*g* ∘ *f*) *A B* ⟹ *surj-on g* (*f* ' *A*) *B*
⟨*proof*⟩
lemma *surj-on-insert-right*: *surj-on f A* (*insert b B*) = (*surj-on f A B* ∧ *surj-on f A* {*b*})

⟨*proof*⟩
lemma *surj-on-insert-left*: *surj-on f* (*insert a A*) *B* = (*surj-on f A* (*B* − {*f a*}))
⟨*proof*⟩
lemma *surj-on-subset-right*: ⟦ *surj-on f A B*; *B'* ⊆ *B* ⟧ ⟹ *surj-on f A B'*
⟨*proof*⟩
lemma *surj-on-subset-left*: ⟦ *surj-on f A B*; *A* ⊆ *A'* ⟧ ⟹ *surj-on f A' B*
⟨*proof*⟩

lemma *bij-betw-imp-surj-on*: *bij-betw f A B* ⟹ *surj-on f A B*
⟨*proof*⟩

lemma *bij-betw-inj-on-surj-on-conv*:
bij-betw f A B = (*inj-on f A* ∧ *surj-on f A B* ∧ *f* ‘ *A* ⊆ *B*)
⟨*proof*⟩

6.1.4 Induction over natural sets

lemma *image-nat-induct*:
⟦ *P* (*f* 0); ⋀*n*. *P* (*f n*) ⟹ *P* (*f* (*Suc n*)); *surj-on f UNIV I*; *a* ∈ *I* ⟧ ⟹ *P a*
⟨*proof*⟩
thm *image-nat-induct*

thm *nat-induct*
lemma *nat-induct'*[*rule-format*]:
⟦ *P n0*; ⋀*n*. ⟦ *n0* ≤ *n*; *P n* ⟧ ⟹ *P* (*Suc n*); *n0* ≤ *n* ⟧ ⟹ *P n*
thm *nat-induct*
thm *nat-induct*[**where** *n*=*n*−*n0* **and** *P*=λ*n*. *P* (*n0*+*n*)]
⟨*proof*⟩

thm
nat-induct'
nat-induct'[**where** *?n0.0*=*0*, *simplified*]
nat-induct

lemma *inat-induct*:
⟦ *P* 0; *P* ∞; ⋀*n*. *P n* ⟹ *P* (*iSuc n*)⟧ ⟹ *P n*
⟨*proof*⟩

lemma *iSuc-imp-Suc-aux-0*:
⟦ ⋀*n*. *P n* ⟹ *P* (*iSuc n*); *n0'* ≤ *n'*; *P* (*Fin n'*)⟧ ⟹ *P* (*Fin* (*Suc n'*))
⟨*proof*⟩
lemma *iSuc-imp-Suc-aux-n0*:
⟦ ⋀*n*. ⟦*Fin n0'* ≤ *n*; *P n*⟧ ⟹ *P* (*iSuc n*); *n0'* ≤ *n'*; *P* (*Fin n'*)⟧ ⟹ *P* (*Fin* (*Suc n'*))

thm *inat-defs*
⟨*proof*⟩

lemma *inat-induct'*:
⟦ *P* (*n0*::*inat*); *P* ∞; ⋀*n*. ⟦ *n0* ≤ *n*; *P n* ⟧ ⟹ *P* (*iSuc n*); *n0* ≤ *n* ⟧ ⟹ *P n*
⟨*proof*⟩

thm *nat-induct'*[**where** *?n0.0*=*n0'* **and** *n*=*n'* **and** *P*=λ*n*. *P* (*Fin n*)]
⟨*proof*⟩

thm
inat-induct'
inat-induct'[**where** *?n0.0*=*0*, *simplified*]
inat-induct
thm *inat-induct'*

thm
nat-induct
nat-induct'
thm
inat-induct
inat-induct'

thm *wellorder-class.intro*
thm *wf-def*
thm
wf-less
wf-subset

lemma *wf-less-interval*:
wf { (*x*,*y*). *x* ∈ (*I*::*nat set*) ∧ *y* ∈ *I* ∧ *x* < *y* }
thm *wf-subset*
thm *wf-subset*[**where**
p={ (*x*,*y*). *x* ∈ *I* ∧ *y* ∈ *I* ∧ *x* < *y* } **and**
r={(*x*,*y*). *x* < *y*}]
⟨*proof*⟩
thm *wf-less*
⟨*proof*⟩
thm *wf-less-interval*

thm *wf-induct*
lemma *interval-induct*:
⟦ ⋀*x*. ∀ *y*. (*x*∈(*I*::*nat set*) ∧ *y* ∈ *I* ∧ *y* < *x* ⟶ *P y*) ⟹ *P x* ⟧
⟹ *P a*
(**is** ⟦ ⋀*x*. ∀ *y*. *?IA x y* ⟹ *P x* ⟧ ⟹ *P a*)
thm *wf-induct*
thm *wf-induct*[**where** *r*={ (*x*,*y*). *x* ∈ *I* ∧ *y* ∈ *I* ∧ *x* < *y* }]
⟨*proof*⟩

corollary *interval-induct-rule*:
$[\![\bigwedge x.\ (\bigwedge y.\ (x{\in}(I{::}nat\ set) \wedge y \in I \wedge y < x \Longrightarrow P\ y)) \Longrightarrow P\ x\ ]\!]$
$\Longrightarrow P\ a$
⟨*proof*⟩

thm
wf-induct
wf-induct-rule
interval-induct
interval-induct-rule

6.1.5 Monotonicity and injectivity of artithmetic operators

lemma *add-left-inj*: *inj* $(\lambda x.\ n + (x{::}'a{::}cancel\text{-}ab\text{-}semigroup\text{-}add))$
⟨*proof*⟩
lemma *add-right-inj*: *inj* $(\lambda x.\ x + (n{::}'a{::}cancel\text{-}ab\text{-}semigroup\text{-}add))$
⟨*proof*⟩
thm
add-left-inj
add-right-inj

lemma *mult-left-inj*: $0 < n \Longrightarrow$ *inj* $(\lambda x.\ (n{::}nat) * x)$
⟨*proof*⟩
lemma *mult-right-inj*: $0 < n \Longrightarrow$ *inj* $(\lambda x.\ x * (n{::}nat))$
⟨*proof*⟩
thm
mult-left-inj
mult-right-inj

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

lemma *add-left-strict-mono*: *strict-mono* $(\lambda x.\ n + (x{::}'a{::}ordered\text{-}cancel\text{-}ab\text{-}semigroup\text{-}add))$
⟨*proof*⟩
lemma *add-right-strict-mono*: *strict-mono* $(\lambda x.\ x + (n{::}'a{::}ordered\text{-}cancel\text{-}ab\text{-}semigroup\text{-}add))$
⟨*proof*⟩

lemma *mult-left-strict-mono*: $0 < n \Longrightarrow$ *strict-mono* $(\lambda x.\ n * (x{::}nat))$
⟨*proof*⟩
lemma *mult-right-strict-mono*: $0 < n \Longrightarrow$ *strict-mono* $(\lambda x.\ x * (n{::}nat))$
⟨*proof*⟩
lemma *sub-left-strict-mono-on*: *strict-mono-on* $(\lambda x.\ (x{::}nat) - k)\ \{k..\}$
⟨*proof*⟩

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

$[\![\ 0 < (k::nat);\ \forall x{\in}I.\ \forall y{\in}I.\ x < y \longrightarrow x + k \leq y\ ]\!] \Longrightarrow$
strict-mono-on $(\lambda x.\ x\ div\ k)\ I$
⟨*proof*⟩

lemma *mod-eq-div-right-strict-mono-on*:
$[\![\ 0 < (k::nat);\ \forall x{\in}I.\ \forall y{\in}I.\ x\ mod\ k = y\ mod\ k\ ]\!] \Longrightarrow$
strict-mono-on $(\lambda x.\ x\ div\ k)\ I$
⟨*proof*⟩
thm *less-mod-eq-imp-add-divisor-le*
⟨*proof*⟩

corollary *div-right-inj-on*:
$[\![\ 0 < (k::nat);\ \forall x{\in}I.\ \forall y{\in}I.\ x < y \longrightarrow x + k \leq y\ ]\!] \Longrightarrow$
inj-on $(\lambda x.\ x\ div\ k)\ I$
⟨*proof*⟩
corollary *mod-eq-imp-div-right-inj-on*:
$[\![\ 0 < (k::nat);\ \forall x{\in}I.\ \forall y{\in}I.\ x\ mod\ k = y\ mod\ k\ ]\!] \Longrightarrow$
inj-on $(\lambda x.\ x\ div\ k)\ I$
⟨*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.

thm *Least-def*
definition
iMin :: *'a::wellorder set* ⇒ *'a*
where
iMin I ≡ *LEAST x. x* ∈ *I*

thm
Least-def
Nat.Least-Suc
Set.Least-mono
Orderings.not-less-Least
Orderings.LeastI2-ex
Orderings.LeastI2
Orderings.LeastI-ex
Orderings.Least-le
Orderings.LeastI
Orderings.wellorder-Least-lemma
Orderings.LeastI2-order
Orderings.Least-equality

6.2.1 Basic results, as for *Least*

thm *LeastI*
lemma *iMinI*: $k \in I \Longrightarrow iMin\ I \in I$
⟨*proof*⟩
thm *LeastI*
⟨*proof*⟩

thm *LeastI-ex*
lemma *iMinI-ex*: $\exists x.\ x \in I \Longrightarrow iMin\ I \in I$
⟨*proof*⟩
corollary *iMinI-ex2*: $I \neq \{\} \Longrightarrow iMin\ I \in I$
⟨*proof*⟩

thm *LeastI2*
lemma *iMinI2*: $[\![\ k \in I;\ \bigwedge x.\ x \in I \Longrightarrow P\ x\ ]\!] \Longrightarrow P\ (iMin\ I)$
thm *iMinI*
⟨*proof*⟩
thm *LeastI2-ex*
lemma *iMinI2-ex*: $[\![\ \exists x.\ x \in I;\ \bigwedge x.\ x \in I \Longrightarrow P\ x\ ]\!] \Longrightarrow P\ (iMin\ I)$
⟨*proof*⟩
lemma *iMinI2-ex2*: $[\![\ I \neq \{\}\ ;\ \bigwedge x.\ x \in I \Longrightarrow P\ x\ ]\!] \Longrightarrow P\ (iMin\ I)$
⟨*proof*⟩

thm *Least-le*
lemma *iMin-le*[*dest*]: $k \in I \Longrightarrow iMin\ I \leq k$
⟨*proof*⟩

lemma *iMin-neq-imp-greater*[*dest*]: $[\![\ k \in I;\ k \neq iMin\ I\ ]\!] \Longrightarrow iMin\ I < k$
⟨*proof*⟩

thm *Least-mono*
lemma *iMin-mono*:
$[\![\ mono\ f;\ \exists x.\ x \in I\ ]\!] \Longrightarrow iMin\ (f\ `\ I) = f\ (iMin\ I)$
⟨*proof*⟩
thm *Least-mono*
thm *Least-mono*[*of f I*]
⟨*proof*⟩
thm *iMin-le*
⟨*proof*⟩
thm *iMinI-ex*
⟨*proof*⟩
corollary *iMin-mono2*:
$[\![\ mono\ f;\ I \neq \{\}\ ]\!] \Longrightarrow iMin\ (f\ `\ I) = f\ (iMin\ I)$
⟨*proof*⟩

thm *not-less-Least*
lemma *not-less-iMin*: $k < iMin\ I \Longrightarrow k \notin I$
⟨*proof*⟩
thm *not-less-Least*
⟨*proof*⟩

lemma *Collect-not-less-iMin*: $k < iMin\ \{x.\ P\ x\} \Longrightarrow \neg\ P\ k$
⟨*proof*⟩
lemma *Collect-iMin-le*: $P\ k \Longrightarrow iMin\ \{x.\ P\ x\} \leq k$
⟨*proof*⟩

lemma *Collect-minI*: $[\![\ k \in I;\ P\ (k::('a::wellorder))\ ]\!] \Longrightarrow \exists\, x{\in}I.\ P\ x \wedge (\forall\, y{\in}I.\ y < x \longrightarrow \neg\ P\ y)$
⟨*proof*⟩
thm *iMinI*
thm *subsetD*
thm *Collect-is-subset*
thm *subsetD*[*OF - iMinI*, *OF Collect-is-subset*]
⟨*proof*⟩
corollary *Collect-minI-ex*: $\exists\, k{\in}I.\ P\ (k::('a::wellorder)) \Longrightarrow \exists\, x{\in}I.\ P\ x \wedge (\forall\, y{\in}I.\ y < x \longrightarrow \neg\ P\ y)$
⟨*proof*⟩
corollary *Collect-minI-ex2*: $\{k{\in}I.\ P\ (k::('a::wellorder))\} \neq \{\} \Longrightarrow \exists\, x{\in}I.\ P\ x \wedge (\forall\, y{\in}I.\ y < x \longrightarrow \neg\ P\ y)$
⟨*proof*⟩

thm
Orderings.wellorder-Least-lemma
Orderings.Least-equality
Orderings.LeastI2-order
Least-def

thm *Least-def*
lemma *iMin-the*: $iMin\ I = (THE\ x.\ x \in I \wedge (\forall\, y.\ y \in I \longrightarrow x \leq y))$
⟨*proof*⟩
lemma *iMin-the2*: $iMin\ I = (THE\ x.\ x \in I \wedge (\forall\, y{\in}I.\ x \leq y))$
⟨*proof*⟩

thm *Least-equality*
lemma *iMin-equality*:
$[\![\ k \in I;\ \bigwedge x.\ x \in I \Longrightarrow k \leq x\ ]\!] \Longrightarrow iMin\ I = k$
⟨*proof*⟩
thm *Least-equality*

⟨*proof*⟩

lemma *iMin-mono-on*:
⟦ *mono-on f I*; ∃ *x*. *x* ∈ *I* ⟧ ⟹ *iMin* (*f* ' *I*) = *f* (*iMin I*)
⟨*proof*⟩
thm *iMinI-ex*
⟨*proof*⟩
lemma *iMin-mono-on2*:
⟦ *mono-on f I*; *I* ≠ {} ⟧ ⟹ *iMin* (*f* ' *I*) = *f* (*iMin I*)
⟨*proof*⟩

thm *LeastI2-order*
lemma *iMinI2-order*:
⟦ *k* ∈ *I*; ⋀*y*. *y* ∈ *I* ⟹ *k* ≤ *y*;
⋀*x*. ⟦ *x* ∈ *I*; ∀ *y*∈*I*. *x* ≤ *y* ⟧ ⟹ *P x* ⟧ ⟹
P (*iMin I*)
thm *LeastI2-order*
thm *LeastI2-order*[*of* λ*x*. *x* ∈ *i k P*]
⟨*proof*⟩

thm
iMinI2-order
thm
iMinI2
iMinI2-ex
iMinI2-ex2

thm *wellorder-Least-lemma*
lemma *wellorder-iMin-lemma*:
k ∈ *I* ⟹ *iMin I* ∈ *I* ∧ *iMin I* ≤ *k*
thm
iMinI
iMin-le
⟨*proof*⟩

thm
iMin-the *iMin-the2*
thm
iMin-mono *iMin-mono2*
thm
iMin-le

not-less-iMin
thm
iMinI
iMinI-ex *iMinI-ex2*
thm
iMinI2
iMinI2-ex *iMinI2-ex2*
thm
wellorder-iMin-lemma
thm
iMin-equality
thm
iMinI2-order

thm *iMinI*

lemma *iMin-Min-conv*:
⟦ *finite I*; $I \neq \{\}$ ⟧ ⟹ *iMin I* = *Min I*
⟨*proof*⟩

thm *Min-ge-iff*[*THEN iffD2*]
⟨*proof*⟩
thm *Min-le-iff*[*THEN iffD2*]
⟨*proof*⟩

lemma *Min-neq-imp-greater*[*dest*]: ⟦ *finite I*; $k \in I$; $k \neq$ *Min I* ⟧ ⟹ *Min I* $< k$
⟨*proof*⟩
lemma *Max-neq-imp-less*[*dest*]: ⟦ *finite I*; $k \in I$; $k \neq$ *Max I* ⟧ ⟹ $k <$ *Max I*
⟨*proof*⟩

lemma *nat-Least-mono*:
⟦ $A \neq \{\}$; *mono* (f::(*nat*⇒*nat*)) ⟧ ⟹
(*LEAST x*. $x \in f$ ‘ A) = f (*LEAST x*. $x \in A$)
⟨*proof*⟩

lemma *Least-disj*:
⟦ ∃ *x*. *P x*; ∃ *x*. *Q x* ⟧ ⟹
(*LEAST* (*x*::'*a*::*wellorder*). (*P x* ∨ *Q x*)) = *min* (*LEAST x*. *P x*) (*LEAST x*. *Q x*)
⟨*proof*⟩

lemma *Least-imp-le*:
⟦ ∃ *x*. *P x*; ⋀*x*. *P x* ⟹ *Q x* ⟧ ⟹
(*LEAST* (*x*::'*a*::*wellorder*). *Q x*) ≤ (*LEAST x*. *P x*)
thm *Least-le LeastI2-ex*
⟨*proof*⟩

lemma *Least-imp-disj-eq*:
$[\![\exists x.\ P\ x;\ \bigwedge x.\ P\ x \Longrightarrow Q\ x\ ]\!] \Longrightarrow$
$(LEAST\ (x::'a::wellorder).\ P\ x \vee Q\ x) = (LEAST\ x.\ Q\ x)$
⟨*proof*⟩

lemma *Least-le-imp-le*:
$[\![\exists x.\ P\ x;\ \exists x.\ Q\ x;\ \bigwedge x\ y.\ [\![P\ x;\ Q\ y\ ]\!] \Longrightarrow x \leq y\ ]\!] \Longrightarrow$
$(LEAST\ (x::'a::wellorder).\ P\ x) \leq (LEAST\ (x::'a::wellorder).\ Q\ x)$
⟨*proof*⟩
lemma *Least-le-imp-le-disj*:
$[\![\exists x.\ P\ x;\ \bigwedge x\ y.\ [\![P\ x;\ Q\ y\ ]\!] \Longrightarrow x \leq y\ ]\!] \Longrightarrow$
$(LEAST\ (x::'a::wellorder).\ P\ x \vee Q\ x) = (LEAST\ (x::'a::wellorder).\ P\ x)$
thm *Least-imp-disj-eq*
⟨*proof*⟩

thm *Max-le-iff*
thm *Max-less-iff*

thm *iMin-equality*
lemma *Max-equality*: $[\![(k::'a::linorder) \in A;\ finite\ A;\ \bigwedge x.\ x \in A \Longrightarrow x \leq k\ ]\!] \Longrightarrow$

$Max\ A = k$
⟨*proof*⟩

thm
iMin-le
Max-ge

thm *not-less-iMin*
lemma *not-greater-Max*: $[\![finite\ A;\ Max\ A < k\ ]\!] \Longrightarrow k \notin A$
⟨*proof*⟩
thm *Max-ge*[*of I k*]
⟨*proof*⟩
thm *order-le-less-trans*[*of k Max A k*]
⟨*proof*⟩

lemma *Collect-not-greater-Max*: $[\![finite\ \{x.\ P\ x\};\ Max\ \{x.\ P\ x\} < k\ ]\!] \Longrightarrow \neg\ P\ k$
⟨*proof*⟩
lemma *Collect-Max-ge*: $[\![finite\ \{x.\ P\ x\};\ P\ k\ ]\!] \Longrightarrow k \leq Max\ \{x.\ P\ x\}$
⟨*proof*⟩

thm
iMinI-ex2
Max-in

thm *iMinI*
lemma *MaxI*: $[\![\ k \in A;\ \mathit{finite}\ A\ ]\!] \Longrightarrow \mathit{Max}\ A \in A$
⟨*proof*⟩

thm *iMinI-ex*
lemma *MaxI-ex*: $[\![\ \exists x.\ x \in A;\ \mathit{finite}\ A\ ]\!] \Longrightarrow \mathit{Max}\ A \in A$
⟨*proof*⟩
thm *iMinI-ex2*
lemma *MaxI-ex2*: $[\![\ A \neq \{\};\ \mathit{finite}\ A\ ]\!] \Longrightarrow \mathit{Max}\ A \in A$
⟨*proof*⟩

thm *iMinI2*
lemma *MaxI2*: $[\![\ k \in A;\ \bigwedge x.\ x \in A \Longrightarrow P\ x;\ \mathit{finite}\ A\ ]\!] \Longrightarrow P\ (\mathit{Max}\ A)$
thm *Max-in*
⟨*proof*⟩
thm *iMinI2-ex*
lemma *MaxI2-ex*:$[\![\ \exists x.\ x \in A;\ \bigwedge x.\ x \in A \Longrightarrow P\ x;\ \mathit{finite}\ A\ ]\!] \Longrightarrow P\ (\mathit{Max}\ A)$
⟨*proof*⟩
thm *iMinI2-ex2*
lemma *MaxI2-ex2*:$[\![\ A \neq \{\};\ \bigwedge x.\ x \in A \Longrightarrow P\ x;\ \mathit{finite}\ A\ ]\!] \Longrightarrow P\ (\mathit{Max}\ A)$
⟨*proof*⟩

thm *iMin-mono*
lemma *Max-mono*: $[\![\ \mathit{mono}\ f;\ \exists x.\ x \in A;\ \mathit{finite}\ A\ ]\!] \Longrightarrow \mathit{Max}\ (f\ `\ A) = f\ (\mathit{Max}\ A)$
⟨*proof*⟩
thm *Max-equality*
⟨*proof*⟩

thm *iMin-mono2*
lemma *Max-mono2*:$[\![\ \mathit{mono}\ f;\ A \neq \{\};\ \mathit{finite}\ A\ ]\!] \Longrightarrow \mathit{Max}\ (f\ `\ A) = f\ (\mathit{Max}\ A)$
⟨*proof*⟩

thm *iMin-mono-on*
lemma *Max-mono-on*: $[\![\ \mathit{mono\text{-}on}\ f\ A;\ \exists x.\ x \in A;\ \mathit{finite}\ A\ ]\!] \Longrightarrow \mathit{Max}\ (f\ `\ A) = f$
$(\mathit{Max}\ A)$
⟨*proof*⟩

lemma *Max-mono-on2*:
$[\![\ \mathit{mono\text{-}on}\ f\ A;\ A \neq \{\};\ \mathit{finite}\ A\ ]\!] \Longrightarrow \mathit{Max}\ (f\ `\ A) = f\ (\mathit{Max}\ A)$
⟨*proof*⟩

thm *iMin-the*
lemma *Max-the*:
⟦ $A \neq \{\}$; *finite* A ⟧ ⟹
$Max\ A = (THE\ x.\ x \in A \wedge (\forall y.\ y \in A \longrightarrow y \leq x))$
thm *iffD1*[*OF eq-commute*]
⟨*proof*⟩
thm *the-equality*
⟨*proof*⟩
thm *Max-equality*
⟨*proof*⟩

thm *iMin-the2*
lemma *Max-the2*: ⟦ $A \neq \{\}$; *finite* A ⟧ ⟹
$Max\ A = (THE\ x.\ x \in A \wedge (\forall y{\in}A.\ y \leq x))$
⟨*proof*⟩

thm *wellorder-iMin-lemma*
lemma *wellorder-Max-lemma*: ⟦ $k \in A$; *finite* A ⟧ ⟹ $Max\ A \in A \wedge k \leq Max\ A$
⟨*proof*⟩

thm *iMinI2-order*
lemma *MaxI2-order*: ⟦ $k \in A$; *finite* A; $\bigwedge y.\ y \in A \Longrightarrow y \leq k$;
$\bigwedge x.$ ⟦ $x \in A$; $\forall y{\in}A.\ y \leq x$ ⟧ ⟹ $P\ x$ ⟧
⟹ $P\ (Max\ A)$
thm *Max-equality*
⟨*proof*⟩

thm
iMin-equality
Max-equality
thm
iMin-the iMin-the2
Max-the Max-the2
thm
iMin-mono iMin-mono2
Max-mono Max-mono2
thm
iMin-le
Max-ge
thm
not-less-iMin
not-greater-Max
thm
iMinI
MaxI
thm
iMinI-ex iMinI-ex2
MaxI-ex MaxI-ex2

thm
iMinI2
MaxI2
thm
iMinI2-ex *iMinI2-ex2*
MaxI2-ex *MaxI2-ex2*
thm
wellorder-iMin-lemma
wellorder-Max-lemma
thm
iMinI2-order
MaxI2-order

lemma *Min-le-Max*: $[\![\mathit{finite}\ A;\ A \neq \{\}\ ]\!] \Longrightarrow \mathit{Min}\ A \leq \mathit{Max}\ A$
⟨*proof*⟩
lemma *iMin-le-Max*: $[\![\mathit{finite}\ A;\ A \neq \{\}\ ]\!] \Longrightarrow \mathit{iMin}\ A \leq \mathit{Max}\ A$
thm *subst*[*OF iMin-Min-conv*]
⟨*proof*⟩

6.2.2 *Max* for sets over *inat*

definition
$\mathit{iMax} :: \mathit{nat\ set} \Rightarrow \mathit{inat}$
where
$\mathit{iMax}\ i \equiv \mathit{if}\ (\mathit{finite}\ i)\ \mathit{then}\ (\mathit{Fin}\ (\mathit{Max}\ i))\ \mathit{else}\ \infty$

lemma *iMax-finite-conv*: $\mathit{finite}\ I = (\mathit{iMax}\ I \neq \infty)$
⟨*proof*⟩
lemma *iMax-infinite-conv*: $\mathit{infinite}\ I = (\mathit{iMax}\ I = \infty)$
⟨*proof*⟩

thm *lattice.inf-sup-aci*
thm *lattice-class.inf-sup-aci*
thm *semilattice-inf-class.inf-aci*
thm *semilattice-sup-class.sup-aci*

thm *lattice-class-def*
thm *lattice-class.axioms*

thm *distrib-lattice-class-def*

print-locale *distrib-lattice*
print-locale *lattice*
print-locale *distrib-lattice*

print-locale! *distrib-lattice*

thm *distrib-lattice-class.axioms*
interpretation *min-max2*:
  distrib-lattice op $\leq$ :: *'a*::*linorder* $\Rightarrow$ *'a* $\Rightarrow$ *bool op* $<$ *min max*
⟨*proof*⟩
print-theorems
term *distrib-lattice-class*
lemma *class.distrib-lattice* (*op* $\leq$) (*op* $<$) (*min*::(*'a*::*linorder* $\Rightarrow$ *'a* $\Rightarrow$ *'a*)) *max*
print-locale *distrib-lattice*
thm *distrib-lattice-class.intro*
⟨*proof*⟩
  thm *class.semilattice-inf.intro*
  ⟨*proof*⟩

lemma *class.distrib-lattice* (*op* $\geq$) (*op* $>$) (*max*::(*'a*::*linorder* $\Rightarrow$ *'a* $\Rightarrow$ *'a*)) *min*
thm *linorder-class.min-max.dual-distrib-lattice*
⟨*proof*⟩

print-locale *Lattices.distrib-lattice*

thm *Big-Operators.distrib-lattice.sup-Inf1-distrib*
lemma *max-Min-eq-Min-max*[*rule-format*]:
  finite A $\Longrightarrow$
  $A \neq \{\}$ $\longrightarrow$
  max x (*Min A*) = *Min* $\{$*max x a* |*a*. $a \in A\}$
thm *finite.induct*[*of A*]
⟨*proof*⟩
thm *max-le-monoR*[*OF less-imp-le*]
⟨*proof*⟩
thm *max-Min-eq-Min-max*

lemma *max-iMin-eq-iMin-max*:
  $[\![$ *finite A*; $A \neq \{\}$ $]\!] \Longrightarrow$
  max x (*iMin A*) = *iMin* $\{$*max x a* |*a*. $a \in A\}$
thm *iMin-Min-conv*
⟨*proof*⟩
thm *iMin-Min-conv*[*of* $\{$*max x a* |*a*. $a \in A\}$]
⟨*proof*⟩
thm *max-Min-eq-Min-max*
⟨*proof*⟩

lemma $[\![$ *finite A*; $A \neq \{\}$ $]\!] \Longrightarrow \forall x \in A.\ x \leq Max\ A$
⟨*proof*⟩

6.2.3 *Min* and *Max* for set operations

lemma *iMin-subset*: $[\![$ $A \neq \{\}$; $A \subseteq B$ $]\!] \Longrightarrow iMin\ B \leq iMin\ A$
thm *iMin-le*[*of iMin A j*]

thm *iMinI-ex2*[*of A*]
⟨*proof*⟩

lemma *Max-subset*: ⟦ $A \neq \{\}$; $A \subseteq B$; *finite* B ⟧ ⟹ $Max\ A \leq Max\ B$
⟨*proof*⟩
lemma *Min-subset*: ⟦ $A \neq \{\}$; $A \subseteq B$; *finite* B ⟧ ⟹ $Min\ B \leq Min\ A$
⟨*proof*⟩

lemma *iMin-Int-ge1*: $(A \cap B) \neq \{\} \Longrightarrow iMin\ A \leq iMin\ (A \cap B)$
thm *iMin-subset*[*OF - Int-lower1*]
⟨*proof*⟩
lemma *iMin-Int-ge2*: $(A \cap B) \neq \{\} \Longrightarrow iMin\ B \leq iMin\ (A \cap B)$
⟨*proof*⟩
lemma *iMin-Int-ge*: $(A \cap B) \neq \{\} \Longrightarrow max\ (iMin\ A)\ (iMin\ B) \leq iMin\ (A \cap B)$
⟨*proof*⟩

lemma *Max-Int-le1*: ⟦ $(A \cap B) \neq \{\}$; *finite* A ⟧ ⟹ $Max\ (A \cap B) \leq Max\ A$
thm *Max-subset*[*OF - Int-lower1*]
⟨*proof*⟩
lemma *Max-Int-le2*: ⟦ $(A \cap B) \neq \{\}$; *finite* B ⟧ ⟹ $Max\ (A \cap B) \leq Max\ B$
⟨*proof*⟩
lemma *Max-Int-le*: ⟦ $(A \cap B) \neq \{\}$; *finite* A; *finite* B ⟧ ⟹
$Max\ (A \cap B) \leq min\ (Max\ A)\ (Max\ B)$
⟨*proof*⟩

lemma *iMin-Un*[*rule-format*]:
⟦ $A \neq \{\}$; $B \neq \{\}$ ⟧ ⟹
$iMin\ (A \cup B) = min\ (iMin\ A)\ (iMin\ B)$
⟨*proof*⟩
thm *min-le-iff-disj*
⟨*proof*⟩
thm *iMinI-ex2*[*of A∪B*]
⟨*proof*⟩
thm *Min-Un*
thm *Max-Un*

thm *linorder-class.Min-singleton linorder-class.Max-singleton*
thm *singletonI*[*THEN iMinI*, *THEN singletonD*]
lemma *iMin-singleton*[*simp*]: $iMin\ \{a\} = a$
⟨*proof*⟩
lemma *iMax-singleton*[*simp*]: $iMax\ \{a\} = Fin\ a$
⟨*proof*⟩

lemma *Max-le-Min-imp-singleton*:
⟦ *finite A*; *A* ≠ {}; *Max A* ≤ *Min A* ⟧ ⟹ *A* = {*Min A*}
thm *Max-ge*
thm *Max-le-iff* [*THEN iffD1*]
⟨*proof*⟩
lemma *Max-le-Min-conv-singleton*:
⟦ *finite A*; *A* ≠ {} ⟧ ⟹ (*Max A* ≤ *Min A*) = (∃ *x*. *A* = {*x*})
⟨*proof*⟩

lemma *Max-le-iMin-imp-le*:
⟦ *finite A*; *Max A* ≤ *iMin B*; *a* ∈ *A*; *b* ∈ *B* ⟧ ⟹ *a* ≤ *b*
⟨*proof*⟩

lemma *le-imp-Max-le-iMin*:
⟦ *finite A*; *A* ≠ {}; *B* ≠ {}; ∀ *a*∈*A*. ∀ *b*∈*B*. *a* ≤ *b* ⟧ ⟹ *Max A* ≤ *iMin B*
⟨*proof*⟩

lemma *Max-le-iMin-conv-le*:
⟦ *finite A*; *A* ≠ {}; *B* ≠ {} ⟧ ⟹ (*Max A* ≤ *iMin B*) = (∀ *a*∈*A*. ∀ *b*∈*B*. *a* ≤ *b*)
⟨*proof*⟩

lemma *Max-less-iMin-imp-less*:
⟦ *finite A*; *Max A* < *iMin B*; *a* ∈ *A*; *b* ∈ *B* ⟧ ⟹ *a* < *b*
thm *Max-less-iff* [*THEN iffD1*, *of* - *iMin B*]
⟨*proof*⟩

lemma *less-imp-Max-less-iMin*:
⟦ *finite A*; *A* ≠ {}; *B* ≠ {}; ∀ *a*∈*A*. ∀ *b*∈*B*. *a* < *b* ⟧ ⟹ *Max A* < *iMin B*
⟨*proof*⟩

lemma *Max-less-iMin-conv-less*:
⟦ *finite A*; *A* ≠ {}; *B* ≠ {} ⟧ ⟹ (*Max A* < *iMin B*) = (∀ *a*∈*A*. ∀ *b*∈*B*. *a* < *b*)
⟨*proof*⟩

lemma *Max-less-iMin-imp-disjoint*:
⟦ *finite A*; *Max A* < *iMin B* ⟧ ⟹ *A* ∩ *B* = {}
⟨*proof*⟩

thm *Min.in-idem*
lemma *iMin-in-idem*: *n* ∈ *I* ⟹ *min n* (*iMin I*) = *iMin I*
⟨*proof*⟩

thm *Min-insert*
lemma *iMin-insert*: *I* ≠ {} ⟹ *iMin* (*insert n I*) = *min n* (*iMin I*)
⟨*proof*⟩

thm *Min.insert-remove*
lemma *iMin-insert-remove*:
iMin (*insert n I*) =
(*if I* − {*n*} = {} *then n else min n* (*iMin* (*I* − {*n*})))
⟨*proof*⟩

thm *Min.remove*
lemma *iMin-remove*: *n* ∈ *I* ⟹ *iMin I* = (*if I* − {*n*} = {} *then n else min n* (*iMin* (*I* − {*n*})))
⟨*proof*⟩

thm *Min.subset-idem*
lemma *iMin-subset-idem*: ⟦ *B* ≠ {}; *B* ⊆ *A* ⟧ ⟹ *min* (*iMin B*) (*iMin A*) = *iMin A*
⟨*proof*⟩

thm *Min.union-inter*
lemma *iMin-union-inter*: *A* ∩ *B* ≠ {} ⟹ *min* (*iMin* (*A* ∪ *B*)) (*iMin* (*A* ∩ *B*)) = *min* (*iMin A*) (*iMin B*)
⟨*proof*⟩

thm *Min-ge-iff*
lemma *iMin-ge-iff*: *I* ≠ {} ⟹ (*n* ≤ *iMin I*) = (∀ *a*∈*I*. *n* ≤ *a*)
⟨*proof*⟩

thm *Min-gr-iff*
lemma *iMin-gr-iff*: *I* ≠ {} ⟹ (*n* < *iMin I*) = (∀ *a*∈*I*. *n* < *a*)
⟨*proof*⟩

thm *Min-le-iff*
lemma *iMin-le-iff*: *I* ≠ {} ⟹ (*iMin I* ≤ *n*) = (∃ *a*∈*I*. *a* ≤ *n*)
⟨*proof*⟩

thm *Min-less-iff*
lemma *iMin-less-iff*: *I* ≠ {} ⟹ (*iMin I* < *n*) = (∃ *a*∈*I*. *a* < *n*)
⟨*proof*⟩

thm *hom-Min-commute*
lemma *hom-iMin-commute*: ⟦ ⋀*x y*. *h* (*min x y*) = *min* (*h x*) (*h y*); *I* ≠ {} ⟧ ⟹ *iMin* (*h* ‘ *I*) = *h* (*iMin I*)
⟨*proof*⟩
thm *min-eqL*[*OF iMin-le*]
⟨*proof*⟩

Synonyms for similarity with theorem names for *Min*"

thm *Min-eqI iMin-equality*
lemmas *iMin-eqI* = *iMin-equality*
thm *Min-in iMinI-ex2*
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* similar to *transP* from HOL/Predicate.

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

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

abbreviation *equivP* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$ **where**
$equivP\ r \equiv reflP\ r \wedge symP\ r \wedge transP\ r$

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

Example for *reflP*

lemma *reflP* $((op \leq)::('a::preorder \Rightarrow 'a \Rightarrow bool))$
⟨*proof*⟩

Example for *symP*

lemma *symP* $(op =)$
⟨*proof*⟩

Example for *equivP*

lemma *equivP* $(op =)$
⟨*proof*⟩

Example for *irreflP*

lemma *irreflP* $((op <)::('a::preorder \Rightarrow 'a \Rightarrow bool))$
⟨*proof*⟩

6.3.2 Auxiliary results for *singletons*

lemma *singleton-not-empty*: $\{a\} \neq \{\}$ ⟨*proof*⟩
lemma *singleton-finite*: *finite* $\{a\}$ ⟨*proof*⟩

lemma *singleton-ball*: $(\forall x{\in}\{a\}.\ P\ x) = P\ a$ ⟨*proof*⟩
lemma *singleton-bex*: $(\exists x{\in}\{a\}.\ P\ x) = P\ a$ ⟨*proof*⟩

thm *Set.subset-singletonD*
lemma *subset-singleton-conv*: $(A \subseteq \{a\}) = (A = \{\} \vee A = \{a\})$ ⟨*proof*⟩
lemma *singleton-subset-conv*: $(\{a\} \subseteq A) = (a \in A)$ ⟨*proof*⟩
thm *Set.singleton-inject*
lemma *singleton-eq-conv*: $(\{a\} = \{b\}) = (a = b)$ ⟨*proof*⟩
lemma *singleton-subset-singleton-conv*: $(\{a\} \subseteq \{b\}) = (a = b)$ ⟨*proof*⟩

lemma *singleton-Int1-if*: $\{a\} \cap A = (\mathit{if}\ a \in A\ \mathit{then}\ \{a\}\ \mathit{else}\ \{\})$
⟨*proof*⟩
lemma *singleton-Int2-if*: $A \cap \{a\} = (\mathit{if}\ a \in A\ \mathit{then}\ \{a\}\ \mathit{else}\ \{\})$
⟨*proof*⟩

lemma *singleton-image*: $f\ `\ \{a\} = \{f\ a\}$ ⟨*proof*⟩
lemma *inj-on-singleton*: *inj-on* $f\ \{a\}$ ⟨*proof*⟩
lemma *strict-mono-on-singleton*: *strict-mono-on* $f\ \{a\}$
⟨*proof*⟩

Auxiliary results for *empty* sets

thm *empty-imp-not-in*
thm *ex-imp-not-empty*
thm *in-imp-not-empty*

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

thm *infinite-imp-nonempty*
lemma *infinite-imp-not-singleton*: *infinite* $A \Longrightarrow \neg\ (\exists\, a.\ A = \{a\})$ ⟨*proof*⟩

lemma *infinite-insert*: *infinite* $(\mathit{insert}\ a\ A) = \mathit{infinite}\ A$
⟨*proof*⟩
lemma *infinite-Diff-insert*: *infinite* $(A - \mathit{insert}\ a\ B) = \mathit{infinite}\ (A - B)$
⟨*proof*⟩

lemma *subset-finite-imp-finite*: $[\![\ \mathit{finite}\ A;\ B \subseteq A\ ]\!] \Longrightarrow \mathit{finite}\ B$
⟨*proof*⟩
lemma *infinite-not-subset-finite*: $[\![\ \mathit{infinite}\ A;\ \mathit{finite}\ B\ ]\!] \Longrightarrow \neg\ A \subseteq B$
⟨*proof*⟩

thm *Un-infinite*
lemma *Un-infinite-right*: *infinite* $T \Longrightarrow \mathit{infinite}\ (S \cup T)$ ⟨*proof*⟩
lemma *Un-infinite-iff*: *infinite* $(S \cup T) = (\mathit{infinite}\ S \vee \mathit{infinite}\ T)$ ⟨*proof*⟩

thm *transfer-nat-int-set-relations*

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

corollary *finite-A-int-A-conv*: *finite* $A = \mathit{finite}\ (\mathit{int}\ `\ A)$
⟨*proof*⟩

Corresponding fact fo infinite sets

corollary *infinite-A-int-A-conv*: *infinite* $A = \mathit{infinite}\ (\mathit{int}\ `\ A)$
⟨*proof*⟩

lemma *cartesian-product-infiniteL-imp-infinite*: $[\![\ \mathit{infinite}\ A;\ B \neq \{\}\ ]\!] \Longrightarrow \mathit{infinite}\ (A \times B)$
⟨*proof*⟩

lemma *cartesian-product-infiniteR-imp-infinite*: $[\![\mathit{infinite}\ B;\ A \neq \{\}]\!] \Longrightarrow \mathit{infinite}\ (A \times B)$
⟨*proof*⟩

thm *finite-nat-iff-bounded*
lemma *finite-nat-iff-bounded2*:
$\mathit{finite}\ S = (\exists\,(k{::}\mathit{nat}).\forall\, n{\in}S.\ n < k)$
⟨*proof*⟩
thm *finite-nat-iff-bounded-le*
lemma *finite-nat-iff-bounded-le2*:
$\mathit{finite}\ S = (\exists\,(k{::}\mathit{nat}).\forall\, n{\in}S.\ n \leq k)$
⟨*proof*⟩

lemma *nat-asc-chain-imp-unbounded*:
$[\![S \neq \{\};\ (\forall\, m{\in}S.\ \exists\, n{\in}S.\ m < (n{::}\mathit{nat}))]\!] \Longrightarrow \forall\, m.\ \exists\, n{\in}S.\ m \leq n$
⟨*proof*⟩

thm *infinite-nat-iff-unbounded-le*
lemma *infinite-nat-iff-asc-chain*:
$S \neq \{\} \Longrightarrow \mathit{infinite}\ S = (\forall\, m{\in}S.\ \exists\, n{\in}S.\ m < (n{::}\mathit{nat}))$
⟨*proof*⟩
lemma *infinite-imp-asc-chain*: $\mathit{infinite}\ S \Longrightarrow \forall\, m{\in}S.\ \exists\, n{\in}S.\ m < (n{::}\mathit{nat})$
thm *infinite-nat-iff-asc-chain*[*THEN iffD1*, *OF infinite-imp-nonempty*]
⟨*proof*⟩

lemma *infinite-image-imp-infinite*: $\mathit{infinite}\ (f\ `\ A) \Longrightarrow \mathit{infinite}\ A$
⟨*proof*⟩

lemma *inj-on-imp-infinite-image*: $[\![\mathit{infinite}\ A;\ \mathit{inj\text{-}on}\ f\ A]\!] \Longrightarrow \mathit{infinite}\ (f\ `\ A)$
⟨*proof*⟩

lemma *inj-on-infinite-image-iff*: $\mathit{inj\text{-}on}\ f\ A \Longrightarrow \mathit{infinite}\ (f\ `\ A) = \mathit{infinite}\ A$
⟨*proof*⟩

lemma *inj-on-finite-image-iff*: $\mathit{inj\text{-}on}\ f\ A \Longrightarrow \mathit{finite}\ (f\ `\ A) = \mathit{finite}\ A$
⟨*proof*⟩

lemma *nat-ex-greater-finite-Max-conv*:
$A \neq \{\} \Longrightarrow (\exists\, x{\in}A.\ (n{::}\mathit{nat}) < x) = (\mathit{finite}\ A \longrightarrow n < \mathit{Max}\ A)$
⟨*proof*⟩
thm *infinite-nat-iff-unbounded*[*THEN iffD1*, *rule-format*]
⟨*proof*⟩
corollary *nat-ex-greater-infinite-finite-Max-conv'*:
$(\exists\, x{\in}A.\ (n{::}\mathit{nat}) < x) = (\mathit{finite}\ A \wedge A \neq \{\} \wedge n < \mathit{Max}\ A \vee \mathit{infinite}\ A)$
⟨*proof*⟩

6.3.4 Some auxiliary results for disjoint sets

thm *Set.disjoint-iff-not-equal*
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*:
$[\![A \cap B = \{\};\ x \in A \cup B]\!] \Longrightarrow x \notin A \vee x \notin B$
thm *disjoint-iff-in-not-in1*[*THEN iffD1*, *rule-format*]
⟨*proof*⟩

lemma *partition-Union*: $A \subseteq \bigcup C \Longrightarrow (\bigcup c{\in}C.\ A \cap c) = A$ ⟨*proof*⟩

lemma *disjoint-partition-Int*:
$\forall c1{\in}C.\ \forall c2{\in}C.\ c1 \neq c2 \longrightarrow c1 \cap c2 = \{\} \Longrightarrow$
$\forall a1{\in}\{A \cap c|\ c.\ c \in C\}.\ \forall a2{\in}\{A \cap c|\ c.\ c \in C\}.$
$a1 \neq a2 \longrightarrow a1 \cap a2 = \{\}$
⟨*proof*⟩
thm
image-Union
image-eq-UN
image-def
image-Collect
lemma $\{f\ x\ |x.\ x \in A\} = (\bigcup x{\in}A.\ \{f\ x\})$
⟨*proof*⟩

thm *Finite-Set.card-partition*

This lemma version drops the superfluous precondition *finite* $(\bigcup C)$ (and turns the resulting equation to the sensible order *card* .. $= k * card$ ..).

lemma *card-partition*:
$[\![$ *finite* C; $\bigwedge c.\ c \in C \Longrightarrow card\ c = k$; $\bigwedge c1\ c2.\ [\![c1 \in C;\ c2 \in C;\ c1 \neq c2]\!] \Longrightarrow c1 \cap c2 = \{\}\ ]\!] \Longrightarrow$
card $(\bigcup C) = k * card\ C$
⟨*proof*⟩

6.3.5 Some auxiliary results for subset relation

thm *bex-subset-imp-bex*
thm *bex-imp-ex*
thm
ball-subset-imp-ball
ball-subset-imp-ball[*rule-format*]
thm
all-imp-ball
all-imp-ball[*rule-format*]

thm *image-mono*

lemma *subset-image-Int*: $A \subseteq B \Longrightarrow f\ `\ (A \cap B) = f\ `\ A \cap f\ `\ B$
⟨*proof*⟩

lemma *image-diff-disjoint-image-Int*:
$[\![\ f\ `\ (A - B) \cap f\ `\ B = \{\}\ ]\!] \Longrightarrow$
$f\ `\ (A \cap B) = f\ `\ A \cap f\ `\ B$
⟨*proof*⟩

lemma *subset-imp-Int-subset1*: $A \subseteq C \Longrightarrow A \cap B \subseteq C$
thm *subset-trans*[*of* - $C \cap B$]
thm *subset-trans*[*of* - $C \cap B$, *OF Int-mono*]
thm *subset-trans*[*of* - $C \cap B$, *OF Int-mono*, *OF* - *subset-refl Int-lower1*]
⟨*proof*⟩
lemma *subset-imp-Int-subset2*: $B \subseteq C \Longrightarrow A \cap B \subseteq C$
⟨*proof*⟩

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
thm *set-interval-defs*

thm *image-add-atLeastAtMost*
lemma *image-add-atLeast*:
$(\lambda n{::}nat.\ n{+}k)\ `\ \{i..\} = \{i{+}k..\}$ (**is** *?A* = *?B*)
⟨*proof*⟩

lemma *image-add-atMost*:
$(\lambda n{::}nat.\ n{+}k)\ `\ \{..i\} = \{k..i{+}k\}$ (**is** *?A* = *?B*)
⟨*proof*⟩

thm *image-add-atLeastAtMost*
thm *image-add-atLeast*
thm *image-add-atMost*

thm *image-Suc-atLeastAtMost*

thm *image-add-atLeast*
corollary *image-Suc-atLeast*: *Suc* ` $\{i..\} = \{Suc\ i..\}$
⟨*proof*⟩

thm *image-add-atMost*
corollary *image-Suc-atMost*: *Suc* ` $\{..i\} = \{Suc\ 0..Suc\ i\}$
thm *image-add-atMost*[*of Suc 0*]
⟨*proof*⟩

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*: $(k::'a::order) \leq k' \Longrightarrow \{l..k\} \subseteq \{..k'\}$
⟨*proof*⟩

thm *image-add-lemmas*
thm *image-Suc-lemmas*
thm *atMost-atLeastAtMost-0-conv*

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*⟩

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

thm *atLeastAtMost-def atLeastLessThan-def*
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* $A \Longrightarrow A \subseteq \{Min\ A..Max\ A\}$

⟨*proof*⟩

lemma *Max-le-imp-subset-atMost*:
⟦ *finite A*; *Max A* ≤ *n* ⟧ ⟹ *A* ⊆ {..*n*}
thm *subset-trans*[*OF finite-subset-atLeastAtMost atLeastAtMost-subset-atMost*]
⟨*proof*⟩

lemma *subset-atMost-imp-Max-le*:
⟦ *finite A*; *A* ≠ {}; *A* ⊆ {..*n*} ⟧ ⟹ *Max A* ≤ *n*
⟨*proof*⟩
lemma *subset-atMost-Max-le-conv*:
⟦ *finite A*; *A* ≠ {} ⟧ ⟹ (*A* ⊆ {..*n*}) = (*Max A* ≤ *n*)
⟨*proof*⟩

lemma *iMin-atLeast*: *iMin* {*n*..} = *n*
⟨*proof*⟩
lemma *iMin-greaterThan*: *iMin* {*n*<..} = *Suc n*
⟨*proof*⟩
lemma *iMin-atMost*: *iMin* {..(*n*::*nat*)} = *0*
⟨*proof*⟩
lemma *iMin-lessThan*: *0* < *n* ⟹ *iMin* {..<(*n*::*nat*)} = *0*
⟨*proof*⟩

lemma *Max-atMost*: *Max* {..(*n*::*nat*)} = *n*
⟨*proof*⟩
lemma *Max-lessThan*: *0* < *n* ⟹ *Max* {..<*n*} = *n* − *Suc 0*
⟨*proof*⟩
lemma *iMin-atLeastLessThan*: *m* < *n* ⟹ *iMin* {*m*..<*n*} = *m*
⟨*proof*⟩
lemma *Max-atLeastLessThan*: *m* < *n* ⟹ *Max* {*m*..<*n*} = *n* − *Suc 0*
⟨*proof*⟩
lemma *iMin-greaterThanLessThan*: *Suc m* < *n* ⟹ *iMin* {*m*<..<*n*} = *Suc m*
⟨*proof*⟩
lemma *Max-greaterThanLessThan*: *Suc m* < *n* ⟹ *Max* {*m*<..<*n*} = *n* − *Suc 0*
⟨*proof*⟩
lemma *iMin-greaterThanAtMost*: *m* < *n* ⟹ *iMin* {*m*<..*n*} = *Suc m*
⟨*proof*⟩
lemma *Max-greaterThanAtMost*: *m* < *n* ⟹ *Max* {*m*<..(*n*::*nat*)} = *n*
⟨*proof*⟩
lemma *iMin-atLeastAtMost*: *m* ≤ *n* ⟹ *iMin* {*m*..*n*} = *m*
⟨*proof*⟩
lemma *Max-atLeastAtMost*: *m* ≤ *n* ⟹ *Max* {*m*..(*n*::*nat*)} = *n*
⟨*proof*⟩

lemma *infinite-atLeast*: *infinite* {(*n*::*nat*)..}
⟨*proof*⟩
lemma *infinite-greaterThan*: *infinite* {(*n*::*nat*)<..}

⟨*proof*⟩

lemma *infinite-atLeast-int*: *infinite* {(*n*::*int*)..}
⟨*proof*⟩

lemma *infinite-greaterThan-int*: *infinite* {(*n*::*int*)<..}
thm *atLeast-remove*
⟨*proof*⟩

lemma *infinite-atMost-int*: *infinite* {..(*n*::*int*)}
⟨*proof*⟩

lemma *infinite-lessThan-int*: *infinite* {..<(*n*::*int*)}
thm *atMost-remove*
⟨*proof*⟩

6.3.7 Auxiliary results for *card*

lemma *setsum-singleton*: $(\sum x{\in}\{a\}.\ f\ x) = f\ a$
⟨*proof*⟩

lemma *card-singleton*: *card* $\{a\} = Suc\ 0$
⟨*proof*⟩

thm *card-cartesian-product-singleton*
lemma *card-cartesian-product-singleton-right*: *card* $(A \times \{x\}) = card\ A$
⟨*proof*⟩

lemma *card-1-imp-singleton*: *card* $A = Suc\ 0 \Longrightarrow (\exists\, a.\ A = \{a\})$
⟨*proof*⟩

lemma *card-1-singleton-conv*: $(card\ A = Suc\ 0) = (\exists\, a.\ A = \{a\})$
⟨*proof*⟩

lemma *card-gr0-imp-finite*: $0 < card\ A \Longrightarrow finite\ A$
⟨*proof*⟩
lemma *card-gr0-imp-not-empty*: $(0 < card\ A) \Longrightarrow A \neq \{\}$
⟨*proof*⟩
lemma *not-empty-card-gr0-conv*: $finite\ A \Longrightarrow (A \neq \{\}) = (0 < card\ A)$
⟨*proof*⟩

lemma *nat-card-le-Max*: *card* (*A*::*nat set*) ≤ *Suc* (*Max A*)
⟨*proof*⟩
thm *card-mono*[*OF finite-atMost*, *of A Max A*]
⟨*proof*⟩

lemma *Int-card1*: $finite\ A \Longrightarrow card\ (A \cap B) \leq card\ A$
⟨*proof*⟩
lemma *Int-card2*: $finite\ B \Longrightarrow card\ (A \cap B) \leq card\ B$

⟨*proof*⟩
lemma *Un-card1*: ⟦ *finite A*; *finite B* ⟧ ⟹ *card A* ≤ *card* (*A* ∪ *B*)
⟨*proof*⟩
lemma *Un-card2*: ⟦ *finite A*; *finite B* ⟧ ⟹ *card B* ≤ *card* (*A* ∪ *B*)
⟨*proof*⟩

thm *Finite-Set.card-Un-Int*
lemma *card-Un-conv*:
⟦ *finite A*; *finite B* ⟧ ⟹
card (*A* ∪ *B*) = *card A* + *card B* − *card* (*A* ∩ *B*)
⟨*proof*⟩
lemma *card-Int-conv*:
⟦ *finite A*; *finite B* ⟧ ⟹
card (*A* ∩ *B*) = *card A* + *card B* − *card* (*A* ∪ *B*)
⟨*proof*⟩

Pigeonhole principle, dirichlet's box principle

lemma *pigeonhole-principle*[*rule-format*]:
card (*f* ‘ *A*) < *card A* ⟶ (∃ *x*∈*A*. ∃ *y*∈*A*. *x* ≠ *y* ∧ *f x* = *f y*)
⟨*proof*⟩
thm *finite.induct*[*of A*]
⟨*proof*⟩

thm *pigeonhole-principle*
corollary *pigeonhole-principle-linorder*[*rule-format*]:
card (*f* ‘ *A*) < *card* (*A*::*'a*::*linorder set*) ⟹ (∃ *x*∈*A*. ∃ *y*∈*A*. *x* < *y* ∧ *f x* = *f y*)
⟨*proof*⟩

corollary *pigeonhole-mod*:
⟦ *0* < *m*; *m* < *card A* ⟧ ⟹ ∃ *x*∈*A*. ∃ *y*∈*A*. *x* < *y* ∧ *x mod m* = *y mod m*
⟨*proof*⟩
corollary *pigeonhole-mod2*:
⟦ (*0*::*nat*) < *m*; *m* ≤ *c*; *inj-on f* {..*c*} ⟧ ⟹ ∃ *x*≤*c*. ∃ *y*≤*c*. *x* < *y* ∧ *f x mod m*
= *f y mod m*
thm *pigeonhole-mod*[*of m f* ‘ {..*c*}]
⟨*proof*⟩

end

7 SetIntervalCut: 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 :: $('a\ set \Rightarrow 'a\ set) \Rightarrow bool$
where
$set\text{-}restriction\ f \equiv \exists P.\ \forall A.\ f\ A = \{x \in A.\ P\ x\}$

lemma *set-restrictionD*: $set\text{-}restriction\ f \Longrightarrow \exists P.\ \forall A.\ f\ A = \{x \in A.\ P\ x\}$
⟨*proof*⟩
lemma *set-restrictionD-spec*: $set\text{-}restriction\ f \Longrightarrow \exists P.\ f\ A = \{x \in A.\ P\ x\}$
⟨*proof*⟩
lemma *set-restrictionI*: $f = (\lambda A.\ \{x \in A.\ P\ x\}) \Longrightarrow set\text{-}restriction\ f$
⟨*proof*⟩

lemma *set-restriction-comp*:
$[\![\ set\text{-}restriction\ f;\ set\text{-}restriction\ g\ ]\!] \Longrightarrow set\text{-}restriction\ (f \circ g)$
⟨*proof*⟩
lemma *set-restriction-commute*:
$[\![\ set\text{-}restriction\ f;\ set\text{-}restriction\ g\ ]\!] \Longrightarrow f\ (g\ I) = g\ (f\ I)$
⟨*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
$set\text{-}restriction\text{-}fun\ P \equiv \lambda A.\ \{x \in A.\ P\ x\}$

lemma *set-restriction-fun-is-set-restriction*:
$set\text{-}restriction\ (set\text{-}restriction\text{-}fun\ P)$
⟨*proof*⟩

lemma *set-restriction-Int-conv*:
$set\text{-}restriction\ f = (\exists B.\ \forall A.\ f\ A = A \cap B)$
⟨*proof*⟩

lemma *set-restriction-Un*:
$set\text{-}restriction\ f \Longrightarrow f\ (A \cup B) = f\ A \cup f\ B$
⟨*proof*⟩
lemma *set-restriction-Int*:
$set\text{-}restriction\ f \Longrightarrow f\ (A \cap B) = f\ A \cap f\ B$
⟨*proof*⟩
lemma *set-restriction-Diff*:
$set\text{-}restriction\ f \Longrightarrow f\ (A - B) = f\ A - f\ B$
⟨*proof*⟩
lemma *set-restriction-mono*:
$[\![\ set\text{-}restriction\ f;\ A \subseteq B\ ]\!] \Longrightarrow f\ A \subseteq f\ B$

⟨*proof*⟩
lemma *set-restriction-absorb*:
set-restriction f ⟹ *f* (*f A*) = *f A*
⟨*proof*⟩
lemma *set-restriction-empty*:
set-restriction f ⟹ *f* {} = {}
⟨*proof*⟩
lemma *set-restriction-non-empty-imp*:
⟦ *set-restriction f*; *f A* ≠ {} ⟧ ⟹ *A* ≠ {}
⟨*proof*⟩
lemma *set-restriction-subset*:
set-restriction f ⟹ *f A* ⊆ *A*
⟨*proof*⟩
lemma *set-restriction-finite*:
⟦ *set-restriction f*; *finite A* ⟧ ⟹ *finite* (*f A*)
⟨*proof*⟩
lemma *set-restriction-card*:
⟦ *set-restriction f*; *finite A* ⟧ ⟹
card (*f A*) = *card A* − *card* {*a* ∈ *A*. *f* {*a*} = {}}
⟨*proof*⟩
thm *card-Diff-subset*[*symmetric*]
⟨*proof*⟩
thm *card-Diff-subset*[*symmetric*]
⟨*proof*⟩
lemma *set-restriction-card-le*:
⟦ *set-restriction f*; *finite A* ⟧ ⟹ *card* (*f A*) ≤ *card A*
⟨*proof*⟩

lemma *set-restriction-not-in-imp*:
⟦ *set-restriction f*; *x* ∉ *A* ⟧ ⟹ *x* ∉ *f A*
⟨*proof*⟩
lemma *set-restriction-in-imp*:
⟦ *set-restriction f*; *x* ∈ *f A* ⟧ ⟹ *x* ∈ *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*) = (∀ *x*∈*A*. *P x*)
⟨*proof*⟩
lemma *set-restriction-fun-empty-conv*:
((*set-restriction-fun P*) *A* = {}) = (∀ *x*∈*A*. ¬ *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* ⇒ *'a* ⇒ *'a set* (**infixl** ↓≤ *100*)
where
$I \downarrow\leq t \equiv \{ x{\in}I.\ x \leq t \}$
definition
cut-less :: *'a*::*linorder set* ⇒ *'a* ⇒ *'a set* (**infixl** ↓< *100*)
where
$I \downarrow< t \equiv \{ x{\in}I.\ x < t \}$

definition
cut-ge :: *'a*::*linorder set* ⇒ *'a* ⇒ *'a set* (**infixl** ↓≥ *100*)
where
$I \downarrow\geq t \equiv \{ x{\in}I.\ t \leq x \}$
definition
cut-greater :: *'a*::*linorder set* ⇒ *'a* ⇒ *'a set* (**infixl** ↓> *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
thm *i-cut-defs*

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
thm *i-cut-mem-iff*

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

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

lemma
cut-less-bound: $n \in I \downarrow< t \Longrightarrow n < t$ **and**
cut-le-bound: $n \in I \downarrow\leq t \Longrightarrow n \leq t$ **and**
cut-greater-bound: $n \in i \downarrow> t \Longrightarrow t < n$ **and**
cut-ge-bound: $n \in i \downarrow\geq t \Longrightarrow t \leq n$
⟨*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
thm *i-cut-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
thm *i-cut-Diff-conv*

7.2.2 Basic results for cut operators

lemma
cut-less-eq-set-restriction-fun': $(\lambda I.\ I \downarrow< t) = \textit{set-restriction-fun}\ (\lambda x.\ x < t)$
and
cut-le-eq-set-restriction-fun': $(\lambda I.\ I \downarrow\leq t) = \textit{set-restriction-fun}\ (\lambda x.\ x \leq t)$
and
cut-greater-eq-set-restriction-fun': $(\lambda I.\ I \downarrow> t) = \textit{set-restriction-fun}\ (\lambda x.\ x > t)$
and
cut-ge-eq-set-restriction-fun': $(\lambda I.\ I \downarrow\geq t) = \textit{set-restriction-fun}\ (\lambda x.\ x \geq t)$
⟨*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: $I \downarrow< t = \textit{set-restriction-fun}\ (\lambda x.\ x < t)\ I$ **and**
cut-le-eq-set-restriction-fun: $I \downarrow\leq t = \textit{set-restriction-fun}\ (\lambda x.\ x \leq t)\ I$ **and**
cut-greater-eq-set-restriction-fun: $I \downarrow> t = \textit{set-restriction-fun}\ (\lambda x.\ x > t)\ I$ **and**
cut-ge-eq-set-restriction-fun: $I \downarrow\geq t = \textit{set-restriction-fun}\ (\lambda x.\ x \geq t)\ I$
⟨*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*:
$[\![$ *cut-op* = *op* $\downarrow<$ ∨ *cut-op* = *op* $\downarrow\leq$ ∨
cut-op = *op* $\downarrow>$ ∨ *cut-op* = *op* $\downarrow\geq$;
$f = (\lambda I.\ \textit{cut-op}\ I\ t)$ $]\!] \Longrightarrow$ *set-restriction f*
⟨*proof*⟩
thm *set-restriction-def*
corollary
i-cut-less-set-restriction: *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*: $[\![$
cut-op1 = *op* $\downarrow<$ ∨ *cut-op1* = *op* $\downarrow\leq$ ∨
cut-op1 = *op* $\downarrow>$ ∨ *cut-op1* = *op* $\downarrow\geq$;
cut-op2 = *op* $\downarrow<$ ∨ *cut-op2* = *op* $\downarrow\leq$ ∨
cut-op2 = *op* $\downarrow>$ ∨ *cut-op2* = *op* $\downarrow\geq$ $]\!] \Longrightarrow$
cut-op2 (*cut-op1 I t1*) *t2* = *cut-op1* (*cut-op2 I t2*) *t1*
thm
set-restriction-commute[*of* $\lambda I.$ *cut-op1 I t*]
i-cut-set-restriction-disj
⟨*proof*⟩
thm *i-cut-commute-disj*
thm *i-cut-commute-disj*[*of op* $\downarrow<$ *op* $\downarrow<$, *simplified*]
thm *i-cut-commute-disj*[*of op* $\downarrow<$ *op* $\downarrow>$, *simplified*]
thm *i-cut-commute-disj*[*of op* $\downarrow\leq$ *op* $\downarrow>$, *simplified*]
thm *i-cut-commute-disj*[*of op* $\downarrow\geq$ *op* $\downarrow>$, *simplified*]

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
thm *i-cut-empty*

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

lemma
cut-less-singleton: $\{a\} \downarrow< t = (\textit{if } a < t \textit{ then } \{a\} \textit{ else } \{\})$ **and**
cut-le-singleton: $\{a\} \downarrow\leq t = (\textit{if } a \leq t \textit{ then } \{a\} \textit{ else } \{\})$ **and**
cut-greater-singleton: $\{a\} \downarrow> t = (\textit{if } a > t \textit{ then } \{a\} \textit{ else } \{\})$ **and**
cut-ge-singleton: $\{a\} \downarrow\geq t = (\textit{if } a \geq t \textit{ then } \{a\} \textit{ else } \{\})$
⟨*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: $I \downarrow\geq t \subseteq I$
thm *i-cut-set-restriction*[*THEN set-restriction-subset*]
⟨*proof*⟩
lemmas *i-cut-subset* =
cut-less-subset cut-le-subset
cut-greater-subset cut-ge-subset
thm *i-cut-subset*

thm *set-restriction-Un*
lemma *i-cut-Un-disj*:
$[\![$ *cut-op* = *op* $\downarrow<$ $\vee$ *cut-op* = *op* $\downarrow\leq$ $\vee$
cut-op = *op* $\downarrow>$ $\vee$ *cut-op* = *op* $\downarrow\geq$ $]\!]$
$\Longrightarrow$ *cut-op* $(A \cup B)$ t = *cut-op* A $t \cup$ *cut-op* B t
thm *i-cut-set-restriction-disj*[*of cut-op* $\lambda I.$ *cut-op* I t t]
⟨*proof*⟩
thm *set-restriction-Un*
⟨*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*:
$[\![$ *cut-op* = *op* $\downarrow<$ $\vee$ *cut-op* = *op* $\downarrow\leq$ $\vee$
cut-op = *op* $\downarrow>$ $\vee$ *cut-op* = *op* $\downarrow\geq$ $]\!]$
$\Longrightarrow$ *cut-op* $(A \cap B)$ t = *cut-op* A $t \cap$ *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*:
⟦ *cut-op* = *op* ↓< ∨ *cut-op* = *op* ↓≤ ∨
cut-op = *op* ↓> ∨ *cut-op* = *op* ↓≥ ⟧
⟹ *cut-op* (*A* − *B*) *t* = *cut-op A t* − *cut-op B t*
⟨*proof*⟩
corollary
cut-less-Diff: (*A* − *B*) ↓< *t* = *A* ↓< *t* − *B* ↓< *t* **and**
cut-le-Diff: (*A* − *B*) ↓≤ *t* = *A* ↓≤ *t* − *B* ↓≤ *t* **and**
cut-greater-Diff: (*A* − *B*) ↓> *t* = *A* ↓> *t* − *B* ↓> *t* **and**
cut-ge-Diff: (*A* − *B*) ↓≥ *t* = *A* ↓≥ *t* − *B* ↓≥ *t*
⟨*proof*⟩
lemmas *i-cut-Diff* =
cut-less-Diff cut-le-Diff
cut-greater-Diff cut-ge-Diff

thm *set-restriction-mono*
lemma *i-cut-subset-mono-disj*:
⟦ *cut-op* = *op* ↓< ∨ *cut-op* = *op* ↓≤ ∨
cut-op = *op* ↓> ∨ *cut-op* = *op* ↓≥; *A* ⊆ *B* ⟧
⟹ *cut-op A t* ⊆ *cut-op B t*
⟨*proof*⟩
thm *set-restriction-mono*[**where** *f*=λ*I*. *cut-op I t*]
⟨*proof*⟩
corollary
cut-less-subset-mono: *A* ⊆ *B* ⟹ *A* ↓< *t* ⊆ *B* ↓< *t* **and**
cut-le-subset-mono: *A* ⊆ *B* ⟹ *A* ↓≤ *t* ⊆ *B* ↓≤ *t* **and**
cut-greater-subset-mono: *A* ⊆ *B* ⟹ *A* ↓> *t* ⊆ *B* ↓> *t* **and**
cut-ge-subset-mono: *A* ⊆ *B* ⟹ *A* ↓≥ *t* ⊆ *B* ↓≥ *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* ≤ *t*′ ⟹ *I* ↓< *t* ⊆ *I* ↓< *t*′ **and**
cut-greater-mono: *t*′ ≤ *t* ⟹ *I* ↓> *t* ⊆ *I* ↓> *t*′ **and**
cut-le-mono: *t* ≤ *t*′ ⟹ *I* ↓≤ *t* ⊆ *I* ↓≤ *t*′ **and**
cut-ge-mono: *t*′ ≤ *t* ⟹ *I* ↓≥ *t* ⊆ *I* ↓≥ *t*′
⟨*proof*⟩
lemmas *i-cut-mono* =
cut-le-mono cut-less-mono
cut-ge-mono cut-greater-mono

lemma
cut-cut-le: $i \downarrow\leq a \downarrow\leq b = i \downarrow\leq \mathit{min}\ a\ b$ **and**
cut-cut-less: $i \downarrow< a \downarrow< b = i \downarrow< \mathit{min}\ a\ b$ **and**
cut-cut-ge: $i \downarrow\geq a \downarrow\geq b = i \downarrow\geq \mathit{max}\ a\ b$ **and**
cut-cut-greater: $i \downarrow> a \downarrow> b = i \downarrow> \mathit{max}\ a\ b$
⟨*proof*⟩
lemmas *i-cut-cut* =
cut-cut-le cut-cut-less
cut-cut-ge cut-cut-greater

lemma *i-cut-absorb-disj*:
$[\![\ \mathit{cut\text{-}op} = \mathit{op} \downarrow< \vee\ \mathit{cut\text{-}op} = \mathit{op} \downarrow\leq \vee$
$\mathit{cut\text{-}op} = \mathit{op} \downarrow> \vee\ \mathit{cut\text{-}op} = \mathit{op} \downarrow\geq\ ]\!]$
$\Longrightarrow \mathit{cut\text{-}op}\ (\mathit{cut\text{-}op}\ I\ t)\ t = \mathit{cut\text{-}op}\ I\ t$
thm *i-cut-set-restriction-disj*[**where** $f{=}\lambda I.\ \mathit{cut\text{-}op}\ I\ t$]
⟨*proof*⟩
thm *set-restriction-absorb*
⟨*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: $I \downarrow> t \downarrow> t = I \downarrow> t$
thm *i-cut-absorb-disj*
⟨*proof*⟩
lemmas *i-cut-absorb* =
cut-le-absorb cut-less-absorb
cut-ge-absorb cut-greater-absorb

lemma
cut-less-0-empty: $I \downarrow< (0{::}\mathit{nat}) = \{\}$ **and**
cut-ge-0-all: $I \downarrow\geq (0{::}\mathit{nat}) = I$
⟨*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
thm *i-cut-not-empty-iff*

lemma *nat-cut-ge-infinite-not-empty*: *infinite* $I \Longrightarrow I \downarrow\geq (t{::}nat) \neq \{\}$
⟨*proof*⟩
lemma *nat-cut-greater-infinite-not-empty*: *infinite* $I \Longrightarrow I \downarrow> (t{::}nat) \neq \{\}$
⟨*proof*⟩

thm *set-restriction-not-in-imp*
corollary
cut-le-not-in-imp: $x \notin I \Longrightarrow x \notin I \downarrow\leq t$ **and**
cut-less-not-in-imp: $x \notin I \Longrightarrow x \notin I \downarrow< t$ **and**
cut-ge-not-in-imp: $x \notin I \Longrightarrow x \notin I \downarrow\geq t$ **and**
cut-greater-not-in-imp: $x \notin I \Longrightarrow x \notin I \downarrow> t$
thm *i-cut-set-restriction*[*THEN set-restriction-not-in-imp*]
⟨*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

thm *set-restriction-in-imp*
corollary
cut-le-in-imp: $x \in I \downarrow\leq t \Longrightarrow x \in I$ **and**
cut-less-in-imp: $x \in I \downarrow< t \Longrightarrow x \in I$ **and**
cut-ge-in-imp: $x \in I \downarrow\geq t \Longrightarrow x \in I$ **and**
cut-greater-in-imp: $x \in I \downarrow> t \Longrightarrow x \in I$
thm *i-cut-set-restriction*[*THEN set-restriction-in-imp*]
⟨*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*: $[\![\ k \in I;\ P\ (k::('a::wellorder))\ ]\!] \Longrightarrow \exists\, x{\in}I.\ P\ x \wedge (\forall\, y{\in}(I \downarrow< x).\ \neg\ P\ y)$
⟨*proof*⟩
corollary *Collect-minI-ex-cut*: $\exists\, k{\in}I.\ P\ (k::('a::wellorder)) \Longrightarrow \exists\, x{\in}I.\ P\ x \wedge (\forall\, y{\in}(I \downarrow< x).\ \neg\ P\ y)$
⟨*proof*⟩
corollary *Collect-minI-ex2-cut*: $\{k{\in}I.\ P\ (k::('a::wellorder))\} \neq \{\} \Longrightarrow \exists\, x{\in}I.\ P\ x \wedge (\forall\, y{\in}(I \downarrow< x).\ \neg\ P\ y)$
⟨*proof*⟩

lemma *cut-le-cut-greater-ident*: $t2 \leq t1 \Longrightarrow I \downarrow\leq t1 \cup I \downarrow> t2 = I$
⟨*proof*⟩
lemma *cut-less-cut-ge-ident*: $t2 \leq t1 \Longrightarrow I \downarrow< t1 \cup I \downarrow\geq t2 = I$
⟨*proof*⟩
lemma *cut-le-cut-ge-ident*: $t2 \leq t1 \Longrightarrow I \downarrow\leq t1 \cup I \downarrow\geq t2 = I$
⟨*proof*⟩

lemma *cut-less-cut-greater-ident*:
$[\![\ t2 \leq t1;\ I \cap \{t1..t2\} = \{\}\ ]\!] \Longrightarrow I \downarrow< t1 \cup I \downarrow> t2 = I$
⟨*proof*⟩
corollary *cut-less-cut-greater-ident'*:
$t \notin I \Longrightarrow I \downarrow< t \cup I \downarrow> t = I$
⟨*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*: $A \cap (insert\ x\ B) =$ (
if $x \in A$ *then 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 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 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*: $I \downarrow\leq t = I \downarrow< Suc\ t$
⟨*proof*⟩
lemma *nat-cut-less-le-conv*: $0 < t \Longrightarrow I \downarrow< t = I \downarrow\leq (t - Suc\ 0)$
⟨*proof*⟩
lemma *nat-cut-ge-greater-conv*: $I \downarrow\geq Suc\ t = I \downarrow> t$
⟨*proof*⟩
lemma *nat-cut-greater-ge-conv*: $0 < t \Longrightarrow I \downarrow> (t - Suc\ 0) = I \downarrow\geq t$
⟨*proof*⟩

7.2.4 Function images with cut operators

lemma *cut-less-image*:
  $[\![$ *strict-mono-on* $f\ A$; $I \subseteq A$; $n \in A\ ]\!] \Longrightarrow$
  $(f\ `\ I) \downarrow< f\ n = f\ `\ (I \downarrow< n)$
⟨*proof*⟩

lemma *cut-le-image*:
  $[\![$ *strict-mono-on* $f\ A$; $I \subseteq A$; $n \in A\ ]\!] \Longrightarrow$
  $(f\ `\ I) \downarrow\leq f\ n = f\ `\ (I \downarrow\leq n)$
⟨*proof*⟩
thm *cut-le-less-conv-if*
⟨*proof*⟩

lemma *cut-greater-image*:

⟦ *strict-mono-on f A*; $I \subseteq A$; $n \in A$ ⟧ ⟹
$(f \text{ ` } I) \downarrow> f\ n = f \text{ ` } (I \downarrow> n)$
⟨*proof*⟩

lemma *cut-ge-image*:
⟦ *strict-mono-on f A*; $I \subseteq A$; $n \in A$ ⟧ ⟹
$(f \text{ ` } I) \downarrow\geq f\ n = f \text{ ` } (I \downarrow\geq n)$
⟨*proof*⟩
thm *cut-ge-greater-conv-if*
⟨*proof*⟩

lemmas *i-cut-image* =
cut-le-image cut-less-image
cut-ge-image cut-greater-image
thm *i-cut-image*
thm *i-cut-image*[*OF - subset-refl*]

7.2.5 Finiteness and cardinality with cut operators

thm *set-restriction-finite*
lemma
cut-le-finite: *finite I* ⟹ *finite* $(I \downarrow\leq t)$ **and**
cut-less-finite: *finite I* ⟹ *finite* $(I \downarrow< t)$ **and**
cut-ge-finite: *finite I* ⟹ *finite* $(I \downarrow\geq t)$ **and**
cut-greater-finite: *finite I* ⟹ *finite* $(I \downarrow> t)$
thm *finite-subset*
⟨*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*⟩
thm *cut-less-cut-ge-ident*[*OF order-refl*]
⟨*proof*⟩
lemma *nat-cut-greater-finite-iff*: *finite* $(I \downarrow> (t::nat)) =$ *finite I*
thm *cut-ge-greater-conv*
⟨*proof*⟩

lemma
cut-le-card: *finite I* ⟹ *card* $(I \downarrow\leq t) \leq$ *card I* **and**
cut-less-card: *finite I* ⟹ *card* $(I \downarrow< t) \leq$ *card I* **and**
cut-ge-card: *finite I* ⟹ *card* $(I \downarrow\geq t) \leq$ *card I* **and**
cut-greater-card: *finite I* ⟹ *card* $(I \downarrow> t) \leq$ *card I*
⟨*proof*⟩

lemma *nat-cut-greater-card*: *card* $(I \downarrow> (t::nat)) \leq$ *card I*

⟨*proof*⟩
thm *nat-cut-greater-finite-iff*
⟨*proof*⟩
lemma *nat-cut-ge-card*: *card* $(I \downarrow\geq (t::nat)) \leq$ *card* I
⟨*proof*⟩
thm *nat-cut-ge-finite-iff*
⟨*proof*⟩

7.2.6 Cutting a set at *Min* or *Max* element

lemma *cut-greater-Min-eq-Diff*: $I \downarrow> (iMin\ I) = I - \{iMin\ I\}$
⟨*proof*⟩
lemma *cut-less-Max-eq-Diff*: *finite* $I \Longrightarrow I \downarrow< (Max\ I) = I - \{Max\ I\}$
⟨*proof*⟩

lemma *cut-le-Min-empty*: $t < iMin\ I \Longrightarrow I \downarrow\leq t = \{\}$
⟨*proof*⟩
lemma *cut-less-Min-empty*: $t \leq iMin\ I \Longrightarrow I \downarrow< t = \{\}$
⟨*proof*⟩

lemma *cut-le-Min-not-empty*: $[\![\ I \neq \{\};\ iMin\ I \leq t\ ]\!] \Longrightarrow I \downarrow\leq t \neq \{\}$
⟨*proof*⟩
thm *iMinI-ex2*
⟨*proof*⟩
lemma *cut-less-Min-not-empty*: $[\![\ I \neq \{\};\ iMin\ I < t\ ]\!] \Longrightarrow I \downarrow< t \neq \{\}$
⟨*proof*⟩

lemma *cut-ge-Min-all*: $t \leq iMin\ I \Longrightarrow I \downarrow\geq t = I$
⟨*proof*⟩
thm *iMin-le*
⟨*proof*⟩

lemma *cut-greater-Min-all*: $t < iMin\ I \Longrightarrow I \downarrow> t = I$
⟨*proof*⟩
thm *iMin-le*
⟨*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

thm
  i-cut-min-empty

i-cut-min-all

lemma *cut-ge-Max-empty*: *finite* $I \Longrightarrow Max\ I < t \Longrightarrow I \downarrow\geq t = \{\}$
⟨*proof*⟩

lemma *cut-greater-Max-empty*: *finite* $I \Longrightarrow Max\ I \leq t \Longrightarrow I \downarrow> t = \{\}$
⟨*proof*⟩

lemma *cut-ge-Max-not-empty*: $[\![\ I \neq \{\};$ *finite* $I;\ t \leq Max\ I\ ]\!] \Longrightarrow I \downarrow\geq t \neq \{\}$
⟨*proof*⟩
thm *MaxI-ex2*
⟨*proof*⟩

lemma *cut-greater-Max-not-empty*: $[\![\ I \neq \{\};$ *finite* $I;\ t < Max\ I\ ]\!] \Longrightarrow I \downarrow> t \neq \{\}$
⟨*proof*⟩

lemma *cut-le-Max-all*: *finite* $I \Longrightarrow Max\ I \leq t \Longrightarrow I \downarrow\leq t = I$
⟨*proof*⟩

lemma *cut-less-Max-all*: *finite* $I \Longrightarrow Max\ I < t \Longrightarrow I \downarrow< t = I$
⟨*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

thm
i-cut-max-empty
i-cut-max-all

lemma *cut-less-Max-less*:
$[\![$ *finite* $(I \downarrow< t);\ I \downarrow< t \neq \{\}\ ]\!] \Longrightarrow Max\ (I \downarrow< t) < t$
⟨*proof*⟩
lemma *cut-le-Max-le*:
$[\![$ *finite* $(I \downarrow\leq t);\ I \downarrow\leq t \neq \{\}\ ]\!] \Longrightarrow Max\ (I \downarrow\leq t) \leq t$
⟨*proof*⟩
lemma *nat-cut-less-Max-less*:
$I \downarrow< t \neq \{\} \Longrightarrow Max\ (I \downarrow< t) < (t::nat)$
⟨*proof*⟩
lemma *nat-cut-le-Max-le*:
$I \downarrow\leq t \neq \{\} \Longrightarrow Max\ (I \downarrow\leq t) \leq (t::nat)$
⟨*proof*⟩

lemma *cut-greater-Min-greater*:
$I \downarrow> t \neq \{\} \Longrightarrow iMin\ (I \downarrow> t) > t$
⟨*proof*⟩
lemma *cut-ge-Min-greater*:
$I \downarrow\geq t \neq \{\} \Longrightarrow iMin\ (I \downarrow\geq t) \geq t$
⟨*proof*⟩

lemma *cut-less-Min-eq*: $I \downarrow< t \neq \{\} \Longrightarrow iMin\ (I \downarrow< t) = iMin\ I$
⟨*proof*⟩

lemma *cut-le-Min-eq*: $I \downarrow\leq t \neq \{\} \Longrightarrow iMin\ (I \downarrow\leq t) = iMin\ I$
⟨*proof*⟩

lemma *cut-ge-Max-eq*: $[\![\ finite\ I;\ I \downarrow\geq t \neq \{\}\ ]\!] \Longrightarrow Max\ (I \downarrow\geq t) = Max\ I$
⟨*proof*⟩

lemma *cut-greater-Max-eq*: $[\![\ finite\ I;\ I \downarrow> t \neq \{\}\ ]\!] \Longrightarrow Max\ (I \downarrow> t) = Max\ I$
⟨*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: $\{..<n\} \downarrow\leq t = (if\ n \leq t\ then\ \{..<n\}\ else\ \{..t\})$ **and**
lessThan-cut-less: $\{..<n\} \downarrow< t = (if\ n \leq t\ then\ \{..<n\}\ else\ \{..<t\})$ **and**
lessThan-cut-ge: $\{..<n\} \downarrow\geq t = \{t..<n\}$ **and**
lessThan-cut-greater: $\{..<n\} \downarrow> t = \{t<..<n\}$ **and**
atMost-cut-le: $\{..n\} \downarrow\leq t = (if\ n \leq t\ then\ \{..n\}\ else\ \{..t\})$ **and**
atMost-cut-less: $\{..n\} \downarrow< t = (if\ n < t\ then\ \{..n\}\ else\ \{..<t\})$ **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: $\{n<..\} \downarrow\geq t = (if\ t \leq n\ then\ \{n<..\}\ else\ \{t..\})$ **and**
greaterThan-cut-greater: $\{n<..\} \downarrow> t = (if\ t \leq n\ then\ \{n<..\}\ else\ \{t<..\})$ **and**
atLeast-cut-le: $\{n..\} \downarrow\leq t = \{n..t\}$ **and**
atLeast-cut-less: $\{n..\} \downarrow< t = \{n..<t\}$ **and**
atLeast-cut-ge: $\{n..\} \downarrow\geq t = (if\ t \leq n\ then\ \{n..\}\ else\ \{t..\})$ **and**
atLeast-cut-greater: $\{n..\} \downarrow\geq t = (if\ t \leq n\ then\ \{n..\}\ else\ \{t..\})$
⟨*proof*⟩

lemma
greaterThanLessThan-cut-le: $\{m<..<n\} \downarrow\leq t = (\mathit{if}\ n \leq t\ \mathit{then}\ \{m<..<n\}\ \mathit{else}\ \{m<..t\})$ **and**
greaterThanLessThan-cut-less: $\{m<..<n\} \downarrow< t = (\mathit{if}\ n \leq t\ \mathit{then}\ \{m<..<n\}\ \mathit{else}\ \{m<..<t\})$ **and**
greaterThanLessThan-cut-ge: $\{m<..<n\} \downarrow\geq t = (\mathit{if}\ t \leq m\ \mathit{then}\ \{m<..<n\}\ \mathit{else}\ \{t..<n\})$ **and**
greaterThanLessThan-cut-greater: $\{m<..<n\} \downarrow> t = (\mathit{if}\ t \leq m\ \mathit{then}\ \{m<..<n\}\ \mathit{else}\ \{t<..<n\})$ **and**
atLeastLessThan-cut-le: $\{m..<n\} \downarrow\leq t = (\mathit{if}\ n \leq t\ \mathit{then}\ \{m..<n\}\ \mathit{else}\ \{m..t\})$ **and**
atLeastLessThan-cut-less: $\{m..<n\} \downarrow< t = (\mathit{if}\ n \leq t\ \mathit{then}\ \{m..<n\}\ \mathit{else}\ \{m..<t\})$ **and**
atLeastLessThan-cut-ge: $\{m..<n\} \downarrow\geq t = (\mathit{if}\ t \leq m\ \mathit{then}\ \{m..<n\}\ \mathit{else}\ \{t..<n\})$ **and**
atLeastLessThan-cut-greater: $\{m..<n\} \downarrow> t = (\mathit{if}\ t < m\ \mathit{then}\ \{m..<n\}\ \mathit{else}\ \{t<..<n\})$ **and**
greaterThanAtMost-cut-le: $\{m<..n\} \downarrow\leq t = (\mathit{if}\ n \leq t\ \mathit{then}\ \{m<..n\}\ \mathit{else}\ \{m<..t\})$ **and**
greaterThanAtMost-cut-less: $\{m<..n\} \downarrow< t = (\mathit{if}\ n < t\ \mathit{then}\ \{m<..n\}\ \mathit{else}\ \{m<..<t\})$ **and**
greaterThanAtMost-cut-ge: $\{m<..n\} \downarrow\geq t = (\mathit{if}\ t \leq m\ \mathit{then}\ \{m<..n\}\ \mathit{else}\ \{t..n\})$ **and**
greaterThanAtMost-cut-greater: $\{m<..n\} \downarrow> t = (\mathit{if}\ t \leq m\ \mathit{then}\ \{m<..n\}\ \mathit{else}\ \{t<..n\})$ **and**
atLeastAtMost-cut-le: $\{m..n\} \downarrow\leq t = (\mathit{if}\ n \leq t\ \mathit{then}\ \{m..n\}\ \mathit{else}\ \{m..t\})$ **and**
atLeastAtMost-cut-less: $\{m..n\} \downarrow< t = (\mathit{if}\ n < t\ \mathit{then}\ \{m..n\}\ \mathit{else}\ \{m..<t\})$ **and**
atLeastAtMost-cut-ge: $\{m..n\} \downarrow\geq t = (\mathit{if}\ t \leq m\ \mathit{then}\ \{m..n\}\ \mathit{else}\ \{t..n\})$ **and**
atLeastAtMost-cut-greater: $\{m..n\} \downarrow> t = (\mathit{if}\ t < m\ \mathit{then}\ \{m..n\}\ \mathit{else}\ \{t<..n\})$
⟨*proof*⟩

7.2.8 Mirroring finite natural sets between their *Min* and *Max* element

Mirroring a number at the middle of the interval min l r..max l r

Mirroring a single element n between the interval boundaries l and r

definition
nat-mirror :: $nat \Rightarrow nat \Rightarrow nat \Rightarrow nat$
where
$\mathit{nat\text{-}mirror}\ n\ l\ r \equiv l + r - n$

lemma *nat-mirror-commute*: $\mathit{nat\text{-}mirror}\ n\ l\ r = \mathit{nat\text{-}mirror}\ n\ r\ l$
⟨*proof*⟩

lemma *nat-mirror-inj-on*: *inj-on* ($\lambda x.$ *nat-mirror* x l r) $\{..l + r\}$
⟨*proof*⟩

lemma *nat-mirror-nat-mirror-ident*:
$n \leq l + r \Longrightarrow$ *nat-mirror* (*nat-mirror* n l r) l $r = n$
⟨*proof*⟩

lemma *nat-mirror-add*:
nat-mirror $(n + k)$ l r = (*nat-mirror* n l r) $- k$
⟨*proof*⟩
lemma *nat-mirror-diff*:
$[\![k \leq n; n \leq l + r]\!] \Longrightarrow$
nat-mirror $(n - k)$ l r = (*nat-mirror* n l r) $+ k$
⟨*proof*⟩

lemma *nat-mirror-le*: $a \leq b \Longrightarrow$ *nat-mirror* b l $r \leq$ *nat-mirror* a l r
⟨*proof*⟩
lemma *nat-mirror-le-conv*:
$a \leq l + r \Longrightarrow$ (*nat-mirror* b l $r \leq$ *nat-mirror* a l r) = $(a \leq b)$
⟨*proof*⟩

lemma *nat-mirror-less*:
$[\![a < b; a < l + r]\!] \Longrightarrow$
nat-mirror b l $r <$ *nat-mirror* a l r
⟨*proof*⟩
lemma *nat-mirror-less-imp-less*:
nat-mirror b l $r <$ *nat-mirror* a l $r \Longrightarrow a < b$
⟨*proof*⟩
lemma *nat-mirror-less-conv*:
$a < l + r \Longrightarrow$ (*nat-mirror* b l $r <$ *nat-mirror* a l r) = $(a < b)$
⟨*proof*⟩

lemma *nat-mirror-eq-conv*:
$[\![a \leq l + r; b \leq l + r]\!] \Longrightarrow$
(*nat-mirror* a l r = *nat-mirror* b l r) = $(a = b)$
⟨*proof*⟩

Mirroring a single element n between the interval boundaries of I

definition
mirror-elem :: *nat* ⇒ *nat set* ⇒ *nat*
where
mirror-elem n $I \equiv$ *nat-mirror* n (*iMin* I) (*Max* I)

lemma *mirror-elem-inj-on*: *finite* $I \Longrightarrow$ *inj-on* ($\lambda x.$ *mirror-elem* x I) I
⟨*proof*⟩

lemma *mirror-elem-add*:
finite $I \Longrightarrow$ *mirror-elem* $(n + k)$ I = *mirror-elem* n $I - k$
⟨*proof*⟩

lemma *mirror-elem-diff*:
$[\![\mathit{finite}\ I;\ k \leq n;\ n \in I\ ]\!] \Longrightarrow \mathit{mirror\text{-}elem}\ (n - k)\ I = \mathit{mirror\text{-}elem}\ n\ I + k$
⟨*proof*⟩
lemma *mirror-elem-Min*:
$[\![\mathit{finite}\ I;\ I \neq \{\}\ ]\!] \Longrightarrow \mathit{mirror\text{-}elem}\ (\mathit{iMin}\ I)\ I = \mathit{Max}\ I$
⟨*proof*⟩
lemma *mirror-elem-Max*:
$[\![\mathit{finite}\ I;\ I \neq \{\}\ ]\!] \Longrightarrow \mathit{mirror\text{-}elem}\ (\mathit{Max}\ I)\ I = \mathit{iMin}\ I$
⟨*proof*⟩

lemma *mirror-elem-mirror-elem-ident*:
$[\![\mathit{finite}\ I;\ n \leq \mathit{iMin}\ I + \mathit{Max}\ I\ ]\!] \Longrightarrow \mathit{mirror\text{-}elem}\ (\mathit{mirror\text{-}elem}\ n\ I)\ I = n$
⟨*proof*⟩

thm *nat-mirror-le-conv*
lemma *mirror-elem-le-conv*:
$[\![\mathit{finite}\ I;\ a \in I;\ b \in I\ ]\!] \Longrightarrow$
$(\mathit{mirror\text{-}elem}\ b\ I \leq \mathit{mirror\text{-}elem}\ a\ I) = (a \leq b)$
⟨*proof*⟩
lemma *mirror-elem-less-conv*:
$[\![\mathit{finite}\ I;\ a \in I;\ b \in I\ ]\!] \Longrightarrow$
$(\mathit{mirror\text{-}elem}\ b\ I < \mathit{mirror\text{-}elem}\ a\ I) = (a < b)$
⟨*proof*⟩

thm *nat-mirror-eq-conv*
lemma *mirror-elem-eq-conv*:
$[\![a \leq \mathit{iMin}\ I + \mathit{Max}\ I;\ b \leq \mathit{iMin}\ I + \mathit{Max}\ I\ ]\!] \Longrightarrow$
$(\mathit{mirror\text{-}elem}\ a\ I = \mathit{mirror\text{-}elem}\ b\ I) = (a = b)$
⟨*proof*⟩
lemma *mirror-elem-eq-conv'*:
$[\![\mathit{finite}\ I;\ a \in I;\ b \in I\ ]\!] \Longrightarrow (\mathit{mirror\text{-}elem}\ a\ I = \mathit{mirror\text{-}elem}\ b\ I) = (a = b)$
⟨*proof*⟩

definition
$\mathit{imirror\text{-}bounds} :: \mathit{nat\ set} \Rightarrow \mathit{nat} \Rightarrow \mathit{nat} \Rightarrow \mathit{nat\ set}$
where
$\mathit{imirror\text{-}bounds}\ I\ l\ r \equiv (\lambda x.\ \mathit{nat\text{-}mirror}\ x\ l\ r)\ `\ I$

Mirroring all elements between the interval boundaries of I

definition
$\mathit{imirror} :: \mathit{nat\ set} \Rightarrow \mathit{nat\ set}$
where
$\mathit{imirror}\ I \equiv (\lambda x.\ \mathit{iMin}\ I + \mathit{Max}\ I - x)\ `\ I$

lemma *imirror-eq-nat-mirror-image*:
$\mathit{imirror}\ I = (\lambda x.\ \mathit{nat\text{-}mirror}\ x\ (\mathit{iMin}\ I)\ (\mathit{Max}\ I))\ `\ I$

⟨*proof*⟩
lemma *imirror-eq-mirror-elem-image*:
imirror I = (λ*x*. *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* ≠ {} ⟹ *imirror I* ≠ {}
⟨*proof*⟩

lemma *imirror-singleton*: *imirror* {*a*} = {*a*}
⟨*proof*⟩

lemma *imirror-finite*: *finite I* ⟹ *finite* (*imirror I*)
⟨*proof*⟩

lemma *imirror-bounds-iMin*:
⟦ *finite I*; *I* ≠ {}; *iMin I* ≤ *l* + *r* ⟧ ⟹
iMin (*imirror-bounds I l r*) = *l* + *r* − *Max I*
⟨*proof*⟩
thm *iMin-equality*
⟨*proof*⟩

lemma *imirror-bounds-Max*:
⟦ *finite I*; *I* ≠ {}; *Max I* ≤ *l* + *r* ⟧ ⟹
Max (*imirror-bounds I l r*) = *l* + *r* − *iMin I*
⟨*proof*⟩

lemma *imirror-iMin*: *finite I* ⟹ *iMin* (*imirror I*) = *iMin I*
⟨*proof*⟩

lemma *imirror-Max*: *finite I* ⟹ *Max* (*imirror I*) = *Max I*
⟨*proof*⟩
corollary *imirror-iMin-Max*: ⟦ *finite I*; *n* ∈ *imirror I* ⟧ ⟹ *iMin I* ≤ *n* ∧ *n* ≤ *Max I*
⟨*proof*⟩
thm *imirror-iMin imirror-Max*
⟨*proof*⟩

thm *image-iff*
lemma *imirror-bounds-iff*:
$(n \in \mathit{imirror\text{-}bounds}\ I\ l\ r) = (\exists\, x{\in}I.\ n = l + r - x)$
⟨*proof*⟩
lemma *imirror-iff*: $(n \in \mathit{imirror}\ I) = (\exists\, x{\in}I.\ n = \mathit{iMin}\ I + \mathit{Max}\ I - x)$
⟨*proof*⟩

lemma *imirror-bounds-imirror-bounds-ident*:
$[\![\ \mathit{finite}\ I;\ \mathit{Max}\ I \le l + r\ ]\!] \Longrightarrow$
$\mathit{imirror\text{-}bounds}\ (\mathit{imirror\text{-}bounds}\ I\ l\ r)\ l\ r = I$
⟨*proof*⟩
thm *nat-mirror-nat-mirror-ident*
⟨*proof*⟩

lemma *imirror-imirror-ident*: $\mathit{finite}\ I \Longrightarrow \mathit{imirror}\ (\mathit{imirror}\ I) = I$
⟨*proof*⟩
thm *imirror-eq-imirror-bounds*
thm *imirror-bounds-imirror-bounds-ident*
⟨*proof*⟩
lemma *mirror-elem-imirror*:
$\mathit{finite}\ I \Longrightarrow \mathit{mirror\text{-}elem}\ t\ (\mathit{imirror}\ I) = \mathit{mirror\text{-}elem}\ t\ I$
⟨*proof*⟩

lemma *imirror-card*: $\mathit{finite}\ I \Longrightarrow \mathit{card}\ (\mathit{imirror}\ I) = \mathit{card}\ I$
⟨*proof*⟩

lemma *imirror-bounds-elem-conv*:
$[\![\ \mathit{finite}\ I;\ n \le l + r;\ \mathit{Max}\ I \le l + r\ ]\!] \Longrightarrow$
$((\mathit{nat\text{-}mirror}\ n\ l\ r) \in \mathit{imirror\text{-}bounds}\ I\ l\ r) = (n \in I)$
⟨*proof*⟩
thm *inj-on-image-mem-iff*[**where** $A{=}\{..l + r\}$]
⟨*proof*⟩

lemma *imirror-mem-conv*:
$[\![\ \mathit{finite}\ I;\ n \le \mathit{iMin}\ I + \mathit{Max}\ I\ ]\!] \Longrightarrow ((\mathit{mirror\text{-}elem}\ n\ I) \in \mathit{imirror}\ I) = (n \in I)$
thm *imirror-bounds-elem-conv*
⟨*proof*⟩

corollary *in-imp-mirror-elem-in*:
$[\![\ \mathit{finite}\ I;\ n \in I\ ]\!] \Longrightarrow (\mathit{mirror\text{-}elem}\ n\ I) \in \mathit{imirror}\ I$
thm *imirror-mem-conv*[*OF - trans-le-add2*[*OF Max-ge*], *THEN iffD2*]
⟨*proof*⟩

lemma *imirror-cut-less*:
$\mathit{finite}\ I \Longrightarrow$
$\mathit{imirror}\ I \downarrow{<}\ (\mathit{mirror\text{-}elem}\ t\ I) =$
$\mathit{imirror\text{-}bounds}\ (I \downarrow{>}\ t)\ (\mathit{iMin}\ I)\ (\mathit{Max}\ I)$
⟨*proof*⟩

thm *nat-mirror-less-imp-less*
⟨*proof*⟩
thm *nat-mirror-less*
⟨*proof*⟩

lemma *imirror-cut-le*:
⟦ *finite I*; $t \leq iMin\ I + Max\ I$ ⟧ ⟹
imirror I $\downarrow\leq$ (*mirror-elem t I*) =
imirror-bounds (*I* $\downarrow\geq$ *t*) (*iMin I*) (*Max I*)
thm *nat-cut-le-less-conv nat-cut-greater-ge-conv*
⟨*proof*⟩
thm *imirror-cut-less*
⟨*proof*⟩

lemma *imirror-cut-ge*:
finite I ⟹
imirror I $\downarrow\geq$ (*mirror-elem t I*) =
imirror-bounds (*I* $\downarrow\leq$ *t*) (*iMin I*) (*Max I*)
(**is** *?P* ⟹ *?lhs I* = *?rhs I t*)
⟨*proof*⟩
thm *mirror-elem-mirror-elem-ident*
⟨*proof*⟩
thm *imirror-cut-le*[*of imirror I mirror-elem t I*, *OF imirror-finite*]
⟨*proof*⟩

lemma *imirror-cut-greater*: ⟦ *finite I*; $t \leq iMin\ I + Max\ I$ ⟧ ⟹
imirror I $\downarrow>$ (*mirror-elem t I*) =
imirror-bounds (*I* $\downarrow<$ *t*) (*iMin I*) (*Max I*)
⟨*proof*⟩
thm *nat-cut-ge-greater-conv*
⟨*proof*⟩
thm *subst*[*of mirror-elem* (*t* − *Suc 0*) *I Suc* (*mirror-elem t I*)]
⟨*proof*⟩
thm *nat-mirror-diff*
⟨*proof*⟩
thm *imirror-cut-ge nat-cut-less-le-conv*
⟨*proof*⟩

lemmas *imirror-cut* =
imirror-cut-less imirror-cut-ge
imirror-cut-le imirror-cut-greater
thm *imirror-cut*

corollary *imirror-cut-le′*:
⟦ *finite I*; *t* ∈ *I* ⟧ ⟹
imirror I $\downarrow\leq$ *mirror-elem t I* =
imirror-bounds (*I* $\downarrow\geq$ *t*) (*iMin I*) (*Max I*)
thm *imirror-cut-le*[*OF - trans-le-add2*[*OF Max-ge*]]
⟨*proof*⟩

corollary *imirror-cut-greater'*:
⟦ *finite I*; *t* ∈ *I* ⟧ ⟹
imirror I ↓> *mirror-elem t I* =
imirror-bounds (*I* ↓< *t*) (*iMin I*) (*Max I*)
⟨*proof*⟩
lemmas *imirror-cut'* =
imirror-cut-le' imirror-cut-greater'
thm
imirror-cut
imirror-cut'

lemma *imirror-bounds-Un*:
imirror-bounds (*A* ∪ *B*) *l r* =
imirror-bounds A l r ∪ *imirror-bounds B l r*
⟨*proof*⟩
lemma *imirror-bounds-Int*:
⟦ *A* ⊆ {..*l* + *r*}; *B* ⊆ {..*l* + *r*} ⟧ ⟹
imirror-bounds (*A* ∩ *B*) *l r* =
imirror-bounds A l r ∩ *imirror-bounds B l r*
⟨*proof*⟩
thm *inj-on-image-Int*[*OF - Un-upper1 Un-upper2*]
⟨*proof*⟩
thm *nat-mirror-inj-on*
thm *subset-inj-on*[*OF nat-mirror-inj-on*]
⟨*proof*⟩

end

8 SetIntervalStep: Stepping through sets of natural numbers

theory *SetIntervalStep*
imports *SetIntervalCut*
begin

8.1 Function *inext* and *iprev* for stepping through natural sets

definition
inext :: *nat* ⇒ *nat set* ⇒ *nat*
where
inext n I ≡ (
if (*n* ∈ *I* ∧ (*I* ↓> *n* ≠ {}))
then iMin (*I* ↓> *n*)
else n)

definition

iprev :: *nat* ⇒ *nat set* ⇒ *nat*
where
iprev n I ≡ (
if (*n* ∈ *I* ∧ (*I* ↓< *n* ≠ {}))
then Max (*I* ↓< *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* − *Suc 0*
⟨*proof*⟩

lemma *inext-empty*: *inext n* {} = *n*
⟨*proof*⟩
lemma *iprev-empty*: *iprev n* {} = *n*
⟨*proof*⟩

thm
finite-nat-iff-bounded-le
finite-nat-iff-bounded-le2

lemma *not-in-inext-fix*: *n* ∉ *I* ⟹ *inext n I* = *n*
⟨*proof*⟩
lemma *not-in-iprev-fix*: *n* ∉ *I* ⟹ *iprev n I* = *n*
⟨*proof*⟩

lemma *inext-all-le-fix*: ∀ *x*∈*I*. *x* ≤ *n* ⟹ *inext n I* = *n*
⟨*proof*⟩
lemma *iprev-all-ge-fix*: ∀ *x*∈*I*. *n* ≤ *x* ⟹ *iprev n I* = *n*
⟨*proof*⟩

lemma *inext-Max*: *finite I* ⟹ *inext* (*Max I*) *I* = *Max I*
⟨*proof*⟩
lemma *iprev-iMin*: *iprev* (*iMin I*) *I* = *iMin I*
⟨*proof*⟩

lemma *inext-ge-Max*: ⟦ *finite I*; *Max I* ≤ *n* ⟧ ⟹ *inext n I* = *n*
⟨*proof*⟩
thm *iprev-iMin*
lemma *iprev-le-iMin*: *n* ≤ *iMin I* ⟹ *iprev n I* = *n*
⟨*proof*⟩

lemma *inext-singleton*: *inext n* {*a*} = *n*
⟨*proof*⟩

lemma *iprev-singleton*: *iprev n* {*a*} = *n*
⟨*proof*⟩

lemma *inext-closed*: $n \in I \Longrightarrow inext\ n\ I \in I$
⟨*proof*⟩
thm *subsetD*[*of* I $\downarrow>$ n I]
⟨*proof*⟩

lemma *iprev-closed*: $n \in I \Longrightarrow iprev\ n\ I \in I$
⟨*proof*⟩
thm *subsetD*[*of* I $\downarrow<$ n I]
⟨*proof*⟩
thm *Max-in*[*OF nat-cut-less-finite*]
⟨*proof*⟩

thm *inext-closed*
lemma *inext-in-imp-in*: $inext\ n\ I \in I \Longrightarrow n \in I$
⟨*proof*⟩

lemma *inext-in-iff*: $(inext\ n\ I \in I) = (n \in I)$
⟨*proof*⟩

lemma *subset-inext-closed*: $[\![\ n \in B;\ A \subseteq B\ ]\!] \Longrightarrow inext\ n\ A \in B$
⟨*proof*⟩
lemma *subset-inext-in-imp-in*: $[\![\ inext\ n\ A \in B;\ A \subseteq B\ ]\!] \Longrightarrow n \in B$
⟨*proof*⟩
lemma *subset-inext-in-iff*: $A \subseteq B \Longrightarrow (inext\ n\ A \in B) = (n \in B)$
⟨*proof*⟩

thm *iprev-closed*
lemma *iprev-in-imp-in*: $iprev\ n\ I \in I \Longrightarrow n \in I$
⟨*proof*⟩
lemma *iprev-in-iff*: $(iprev\ n\ I \in I) = (n \in I)$
⟨*proof*⟩

lemma *subset-iprev-closed*: $[\![\ n \in B;\ A \subseteq B\ ]\!] \Longrightarrow iprev\ n\ A \in B$
⟨*proof*⟩
lemma *subset-iprev-in-imp-in*: $[\![\ iprev\ n\ A \in B;\ A \subseteq B\ ]\!] \Longrightarrow n \in B$
⟨*proof*⟩
lemma *subset-iprev-in-iff*: $A \subseteq B \Longrightarrow (iprev\ n\ A \in B) = (n \in B)$
⟨*proof*⟩

lemma *inext-mono*: $n \leq inext\ n\ I$
⟨*proof*⟩

corollary *inext-neq-imp-less*: $n \neq inext\ n\ I \Longrightarrow n < inext\ n\ I$
⟨*proof*⟩

lemma *inext-mono2*: $[\![\ n \in I;\ \exists\, x{\in}I.\ n < x\ ]\!] \Longrightarrow n < inext\ n\ I$
⟨*proof*⟩

lemma *inext-mono2-infin*: $[\![\ n \in I;\ infinite\ I\ ]\!] \Longrightarrow n < inext\ n\ I$
⟨*proof*⟩

lemma *inext-mono2-fin*: $[\![\ n \in I;\ finite\ I;\ n \neq Max\ I\ ]\!] \Longrightarrow n < inext\ n\ I$
⟨*proof*⟩

thm *inext-mono2*
lemma *inext-mono2-infin-fin*:
$[\![\ n \in I;\ n \neq Max\ I \vee infinite\ I\ ]\!] \Longrightarrow n < inext\ n\ I$
⟨*proof*⟩

thm *Nat.zero-less-Suc*
lemma *inext-neq-iMin*: $\exists\, x{\in}I.\ n < x \Longrightarrow inext\ n\ I \neq iMin\ I$
⟨*proof*⟩
thm *le-less-trans*[*OF iMin-le*[*of n*], *OF - inext-mono2*]
⟨*proof*⟩

lemma *inext-neq-iMin-infin*: $infinite\ I \Longrightarrow inext\ n\ I \neq iMin\ I$
⟨*proof*⟩
thm *infinite-nat-iff-unbounded*[*THEN iffD1*]
⟨*proof*⟩

thm *Max-le-Min-imp-singleton*
lemma *Max-le-iMin-imp-singleton*: $[\![\ finite\ I;\ I \neq \{\};\ Max\ I \leq iMin\ I\ ]\!] \Longrightarrow I = \{iMin\ I\}$
⟨*proof*⟩

lemma *inext-neq-iMin-not-singleton*:
$[\![\ I \neq \{\};\ \neg(\exists\, a.\ I = \{a\})\ ]\!] \Longrightarrow inext\ n\ I \neq iMin\ I$
⟨*proof*⟩
corollary *inext-neq-iMin-not-card-1*:
$[\![\ I \neq \{\};\ card\ I \neq Suc\ 0\ ]\!] \Longrightarrow inext\ n\ I \neq iMin\ I$
⟨*proof*⟩

lemma *inext-neq-imp-Max*: $n \neq inext\ n\ I \Longrightarrow n < Max\ I \vee infinite\ I$
⟨*proof*⟩

lemma *inext-less-conv*: $(n \in I \wedge (n < Max\ I \vee infinite\ I)) = (n < inext\ n\ I)$
⟨*proof*⟩

lemma *inext-min-step*: $[\![\ n < k;\ k < inext\ n\ I\ ]\!] \Longrightarrow k \notin I$
⟨*proof*⟩
thm *contrapos-pn*[*of* $k < inext\ n\ I\ k \in I$]
⟨*proof*⟩
thm *not-less-iMin*
thm *not-less-iMin*[*of* $k\ \{x \in I.\ n < x\}$]
⟨*proof*⟩
corollary *inext-min-step2*: $\neg(\exists\, k{\in}I.\ n < k \wedge k < inext\ n\ I)$
⟨*proof*⟩

lemma *min-step-inext*[*rule-format*]:
$[\![\ x < y;\ x \in I;\ y \in I;\ \bigwedge k.\ [\![\ x < k;\ k < y\ ]\!] \Longrightarrow k \notin I\ ]\!] \Longrightarrow$
$inext\ x\ I = y$
⟨*proof*⟩
thm *nat-neq-iff*
⟨*proof*⟩
thm *inext-closed*[*of* $x\ I$]
thm *inext-mono2*[*of* $x\ I$]
⟨*proof*⟩
thm *inext-min-step*[*of* $x\ y\ I$]
⟨*proof*⟩

corollary *min-step-inext2*[*rule-format*]:
$[\![\ x < y;\ x \in I;\ y \in I;\ \neg(\exists\, k \in I.\ x < k \wedge k < y)\ ]\!] \Longrightarrow$
$inext\ x\ I = y$
⟨*proof*⟩
lemma *between-empty-imp-inext-eq*:
$[\![\ n \in A;\ n < inext\ n\ A;\ n \in B;\ inext\ n\ A \in B;\ B\ {\downarrow}{>}\ n\ {\downarrow}{<}\ (inext\ n\ A) = \{\}\ ]\!]$
$\Longrightarrow$
$inext\ n\ B = inext\ n\ A$
⟨*proof*⟩

lemma *inext-le-mono*: $[\![\ a \leq b;\ a \in I;\ b \in I\ ]\!] \Longrightarrow inext\ a\ I \leq inext\ b\ I$
⟨*proof*⟩
thm *inext-min-step*
⟨*proof*⟩

thm *inext-mono2*
lemma *inext-less-mono*:
$[\![\ a < b;\ a \in I;\ b \in I;\ \exists\, x{\in}I.\ b < x\ ]\!] \Longrightarrow inext\ a\ I < inext\ b\ I$
⟨*proof*⟩
thm *inext-min-step*
⟨*proof*⟩

thm *inext-mono2-fin*
lemma *inext-less-mono-fin*:

$[\![a < b; a \in I; b \in I; \mathit{finite}\ I; b \neq \mathit{Max}\ I]\!] \Longrightarrow \mathit{inext}\ a\ I < \mathit{inext}\ b\ I$
thm *inext-less-mono Max-in*
⟨*proof*⟩

thm *inext-mono2-infin*
lemma *inext-less-mono-infin*:
$[\![a < b; a \in I; b \in I; \mathit{infinite}\ I]\!] \Longrightarrow \mathit{inext}\ a\ I < \mathit{inext}\ b\ I$
⟨*proof*⟩
thm *inext-mono2-infin-fin*
lemma *inext-less-mono-infin-fin*:
$[\![a < b; a \in I; b \in I; b \neq \mathit{Max}\ I \vee \mathit{infinite}\ I]\!] \Longrightarrow \mathit{inext}\ a\ I < \mathit{inext}\ b\ I$
⟨*proof*⟩

lemma *inext-le-mono-rev*:
$[\![\mathit{inext}\ a\ I \leq \mathit{inext}\ b\ I; a \in I; b \in I; \exists\, x{\in}I.\ \mathit{inext}\ a\ I < x]\!] \Longrightarrow a \leq b$
⟨*proof*⟩
thm *inext-less-mono*
⟨*proof*⟩
lemma *inext-le-mono-fin-rev*:
$[\![\mathit{inext}\ a\ I \leq \mathit{inext}\ b\ I; a \in I; b \in I; \mathit{finite}\ I; \mathit{inext}\ a\ I \neq \mathit{Max}\ I]\!] \Longrightarrow a \leq b$
⟨*proof*⟩
lemma *inext-le-mono-infin-rev*:
$[\![\mathit{inext}\ a\ I \leq \mathit{inext}\ b\ I; a \in I; b \in I; \mathit{infinite}\ I]\!] \Longrightarrow a \leq b$
⟨*proof*⟩
lemma *inext-le-mono-infin-fin-rev*:
$[\![\mathit{inext}\ a\ I \leq \mathit{inext}\ b\ I; a \in I; b \in I; \mathit{inext}\ a\ I \neq \mathit{Max}\ I \vee \mathit{infinite}\ I]\!] \Longrightarrow a \leq b$
⟨*proof*⟩

lemma *inext-less-mono-rev*:
$[\![\mathit{inext}\ a\ I < \mathit{inext}\ b\ I; a \in I; b \in I]\!] \Longrightarrow a < b$
⟨*proof*⟩

lemma *less-imp-inext-le*: $[\![a < b; a \in I; b \in I]\!] \Longrightarrow \mathit{inext}\ a\ I \leq b$
⟨*proof*⟩

lemma *iprev-mono*: $\mathit{iprev}\ n\ I \leq n$
⟨*proof*⟩
corollary *iprev-neq-imp-greater*: $n \neq \mathit{iprev}\ n\ I \Longrightarrow \mathit{iprev}\ n\ I < n$
⟨*proof*⟩

lemma *iprev-mono2*: $[\![n \in I; \exists\, x{\in}I.\ x < n]\!] \Longrightarrow \mathit{iprev}\ n\ I < n$
⟨*proof*⟩
thm *finite-nat-iff-bounded*
⟨*proof*⟩

thm *inext-mono2-fin*

lemma *iprev-mono2-if-neq-iMin*: $[\![n \in I; iMin\ I \neq n]\!] \Longrightarrow iprev\ n\ I < n$
thm *iMinI*
thm *iprev-mono2*
⟨*proof*⟩

thm *inext-neq-iMin*
lemma *iprev-neq-Max*: $[\![finite\ I; \exists\, x{\in}I.\ x < n]\!] \Longrightarrow iprev\ n\ I \neq Max\ I$
⟨*proof*⟩
thm *less-le-trans*[*OF iprev-mono2 Max-ge*]
⟨*proof*⟩

thm *inext-neq-iMin-not-singleton*
lemma *iprev-neq-Max-not-singleton*:
$[\![finite\ I; I \neq \{\}; \neg(\exists\, a.\ I = \{a\})]\!] \Longrightarrow iprev\ n\ I \neq Max\ I$
⟨*proof*⟩
thm *not-in-iprev-fix*
⟨*proof*⟩
corollary *iprev-neq-Max-not-card-1*:
$[\![finite\ I; I \neq \{\}; card\ I \neq Suc\ 0]\!] \Longrightarrow iprev\ n\ I \neq Max\ I$
⟨*proof*⟩

lemma *iprev-neq-imp-iMin*: $iprev\ n\ I \neq n \Longrightarrow iMin\ I < n$
⟨*proof*⟩

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*: $[\![iprev\ n\ I < k; k < n]\!] \Longrightarrow k \notin I$
⟨*proof*⟩
thm *contrapos-pn*[*of iprev n I < k k ∈ I*]
⟨*proof*⟩
thm *Max-ge*[*of* $\{x \in I.\ x < n\}$ *k*]
⟨*proof*⟩

corollary *iprev-min-step2*: $\neg(\exists\, x{\in}I.\ iprev\ n\ I < x \wedge x < n)$
⟨*proof*⟩

lemma *min-step-iprev*:
$[\![\ x < y;\ x \in I;\ y \in I;\ \bigwedge k.\ [\![\ x < k;\ k < y\ ]\!] \Longrightarrow k \notin I\ ]\!] \Longrightarrow$
$iprev\ y\ I = x$
thm *ccontr*
⟨*proof*⟩
thm *nat-neq-iff*
⟨*proof*⟩
thm *iprev-min-step*
⟨*proof*⟩
thm *iprev-closed*
thm *iprev-mono2*
⟨*proof*⟩
corollary *min-step-iprev2*[*rule-format*]:
$[\![\ x < y;\ x \in I;\ y \in I;\ \neg(\exists\, k \in I.\ x < k \wedge k < y)\ ]\!] \Longrightarrow$
$iprev\ y\ I = x$
⟨*proof*⟩
lemma *between-empty-imp-iprev-eq*:
$[\![\ n \in A;\ iprev\ n\ A < n;\ n \in B;\ iprev\ n\ A \in B;\ B \downarrow> (iprev\ n\ A) \downarrow< n = \{\}\ ]\!]$
$\Longrightarrow$
$iprev\ n\ B = iprev\ n\ A$
⟨*proof*⟩

lemma *iprev-le-mono*: $[\![\ a \leq b;\ a \in I;\ b \in I\ ]\!] \Longrightarrow iprev\ a\ I \leq iprev\ b\ I$
⟨*proof*⟩
thm *iprev-min-step*
⟨*proof*⟩
lemma *iprev-less-mono*:
$[\![\ a < b;\ a \in I;\ b \in I;\ \exists\, x{\in}I.\ x < a\ ]\!] \Longrightarrow iprev\ a\ I < iprev\ b\ I$
⟨*proof*⟩
thm *iprev-min-step*
⟨*proof*⟩

lemma *iprev-less-mono-if-neq-iMin*:
$[\![\ a < b;\ a \in I;\ b \in I;\ iMin\ I \neq a\ ]\!] \Longrightarrow iprev\ a\ I < iprev\ b\ I$
⟨*proof*⟩

thm *inext-le-mono-rev*
lemma *iprev-le-mono-rev*:
$[\![\ iprev\ a\ I \leq iprev\ b\ I;\ a \in I;\ b \in I;\ iMin\ I \neq iprev\ b\ I\ ]\!] \Longrightarrow a \leq b$
⟨*proof*⟩

thm *inext-less-mono-rev*
lemma *iprev-less-mono-rev*:
$[\![\ iprev\ a\ I < iprev\ b\ I;\ a \in I;\ b \in I\ ]\!] \Longrightarrow a < b$
⟨*proof*⟩

lemma *set-restriction-inext-eq*:
⟦ *set-restriction interval-fun*; $n \in$ *interval-fun I*; *inext n I* $\in$ *interval-fun I* ⟧ ⟹

inext n (*interval-fun I*) = *inext n I*
⟨*proof*⟩
thm *inext-fix-iff*
thm *inext-fix-iff*[*THEN iffD1*]
⟨*proof*⟩
thm *inext-fix-iff*
⟨*proof*⟩
thm *inext-mono*
thm *le-neq-implies-less*[*OF inext-mono*, *OF not-sym*]
⟨*proof*⟩
thm *not-ex-in-conv*
⟨*proof*⟩

thm *set-restriction-inext-eq*
lemma *set-restriction-inext-singleton-eq*:
⟦ *set-restriction interval-fun*; $n \in$ *interval-fun I*; *inext n I* $\in$ *interval-fun I* ⟧ ⟹

{*inext n* (*interval-fun I*)} = *interval-fun* {*inext n I*}
⟨*proof*⟩

lemma *inext-iprev*: *iMin I* $\neq n \Longrightarrow$ *inext* (*iprev n I*) *I* = *n*
⟨*proof*⟩
thm *iMinI iprev-min-step*
thm *min-step-inext iprev-mono2 iprev-closed*
⟨*proof*⟩

lemma *iprev-inext-infin*: *infinite I* ⟹ *iprev* (*inext n I*) *I* = *n*
⟨*proof*⟩

lemma *iprev-inext-fin*:
⟦ *finite I*; $n \neq$ *Max I* ⟧ ⟹ *iprev* (*inext n I*) *I* = *n*
⟨*proof*⟩

lemma *iprev-inext*:
$n \neq$ *Max I* $\vee$ *infinite I* ⟹ *iprev* (*inext n I*) *I* = *n*
⟨*proof*⟩

lemma *inext-eq-infin*:
⟦ *inext a I* = *inext b I*; *infinite I* ⟧ ⟹ *a* = *b*
thm *arg-cong*[**where** *f*=λ*x*. *iprev x I*]
⟨*proof*⟩

lemma *inext-eq-fin*:
⟦ *inext a I* = *inext b I*; *finite I*; *a* ≠ *Max I*; *b* ≠ *Max I* ⟧ ⟹ *a* = *b*
⟨*proof*⟩
thm *inext-mono2-infin-fin*
lemma *inext-eq-infin-fin*:
⟦ *inext a I* = *inext b I*; *a* ≠ *Max I* ∧ *b* ≠ *Max I* ∨ *infinite I* ⟧ ⟹ *a* = *b*
thm *inext-eq-fin inext-eq-infin*
⟨*proof*⟩
lemma *inext-eq*:
⟦ *inext a I* = *inext b I*; ∃ *x*∈*I*. *a* < *x*; ∃ *x*∈*I*. *b* < *x* ⟧ ⟹ *a* = *b*
⟨*proof*⟩

lemma *iprev-eq-if-neq-iMin*:
⟦ *iprev a I* = *iprev b I*; *iMin I* ≠ *a*; *iMin I* ≠ *b* ⟧ ⟹ *a* = *b*
⟨*proof*⟩
lemma *iprev-eq*:
⟦ *iprev a I* = *iprev b I*; ∃ *x*∈*I*. *x* < *a*; ∃ *x*∈*I*. *x* < *b* ⟧ ⟹ *a* = *b*
⟨*proof*⟩

lemma *greater-imp-iprev-ge*: ⟦ *b* < *a*; *a* ∈ *I*; *b* ∈ *I* ⟧ ⟹ *b* ≤ *iprev a I*
⟨*proof*⟩

lemma *inext-cut-less-conv*: *inext n I* < *t* ⟹ *inext n* (*I* ↓< *t*) = *inext n I*
thm *le-less-trans*[*OF inext-mono*]
⟨*proof*⟩
thm *i-cut-commute-disj*[*of op* ↓< *op* ↓>, *simplified*]
⟨*proof*⟩
lemma *inext-cut-le-conv*: *inext n I* ≤ *t* ⟹ *inext n* (*I* ↓≤ *t*) = *inext n I*
thm *nat-cut-le-less-conv inext-cut-less-conv*
⟨*proof*⟩

lemma *inext-cut-greater-conv*: *t* < *n* ⟹ *inext n* (*I* ↓> *t*) = *inext n I*
⟨*proof*⟩
thm *cut-greater-mem-iff*[*THEN iffD2*, *OF conjI*]
⟨*proof*⟩
thm *i-cut-commute-disj*[*of op* ↓> *op* ↓>, *simplified*]
thm *cut-cut-greater*
⟨*proof*⟩
lemma *inext-cut-ge-conv*: *t* ≤ *n* ⟹ *inext n* (*I* ↓≥ *t*) = *inext n I*
⟨*proof*⟩
thm *nat-cut-greater-ge-conv*[*symmetric*]
⟨*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*: $t < iprev\ n\ I \Longrightarrow iprev\ n\ (I \downarrow> t) = iprev\ n\ I$
thm *less-le-trans*[*OF - iprev-mono*]
⟨*proof*⟩
thm *i-cut-commute-disj*[*of op* $\downarrow>$ *op* $\downarrow<$, *simplified*]
⟨*proof*⟩
lemma *iprev-cut-ge-conv*: $t \leq iprev\ n\ I \Longrightarrow iprev\ n\ (I \downarrow\geq t) = iprev\ n\ I$
⟨*proof*⟩
thm *nat-cut-greater-ge-conv*
⟨*proof*⟩
lemma *iprev-cut-less-conv*: $n < t \Longrightarrow iprev\ n\ (I \downarrow< t) = iprev\ n\ I$
⟨*proof*⟩
thm *cut-less-mem-iff*[*THEN iffD2*, *OF conjI*]
⟨*proof*⟩
thm *i-cut-commute-disj*[*of op* $\downarrow<$ *op* $\downarrow<$, *simplified*]
⟨*proof*⟩
lemma *iprev-cut-le-conv*: $n \leq t \Longrightarrow iprev\ n\ (I \downarrow\leq t) = iprev\ n\ I$
thm *nat-cut-le-less-conv iprev-cut-less-conv*
⟨*proof*⟩

lemmas *iprev-cut-conv* =
iprev-cut-less-conv iprev-cut-le-conv
iprev-cut-greater-conv iprev-cut-ge-conv
thm
inext-cut-conv
iprev-cut-conv

thm *inext-cut-less-conv*
lemma *inext-cut-less-fix*: $t \leq inext\ n\ I \Longrightarrow inext\ n\ (I \downarrow< t) = n$
⟨*proof*⟩
thm *contra-subsetD*[*OF cut-less-subset*]
⟨*proof*⟩
thm *inext-min-step*
⟨*proof*⟩
thm *inext-Max nat-cut-less-finite*
⟨*proof*⟩
lemma *inext-cut-le-fix*: $t < inext\ n\ I \Longrightarrow inext\ n\ (I \downarrow\leq t) = n$
thm *nat-cut-le-less-conv*
⟨*proof*⟩

lemma *iprev-cut-greater-fix*: $iprev\ n\ I \leq t \Longrightarrow iprev\ n\ (I \downarrow> t) = n$
⟨*proof*⟩
thm *contra-subsetD*[*OF cut-greater-subset*]

⟨*proof*⟩
thm *iprev-iMin*
⟨*proof*⟩
lemma *iprev-cut-ge-fix*: *iprev n I* < *t* ⟹ *iprev n* (*I* ↓≥ *t*) = *n*
⟨*proof*⟩
thm *nat-cut-greater-ge-conv*[*symmetric*] *iprev-cut-greater-fix*
⟨*proof*⟩

definition
CommuteWithIntervalCut4 :: ((′*a*::*linorder*) *set* ⇒ ′*a set*) ⇒ *bool*
where
CommuteWithIntervalCut4 fun ≡
∀ *t fun2 I*.
(*fun2* = (λ*I*. *I* ↓< *t*) ∨ *fun2* = (λ*I*. *I* ↓≤ *t*) ∨ *fun2* = (λ*I*. *I* ↓> *t*) ∨ *fun2* = (λ*I*. *I* ↓≥ *t*)) ⟶
fun (*fun2 I*) = *fun2* (*fun I*)
definition *CommuteWithIntervalCut2* :: ((′*a*::*linorder*) *set* ⇒ ′*a set*) ⇒ *bool*
where
CommuteWithIntervalCut2 fun ≡
∀ *t fun2 I*.
(*fun2* = (λ*I*. *I* ↓< *t*) ∨ *fun2* = (λ*I*. *I* ↓> *t*)) ⟶
fun (*fun2 I*) = *fun2* (*fun I*)

lemma *CommuteWithIntervalCut4-imp-2*: *CommuteWithIntervalCut4 fun* ⟹ *CommuteWithIntervalCut2 fun*
⟨*proof*⟩

lemma *nat-CommuteWithIntervalCut2-4-eq*:
CommuteWithIntervalCut4 (*fun*::*nat set* ⇒ *nat set*) = *CommuteWithIntervalCut2 fun*
⟨*proof*⟩

lemma
cut-less-CommuteWithIntervalCut4: *CommuteWithIntervalCut4* (λ*I*. *I* ↓< *t*)
and
cut-le-CommuteWithIntervalCut4: *CommuteWithIntervalCut4* (λ*I*. *I* ↓≤ *t*)
and
cut-greater-CommuteWithIntervalCut4: *CommuteWithIntervalCut4* (λ*I*. *I* ↓> *t*)
and
cut-ge-CommuteWithIntervalCut4: *CommuteWithIntervalCut4* (λ*I*. *I* ↓≥ *t*)
thm *i-cut-commute-disj*
⟨*proof*⟩
lemmas *i-cut-CommuteWithIntervalCut4* =
cut-less-CommuteWithIntervalCut4 cut-le-CommuteWithIntervalCut4
cut-greater-CommuteWithIntervalCut4 cut-ge-CommuteWithIntervalCut4

thm *cut-greater-image*
lemma *inext-image*:
  ⟦ $n \in I$; *strict-mono-on f I* ⟧ ⟹ *inext* (*f n*) (*f* ‘ *I*) = *f* (*inext n I*)
⟨*proof*⟩
 thm *inext-mono2*
 ⟨*proof*⟩
 thm *cut-greater-not-empty-iff*[*THEN iffD2*]
 ⟨*proof*⟩
 thm *cut-greater-image*
 ⟨*proof*⟩
 thm *iMin-mono-on2*[*OF strict-mono-on-imp-mono-on*]
 thm *strict-mono-on-subset*
 ⟨*proof*⟩
thm *inext-all-le-fix*
⟨*proof*⟩

lemma *iprev-image*:
  ⟦ $n \in I$; *strict-mono-on f I* ⟧ ⟹ *iprev* (*f n*) (*f* ‘ *I*) = *f* (*iprev n I*)
⟨*proof*⟩
 thm *iprev-mono2*
 ⟨*proof*⟩
 thm *cut-less-not-empty-iff*[*THEN iffD2*]
 ⟨*proof*⟩
 thm *cut-less-image*
 ⟨*proof*⟩
 thm *Max-mono-on2*[*OF strict-mono-on-imp-mono-on*]
 thm *strict-mono-on-subset*
 thm *nat-cut-less-finite*
 ⟨*proof*⟩
thm *inext-all-le-fix*
⟨*proof*⟩

lemma *inext-image2*:
  strict-mono f ⟹ *inext* (*f n*) (*f* ‘ *I*) = *f* (*inext n I*)
⟨*proof*⟩
thm *inj-image-mem-iff strict-mono-imp-inj*
⟨*proof*⟩
lemma *iprev-image2*:
  strict-mono f ⟹ *iprev* (*f n*) (*f* ‘ *I*) = *f* (*iprev n I*)
⟨*proof*⟩

lemma *inext-imirror-iprev-conv*:

⟦ *finite I*; *n* ≤ *iMin I* + *Max I* ⟧ ⟹
inext (*mirror-elem n I*) (*imirror I*) = *mirror-elem* (*iprev n I*) *I*
⟨*proof*⟩
thm *imirror-mem-conv*
⟨*proof*⟩
thm *cut-less-Min-empty*
thm *mirror-elem-Min*
⟨*proof*⟩
thm *imirror-Max*
⟨*proof*⟩
thm *cut-greater-Max-empty*[*OF - order-refl*]
⟨*proof*⟩
thm *imirror-cut-greater'*
⟨*proof*⟩
thm *imirror-bounds-iMin nat-cut-less-finite*
⟨*proof*⟩
thm *cut-less-Min-not-empty*
⟨*proof*⟩
corollary *inext-imirror-iprev-conv'*:
⟦ *finite I*; *n* ∈ *I* ⟧ ⟹
inext (*mirror-elem n I*) (*imirror I*) = *mirror-elem* (*iprev n I*) *I*
thm *inext-imirror-iprev-conv*[*OF - trans-le-add2*[*OF Max-ge*]]
⟨*proof*⟩

lemma *iprev-imirror-inext-conv*:
⟦ *finite I*; *n* ≤ *iMin I* + *Max I* ⟧ ⟹
iprev (*mirror-elem n I*) (*imirror I*) = *mirror-elem* (*inext n I*) *I*
⟨*proof*⟩
thm *imirror-mem-conv*
⟨*proof*⟩
thm *cut-greater-Max-empty*
thm *mirror-elem-Max*
⟨*proof*⟩
thm *imirror-iMin*
⟨*proof*⟩
thm *cut-less-Min-empty*[*OF order-refl*]
⟨*proof*⟩
thm *Max-ge*[*of I n*]
⟨*proof*⟩
thm *imirror-cut-less*
⟨*proof*⟩
thm *imirror-bounds-Max cut-greater-finite cut-greater-Max-eq*
thm *imirror-bounds-Max*[*OF cut-greater-finite*]
thm *Max-le-iff*
⟨*proof*⟩
thm *cut-greater-Max-not-empty*[*of I n*]
⟨*proof*⟩
corollary *iprev-imirror-inext-conv'*:
⟦ *finite I*; *n* ∈ *I* ⟧ ⟹

iprev (*mirror-elem n I*) (*imirror I*) = *mirror-elem* (*inext n I*) *I*
⟨*proof*⟩

thm
inext-imirror-iprev-conv
inext-imirror-iprev-conv'
iprev-imirror-inext-conv
iprev-imirror-inext-conv'

lemma *inext-insert-ge-Max*:
⟦ *finite I*; *I* ≠ {}; *Max I* ≤ *a* ⟧ ⟹ *inext* (*Max I*) (*insert a I*) = *a*
⟨*proof*⟩
thm *le-neq-trans*
⟨*proof*⟩

lemma *iprev-insert-le-iMin*:
⟦ *finite I*; *I* ≠ {}; *a* ≤ *iMin I* ⟧ ⟹ *iprev* (*iMin I*) (*insert a I*) = *a*
⟨*proof*⟩
thm *le-neq-trans*
⟨*proof*⟩

thm *cut-le-less-conv*
lemma *cut-less-le-iprev-conv*:
⟦ *t* ∈ *I*; *t* ≠ *iMin I* ⟧ ⟹ *I* ↓< *t* = *I* ↓≤ (*iprev t I*)
⟨*proof*⟩
thm *Max-ge-iff nat-cut-less-finite*
⟨*proof*⟩

lemma *neq-Max-imp-inext-neq-iMin*:
⟦ *t* ∈ *I*; *t* ≠ *Max I* ∨ *infinite I* ⟧ ⟹ *inext t I* ≠ *iMin I*
⟨*proof*⟩
corollary *neq-Max-imp-inext-gr-iMin*:
⟦ *t* ∈ *I*; *t* ≠ *Max I* ∨ *infinite I*⟧ ⟹ *iMin I* < *inext t I*
⟨*proof*⟩
thm *inext-closed*
⟨*proof*⟩

lemma *cut-le-less-inext-conv*:
⟦ *t* ∈ *I*; *t* ≠ *Max I* ∨ *infinite I*⟧ ⟹ *I* ↓≤ *t* = *I* ↓< (*inext t I*)
thm *cut-less-le-iprev-conv*[*of inext t I I*]
⟨*proof*⟩
thm *iprev-inext*[*of t I*]
⟨*proof*⟩
lemma *cut-ge-greater-iprev-conv*:
⟦ *t* ∈ *I*; *t* ≠ *iMin I* ⟧ ⟹ *I* ↓≥ *t* = *I* ↓> (*iprev t I*)
⟨*proof*⟩
lemma *cut-greater-ge-inext-conv*:

⟦ $t \in I$; $t \neq Max\ I \vee infinite\ I$ ⟧ ⟹ $I \downarrow> t = I \downarrow\geq (inext\ t\ I)$
thm *cut-ge-greater-iprev-conv*[*of inext t I I*]
⟨*proof*⟩
thm *iprev-inext*[*of t I*]
⟨*proof*⟩

thm
cut-less-le-iprev-conv
cut-le-less-inext-conv
cut-ge-greater-iprev-conv
cut-greater-ge-inext-conv

lemma *inext-append*:
⟦ *finite A*; $A \neq \{\}$; $B \neq \{\}$; $Max\ A < iMin\ B$ ⟧ ⟹
inext n $(A \cup B)$ = (*if* $n \in B$ *then inext n B else* (*if* $n = Max\ A$ *then iMin B else inext n A*))
⟨*proof*⟩
corollary *inext-append-eq1*:
⟦ *finite A*; $A \neq \{\}$; $B \neq \{\}$; $Max\ A < iMin\ B$; $n \in A$; $n \neq Max\ A$ ⟧ ⟹
inext n $(A \cup B) = inext\ n\ A$
⟨*proof*⟩
corollary *inext-append-eq2*:
⟦ *finite A*; $A \neq \{\}$; $B \neq \{\}$; $Max\ A < iMin\ B$; $n \in B$ ⟧ ⟹
inext n $(A \cup B) = inext\ n\ B$
⟨*proof*⟩
corollary *inext-append-eq3*:
⟦ *finite A*; $A \neq \{\}$; $B \neq \{\}$; $Max\ A < iMin\ B$ ⟧ ⟹
inext $(Max\ A)$ $(A \cup B) = iMin\ B$
⟨*proof*⟩

lemma *iprev-append*: ⟦ *finite A*; $A \neq \{\}$; $B \neq \{\}$; $Max\ A < iMin\ B$ ⟧ ⟹
iprev n $(A \cup B)$ = (*if* $n \in A$ *then iprev n A else* (*if* $n = iMin\ B$ *then Max A else iprev n B*))
⟨*proof*⟩
thm *Max-in*[*OF nat-cut-less-finite*, *THEN cut-less-in-imp*]
⟨*proof*⟩

corollary *iprev-append-eq1*:
⟦ *finite A*; $A \neq \{\}$; $B \neq \{\}$; $Max\ A < iMin\ B$; $n \in A$ ⟧ ⟹
iprev n $(A \cup B) = iprev\ n\ A$
⟨*proof*⟩
corollary *iprev-append-eq2*:
⟦ *finite A*; $A \neq \{\}$; $B \neq \{\}$; $Max\ A < iMin\ B$; $n \in B$; $n \neq iMin\ B$ ⟧ ⟹

iprev n (A ∪ B) = iprev n B
⟨*proof*⟩
corollary *iprev-append-eq3*:
⟦ finite A; A ≠ {}; B ≠ {}; Max A < iMin B ⟧ ⟹
iprev (iMin B) (A ∪ B) = Max A
⟨*proof*⟩

lemma *inext-predicate-change-exists-aux*: *⋀a.*
⟦ c = card (I ↓≥ a ↓< b); a < b; a ∈ I; b ∈ I; ¬ P a; P b ⟧ ⟹
∃ n ∈ (I ↓≥ a ↓< b). ¬ P n ∧ P (inext n I)
⟨*proof*⟩
thm *less-imp-inext-le*[*of - b I*]
⟨*proof*⟩
thm *Max-gr-iff*[*OF - in-imp-not-empty*]
⟨*proof*⟩

lemma *inext-predicate-change-exists*:
⟦ a ≤ b; a ∈ I; b ∈ I; ¬ P a; P b ⟧ ⟹
∃ n∈I. a ≤ n ∧ n < b ∧ ¬ P n ∧ P (inext n I)
⟨*proof*⟩
thm *inext-predicate-change-exists-aux*[*OF refl*]
⟨*proof*⟩

lemma *iprev-predicate-change-exists*:
⟦ a ≤ b; a ∈ I; b ∈ I; ¬ P b; P a ⟧ ⟹
∃ n∈I. a < n ∧ n ≤ b ∧ ¬ P n ∧ P (iprev n I)
⟨*proof*⟩

corollary *nat-Suc-predicate-change-exists*:
⟦ a ≤ b; ¬ P a; P b ⟧ ⟹ ∃ n≥a. n < b ∧ ¬ P n ∧ P (Suc n)
⟨*proof*⟩

corollary *nat-pred-predicate-change-exists*:
⟦ a ≤ b; ¬ P b; P a ⟧ ⟹ ∃ n≤b. a < n ∧ ¬ P n ∧ P (n − Suc 0)
⟨*proof*⟩

lemma *inext-predicate-change-exists2-all*:
⟦ (a::nat) ≤ b; a ∈ I; b ∈ I; ¬ P a; ∀ k ∈ I ↓≥ b. P k ⟧ ⟹
∃ n∈I. a ≤ n ∧ n < b ∧ ¬ P n ∧ (∀ k ∈ I ↓> n. P k)
⟨*proof*⟩
thm *inext-predicate-change-exists*[*of a b I λn. if (n = a) then P n else (∀ k∈I↓≥n. P k)*]
⟨*proof*⟩
thm *cut-greater-ge-inext-conv*

⟨*proof*⟩
thm *inext-neq-imp-less*
⟨*proof*⟩
thm *cut-greater-ge-conv-if*
⟨*proof*⟩

corollary *inext-predicate-change-exists2*:
⟦ (*a*::*nat*) ≤ *b*; *a* ∈ *I*; *b* ∈ *I*; ¬ *P a*; *P b* ⟧ ⟹
∃ *n*∈*I*. *a* ≤ *n* ∧ *n* < *b* ∧ ¬ *P n* ∧ (∀ *k*∈*I*. *n* < *k* ∧ *k* ≤ *b* ⟶ *P k*)
thm *inext-predicate-change-exists2-all*[*of a b I* ↓≤ *b*]
⟨*proof*⟩

corollary *nat-Suc-predicate-change-exists2-all*:
⟦ (*a*::*nat*) ≤ *b*; ¬ *P a*; ∀ *k*≥*b*. *P k* ⟧ ⟹
∃ *n*≥*a*. *n* < *b* ∧ ¬ *P n* ∧ (∀ *k*>*n*. *P k*)
⟨*proof*⟩
corollary *nat-Suc-predicate-change-exists2*:
⟦ (*a*::*nat*) ≤ *b*; ¬ *P a*; *P b* ⟧ ⟹
∃ *n*≥*a*. *n* < *b* ∧ ¬ *P n* ∧ (∀ *k*≤*b*. *n* < *k* ⟶ *P k*)
thm *inext-predicate-change-exists2*[*of a b UNIV*]
⟨*proof*⟩

thm *inext-predicate-change-exists2-all*
lemma *iprev-predicate-change-exists2-all*:
⟦ (*a*::*nat*) ≤ *b*; *a* ∈ *I*; *b* ∈ *I*; ¬ *P b*; ∀ *k*∈*I*↓≤*a*. *P k* ⟧ ⟹
∃ *n*∈*I*. *a* < *n* ∧ *n* ≤ *b* ∧ ¬ *P n* ∧ (∀ *k*∈*I*↓<*n*. *P k*)
⟨*proof*⟩
thm *iprev-predicate-change-exists*[*of a b I* λ*n*. *if* (*n* = *b*) *then P n else* (∀ *k*∈*I*↓≤*n*. *P k*)]
⟨*proof*⟩
thm *cut-less-le-iprev-conv*[*symmetric*]
⟨*proof*⟩

thm *inext-predicate-change-exists2*
corollary *iprev-predicate-change-exists2*:
⟦ (*a*::*nat*) ≤ *b*; *a* ∈ *I*; *b* ∈ *I*; ¬ *P b*; *P a* ⟧ ⟹
∃ *n*∈*I*. *a* < *n* ∧ *n* ≤ *b* ∧ ¬ *P n* ∧ (∀ *k*∈*I*. *a* ≤ *k* ∧ *k* < *n* ⟶ *P k*)
thm *iprev-predicate-change-exists2-all*[*of a b I* ↓≥ *a*]
⟨*proof*⟩

thm *nat-Suc-predicate-change-exists2-all*
corollary *nat-pred-predicate-change-exists2-all*:
⟦ (*a*::*nat*) ≤ *b*; ¬ *P b*; ∀ *k*≤*a*. *P k* ⟧ ⟹
∃ *n*>*a*. *n* ≤ *b* ∧ ¬ *P n* ∧ (∀ *k*<*n*. *P k*)
⟨*proof*⟩

thm *nat-Suc-predicate-change-exists2*

corollary *nat-pred-predicate-change-exists2*:
$[\![(a::nat) \leq b; \neg P\ b; P\ a]\!] \Longrightarrow$
$\exists n{>}a.\ n \leq b \wedge \neg P\ n \wedge (\forall k{\geq}a.\ k < n \longrightarrow P\ k)$
thm *iprev-predicate-change-exists2*[*of a b UNIV*]
⟨*proof*⟩

8.2 *inext-nth* and *iprev-nth* – nth element of a natural set

term *inext*
primrec
inext-nth :: *nat set* ⇒ *nat* ⇒ *nat*
where
inext-nth I 0 = *iMin I*
| *inext-nth I* (*Suc n*) = *inext* (*inext-nth I n*) *I*
notation (*xsymbols*)
inext-nth ((- → -) [*100*, *100*] *60*)
notation (*HTML* **output**)
inext-nth ((- → -) [*100*, *100*] *60*)

term $(I \to a) + b$
term $(I \to n) \in I$
term $(A \to n) + (B \to n)$

lemma *inext-nth-closed*: $I \neq \{\} \Longrightarrow I \to n \in I$
⟨*proof*⟩

thm *inext-image*
lemma *inext-nth-image*:
$[\![I \neq \{\}; \textit{strict-mono-on}\ f\ I]\!] \Longrightarrow (f\ `\ I) \to n = f\ (I \to n)$
⟨*proof*⟩

thm *inext-mono2*
lemma *inext-nth-Suc-mono*: $I \to n \leq I \to Suc\ n$
⟨*proof*⟩

lemma *inext-nth-mono*: $a \leq b \Longrightarrow I \to a \leq I \to b$
⟨*proof*⟩

thm *inext-mono2*
lemma *inext-nth-Suc-mono2*: $\exists x{\in}I.\ I \to n < x \Longrightarrow I \to n < I \to Suc\ n$
⟨*proof*⟩

lemma *inext-nth-mono2*: $\exists x{\in}I.\ I \to a < x \Longrightarrow (I \to a < I \to b) = (a < b)$
⟨*proof*⟩

lemma *inext-nth-mono2-infin*:
infinite I ⟹ $(I \to a < I \to b) = (a < b)$

⟨*proof*⟩

lemma *inext-nth-Max-fix*:
⟦ *finite I*; $I \neq \{\}$; $I \rightarrow a = Max\ I$; $a \leq b$ ⟧ ⟹ $I \rightarrow b = Max\ I$
⟨*proof*⟩

thm *inext-cut-less-conv*
lemma *inext-nth-cut-less-conv*:
$\bigwedge I.\ I \rightarrow n < t \Longrightarrow (I \downarrow< t) \rightarrow n = I \rightarrow n$
⟨*proof*⟩
thm *order-le-less-trans*[*OF inext-mono*]
⟨*proof*⟩
thm *inext-cut-less-conv*
⟨*proof*⟩
thm
inext-cut-less-conv
inext-nth-cut-less-conv

lemma *remove-Min-inext-nth-Suc-conv*: $\bigwedge I.$
Suc 0 < *card I* ∨ *infinite I* ⟹
$(I - \{iMin\ I\}) \rightarrow n = I \rightarrow Suc\ n$

⟨*proof*⟩
thm *card-gr0-imp-not-empty*[*OF gr-implies-gr0*]
⟨*proof*⟩
thm *cut-greater-Min-eq-Diff*[*symmetric*]
⟨*proof*⟩
thm *ssubst*[*OF inext-def*[*THEN meta-eq-to-obj-eq*], *rule-format*]
⟨*proof*⟩
thm *inext-neq-iMin-not-card-1*
thm *inext-neq-iMin-infin*
⟨*proof*⟩
thm *iMin-le*[*OF inext-nth-closed*, *rule-format*]
⟨*proof*⟩

corollary *remove-Min-inext-nth-Suc-conv-finite*: *Suc 0* < *card I* ⟹ $(I - \{iMin\ I\}) \rightarrow n = I \rightarrow Suc\ n$
⟨*proof*⟩
corollary *remove-Min-inext-nth-Suc-conv-infinite*: *infinite I* ⟹ $(I - \{iMin\ I\}) \rightarrow n = I \rightarrow Suc\ n$
⟨*proof*⟩

lemma *remove-Max-eq*: ⟦ *finite I*; $I \neq \{\}$; $n \neq Max\ I$ ⟧ ⟹ $Max\ (I - \{n\}) = Max\ I$
⟨*proof*⟩
lemma *remove-iMin-eq*: ⟦ $I \neq \{\}$; $n \neq iMin\ I$ ⟧ ⟹ $iMin\ (I - \{n\}) = iMin\ I$

⟨*proof*⟩
lemma *remove-Min-eq*: ⟦ *finite I*; $I \neq \{\}$; $n \neq Min\ I$ ⟧ ⟹ $Min\ (I - \{n\}) = Min\ I$
⟨*proof*⟩
lemma *Max-le-iMin-conv-singleton*: ⟦ *finite I*; $I \neq \{\}$ ⟧ ⟹ $(Max\ I \leq iMin\ I) = (\exists x.\ I = \{x\})$
⟨*proof*⟩

lemma *inext-nth-card-less-Max*:
⋀I. $Suc\ n < card\ I \Longrightarrow I \rightarrow n < Max\ I$
thm *card-gr0-imp-not-empty*[*OF less-trans*[*OF zero-less-Suc*]]
⟨*proof*⟩
thm *Max-le-Min-conv-singleton*
⟨*proof*⟩
thm *remove-Min-inext-nth-Suc-conv*
⟨*proof*⟩
thm *Max-le-iMin-conv-singleton*[*THEN iffD1*]
⟨*proof*⟩
thm *remove-Max-eq*
⟨*proof*⟩
thm *inext-nth-card-less-Max*
lemma *inext-nth-card-less-Max'*:
$n < card\ I - Suc\ 0 \Longrightarrow I \rightarrow n < Max\ I$
⟨*proof*⟩

lemma *inext-nth-card-Max-aux*:
⋀I. $card\ I = Suc\ n \Longrightarrow I \rightarrow n = Max\ I$
thm *card-gr0-imp-not-empty*[*OF less-le-trans*[*OF zero-less-Suc*, *OF eq-imp-le*[*OF sym*]]]
⟨*proof*⟩
thm *card-gr0-imp-not-empty*[*OF less-le-trans*[*OF zero-less-Suc*, *OF eq-imp-le*[*OF sym*]], *rule-format*]
⟨*proof*⟩
thm *inext-nth-card-less-Max*
⟨*proof*⟩
lemma *inext-nth-card-Max-aux'*:
⋀I. ⟦ *finite I*; $I \neq \{\}$ ⟧ ⟹ $I \rightarrow (card\ I - Suc\ 0) = Max\ I$
⟨*proof*⟩

thm
inext-nth-card-Max-aux
inext-nth-card-Max-aux'
thm
inext-nth-card-less-Max
inext-nth-Max-fix
lemma *inext-nth-card-Max*:
⟦ *finite I*; $I \neq \{\}$; $card\ I \leq Suc\ n$ ⟧ ⟹ $I \rightarrow n = Max\ I$

thm *inext-nth-Max-fix*[*of - card I − Suc 0*]
⟨*proof*⟩
lemma *inext-nth-card-Max'*:
⟦ *finite I*; *I* ≠ {}; *card I − Suc 0* ≤ *n* ⟧ ⟹ *I* → *n* = *Max I*
⟨*proof*⟩

thm
inext-nth-card-less-Max
inext-nth-card-Max
inext-nth-card-Max'

lemma *inext-nth-singleton*: {*a*} → *n* = *a*
thm *inext-nth-Max-fix*[*OF singleton-finite singleton-not-empty - le0*]
⟨*proof*⟩

lemma *inext-nth-eq-Min-conv*:
I ≠ {} ⟹ (*I* → *n* = *iMin I*) = (*n* = *0* ∨ (∃ *a*. *I* = {*a*}))
⟨*proof*⟩

lemma *inext-nth-gr-Min-conv*:
I ≠ {} ⟹ (*iMin I* < *I* → *n*) = (*0* < *n* ∧ ¬(∃ *a*. *I* = {*a*}))
⟨*proof*⟩

lemma *inext-nth-gr-Min-conv-infinite*:
infinite I ⟹ (*iMin I* < *I* → *n*) = (*0* < *n*)
⟨*proof*⟩

lemma *inext-nth-cut-ge-inext-nth*: ⋀*I b*.
I ≠ {} ⟹ *I* ↓≥ (*I* → *a*) → *b* = *I* → (*a* + *b*)
⟨*proof*⟩
thm *remove-Min-inext-nth-Suc-conv*
⟨*proof*⟩
thm *remove-Min-inext-nth-Suc-conv*
⟨*proof*⟩

thm *inext-append-eq1*
lemma *inext-nth-append-eq1*:
⟦ *finite A*; *A* ≠ {}; *Max A* < *iMin B*; *A* → *n* ≠ *Max A* ⟧ ⟹
(*A* ∪ *B*) → *n* = *A* → *n*
⟨*proof*⟩
thm *iMin-le-Max*
⟨*proof*⟩
thm *Max-ge*[*OF - inext-nth-closed*]
⟨*proof*⟩
thm *order-le-less-trans*[*OF inext-mono*]
⟨*proof*⟩

thm *inext-append-eq1*
⟨*proof*⟩

thm *inext-nth-append-eq1*
thm *inext-append-eq2*
lemma *inext-nth-card-append-eq1*:
⋀*A B*.⟦ *Max A* < *iMin B*; *n* < *card A* ⟧ ⟹
(*A* ∪ *B*) → *n* = *A* → *n*
⟨*proof*⟩
thm *card-gr0-imp-finite*[*OF le-less-trans*[*OF le0*]]
⟨*proof*⟩
thm *inext-nth-card-Max*[*OF - - eq-imp-le*[*OF sym*]]
⟨*proof*⟩
thm *inext-nth-append-eq1*
⟨*proof*⟩
thm *inext-nth-card-Max*
⟨*proof*⟩

thm *inext-append-eq3*
lemma *inext-nth-card-append-eq3*:
⟦ *finite A*; *B* ≠ {}; *Max A* < *iMin B* ⟧ ⟹
(*A* ∪ *B*) → (*card A*) = *iMin B*
⟨*proof*⟩
thm *subst*[*OF Suc-pred*]
⟨*proof*⟩
thm *inext-nth-card-append-eq1 inext-nth-card-Max′ inext-append-eq3*
⟨*proof*⟩

lemma *inext-nth-card-append-eq2*:
⟦ *finite A*; *A* ≠ {}; *B* ≠ {}; *Max A* < *iMin B*; *card A* ≤ *n* ⟧ ⟹
(*A* ∪ *B*) → *n* = *B* → (*n* − *card A*)
thm *inext-nth-cut-ge-inext-nth*
⟨*proof*⟩
thm *inext-nth-cut-ge-inext-nth*[*symmetric*]
⟨*proof*⟩

thm *inext-append*
lemma *inext-nth-card-append*:
⟦ *finite A*; *A* ≠ {}; *B* ≠ {}; *Max A* < *iMin B* ⟧ ⟹
(*A* ∪ *B*) → *n* = (*if n* < *card A then A* → *n else B* → (*n* − *card A*))
⟨*proof*⟩

lemma *inext-nth-insert-Suc*:
⟦ *I* ≠ {}; *a* < *iMin I* ⟧ ⟹ (*insert a I*) → *Suc n* = *I* → *n*

⟨*proof*⟩
thm *remove-Min-inext-nth-Suc-conv*
⟨*proof*⟩

lemma *inext-nth-cut-less-eq*:
$n < card\ (I\ \downarrow<\ t) \Longrightarrow (I\ \downarrow<\ t) \rightarrow n = I \rightarrow n$
⟨*proof*⟩
thm *inext-nth-card-append-eq1*
⟨*proof*⟩

lemma *less-card-cut-less-imp-inext-nth-less*:
$n < card\ (I\ \downarrow<\ t) \Longrightarrow I \rightarrow n < t$
⟨*proof*⟩

lemma *inext-nth-less-less-card-conv*:
$I\ \downarrow\geq\ t \neq \{\} \Longrightarrow (I \rightarrow n < t) = (n < card\ (I\ \downarrow<\ t))$
⟨*proof*⟩
thm *cut-less-empty-iff inext-nth-closed*
⟨*proof*⟩
thm *ssubst*[*OF cut-less-cut-ge-ident*[*OF order-refl*], *of* $\lambda x.\ x \rightarrow n < t$ - *t*]
⟨*proof*⟩
thm *inext-nth-card-append-eq2*[*OF nat-cut-less-finite*, *of I t I* $\downarrow\geq$ *t n*]
⟨*proof*⟩

lemma *cut-less-inext-nth-card-eq1*:
$n < card\ I \vee infinite\ I \Longrightarrow card\ (I\ \downarrow<\ (I \rightarrow n)) = n$
⟨*proof*⟩
thm *inext-nth-card-less-Max*[*THEN less-imp-neq*]
⟨*proof*⟩

lemma *cut-less-inext-nth-card-eq2*:
$[\![\ finite\ I;\ card\ I \leq Suc\ n\ ]\!] \Longrightarrow card\ (I\ \downarrow<\ (I \rightarrow n)) = card\ I - Suc\ 0$
⟨*proof*⟩

lemma *cut-less-inext-nth-card-if*:
$card\ (I\ \downarrow<\ (I \rightarrow n)) = ($
$if\ (n < card\ I \vee infinite\ I)\ then\ n\ else\ card\ I - Suc\ 0)$
⟨*proof*⟩

lemma *cut-le-inext-nth-card-eq1*:
$n < card\ I \vee infinite\ I \Longrightarrow card\ (I\ \downarrow\leq\ (I \rightarrow n)) = Suc\ n$
⟨*proof*⟩
thm *cut-le-less-inext-conv*[*OF inext-nth-closed*]
⟨*proof*⟩
lemma *cut-le-inext-nth-card-eq2*:
$[\![\ finite\ I;\ card\ I \leq Suc\ n\ ]\!] \Longrightarrow card\ (I\ \downarrow\leq\ (I \rightarrow n)) = card\ I$
⟨*proof*⟩

lemma *cut-le-inext-nth-card-if*:
card $(I \downarrow\leq (I \rightarrow n)) = ($
if $(n < \mathit{card}\ I \vee \mathit{infinite}\ I)$ *then Suc n else card I*)
⟨*proof*⟩

term *iprev*
primrec
iprev-nth :: *nat set* ⇒ *nat* ⇒ *nat*
where
iprev-nth I 0 = *Max I*
| *iprev-nth I* (*Suc n*) = *iprev* (*iprev-nth I n*) *I*
thm *inext-iprev*
notation (*xsymbols*)
iprev-nth ((- ← -) [*100*, *100*] *60*)
notation (*HTML* **output**)
iprev-nth ((- ← -) [*100*, *100*] *60*)

lemma *iprev-nth-closed*: ⟦ *finite I*; $I \neq \{\}$ ⟧ ⟹ $I \leftarrow n \in I$
⟨*proof*⟩

thm *iprev-image*
lemma *iprev-nth-image*:
⟦ *finite I*; $I \neq \{\}$; *strict-mono-on f I* ⟧ ⟹ $(f\ `\ I) \leftarrow n = f\ (I \leftarrow n)$
⟨*proof*⟩

thm *iprev-mono*
lemma *iprev-nth-Suc-mono*: $I \leftarrow (\mathit{Suc}\ n) \leq I \leftarrow n$
⟨*proof*⟩
lemma *iprev-nth-mono*: $a \leq b \Longrightarrow I \leftarrow b \leq I \leftarrow a$
⟨*proof*⟩
lemma *iprev-nth-Suc-mono2*:
⟦ *finite I*; $\exists x{\in}I.\ x < I \leftarrow n$ ⟧ ⟹ $I \leftarrow (\mathit{Suc}\ n) < I \leftarrow n$
thm *iprev-mono2*
⟨*proof*⟩
thm *iprev-nth-closed*
⟨*proof*⟩

lemma *iprev-nth-mono2*:
⟦ *finite I*; $\exists x{\in}I.\ x < I \leftarrow a$ ⟧ ⟹ $(I \leftarrow b < I \leftarrow a) = (a < b)$
⟨*proof*⟩

lemma *iprev-nth-iMin-fix*:
⟦ $I \neq \{\}$; $I \leftarrow a = \mathit{iMin}\ I$; $a \leq b$ ⟧ ⟹ $I \leftarrow b = \mathit{iMin}\ I$
⟨*proof*⟩

lemma *iprev-nth-singleton*: $\{a\} \leftarrow n{=}\ a$

thm *iprev-nth-iMin-fix*[*OF singleton-not-empty - le0*]
⟨*proof*⟩

8.3 Induction over arbitrary natural sets using the functions *inext* and *iprev*

lemma *inext-nth-surj-aux1*:
$\{x \in I.\ \neg(\exists\, n.\ I \to n = x)\} = \{\}$
(**is** *?S* = {}
is $\{\ x \in I.\ ?P\ x\} = \{\}$)
⟨*proof*⟩
thm *iMinI-ex2*
⟨*proof*⟩
thm *iprev-closed*[*of iMin S i*]
⟨*proof*⟩
thm *inext-iprev*[*of I iMin S*]
⟨*proof*⟩
thm *iprev-mono2*[*of iMin S I*]
⟨*proof*⟩
thm *not-less-iMin*
⟨*proof*⟩
thm *s-not-ex*[*of prev*]
⟨*proof*⟩
thm *iMin-subset*
⟨*proof*⟩

term *inext-nth*
term $\lambda n.\ I \to n$
lemma *inext-nth-surj-on*:*surj-on* $(\lambda n.\ I \to n)$ *UNIV I*
⟨*proof*⟩
thm *inext-nth-surj-aux1*[*of I*]
⟨*proof*⟩
corollary *in-imp-ex-inext-nth*: $x \in I \Longrightarrow \exists\, n.\ x = I \to n$
thm *surj-onD*
thm *surj-onD*[**where** *A*=*UNIV*, *simplified*]
⟨*proof*⟩

lemma *inext-induct*:
$[\![\ P\ (iMin\ I);\ \bigwedge n.\ [\![\ n \in I;\ P\ n\ ]\!] \Longrightarrow P\ (inext\ n\ I);\ n \in I\ ]\!] \Longrightarrow P\ n$
thm *image-nat-induct*
thm *image-nat-induct*[**where** *P*=*P* **and** *f*=$\lambda n.\ I \to n$ **and** *I*=*I* **and** *a*=*n*]
⟨*proof*⟩
thm *inext-nth-closed*[*OF in-imp-not-empty*]
thm *inext-nth-surj-on*
⟨*proof*⟩
thm *inext-induct*

lemma *iprev-nth-surj-aux1*:

finite I $\Longrightarrow$ $\{ x \in I.\ \neg(\exists n.\ I \leftarrow n = x)\} = \{\}$
⟨*proof*⟩
thm *Max-in*
⟨*proof*⟩
thm *inext-closed*[*of Max S I*]
⟨*proof*⟩
thm *iprev-inext*[*of Max S I*]
⟨*proof*⟩
thm *inext-mono2*[*of Max S I*]
⟨*proof*⟩
thm *Max-ge*[*of S next'*]
⟨*proof*⟩
thm *s-not-ex*[*of next'*]
⟨*proof*⟩
thm *Max-subset*[*of S I*]
⟨*proof*⟩

term *iprev-nth*
term $\lambda n.$ *iprev-nth I n*
lemma *iprev-nth-surj-on*: *finite I* $\Longrightarrow$ *surj-on* $(\lambda n.\ I \leftarrow n)$ *UNIV I*
⟨*proof*⟩
thm *iprev-nth-surj-aux1*[*of I*]
⟨*proof*⟩
corollary *in-imp-ex-iprev-nth*:
$[\![$ *finite I*; $x \in I$ $]\!] \Longrightarrow \exists n.\ x = I \leftarrow n$
thm *surj-onD*
thm *surj-onD*[*of - UNIV I*, *simplified*]
⟨*proof*⟩

lemma *iprev-induct*:
$[\![$ *P* (*Max I*); $\bigwedge n.$ $[\![$ $n \in I$; *P n* $]\!] \Longrightarrow$ *P* (*iprev n I*); *finite I*; $n \in I$ $]\!] \Longrightarrow$ *P n*
thm *image-nat-induct*
thm *image-nat-induct*[**where** *P*=*P* **and** *f*=$\lambda n.$ *I* $\leftarrow$ *n* **and** *I*=*I* **and** *a*=*n*]
⟨*proof*⟩
thm *iprev-nth-closed*[*OF - in-imp-not-empty*]
⟨*proof*⟩
thm
inext-induct
iprev-induct

8.4 Natural intervals with *inext* and *iprev*

lemma *inext-atLeast*: $n \leq t \Longrightarrow$ *inext t* $\{n..\} =$ *Suc t*
⟨*proof*⟩

lemma *iprev-atLeast'*: $n \leq t \Longrightarrow$ *iprev* (*Suc t*) $\{n..\} = t$
⟨*proof*⟩
lemma *iprev-atLeast*: $n < t \Longrightarrow$ *iprev t* $\{n..\} = t -$ *Suc 0*
⟨*proof*⟩

lemma *inext-atMost*: $t < n \Longrightarrow inext\ t\ \{..n\} = Suc\ t$
⟨*proof*⟩
lemma *iprev-atMost*: $t \leq n \Longrightarrow iprev\ t\ \{..n\} = t - Suc\ 0$
⟨*proof*⟩
thm *subst*[*OF iMin-atMost*[*of n*]]
⟨*proof*⟩

lemma *inext-lessThan*: $Suc\ t < n \Longrightarrow inext\ t\ \{..<n\} = Suc\ t$
⟨*proof*⟩
lemma *iprev-lessThan*: $t < n \Longrightarrow iprev\ t\ \{..<n\} = t - Suc\ 0$
⟨*proof*⟩

lemma *inext-atLeastAtMost*: $[\![\ m \leq t;\ t < n\ ]\!] \Longrightarrow inext\ t\ \{m..n\} = Suc\ t$
⟨*proof*⟩
lemma *iprev-atLeastAtMost*: $[\![\ m < t;\ t \leq n\ ]\!] \Longrightarrow iprev\ t\ \{m..n\} = t - Suc\ 0$
⟨*proof*⟩
lemma *iprev-atLeastAtMost'*: $[\![\ m \leq t;\ t < n\ ]\!] \Longrightarrow iprev\ (Suc\ t)\ \{m..n\} = t$
⟨*proof*⟩

lemma *inext-nth-atLeast* : $\{n..\} \rightarrow a = n + a$
⟨*proof*⟩
lemma *inext-nth-atLeastAtMost*: $[\![\ a \leq n - m;\ m \leq n\ ]\!] \Longrightarrow \{m..n\} \rightarrow a = m + a$
⟨*proof*⟩
lemma *iprev-nth-atLeastAtMost*: $[\![\ a \leq n - m;\ m \leq n\ ]\!] \Longrightarrow \{m..n\} \leftarrow a = n - a$
⟨*proof*⟩
lemma *inext-nth-atMost*: $a \leq n \Longrightarrow \{..n\} \rightarrow a = a$
⟨*proof*⟩
lemma *iprev-nth-atMost*: $a \leq n \Longrightarrow \{..n\} \leftarrow a = n - a$
⟨*proof*⟩

lemma *inext-nth-lessThan* : $a < n \Longrightarrow \{..<n\} \rightarrow a = a$
⟨*proof*⟩
lemma *iprev-nth-lessThan*: $a < n \Longrightarrow \{..<n\} \leftarrow a = n - Suc\ a$
⟨*proof*⟩

lemma *inext-nth-UNIV*: $UNIV \rightarrow a = a$
⟨*proof*⟩

8.5 Further result for *inext-nth* and *iprev-nth*

thm *inext-iprev*
lemma *inext-iprev-nth-Suc*:
$iMin\ I \neq I \leftarrow n \Longrightarrow inext\ (I \leftarrow Suc\ n)\ I = I \leftarrow n$
⟨*proof*⟩
lemma *inext-iprev-nth-pred*:
$[\![\ finite\ I;\ iMin\ I \neq I \leftarrow (n - Suc\ 0)\ ]\!] \Longrightarrow$

inext $(I \leftarrow n)\ I = I \leftarrow (n - Suc\ 0)$
⟨*proof*⟩

lemma *iprev-inext-nth-Suc*:
$I \rightarrow n \neq Max\ I \vee infinite\ I \Longrightarrow iprev\ (I \rightarrow Suc\ n)\ I = I \rightarrow n$
⟨*proof*⟩
lemma *iprev-inext-nth-pred*:
$I \rightarrow (n - Suc\ 0) \neq Max\ I \vee infinite\ I \Longrightarrow$
$iprev\ (I \rightarrow n)\ I = I \rightarrow (n - Suc\ 0)$
⟨*proof*⟩

thm *inext-imirror-iprev-conv*
lemma *inext-nth-imirror-iprev-nth-conv*:
$[\![\ finite\ I;\ I \neq \{\}\ ]\!] \Longrightarrow$
$(imirror\ I) \rightarrow n = mirror\text{-}elem\ (I \leftarrow n)\ I$
⟨*proof*⟩
corollary *inext-nth-imirror-iprev-nth-conv2*:
$[\![\ finite\ I;\ I \neq \{\}\ ]\!] \Longrightarrow$
$mirror\text{-}elem\ ((imirror\ I) \leftarrow n)\ I = I \rightarrow n$
thm *inext-nth-imirror-iprev-nth-conv*[*OF imirror-finite imirror-not-empty*]
⟨*proof*⟩

lemma *iprev-nth-imirror-inext-nth-conv*:
$[\![\ finite\ I;\ I \neq \{\}\ ]\!] \Longrightarrow$
$(imirror\ I) \leftarrow n = mirror\text{-}elem\ (I \rightarrow n)\ I$
⟨*proof*⟩
corollary *iprev-nth-imirror-inext-nth-conv2*:
$[\![\ finite\ I;\ I \neq \{\}\ ]\!] \Longrightarrow$
$mirror\text{-}elem\ ((imirror\ I) \rightarrow n)\ I = (I \leftarrow n)$
thm *iprev-nth-imirror-inext-nth-conv*[*OF imirror-finite imirror-not-empty*]
⟨*proof*⟩

thm *inext-nth-card-less-Max*
lemma *iprev-nth-card-greater-iMin*: $Suc\ n < card\ I \Longrightarrow iMin\ I < I \leftarrow n$
⟨*proof*⟩
thm *subst*[*OF iprev-nth-imirror-inext-nth-conv2*]
⟨*proof*⟩
thm *subst*[*OF mirror-elem-Max*]
⟨*proof*⟩
thm *subst*[*OF mirror-elem-imirror*, *of I*]
⟨*proof*⟩
thm *mirror-elem-less-conv*
⟨*proof*⟩

thm *inext-nth-card-less-Max*
⟨*proof*⟩

lemma *iprev-nth-card-iMin*:
⟦ *finite I*; *I* ≠ {}; *card I* ≤ *Suc n* ⟧ ⟹ *I* ← *n* = *iMin I*
thm *subst*[*OF iprev-nth-imirror-inext-nth-conv2*]
⟨*proof*⟩
thm *subst*[*OF mirror-elem-Max*]
⟨*proof*⟩
thm *subst*[*OF mirror-elem-imirror*, *of I*]
⟨*proof*⟩
thm *subst*[*OF imirror-Max*]
⟨*proof*⟩
thm *mirror-elem-eq-conv'*
⟨*proof*⟩
lemma *iprev-nth-card-iMin'*:
⟦ *finite I*; *I* ≠ {}; *card I* − *Suc 0* ≤ *n* ⟧ ⟹ *I* ← *n* = *iMin I*
⟨*proof*⟩

end

9 List2: 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 (*xsymbols*)
f-take' :: *'a list* ⇒ *nat* ⇒ *'a list* (**infixl** ↓ *100*)
where
xs ↓ *n* ≡ *take n xs*
abbreviation (*xsymbols*)
f-drop' :: *'a list* ⇒ *nat* ⇒ *'a list* (**infixl** ↑ *100*)
where
xs ↑ *n* ≡ *drop n xs*
syntax (*HTML* **output**)
f-take' :: *'a list* ⇒ *nat* ⇒ *'a list* (**infixl** ↓ *100*)
f-drop' :: *'a list* ⇒ *nat* ⇒ *'a list* (**infixl** ↑ *100*)

term *xs* ↓ *n*
term *xs* ↑ *n*

thm *List.append-Cons*
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*:$n < length \ xs \Longrightarrow xs \neq []$
⟨*proof*⟩

lemma *length-ge-Suc-imp-not-empty*:$Suc \ n \leq length \ xs \Longrightarrow xs \neq []$
⟨*proof*⟩
thm *length-take*

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*:
$Suc \ n \leq length \ xs \Longrightarrow (tl \ xs) \ ! \ n = xs \ ! \ Suc \ n$
thm *hd-Cons-tl*[*OF length-ge-Suc-imp-not-empty*, *THEN subst*]
⟨*proof*⟩
corollary *nth-tl-eq-nth-Suc2*:
$n < length \ xs \Longrightarrow (tl \ xs) \ ! \ n = xs \ ! \ Suc \ n$
⟨*proof*⟩

lemma *hd-eq-first*: $xs \neq [] \Longrightarrow xs \ ! \ 0 = hd \ xs$
⟨*proof*⟩
corollary *take-first*:$xs \neq [] \Longrightarrow xs \downarrow \ (Suc \ 0) = [xs \ ! \ 0]$
⟨*proof*⟩
corollary *take-hd*:$xs \neq [] \Longrightarrow xs \downarrow \ (Suc \ 0) = [hd \ xs]$

⟨*proof*⟩

thm *last-conv-nth*
theorem *last-nth*: $xs \neq [] \Longrightarrow last\ xs = xs\ !\ (length\ xs - Suc\ 0)$
⟨*proof*⟩

lemma *last-take*: $n < length\ xs \Longrightarrow last\ (xs \downarrow Suc\ n) = xs\ !\ n$
⟨*proof*⟩
corollary *last-take2*:
$[\![\ 0 < n;\ n \leq length\ xs\ ]\!] \Longrightarrow last\ (xs \downarrow n) = xs\ !\ (n - Suc\ 0)$
thm *diff-Suc-less*[*THEN order-less-le-trans*]
⟨*proof*⟩
thm *last-take*[*of n − Suc 0 xs*]
⟨*proof*⟩

thm *List.nth-drop*
corollary *nth-0-drop*: $n \leq length\ xs \Longrightarrow (xs \uparrow n)\ !\ 0 = xs\ !\ n$
⟨*proof*⟩
corollary *hd-drop*: $n < length\ xs \Longrightarrow hd\ (xs \uparrow n) = xs\ !\ n$
⟨*proof*⟩

lemma *drop-eq-tl*: $xs \uparrow (Suc\ 0) = tl\ xs$
⟨*proof*⟩

lemma *drop-take-1*:
$n < length\ xs \Longrightarrow xs \uparrow n \downarrow (Suc\ 0) = [xs\ !\ n]$
thm *take-hd hd-drop*
⟨*proof*⟩

lemma *upt-append*: $m \leq n \Longrightarrow [0..<m]\ @\ [m..<n] = [0..<n]$
thm *upt-add-eq-append*[*of 0 m n − m*]
⟨*proof*⟩

thm *nth-append*
lemma *nth-append1*: $n < length\ xs \Longrightarrow (xs\ @\ ys)\ !\ n = xs\ !\ n$
⟨*proof*⟩
lemma *nth-append2*: $length\ xs \leq n \Longrightarrow (xs\ @\ ys)\ !\ n = ys\ !\ (n - length\ xs)$
⟨*proof*⟩

lemma *list-all-conv*: $list\text{-}all\ P\ xs = (\forall\, i{<}length\ xs.\ P\ (xs\ !\ i))$
⟨*proof*⟩

thm *Fun.fun-eq-iff*
lemma *expand-list-eq*:
$\bigwedge ys.\ (xs = ys) = (length\ xs = length\ ys \wedge (\forall\, i{<}length\ xs.\ xs\ !\ i = ys\ !\ i))$
⟨*proof*⟩
lemmas *list-eq-iff = expand-list-eq*

lemma *list-take-drop-imp-eq*:
$[\![xs \downarrow n = ys \downarrow n;\ xs \uparrow n = ys \uparrow n]\!] \Longrightarrow xs = ys$
⟨*proof*⟩

lemma *list-take-drop-eq-conv*:
$(xs = ys) = (\exists n.\ (xs \downarrow n = ys \downarrow n \wedge xs \uparrow n = ys \uparrow n))$
⟨*proof*⟩

lemma *list-take-eq-conv*: $(xs = ys) = (\forall n.\ xs \downarrow n = ys \downarrow n)$
⟨*proof*⟩

lemma *list-drop-eq-conv*: $(xs = ys) = (\forall n.\ xs \uparrow n = ys \uparrow n)$
⟨*proof*⟩

abbreviation (*xsymbols*)
replicate' :: *'a* ⇒ *nat* ⇒ *'a list* (-[*1000*,*65*])
where
$x^n \equiv replicate\ n\ x$

term *length* $x^{(a+b)}$

thm *List.replicate-Suc*
thm *List.replicate-app-Cons-same*
lemma *replicate-snoc*: $x^n\ @\ [x] = x^{Suc\ n}$
⟨*proof*⟩

thm *List.nth-replicate*
lemma *eq-replicate-conv*: $(\forall i{<}length\ xs.\ xs\ !\ i = m) = (xs = m^{length\ xs})$
⟨*proof*⟩

lemma *replicate-Cons-length*: $length\ (x\ \#\ a^n) = Suc\ n$
⟨*proof*⟩
lemma *replicate-pred-Cons-length*: $0 < n \Longrightarrow length\ (x\ \#\ a^{n\ -\ Suc\ 0}) = n$
⟨*proof*⟩

thm *replicate-add*
lemma *replicate-le-diff*: $m \leq n \Longrightarrow x^m\ @\ x^{n\ -\ m} = x^n$
⟨*proof*⟩
lemma *replicate-le-diff2*: $[\![k \leq m;\ m \leq n]\!] \Longrightarrow x^{m\ -\ k}\ @\ x^{n\ -\ m} = x^{n\ -\ k}$
⟨*proof*⟩

thm *list.induct*
lemma *append-constant-length-induct-aux*: $\bigwedge xs.$
$[\![length\ xs\ div\ k = n;\ \bigwedge ys.\ k = 0 \vee length\ ys < k \Longrightarrow P\ ys;$
$\bigwedge xs\ ys.\ [\![length\ xs = k;\ P\ ys]\!] \Longrightarrow P\ (xs\ @\ ys)]\!] \Longrightarrow P\ xs$
⟨*proof*⟩

lemma *append-constant-length-induct*:
⟦ ⋀*ys*. *k* = *0* ∨ *length ys* < *k* ⟹ *P ys*;
⋀*xs ys*. ⟦ *length xs* = *k*; *P ys* ⟧ ⟹ *P* (*xs* @ *ys*) ⟧ ⟹ *P xs*
⟨*proof*⟩

lemma *zip-swap*: *map* (λ(*y*,*x*). (*x*,*y*)) (*zip ys xs*) = (*zip xs ys*)
⟨*proof*⟩

lemma *zip-takeL*: (*zip xs ys*) ↓ *n* = *zip* (*xs* ↓ *n*) *ys*
⟨*proof*⟩

lemma *zip-takeR*: (*zip xs ys*) ↓ *n* = *zip xs* (*ys* ↓ *n*)
thm *zip-swap*[*of ys*]
⟨*proof*⟩

lemma *zip-take*: (*zip xs ys*) ↓ *n* = *zip* (*xs* ↓ *n*) (*ys* ↓ *n*)
⟨*proof*⟩

thm *nth-zip*
lemma *hd-zip*: ⟦ *xs* ≠ []; *ys* ≠ [] ⟧ ⟹ *hd* (*zip xs ys*) = (*hd xs*, *hd ys*)
⟨*proof*⟩

lemma *map-id*: *map id xs* = *xs*
⟨*proof*⟩

lemma *map-id-subst*: *P* (*map id xs*) ⟹ *P xs*
⟨*proof*⟩

lemma *map-one*: *map f* [*x*] = [*f x*]
⟨*proof*⟩

lemma *map-last*: *xs* ≠ [] ⟹ *last* (*map f xs*) = *f* (*last xs*)
⟨*proof*⟩

lemma *filter-list-all*: *list-all P xs* ⟹ *filter P xs* = *xs*
⟨*proof*⟩

lemma *filter-snoc*: *filter P* (*xs* @ [*x*]) = (*if P x then* (*filter P xs*) @ [*x*] *else filter P xs*)
⟨*proof*⟩

lemma *filter-filter-eq*: *list-all* (λ*x*. *P x* = *Q x*) *xs* ⟹ *filter P xs* = *filter Q xs*
⟨*proof*⟩

lemma *filter-nth*: ⋀*n*.
n < *length* (*filter P xs*) ⟹
(*filter P xs*) ! *n* =

xs ! (*LEAST k.*
k < *length xs* ∧
n < *card* {*i*. *i* ≤ *k* ∧ *i* < *length xs* ∧ *P* (*xs* ! *i*)})
⟨*proof*⟩
thm *filter-snoc*
⟨*proof*⟩
thm *Least-le-imp-le-disj*
⟨*proof*⟩
thm *nth-append1*
⟨*proof*⟩
thm *length-filter-conv-card*
⟨*proof*⟩

9.1.3 Ordered lists

fun
list-ord :: ('*a* ⇒ '*a* ⇒ *bool*) ⇒ ('*a*::*ord*) *list* ⇒ *bool*
where
list-ord ord (*x1* # *x2* # *xs*) = (*ord x1 x2* ∧ *list-ord ord* (*x2* # *xs*))
| *list-ord ord xs* = *True*

thm *list-ord.simps*
definition *list-asc* :: ('*a*::*ord*) *list* ⇒ *bool* **where**
list-asc xs ≡ *list-ord* (*op* ≤) *xs*
definition *list-strict-asc* :: ('*a*::*ord*) *list* ⇒ *bool* **where**
list-strict-asc xs ≡ *list-ord* (*op* <) *xs*
value *list-asc* [*1*::*nat*, *2*, *2*]
value *list-strict-asc* [*1*::*nat*, *2*, *2*]
definition *list-desc* :: ('*a*::*ord*) *list* ⇒ *bool* **where**
list-desc xs ≡ *list-ord* (*op* ≥) *xs*
definition *list-strict-desc* :: ('*a*::*ord*) *list* ⇒ *bool* **where**
list-strict-desc xs ≡ *list-ord* (*op* >) *xs*

lemma *list-ord-Nil*: *list-ord ord* []
⟨*proof*⟩
lemma *list-ord-one*: *list-ord ord* [*x*]
⟨*proof*⟩
lemma *list-ord-Cons*:
list-ord ord (*x* # *xs*) =
(*xs* = [] ∨ (*ord x* (*hd xs*) ∧ *list-ord ord xs*))
⟨*proof*⟩
lemma *list-ord-Cons-imp*: ⟦ *list-ord ord xs*; *ord x* (*hd xs*) ⟧ ⟹ *list-ord ord* (*x* # *xs*)
⟨*proof*⟩
lemma *list-ord-append*: ⋀*ys*.
list-ord ord (*xs* @ *ys*) =
(*list-ord ord xs* ∧
(*ys* = [] ∨ (*list-ord ord ys* ∧ (*xs* = [] ∨ *ord* (*last xs*) (*hd ys*)))))
⟨*proof*⟩

lemma *list-ord-snoc*:
list-ord ord (*xs* @ [*x*]) =
(*xs* = [] ∨ (*ord* (*last xs*) *x* ∧ *list-ord ord xs*))
⟨*proof*⟩

lemma *list-ord-all-conv*:
(*list-ord ord xs*) = (∀ *n* < *length xs* − *1*. *ord* (*xs* ! *n*) (*xs* ! *Suc n*))
⟨*proof*⟩

lemma *list-ord-imp*:
⟦ ⋀*x y*. *ord x y* ⟹ *ord'* *x y*; *list-ord ord xs* ⟧ ⟹
list-ord ord' xs
⟨*proof*⟩
corollary *list-strict-asc-imp-list-asc*:
list-strict-asc (*xs*::'*a*::*preorder list*) ⟹ *list-asc xs*
⟨*proof*⟩
corollary *list-strict-desc-imp-list-desc*:
list-strict-desc (*xs*::'*a*::*preorder list*) ⟹ *list-desc xs*
⟨*proof*⟩

lemma *list-ord-trans-imp*: ⋀*i*.
⟦ *transP ord*; *list-ord ord xs*; *j* < *length xs*; *i* < *j* ⟧ ⟹
ord (*xs* ! *i*) (*xs* ! *j*)
⟨*proof*⟩
thm *trans-def*
⟨*proof*⟩

lemma *list-ord-trans*:
transP ord ⟹
(*list-ord ord xs*) =
(∀ *j* < *length xs*. ∀ *i* < *j*. *ord* (*xs* ! *i*) (*xs* ! *j*))
⟨*proof*⟩

lemma *list-ord-trans-refl-le*:
⟦ *transP ord*; *reflP ord* ⟧ ⟹
(*list-ord ord xs*) =
(∀ *j* < *length xs*. ∀ *i* ≤ *j*. *ord* (*xs* ! *i*) (*xs* ! *j*))
⟨*proof*⟩

lemma *list-ord-trans-refl-le-imp*:
⟦ *transP ord*; ⋀*x y*. *ord x y* ⟹ *ord'* *x y*; *reflP ord'*;
list-ord ord xs ⟧ ⟹
(∀ *j* < *length xs*. ∀ *i* ≤ *j*. *ord'* (*xs* ! *i*) (*xs* ! *j*))
⟨*proof*⟩
thm *list-ord-trans-imp*
⟨*proof*⟩

corollary
list-asc-trans:

(*list-asc* (*xs*::'*a*::*preorder list*)) =
(∀ *j* < *length xs*. ∀ *i* < *j*. *xs* ! *i* ≤ *xs* ! *j*) **and**
list-strict-asc-trans:
(*list-strict-asc* (*xs*::'*a*::*preorder list*)) =
(∀ *j* < *length xs*. ∀ *i* < *j*. *xs* ! *i* < *xs* ! *j*) **and**
list-desc-trans:
(*list-desc* (*xs*::'*a*::*preorder list*)) =
(∀ *j* < *length xs*. ∀ *i* < *j*. *xs* ! *j* ≤ *xs* ! *i*) **and**
list-strict-desc-trans:
(*list-strict-desc* (*xs*::'*a*::*preorder list*)) =
(∀ *j* < *length xs*. ∀ *i* < *j*. *xs* ! *j* < *xs* ! *i*)
⟨*proof*⟩

corollary
list-asc-trans-le:
(*list-asc* (*xs*::'*a*::*preorder list*)) =
(∀ *j* < *length xs*. ∀ *i* ≤ *j*. *xs* ! *i* ≤ *xs* ! *j*) **and**
list-desc-trans-le:
(*list-desc* (*xs*::'*a*::*preorder list*)) =
(∀ *j* < *length xs*. ∀ *i* ≤ *j*. *xs* ! *j* ≤ *xs* ! *i*)
⟨*proof*⟩

corollary
list-strict-asc-trans-le:
(*list-strict-asc* (*xs*::'*a*::*preorder list*)) ⟹
(∀ *j* < *length xs*. ∀ *i* ≤ *j*. *xs* ! *i* ≤ *xs* ! *j*)
⟨*proof*⟩
thm *list-ord-trans-refl-le-imp*
⟨*proof*⟩
thm *list-ord-imp*
⟨*proof*⟩

lemma *list-ord-le-sorted-eq*: *list-asc xs* = *sorted xs*
⟨*proof*⟩
thm *list-asc-trans*
⟨*proof*⟩
corollary *list-asc-upto*: *list-asc* [*m*..*n*]
⟨*proof*⟩

lemma *list-strict-asc-upt*: *list-strict-asc* [*m*..<*n*]
⟨*proof*⟩
thm *list-strict-asc-imp-list-asc*[*OF list-strict-asc-upt*]

lemma *list-ord-distinct-aux*:
⟦ *irrefl* {(*a*, *b*). *ord a b*}; *transP ord*; *list-ord ord xs*;
i < *length xs*; *j* < *length xs*; *i* < *j* ⟧ ⟹
xs ! *i* ≠ *xs* ! *j*
⟨*proof*⟩

thm *list-ord-trans*
⟨*proof*⟩
lemma *list-ord-distinct*:
⟦ *irrefl* {(*a*,*b*). *ord a b*}; *transP ord*; *list-ord ord xs* ⟧ ⟹
distinct xs
thm *distinct-conv-nth*
⟨*proof*⟩
thm *list-ord-distinct-aux*
⟨*proof*⟩
thm *list-ord-distinct-aux*[*THEN not-sym*]
⟨*proof*⟩

lemma *list-strict-asc-distinct*: *list-strict-asc* (*xs*::'*a*::*preorder list*) ⟹ *distinct xs*
⟨*proof*⟩
lemma *list-strict-desc-distinct*: *list-strict-desc* (*xs*::'*a*::*preorder list*) ⟹ *distinct xs*
⟨*proof*⟩

9.1.4 Additional definitions and results for sublists

primrec
sublist-list :: '*a list* ⇒ *nat list* ⇒ '*a list*
where
sublist-list xs [] = []
| *sublist-list xs* (*y* # *ys*) = (*xs* ! *y*) # (*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 *sublist-list* [*0*::*int*,*10*::*int*,*20*,*30*,*40*,*50*] [*1*::*nat*,*1*,*2*,*3*,*10*]

thm *sublist-def*
term *map fst* (*filter* (λ*p*. *snd p* ∈ *A*) (*zip xs* [*0*..<*length xs*]))
term *map fst* ([*p*←(*zip xs* [*0*..<*length xs*]). (*snd p* ∈ *A*)])

lemma *sublist-list-length*: *length* (*sublist-list xs ys*) = *length ys*
⟨*proof*⟩

lemma *sublist-list-append*:
⋀*zs*. *sublist-list xs* (*ys* @ *zs*) = *sublist-list xs ys* @ *sublist-list xs zs*
⟨*proof*⟩

lemma *sublist-list-Nil*: *sublist-list xs* [] =[]
⟨*proof*⟩

lemma *sublist-list-is-Nil-conv*:
(*sublist-list xs ys* = []) = (*ys* = [])
⟨*proof*⟩

lemma *sublist-list-eq-imp-length-eq*:

sublist-list xs ys = sublist-list xs zs ⟹ length ys = length zs
⟨*proof*⟩

lemma *sublist-list-nth*:
⋀n. n < length ys ⟹ sublist-list xs ys ! n = xs ! (ys ! n)
⟨*proof*⟩

lemma *take-drop-eq-sublist-list*:
m + n ≤ length xs ⟹ xs ↑ m ↓ n = sublist-list xs [m..<m+n]
⟨*proof*⟩
thm *sublist-list-nth*
⟨*proof*⟩

primrec
sublist-list-if :: 'a list ⇒ nat list ⇒ 'a list
where
sublist-list-if xs [] = []
| *sublist-list-if xs (y # ys) =*
(if y < length xs then (xs ! y) # (sublist-list-if xs ys)
else (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*: *⋀xs.*
sublist-list-if xs ys = sublist-list xs (filter (λi. i < length xs) ys)
⟨*proof*⟩
corollary *sublist-list-if-sublist-list-eq*: *⋀xs.*
list-all (λi. i < length xs) ys ⟹
sublist-list-if xs ys = sublist-list xs ys
⟨*proof*⟩
corollary *sublist-list-if-sublist-list-eq2*: *⋀xs.*
∀ n<length ys. ys ! n < length xs ⟹
sublist-list-if xs ys = sublist-list xs ys
thm *list-all-conv[THEN iffD2]*
⟨*proof*⟩

lemma *sublist-list-if-Nil-left*: *sublist-list-if [] ys = []*
⟨*proof*⟩
lemma *sublist-list-if-Nil-right*: *sublist-list-if xs [] = []*
⟨*proof*⟩

lemma *sublist-list-if-length*:
length (sublist-list-if xs ys) = length (filter (λi. i < length xs) ys)
⟨*proof*⟩
lemma *sublist-list-if-append*:

sublist-list-if xs (ys @ zs) = sublist-list-if xs ys @ sublist-list-if xs zs
⟨*proof*⟩
lemma *sublist-list-if-snoc*:
sublist-list-if xs (ys @ [y]) = sublist-list-if xs ys @ (if y < length xs then [xs ! y] else [])
⟨*proof*⟩

lemma *sublist-list-if-is-Nil-conv*:
(sublist-list-if xs ys = []) = (list-all (λi. length xs ≤ i) ys)
⟨*proof*⟩

lemma *sublist-list-if-nth*:
n < length ((filter (λi. i < length xs) ys)) ⟹
sublist-list-if xs ys ! n = xs ! ((filter (λi. i < length xs) ys) ! n)
⟨*proof*⟩

lemma *take-drop-eq-sublist-list-if*:
m + n ≤ length xs ⟹ xs ↑ m ↓ n = sublist-list-if xs [m..<m+n]
thm *take-drop-eq-sublist-list*
⟨*proof*⟩

lemma *sublist-empty-conv*: *(sublist xs I = []) = (∀ i∈I. length xs ≤ i)*
⟨*proof*⟩

thm *sublist-singleton*
lemma *sublist-singleton2*: *sublist xs {y} = (if y < length xs then [xs ! y] else [])*
⟨*proof*⟩

lemma *sublist-take-eq*:
⟦ finite I; Max I < n ⟧ ⟹ sublist (xs ↓ n) I = sublist xs I
⟨*proof*⟩
thm *append-take-drop-id*
⟨*proof*⟩

lemma *sublist-drop-eq*:
n ≤ iMin I ⟹ sublist (xs ↑ n) {j. j + n ∈ I} = sublist xs I
⟨*proof*⟩
thm *set-zip-rightD*
⟨*proof*⟩

lemma *sublist-cut-less-eq*:
length xs ≤ n ⟹ sublist xs (I ↓< n) = sublist xs I
⟨*proof*⟩
thm *filter-filter-eq*
⟨*proof*⟩

lemma *sublist-disjoint-Un*:
⟦ finite A; Max A < iMin B ⟧ ⟹ sublist xs (A ∪ B) = sublist xs A @ sublist xs

B
⟨*proof*⟩
thm *sublist-cut-less-eq*
⟨*proof*⟩
thm *sublist-append*
⟨*proof*⟩
thm *sublist-cut-less-eq*
⟨*proof*⟩
thm *sublist-take-eq*
⟨*proof*⟩
thm *sublist-drop-eq*
⟨*proof*⟩
corollary *sublist-disjoint-insert-left*:
⟦ *finite I*; *x* < *iMin I* ⟧ ⟹ *sublist xs* (*insert x I*) = *sublist xs* {*x*} @ *sublist xs I*
⟨*proof*⟩
corollary *sublist-disjoint-insert-right*:
⟦ *finite I*; *Max I* < *x* ⟧ ⟹ *sublist xs* (*insert x I*) = *sublist xs I* @ *sublist xs* {*x*}
⟨*proof*⟩

lemma *sublist-all*: {..<*length xs*} ⊆ *I* ⟹ *sublist xs I* = *xs*
⟨*proof*⟩
corollary *sublist-UNIV*: *sublist xs UNIV* = *xs*
⟨*proof*⟩

lemma *sublist-list-sublist-eq*: ⋀*xs*.
list-strict-asc ys ⟹ *sublist-list-if xs ys* = *sublist xs* (*set ys*)
⟨*proof*⟩
thm *list-ord-append*
⟨*proof*⟩
thm *list-strict-asc-trans*[*unfolded list-strict-asc-def*, *THEN iffD1*, *rule-format*]
⟨*proof*⟩
lemma *set-sublist-list-if*: ⋀*xs*. *set* (*sublist-list-if xs ys*) = {*xs* ! *i* |*i*. *i* < *length xs* ∧ *i* ∈ *set ys*}
⟨*proof*⟩

lemma *set-sublist-list*:
list-all (λ*i*. *i* < *length xs*) *ys* ⟹
set (*sublist-list xs ys*) = {*xs* ! *i* |*i*. *i* < *length xs* ∧ *i* ∈ *set ys*}
⟨*proof*⟩

lemma *set-sublist-list-if-eq-set-sublist*: *set* (*sublist-list-if xs ys*) = *set* (*sublist xs* (*set ys*))
⟨*proof*⟩
lemma *set-sublist-list-eq-set-sublist*:
list-all (λ*i*. *i* < *length xs*) *ys* ⟹
set (*sublist-list xs ys*) = *set* (*sublist xs* (*set ys*))
⟨*proof*⟩

9.1.5 Natural set images with lists

definition
f-image :: *'a list* ⇒ *nat set* ⇒ *'a set* (**infixr** *‘f* *90*)
where
xs ‘f A ≡ {*y*. ∃ *n*∈*A*. *n* < *length xs* ∧ *y* = *xs* ! *n* }

abbreviation
f-range :: *'a list* ⇒ *'a set*
where
f-range xs ≡ *f-image xs UNIV*

thm *Set.image-eqI*
lemma *f-image-eqI*[*simp*, *intro*]:
⟦ *x* = *xs* ! *n*; *n* ∈ *A*; *n* < *length xs* ⟧ ⟹ *x* ∈ *xs ‘f A*
⟨*proof*⟩

thm *Set.imageI*
lemma *f-imageI*: ⟦ *n* ∈ *A*; *n* < *length xs* ⟧ ⟹ *xs* ! *n* ∈ *xs ‘f A*
⟨*proof*⟩

thm *Set.rev-image-eqI*
lemma *rev-f-imageI*: ⟦ *n* ∈ *A*; *n* < *length xs*; *x* = *xs* ! *n* ⟧ ⟹ *x* ∈ *xs ‘f A*
⟨*proof*⟩

thm *Set.imageE*
lemma *f-imageE*[*elim*!]:
⟦ *x* ∈ *xs ‘f A*; ⋀*n*. ⟦ *x* = *xs* ! *n*; *n* ∈ *A*; *n* < *length xs* ⟧ ⟹ *P* ⟧ ⟹ *P*
⟨*proof*⟩

thm *Set.image-Un*
lemma *f-image-Un*: *xs ‘f* (*A* ∪ *B*) = *xs ‘f A* ∪ *xs ‘f B*
⟨*proof*⟩

thm *Set.image-mono*
lemma *f-image-mono*: *A* ⊆ *B* ==> *xs ‘f A* ⊆ *xs ‘f B*
⟨*proof*⟩

thm *Set.image-iff*
lemma *f-image-iff*: (*x* ∈ *xs ‘f A*) = (∃ *n*∈*A*. *n* < *length xs* ∧ *x* = *xs* ! *n*)
⟨*proof*⟩

thm *Set.image-subset-iff*
lemma *f-image-subset-iff*:
(*xs ‘f A* ⊆ *B*) = (∀ *n*∈*A*. *n* < *length xs* ⟶ *xs* ! *n* ∈ *B*)
⟨*proof*⟩
thm *Set.subset-image-iff*
lemma *subset-f-image-iff*: (*B* ⊆ *xs ‘f A*) = (∃ *A′*⊆*A*. *B* = *xs ‘f A′*)
⟨*proof*⟩

thm *image-subsetI*
lemma *f-image-subsetI*:
⟦ ⋀*n*. *n* ∈ *A* ∧ *n* < *length xs* ⟹ *xs* ! *n* ∈ *B* ⟧ ⟹ *xs* ‘*f* *A* ⊆ *B*
⟨*proof*⟩

thm *Set.image-empty*
lemma *f-image-empty*: *xs* ‘*f* {} = {}
⟨*proof*⟩

thm *Set.image-insert*
lemma *f-image-insert-if*:
xs ‘*f* (*insert n A*) = (
if n < *length xs then insert* (*xs* ! *n*) (*xs* ‘*f* *A*) *else* (*xs* ‘*f* *A*))
⟨*proof*⟩
lemma *f-image-insert-eq1*:
n < *length xs* ⟹ *xs* ‘*f* (*insert n A*) = *insert* (*xs* ! *n*) (*xs* ‘*f* *A*)
⟨*proof*⟩
lemma *f-image-insert-eq2*:
length xs ≤ *n* ⟹ *xs* ‘*f* (*insert n A*) = (*xs* ‘*f* *A*)
⟨*proof*⟩

thm *Set.insert-image*
lemma *insert-f-image*:
⟦ *n* ∈ *A*; *n* < *length xs* ⟧ ⟹ *insert* (*xs* ! *n*) (*xs* ‘*f* *A*) = (*xs* ‘*f* *A*)
⟨*proof*⟩
thm *Set.image-is-empty*
lemma *f-image-is-empty*: (*xs* ‘*f* *A* = {}) = ({*x*. *x* ∈ *A* ∧ *x* < *length xs*} = {})
⟨*proof*⟩

thm *Set.image-Collect*
lemma *f-image-Collect*: *xs* ‘*f* {*n*. *P n*} = {*xs* ! *n* |*n*. *P n* ∧ *n* < *length xs*}
⟨*proof*⟩

lemma *f-image-eq-set*: ∀ *n*<*length xs*. *n* ∈ *A* ⟹ *xs* ‘*f* *A* = *set xs*
⟨*proof*⟩
lemma *f-range-eq-set*: *f-range xs* = *set xs*
⟨*proof*⟩

lemma *f-image-eq-set-sublist*: *xs* ‘*f* *A* = *set* (*sublist xs A*)
⟨*proof*⟩
lemma *f-image-eq-set-sublist-list-if*: *xs* ‘*f* (*set ys*) = *set* (*sublist-list-if xs ys*)
⟨*proof*⟩
lemma *f-image-eq-set-sublist-list*:
list-all (λ*i*. *i* < *length xs*) *ys* ⟹ *xs* ‘*f* (*set ys*) = *set* (*sublist-list xs ys*)
⟨*proof*⟩

thm *Set.range-eqI*
lemma *f-range-eqI*: $[\![\ x = xs\ !\ n;\ n < length\ xs\ ]\!] \Longrightarrow x \in f\text{-}range\ xs$
⟨*proof*⟩
thm *Set.rangeI*
lemma *f-rangeI*: $n < length\ xs \Longrightarrow xs\ !\ n \in f\text{-}range\ xs$
⟨*proof*⟩
thm *Set.rangeE*
lemma *f-rangeE*[*elim?*]:
$[\![\ x \in f\text{-}range\ xs;\ \bigwedge n.\ [\![\ n < length\ xs;\ x = xs\ !\ n\ ]\!] \Longrightarrow P\ ]\!] \Longrightarrow P$
⟨*proof*⟩

9.1.6 Mapping lists of functions to lists

primrec
map-list :: $('a \Rightarrow 'b)\ list \Rightarrow 'a\ list \Rightarrow 'b\ list$
where
map-list $[]\ xs = []$
| *map-list* $(f\ \#\ fs)\ xs = f\ (hd\ xs)\ \#\ map\text{-}list\ fs\ (tl\ xs)$

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*: $\bigwedge xs.$
length (*map-list fs xs*) = *length fs*
⟨*proof*⟩
corollary *map-list-empty-conv*:
(*map-list fs xs* = []) = (*fs* = [])
⟨*proof*⟩
corollary *map-list-not-empty-conv*:
(*map-list fs xs* ≠ []) = (*fs* ≠ [])
⟨*proof*⟩

lemma *map-list-nth*: $\bigwedge n\ xs.$
$[\![\ n < length\ fs;\ n < length\ xs\ ]\!] \Longrightarrow$
(*map-list fs xs* ! *n*) =
(*fs* ! *n*) (*xs* ! *n*)
⟨*proof*⟩

lemma *map-list-xs-take*: $\bigwedge n\ xs.$
length fs ≤ *n* ⟹
map-list fs (*xs* ↓ *n*) =
map-list fs xs
⟨*proof*⟩

thm *arg-cong*
⟨*proof*⟩

lemma *map-list-take*: ⋀*n xs*.
(*map-list fs xs*) ↓ *n* =
(*map-list* (*fs* ↓ *n*) *xs*)
⟨*proof*⟩
lemma *map-list-take-take*: ⋀*n xs*.
(*map-list fs xs*) ↓ *n* =
(*map-list* (*fs* ↓ *n*) (*xs* ↓ *n*))
⟨*proof*⟩
lemma *map-list-drop*: ⋀*n xs*.
(*map-list fs xs*) ↑ *n* =
(*map-list* (*fs* ↑ *n*) (*xs* ↑ *n*))
⟨*proof*⟩

lemma *map-list-append-append*: ⋀*xs1* .
length fs1 = *length xs1* ⟹
map-list (*fs1* @ *fs2*) (*xs1* @ *xs2*) =
map-list fs1 xs1 @
map-list fs2 xs2
⟨*proof*⟩
lemma *map-list-snoc-snoc*:
length fs = *length xs* ⟹
map-list (*fs* @ [*f*]) (*xs* @ [*x*]) =
map-list fs xs @ [*f x*]
⟨*proof*⟩
lemma *map-list-snoc*: ⋀*xs*.
length fs < *length xs* ⟹
map-list (*fs* @ [*f*]) *xs* =
map-list fs xs @ [*f* (*xs* ! (*length fs*))]
⟨*proof*⟩

lemma *map-list-Cons-if*:
map-list fs (*x* # *xs*) =
(*if* (*fs* = []) *then* [] *else* (
((*hd fs*) *x*) # *map-list* (*tl fs*) *xs*))
⟨*proof*⟩
lemma *map-list-Cons-not-empty*:
fs ≠ [] ⟹
map-list fs (*x* # *xs*) =
((*hd fs*) *x*) # *map-list* (*tl fs*) *xs*
⟨*proof*⟩

lemma *map-eq-map-list-take*: ⋀*xs*.
⟦ *length fs* ≤ *length xs*; *list-all* (λ*x*. *x* = *f*) *fs* ⟧ ⟹

map-list fs xs = map f (xs ↓ length fs)
⟨*proof*⟩
lemma *map-eq-map-list-take2*:
⟦ length fs = length xs; list-all (λx. x = f) fs ⟧ ⟹
map-list fs xs = map f xs
⟨*proof*⟩
lemma *map-eq-map-list-replicate*:
map-list (f^length xs) xs = map f xs
⟨*proof*⟩

9.1.7 Mapping functions with two arguments to lists

primrec *map2* ::
(∗ *Function taking two parameters* ∗)
('a ⇒ 'b ⇒ 'c) ⇒
(∗ *Lists of parameters* ∗)
'a list ⇒ 'b list ⇒
'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*: *⋀ys. map2 f xs ys = map-list (map f xs) ys*
⟨*proof*⟩

lemma *map2-Nil*: *map2 f [] ys = []*
⟨*proof*⟩
lemma *map2-Cons-Cons*:
map2 f (x # xs) (y # ys) =
(f x y) # map2 f xs ys
⟨*proof*⟩

lemma *map2-length*: *⋀ys. length (map2 f xs ys) = length xs*
⟨*proof*⟩
corollary *map2-empty-conv*:
(map2 f xs ys = []) = (xs = [])
⟨*proof*⟩
corollary *map2-not-empty-conv*:
(map2 f xs ys ≠ []) = (xs ≠ [])
⟨*proof*⟩

lemma *map2-nth*: *⋀n ys.*
⟦ n < length xs; n < length ys ⟧ ⟹
(map2 f xs ys ! n) =
f (xs ! n) (ys ! n)
thm *map-list-nth*
⟨*proof*⟩

lemma *map2-ys-take*: $\bigwedge n\ ys.$
$length\ xs \leq n \Longrightarrow$
$map2\ f\ xs\ (ys \downarrow n) =$
$map2\ f\ xs\ ys$
thm *map-list-xs-take*
⟨*proof*⟩

lemma *map2-take*: $\bigwedge n\ ys.$
$(map2\ f\ xs\ ys) \downarrow n =$
$(map2\ f\ (xs \downarrow n)\ ys)$
thm *map-list-take*
⟨*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*: $\bigwedge n\ ys.$
$(map2\ f\ xs\ ys) \uparrow n =$
$(map2\ f\ (xs \uparrow n)\ (ys \uparrow n))$
thm *map-list-drop*
⟨*proof*⟩

lemma *map2-append-append*: $\bigwedge ys1\ .$
$length\ xs1 = length\ ys1 \Longrightarrow$
$map2\ f\ (xs1\ @\ xs2)\ (ys1\ @\ ys2) =$
$map2\ f\ xs1\ ys1\ @$
$map2\ f\ xs2\ ys2$
thm *map-list-append-append*
⟨*proof*⟩
lemma *map2-snoc-snoc*:
$length\ xs = length\ ys \Longrightarrow$
$map2\ f\ (xs\ @\ [x])\ (ys\ @\ [y]) =$
$map2\ f\ xs\ ys\ @$
$[f\ x\ y]$
⟨*proof*⟩
lemma *map2-snoc*: $\bigwedge ys.$
$length\ xs < length\ ys \Longrightarrow$
$map2\ f\ (xs\ @\ [x])\ ys =$
$map2\ f\ xs\ ys\ @$
$[f\ x\ (ys\ !\ (length\ xs))]$
thm *map-list-snoc*
⟨*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 ≠ [] ⟹
map2 f xs (*y* # *ys*) =
(*f* (*hd xs*) *y*) # *map2 f* (*tl xs*) *ys*
⟨*proof*⟩

lemma *map2-append1-take-drop*:
length xs1 ≤ *length ys* ⟹
map2 f (*xs1* @ *xs2*) *ys* =
map2 f xs1 (*ys* ↓ *length xs1*) @
map2 f xs2 (*ys* ↑ *length xs1*)
thm *map2-append-append*
thm *append-take-drop-id*
⟨*proof*⟩
lemma *map2-append2-take-drop*:
length ys1 ≤ *length xs* ⟹
map2 f xs (*ys1* @ *ys2*) =
map2 f (*xs* ↓ *length ys1*) *ys1* @
map2 f (*xs* ↑ *length ys1*) *ys2*
⟨*proof*⟩

thm *List.map-cong*
lemma *map2-cong*:
⟦ *xs1* = *xs2*; *ys1* = *ys2*; *length xs2* ≤ *length ys2*;
⋀*x y*. ⟦ *x* ∈ *set xs2*; *y* ∈ *set ys2* ⟧ ⟹ *f x y* = *g x y* ⟧ ⟹
map2 f xs1 ys1 = *map2 g xs2 ys2*
⟨*proof*⟩

thm *List.map-eq-conv*
lemma *map2-eq-conv*:
length xs ≤ *length ys* ⟹
(*map2 f xs ys* = *map2 g xs ys*) = (∀ *i*<*length xs*. *f* (*xs* ! *i*) (*ys* ! *i*) = *g* (*xs* ! *i*)
(*ys* ! *i*))
⟨*proof*⟩

thm *List.map-replicate*
lemma *map2-replicate*: *map2 f* x^n y^n = $(f\ x\ y)^n$
⟨*proof*⟩

lemma *map2-zip-conv*: ⋀*ys*.
length xs ≤ *length ys* ⟹
map2 f xs ys = *map* (λ(*x*,*y*). *f x y*) (*zip xs ys*)
⟨*proof*⟩

lemma *map2-rev*: ⋀*ys*.
length xs = *length ys* ⟹
rev (*map2 f xs ys*) = *map2 f* (*rev xs*) (*rev ys*)
⟨*proof*⟩

end

10 InfiniteSet2: Set operations with results of type inat

theory *InfiniteSet2*
imports *SetInterval2*
begin

10.1 Set operations with *inat*

10.1.1 Basic definitions

definition
icard :: *'a set* ⇒ *inat*
where
icard A ≡ *if finite A then Fin* (*card A*) *else* ∞

10.2 Results for *icard*

lemma *icard-UNIV-nat*: *icard* (*UNIV*::*nat set*) = ∞
⟨*proof*⟩

lemma *icard-finite-conv*: (*icard A* = *Fin* (*card A*)) = *finite A*
⟨*proof*⟩
lemma *icard-infinite-conv*: (*icard A* = ∞) = *infinite A*
⟨*proof*⟩

corollary *icard-finite*: *finite A* ⟹ *icard A* = *Fin* (*card A*)
⟨*proof*⟩
corollary *icard-infinite*[*simp*]: *infinite A* ⟹ *icard A* = ∞
⟨*proof*⟩

lemma *icard-eq-Fin-imp*: *icard A* = *Fin n* ⟹ *finite A*
⟨*proof*⟩
lemma *icard-eq-Infty-imp*: *icard A* = ∞ ⟹ *infinite A*
⟨*proof*⟩

lemma *icard-the-Fin*: *finite A* ⟹ *the-Fin* (*icard A*) = *card A*
⟨*proof*⟩

lemma *icard-eq-Fin-imp-card*: *icard A* = *Fin n* ⟹ *card A* = *n*
⟨*proof*⟩

lemma *icard-eq-Fin-card-conv*: *0* < *n* ⟹ (*icard A* = *Fin n*) = (*card A* = *n*)
⟨*proof*⟩

lemma *icard-empty*[*simp*]: *icard* {} = *0*

⟨*proof*⟩
lemma *icard-empty-iff*: (*icard A = 0*) = (*A = {}*)
⟨*proof*⟩
lemmas *icard-empty-iff-Fin = icard-empty-iff*[*unfolded zero-inat-def*]

lemma *icard-not-empty-iff*: (*0 < icard A*) = (*A ≠ {}*)
⟨*proof*⟩
lemmas *icard-not-empty-iff-Fin = icard-not-empty-iff*[*unfolded zero-inat-def*]

lemma *icard-singleton*: *icard {a} = iSuc 0*
⟨*proof*⟩
lemmas *icard-singleton-Fin*[*simp*] = *icard-singleton*[*unfolded zero-inat-def*]
lemma *icard-1-imp-singleton*: *icard A = iSuc 0 ⟹ ∃ a. A = {a}*
⟨*proof*⟩
lemma *icard-1-singleton-conv*: (*icard A = iSuc 0*) = (*∃ a. A = {a}*)
⟨*proof*⟩

thm *Finite-Set.card-insert-disjoint*
lemma *icard-insert-disjoint*: *x ∉ A ⟹ icard* (*insert x A*) = *iSuc* (*icard A*)
⟨*proof*⟩
thm *Finite-Set.card-insert-if*
lemma *icard-insert-if*: *icard* (*insert x A*) = (*if x ∈ A then icard A else iSuc* (*icard A*))
⟨*proof*⟩
thm *Finite-Set.card-0-eq*
lemmas *icard-0-eq = icard-empty-iff*

thm *Finite-Set.card-Suc-Diff1*
lemma *icard-Suc-Diff1*: *x ∈ A ⟹ iSuc* (*icard* (*A − {x}*)) = *icard A*
⟨*proof*⟩

thm *Finite-Set.card-Diff-singleton*
lemma *icard-Diff-singleton*: *x ∈ A ⟹ icard* (*A − {x}*) = *icard A − 1*
⟨*proof*⟩

thm *Finite-Set.card-Diff-singleton-if*
lemma *icard-Diff-singleton-if*: *icard* (*A − {x}*) = (*if x ∈ A then icard A − 1 else icard A*)
⟨*proof*⟩

thm *Finite-Set.card-insert*
lemma *icard-insert*: *icard* (*insert x A*) = *iSuc* (*icard* (*A − {x}*))
⟨*proof*⟩

thm *Finite-Set.card-insert-le*
lemma *icard-insert-le*: *icard A ≤ icard* (*insert x A*)
⟨*proof*⟩

thm *Finite-Set.card-mono*
lemma *icard-mono*: $A \subseteq B \Longrightarrow icard\ A \leq icard\ B$
⟨*proof*⟩

thm *Finite-Set.card-seteq*
lemma *not-icard-seteq*: $\exists (A{::}nat\ set)\ B.\ (A \subseteq B \wedge icard\ B \leq icard\ A \wedge \neg\ A = B)$
⟨*proof*⟩

thm *Finite-Set.psubset-card-mono*
lemma *not-psubset-icard-mono*: $\exists (A{::}nat\ set)\ B.\ A \subset B \wedge \neg\ icard\ A < icard\ B$
⟨*proof*⟩

thm *Finite-Set.card-Un-Int*
lemma *icard-Un-Int*: $icard\ A + icard\ B = icard\ (A \cup B) + icard\ (A \cap B)$
⟨*proof*⟩
thm *card-Un-Int*
⟨*proof*⟩

thm *Finite-Set.card-Un-disjoint*
lemma *icard-Un-disjoint*: $A \cap B = \{\} \Longrightarrow icard\ (A \cup B) = icard\ A + icard\ B$
⟨*proof*⟩

thm *Finite-Set.card-Diff-subset*
lemma *not-icard-Diff-subset*: $\exists (A{::}nat\ set)\ B.\ B \subseteq A \wedge \neg\ icard\ (A - B) = icard\ A - icard\ B$
⟨*proof*⟩

thm
Finite-Set.card-Diff1-less
Finite-Set.card-Diff2-less
lemma *not-icard-Diff1-less*: $\exists (A{::}nat\ set) x.\ x \in A \wedge \neg\ icard\ (A - \{x\}) < icard\ A$
⟨*proof*⟩
lemma *not-icard-Diff2-less*: $\exists (A{::}nat\ set) x\ y.\ x \in A \wedge y \in A \wedge \neg\ icard\ (A - \{x\} - \{y\}) < icard\ A$
⟨*proof*⟩

thm *Finite-Set.card-Diff1-le*
lemma *icard-Diff1-le*: $icard\ (A - \{x\}) \leq icard\ A$
⟨*proof*⟩

thm *Finite-Set.card-psubset*
lemma *icard-psubset*: $[\![\ A \subseteq B;\ icard\ A < icard\ B\ ]\!] \Longrightarrow A \subset B$
⟨*proof*⟩

thm *SetInterval2.card-partition*

lemma *icard-partition*:
⟦ ⋀*c*. *c* ∈ *C* ⟹ *icard c* = *k*; ⋀*c1 c2*. ⟦*c1* ∈ *C*; *c2* ∈ *C*; *c1* ≠ *c2*⟧ ⟹ *c1* ∩ *c2* = {} ⟧ ⟹
icard (⋃ *C*) = *k* ∗ *icard C*
⟨*proof*⟩
thm *SetInterval2.card-partition*
thm *icard-eq-Fin-imp-card*
⟨*proof*⟩
thm *SetInterval2.card-partition*
⟨*proof*⟩
thm *finite-UnionD*
⟨*proof*⟩
thm *Union-upper*
⟨*proof*⟩
thm *infinite-super*
⟨*proof*⟩

thm *Big-Operators.setsum-constant*

thm *Big-Operators.setprod-constant*

thm *Big-Operators.setsum-bounded*

thm *Big-Operators.card-UN-disjoint*

thm *Big-Operators.card-Union-disjoint*

thm *Finite-Set.card-image-le*
lemma *icard-image-le*: *icard* (*f* ‘ *A*) ≤ *icard A*
⟨*proof*⟩

thm *Finite-Set.card-image*
lemma *icard-image*: *inj-on f A* ⟹ *icard* (*f* ‘ *A*) = *icard A*
⟨*proof*⟩

thm *Finite-Set.eq-card-imp-inj-on*
lemma *not-eq-icard-imp-inj-on*: ∃(*f*::*nat*⇒*nat*) (*A*::*nat set*). *icard* (*f* ‘ *A*) = *icard A* ∧ ¬ *inj-on f A*
⟨*proof*⟩

thm *Finite-Set.inj-on-iff-eq-card*
lemma *not-inj-on-iff-eq-icard*: ∃(*f*::*nat*⇒*nat*) (*A*::*nat set*). ¬ (*inj-on f A* = (*icard* (*f* ‘ *A*) = *icard A*))
⟨*proof*⟩

thm *Finite-Set.card-inj-on-le*
lemma *icard-inj-on-le*: ⟦ *inj-on f A*; *f* ‘ *A* ⊆ *B* ⟧ ⟹ *icard A* ≤ *icard B*
⟨*proof*⟩

thm *Finite-Set.card-bij-eq*
thm *Finite-Set.card-bij-eq*[*no-vars*]
lemma *icard-bij-eq*:
⟦ *inj-on f A*; *f ' A* ⊆ *B*; *inj-on g B*; *g ' B* ⊆ *A* ⟧ ⟹
icard A = *icard B*
⟨*proof*⟩

thm *Big-Operators.card-SigmaI*

thm *Big-Operators.card-cartesian-product*
lemma *icard-cartesian-product*: *icard* (*A* × *B*) = *icard A* ∗ *icard B*
⟨*proof*⟩

thm *card-cartesian-product-singleton*
lemma *icard-cartesian-product-singleton*: *icard* ({*x*} × *A*) = *icard A*
⟨*proof*⟩

thm *card-cartesian-product-singleton-right*
lemma *icard-cartesian-product-singleton-right*: *icard* (*A* × {*x*}) = *icard A*
⟨*proof*⟩

thm *Finite-Set.card-Pow*

thm *Finite-Set.dvd-partition*

thm *Equiv-Relations.equiv-imp-dvd-card*

thm *Equiv-Relations.card-quotient-disjoint*

thm
SetInterval.card-lessThan
SetInterval.card-atMost
SetInterval.card-atLeastLessThan
SetInterval.card-atLeastAtMost
SetInterval.card-greaterThanAtMost
SetInterval.card-greaterThanLessThan
lemma
icard-lessThan: *icard* {..<*u*} = *Fin u* **and**
icard-atMost: *icard* {..*u*} = *Fin* (*Suc u*) **and**
icard-atLeastLessThan: *icard* {*l*..<*u*} = *Fin* (*u* − *l*) **and**
icard-atLeastAtMost: *icard* {*l*..*u*} = *Fin* (*Suc u* − *l*) **and**
icard-greaterThanAtMost: *icard* {*l*<..*u*} = *Fin* (*u* − *l*) **and**
icard-greaterThanLessThan: *icard* {*l*<..<*u*} = *Fin* (*u* − *Suc l*)
⟨*proof*⟩

lemma *icard-atLeast*: *icard* $\{(u::nat)..\} = \infty$
⟨*proof*⟩
lemma *icard-greaterThan*: *icard* $\{(u::nat)<..\} = \infty$
⟨*proof*⟩

thm
SetInterval.card-atLeastZeroLessThan-int
SetInterval.card-atLeastLessThan-int
SetInterval.card-atLeastAtMost-int
SetInterval.card-greaterThanAtMost-int
lemma
icard-atLeastZeroLessThan-int: *icard* $\{0..<u\} = Fin\ (nat\ u)$ **and**
icard-atLeastLessThan-int: *icard* $\{l..<u\} = Fin\ (nat\ (u - l))$ **and**
icard-atLeastAtMost-int: *icard* $\{l..u\} = Fin\ (nat\ (u - l + 1))$ **and**
icard-greaterThanAtMost-int: *icard* $\{l<..u\} = Fin\ (nat\ (u - l))$
⟨*proof*⟩

lemma *icard-atLeast-int*: *icard* $\{(u::int)..\} = \infty$
⟨*proof*⟩

lemma *icard-greaterThan-int*: *icard* $\{(u::int)<..\} = \infty$
⟨*proof*⟩

lemma *icard-atMost-int*: *icard* $\{..(u::int)\} = \infty$
⟨*proof*⟩

lemma *icard-lessThan-int*: *icard* $\{..<(u::int)\} = \infty$
⟨*proof*⟩

end

11 ListInf: 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 $nat \Rightarrow 'a$ as infinite lists over elements of $'a$. Mapping functions to intervals lists $[m..<n]$ yiels common finite lists.

11.1.1 Appending a functions to a list

types *'a ilist* = *nat* ⇒ *'a*

definition
i-append :: *'a list* ⇒ *'a ilist* ⇒ *'a ilist* (**infixr** ⌢ *65*)
where
xs ⌢ *f* ≡ λ*n*. *if n* < *length xs then xs* ! *n else f* (*n* − *length xs*)

syntax (*HTML* **output**)
i-append :: *'a list* ⇒ *'a ilist* ⇒ *'a ilist* (**infixr** ⌢ *65*)

Synonym for the lemma *Fun.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 *expand-ilist-eq* = *Fun.fun-eq-iff*
lemmas *ilist-eq-iff* = *expand-ilist-eq*

lemma *i-append-nth*: (*xs* ⌢ *f*) *n* = (*if n* < *length xs then xs* ! *n else f* (*n* − *length xs*))
⟨*proof*⟩
lemma *i-append-nth1*[*simp*]: *n* < *length xs* ⟹ (*xs* ⌢ *f*) *n* = *xs* ! *n*
⟨*proof*⟩
lemma *i-append-nth2*[*simp*]: *length xs* ≤ *n* ⟹ (*xs* ⌢ *f*) *n* = *f* (*n* − *length xs*)
⟨*proof*⟩
lemma *i-append-Nil*[*simp*]: [] ⌢ *f* = *f*
⟨*proof*⟩

lemma *i-append-assoc*[*simp*]: *xs* ⌢ (*ys* ⌢ *f*) = (*xs* @ *ys*) ⌢ *f*
⟨*proof*⟩

thm *append-Cons*
lemma *i-append-Cons*: (*x* # *xs*) ⌢ *f* = [*x*] ⌢ (*xs* ⌢ *f*)
⟨*proof*⟩

thm *List.append-eq-append-conv*
lemma *i-append-eq-i-append-conv*[*simp*]:
length xs = *length ys* ⟹
(*xs* ⌢ *f* = *ys* ⌢ *g*) = (*xs* = *ys* ∧ *f* = *g*)
⟨*proof*⟩

thm *List.append-eq-append-conv2*
lemma *i-append-eq-i-append-conv2-aux*:
⟦ *xs* ⌢ *f* = *ys* ⌢ *g*; *length xs* ≤ *length ys* ⟧ ⟹
∃ *zs*. *xs* @ *zs* = *ys* ∧ *f* = *zs* ⌢ *g*
⟨*proof*⟩

lemma *i-append-eq-i-append-conv2*:
(*xs* ⌢ *f* = *ys* ⌢ *g*) =

$(\exists\, zs.\ xs = ys\ @\ zs \wedge zs \frown f = g \vee xs\ @\ zs = ys \wedge f = zs \frown g)$
⟨*proof*⟩

thm *List.same-append-eq*
lemma *same-i-append-eq*[*iff*]: $(xs \frown f = xs \frown g) = (f = g)$
⟨*proof*⟩

thm *List.append-same-eq*
lemma *NOT-i-append-same-eq*:
$\neg(\forall\, xs\ ys\ f.\ (xs \frown (f::(nat \Rightarrow nat)) = ys \frown f) = (xs = ys))$
⟨*proof*⟩

thm *List.hd-append*
lemma *i-append-hd*: $(xs \frown f)\ 0 = (if\ xs = []\ then\ f\ 0\ else\ hd\ xs)$
⟨*proof*⟩

thm *List.hd-append2*
lemma *i-append-hd2*[*simp*]: $xs \neq [] \Longrightarrow (xs \frown f)\ 0 = hd\ xs$
⟨*proof*⟩

thm *List.eq-Nil-appendI*
lemma *eq-Nil-i-appendI*: $f = g \Longrightarrow f = [] \frown g$
⟨*proof*⟩

thm *List.append-eq-appendI*
lemma *i-append-eq-i-appendI*:
$[\![\ xs\ @\ xs' = ys;\ f = xs' \frown g\ ]\!] \Longrightarrow xs \frown f = ys \frown g$
⟨*proof*⟩

thm *List.map-ext*
lemma *o-ext*:
$(\forall\, x.\ (x \in range\ h \longrightarrow f\ x = g\ x)) \Longrightarrow f \circ h = g \circ h$
⟨*proof*⟩
thm
o-ext
o-ext[*rule-format*]
thm
List.map-ident
Fun.id-o

thm *List.map-append*
lemma *i-append-o*[*simp*]: $g \circ (xs \frown f) = (map\ g\ xs) \frown (g \circ f)$
⟨*proof*⟩

thm

List.map-map[*of f g h*]
Fun.o-assoc[*of f g h*, *symmetric*]
thm *List.map-eq-conv*
lemma *o-eq-conv*: $(f \circ h = g \circ h) = (\forall x{\in}range\ h.\ f\ x = g\ x)$
⟨*proof*⟩

thm *List.map-cong*
lemma *o-cong*:
$[\![\ h = i;\ \bigwedge x.\ x \in range\ i \Longrightarrow f\ x = g\ x\ ]\!] \Longrightarrow f \circ h = f \circ i$
⟨*proof*⟩

thm *List.ex-map-conv*
lemma *ex-o-conv*: $(\exists h.\ g = f \circ h) = (\forall y{\in}range\ g.\ \exists x.\ y = f\ x)$
⟨*proof*⟩
thm *someI-ex*
⟨*proof*⟩

thm *List.map-inj-on*
lemma *o-inj-on*:
$[\![\ f \circ g = f \circ h;\ inj\text{-}on\ f\ (range\ g \cup range\ h)\ ]\!] \Longrightarrow g = h$
⟨*proof*⟩

thm *List.inj-on-map-eq-map*
lemma *inj-on-o-eq-o*:
$inj\text{-}on\ f\ (range\ g \cup range\ h) \Longrightarrow$
$(f \circ g = f \circ h) = (g = h)$
⟨*proof*⟩

thm *List.map-injective*
lemma *o-injective*: $[\![\ f \circ g = f \circ h;\ inj\ f\ ]\!] \Longrightarrow g = h$
⟨*proof*⟩

thm *List.inj-map-eq-map*
lemma *inj-o-eq-o*: $inj\ f \Longrightarrow (f \circ g = f \circ h) = (g = h)$
⟨*proof*⟩

thm *List.inj-mapI*
lemma *inj-oI*: $inj\ f \Longrightarrow inj\ (\lambda g.\ f \circ g)$
⟨*proof*⟩
thm *o-inj-on*[*unfolded inj-on-def*]
⟨*proof*⟩

thm *List.inj-mapD*
lemma *inj-oD*: $inj\ (\lambda g.\ f \circ g) \Longrightarrow inj\ f$
⟨*proof*⟩

thm *List.inj-map*
lemma *inj-o*[*iff*]: $inj\ (\lambda g.\ f \circ g) = inj\ f$
⟨*proof*⟩

thm *List.inj-on-mapI*
lemma *inj-on-oI*:
inj-on f (⋃ (λ*f*. *range f*) ‘ *A*) ⟹ *inj-on* (λ*g*. *f* ∘ *g*) *A*
⟨*proof*⟩

thm *List.map-idI*
lemma *o-idI*: ∀ *x*. *x* ∈ *range g* ⟶ *f x* = *x* ⟹ *f* ∘ *g* = *g*
⟨*proof*⟩
thm
o-idI
o-idI[*rule-format*]

thm *List.map-fun-upd*
lemma *o-fun-upd*[*simp*]: *y* ∉ *range g* ⟹ *f* (*y* := *x*) ∘ *g* = *f* ∘ *g*
⟨*proof*⟩

thm *List.set-append*
lemma *range-i-append*[*simp*]: *range* (*xs* ⌢ *f*) = *set xs* ∪ *range f*
⟨*proof*⟩

thm *List.set-subset-Cons*
lemma *set-subset-i-append*: *set xs* ⊆ *range* (*xs* ⌢ *f*)
⟨*proof*⟩
lemma *range-subset-i-append*: *range f* ⊆ *range* (*xs* ⌢ *f*)
⟨*proof*⟩

thm *List.set-ConsD*
lemma *range-ConsD*: *y* ∈ *range* ([*x*] ⌢ *f*) ⟹ *y* = *x* ∨ *y* ∈ *range f*
⟨*proof*⟩

thm *List.set-map*
lemma *range-o*[*simp*]: *range* (*f* ∘ *g*) = *f* ‘ *range g*
⟨*proof*⟩

thm *List.in-set-conv-decomp*
lemma *in-range-conv-decomp*:
(*x* ∈ *range f*) = (∃ *xs g*. *f* = *xs* ⌢ ([*x*] ⌢ *g*))
⟨*proof*⟩

nth

thm *List.nth-Cons-0*
lemma *i-append-nth-Cons-0*[*simp*]: ((*x* # *xs*) ⌢ *f*) *0* = *x*
⟨*proof*⟩

thm *List.nth-Cons-Suc*
lemma *i-append-nth-Cons-Suc*[*simp*]:

$((x \# xs) \frown f)\ (Suc\ n) = (xs \frown f)\ n$
⟨*proof*⟩

lemma *i-append-nth-Cons*:
$([x] \frown f)\ n = (case\ n\ of\ 0 \Rightarrow x \mid Suc\ k \Rightarrow f\ k)$
⟨*proof*⟩

lemma *i-append-nth-Cons'*:
$([x] \frown f)\ n = (if\ n = 0\ then\ x\ else\ f\ (n - Suc\ 0))$
⟨*proof*⟩

thm
List.nth-append
thm
i-append-def
i-append-nth1
i-append-nth2

thm *List.nth-append-length*
lemma *i-append-nth-length*[*simp*]: $(xs \frown f)\ (length\ xs) = f\ 0$
⟨*proof*⟩

thm *List.nth-append-length-plus*
lemma *i-append-nth-length-plus*[*simp*]: $(xs \frown f)\ (length\ xs + n) = f\ n$
⟨*proof*⟩

thm
List.nth-map
Fun.o-apply

thm
List.set-conv-nth
Set.full-SetCompr-eq

thm *List.in-set-conv-nth*
lemma *range-iff*: $(y \in range\ f) = (\exists x.\ y = f\ x)$
⟨*proof*⟩

thm *List.list-ball-nth*
lemma *range-ball-nth*: $\forall y{\in}range\ f.\ P\ y \Longrightarrow P\ (f\ x)$
⟨*proof*⟩

thm
List.nth-mem
Set.rangeI

thm *List.all-nth-imp-all-set*
lemma *all-nth-imp-all-range*: $[\![\ \forall x.\ P\ (f\ x); y \in range\ f\ ]\!] \Longrightarrow P\ y$
⟨*proof*⟩

thm *List.all-set-conv-all-nth*
lemma *all-range-conv-all-nth*: $(\forall\, y{\in}range\, f.\ P\ y) = (\forall\, x.\ P\ (f\ x))$
⟨*proof*⟩

thm
List.nth-list-update
Fun.fun-upd-def
thm
List.nth-list-update-eq
Fun.fun-upd-same
thm
List.nth-list-update-neq
Fun.fun-upd-other
thm
List.list-update-overwrite
Fun.fun-upd-upd
thm
List.list-update-id
Fun.fun-upd-triv
thm
List.list-update-same-conv
Fun.fun-upd-idem-iff

thm *List.list-update-append1*
lemma *i-append-update1*:
$n < length\ xs \Longrightarrow (xs \frown f)\ (n := x) = xs[n := x] \frown f$
⟨*proof*⟩
lemma *i-append-update2*:
$length\ xs \leq n \Longrightarrow (xs \frown f)\ (n := x) = xs \frown (f(n - length\ xs := x))$
⟨*proof*⟩

thm *List.list-update-append*
lemma *i-append-update*:
$(xs \frown f)\ (n := x) =$
$(if\ n < length\ xs\ then\ xs[n := x] \frown f$
$\ else\ xs \frown (f(n - length\ xs := x)))$
⟨*proof*⟩

thm *List.list-update-length*
lemma *i-append-update-length*[*simp*]:
$(xs \frown f)\ (length\ xs := y) = xs \frown (f(0 := y))$
⟨*proof*⟩

thm *List.set-update-subset-insert*

lemma *range-update-subset-insert*:
range $(f(n := x)) \subseteq$ *insert* x (*range* f)
⟨*proof*⟩

thm *List.set-update-subsetI*
lemma *range-update-subsetI*:
$[\![$ *range* $f \subseteq A$; $x \in A$ $]\!] \Longrightarrow$ *range* $(f(n := x)) \subseteq A$
⟨*proof*⟩

thm *List.set-update-memI*
lemma *range-update-memI*: $x \in$ *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-take f* $n = [f\ 0, f\ 1, \ldots, f\ (n-1)]$. The *i-drop* operator drops the first n elements of an infinite list, i.e. (*i-take f n*) $0 = f\ n$, (*i-take f n*) $1 = f\ (n + 1)$, ...

definition
i-take :: *nat* ⇒ *'a ilist* ⇒ *'a list*
where
i-take n f ≡ *map f* [*0..<n*]
definition
i-drop :: *nat* ⇒ *'a ilist* ⇒ *'a ilist*
where
i-drop n f ≡ (λ*x*. *f* (*n* + *x*))

abbreviation (*xsymbols*)
i-take' :: *'a ilist* ⇒ *nat* ⇒ *'a list* (**infixl** ⇓ *100*)
where
f ⇓ *n* ≡ *i-take n f*
abbreviation (*xsymbols*)
i-drop' :: *'a ilist* ⇒ *nat* ⇒ *'a ilist* (**infixl** ⇑ *100*)
where
f ⇑ *n* ≡ *i-drop n f*

syntax (*HTML* **output**)
i-take' :: *'a ilist* ⇒ *nat* ⇒ *'a list* (**infixl** ⇓ *100*)
i-drop' :: *'a ilist* ⇒ *nat* ⇒ *'a ilist* (**infixl** ⇑ *100*)

term *f* ⇓ *n*
term *f* ⇑ *n*

lemma *f* ⇓ *n* = *map f* [*0..<n*]
⟨*proof*⟩
lemma *f* ⇑ *n* = (λ*x*. *f* (*n* + *x*))
⟨*proof*⟩

Basic results for *i-take* and *i-drop*

thm *take-first*
lemma *i-take-first*: $f \Downarrow Suc\ 0 = [f\ 0]$
⟨*proof*⟩

thm *drop-take-1*
lemma *i-drop-i-take-1*: $f \Uparrow n \Downarrow Suc\ 0 = [f\ n]$
⟨*proof*⟩

thm *List.take-take*
lemma *i-take-take-eq1*: $m \leq n \Longrightarrow (f \Downarrow n) \downarrow m = f \Downarrow m$
⟨*proof*⟩
lemma *i-take-take-eq2*: $n \leq m \Longrightarrow (f \Downarrow n) \downarrow m = f \Downarrow n$
⟨*proof*⟩
lemma *i-take-take*[*simp*]: $(f \Downarrow n) \downarrow m = f \Downarrow min\ n\ m$
⟨*proof*⟩

thm *List.nth-drop*
lemma *i-drop-nth*[*simp*]: $(s \Uparrow n)\ x = s\ (n + x)$
⟨*proof*⟩
lemma *i-drop-nth-sub*: $n \leq x \Longrightarrow (s \Uparrow n)\ (x - n) = s\ x$
⟨*proof*⟩
thm *List.nth-take*
theorem *i-take-nth*[*simp*]: $i < n \Longrightarrow (f \Downarrow n)\ !\ i = f\ i$
⟨*proof*⟩

thm *List.length-take*
lemma *i-take-length*[*simp*]: $length\ (f \Downarrow n) = n$
⟨*proof*⟩

thm *List.take-0*
lemma *i-take-0*[*simp*]: $f \Downarrow 0 = []$
⟨*proof*⟩
thm *List.drop-0*
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*: $last\ (f \Downarrow Suc\ n) = f\ n$
⟨*proof*⟩

lemma *last-i-take2*: $0 < n \Longrightarrow last\ (f \Downarrow n) = f\ (n - Suc\ 0)$
⟨*proof*⟩

lemma *nth-0-i-drop*: $(f \Uparrow n)\ 0 = f\ n$

⟨*proof*⟩

thm *take-replicate drop-replicate*
lemma *i-take-const*[*simp*]: $(\lambda n.\ x) \Downarrow i = replicate\ i\ x$
⟨*proof*⟩
lemma *i-drop-const*[*simp*]: $(\lambda n.\ x) \Uparrow i = (\lambda n.\ x)$
⟨*proof*⟩

thm *List.take-append*
lemma *i-append-i-take-eq1*:
$n \leq length\ xs \Longrightarrow (xs \frown f) \Downarrow n = xs \downarrow n$
⟨*proof*⟩

lemma *i-append-i-take-eq2*:
$length\ xs \leq n \Longrightarrow (xs \frown f) \Downarrow n = xs\ @\ (f \Downarrow (n - length\ xs))$
⟨*proof*⟩
lemma *i-append-i-take-if*:
$(xs \frown f) \Downarrow n = (if\ n \leq length\ xs\ then\ xs \downarrow n\ else\ xs\ @\ (f \Downarrow (n - length\ xs)))$
⟨*proof*⟩
lemma *i-append-i-take*[*simp*]:
$(xs \frown f) \Downarrow n = (xs \downarrow n)\ @\ (f \Downarrow (n - length\ xs))$
⟨*proof*⟩

thm *List.drop-append*
lemma *i-append-i-drop-eq1*:
$n \leq length\ xs \Longrightarrow (xs \frown f) \Uparrow n = (xs \uparrow n) \frown f$
⟨*proof*⟩
lemma *i-append-i-drop-eq2*:
$length\ xs \leq n \Longrightarrow (xs \frown f) \Uparrow n = f \Uparrow (n - length\ xs)$
⟨*proof*⟩
lemma *i-append-i-drop-if*:
$(xs \frown f) \Uparrow n = (if\ n < length\ xs\ then\ (xs \uparrow n) \frown f\ else\ f \Uparrow (n - length\ xs))$
⟨*proof*⟩
lemma *i-append-i-drop*[*simp*]: $(xs \frown f) \Uparrow n = (xs \uparrow n) \frown (f \Uparrow (n - length\ xs))$
⟨*proof*⟩

thm
List.take-append
List.drop-append
i-append-i-take
i-append-i-drop

thm *List.append-take-drop-id*
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*:
$[\![\ f \Downarrow n\ = g \Downarrow n; f \Uparrow n = g \Uparrow n\ ]\!] \Longrightarrow f = g$
⟨*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*:
$f \Downarrow k = (THE\ xs.\ length\ xs = k \wedge (\exists g.\ xs \frown g = f))$
thm *the1I2*
⟨*proof*⟩

lemma *i-drop-the-conv*:
$f \Uparrow k = (THE\ g.\ (\exists xs.\ length\ xs = k \wedge xs \frown g = f))$
⟨*proof*⟩

thm *List.take-Suc-Cons*
lemma *i-take-Suc-append*[*simp*]:
$((x \# xs) \frown f) \Downarrow Suc\ n = x \# ((xs \frown f) \Downarrow n)$
⟨*proof*⟩
corollary *i-take-Suc-Cons*: $([x] \frown f) \Downarrow Suc\ n = x \# (f \Downarrow n)$
⟨*proof*⟩

lemma *i-drop-Suc-append*[*simp*]: $((x \# xs) \frown f) \Uparrow Suc\ n = ((xs \frown f) \Uparrow n)$
⟨*proof*⟩
corollary *i-drop-Suc-Cons*: $([x] \frown f) \Uparrow Suc\ n = f \Uparrow n$
⟨*proof*⟩

thm *List.take-Suc*
lemma *i-take-Suc*: $f \Downarrow Suc\ n = f\ 0 \# (f \Uparrow Suc\ 0 \Downarrow n)$
⟨*proof*⟩
thm *List.take-Suc-conv-app-nth*
lemma *i-take-Suc-conv-app-nth*: $f \Downarrow Suc\ n = (f \Downarrow n) @ [f\ n]$
⟨*proof*⟩

thm *List.drop-drop*
lemma *i-drop-i-drop*[*simp*]: $s \Uparrow a \Uparrow b = s \Uparrow (a + b)$
⟨*proof*⟩
corollary *i-drop-Suc*: $f \Uparrow Suc\ 0 \Uparrow n = f \Uparrow Suc\ n$
⟨*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*⟩

thm *List.drop-tl*
corollary *i-drop-tl*: $f \Uparrow Suc\ 0 \Uparrow n = f \Uparrow n \Uparrow Suc\ 0$
⟨*proof*⟩

thm *List.nth-via-drop*
lemma *nth-via-i-drop*: $(f \Uparrow n)\ 0 = x \Longrightarrow f\ n = x$
⟨*proof*⟩

thm *List.drop-Suc-conv-tl*
lemma *i-drop-Suc-conv-tl*: $[f\ n] \frown (f \Uparrow Suc\ n) = f \Uparrow n$
⟨*proof*⟩

lemma *i-drop-Suc-conv-tl'*: $([f\ n] \frown f) \Uparrow Suc\ n = f \Uparrow n$
⟨*proof*⟩
thm *i-drop-Suc-conv-tl i-drop-Suc-conv-tl'*

thm *List.take-drop*
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*:
$m \leq n \Longrightarrow (f \Downarrow m)\ @\ map\ f\ [m..<n] = f \Downarrow n$
⟨*proof*⟩

lemma *i-take-drop-map-empty-iff*: $(f \Downarrow n \uparrow m = []) = (n \leq m)$
⟨*proof*⟩

lemma *i-take-drop-map*: $f \Downarrow n \uparrow m = map\ f\ [m..<n]$
⟨*proof*⟩

corollary *i-take-drop-append*[*simp*]:
$m \leq n \Longrightarrow (f \Downarrow m)\ @\ (f \Downarrow n \uparrow m) = f \Downarrow n$
⟨*proof*⟩

thm *List.drop-take*
lemma *i-take-drop*: $f \Downarrow n \uparrow m = f \Uparrow m \Downarrow (n - m)$
⟨*proof*⟩

thm *List.take-map*
lemma *i-take-o*[*simp*]: $(f \circ g) \Downarrow n = map\ f\ (g \Downarrow n)$
⟨*proof*⟩
thm *List.drop-map*
lemma *i-drop-o*[*simp*]: $(f \circ g) \Uparrow n = f \circ (g \Uparrow n)$
⟨*proof*⟩

thm *List.set-take-subset*
lemma *set-i-take-subset*: $set\ (f \Downarrow n) \subseteq range\ f$
⟨*proof*⟩

thm *List.set-drop-subset*
lemma *range-i-drop-subset*: $range\ (f \Uparrow n) \subseteq range\ f$
⟨*proof*⟩

thm *List.in-set-takeD*
lemma *in-set-i-takeD*: $x \in set\ (f \Downarrow n) \Longrightarrow x \in range\ f$
⟨*proof*⟩
thm *List.in-set-dropD*
lemma *in-range-i-takeD*: $x \in range\ (f \Uparrow n) \Longrightarrow x \in range\ f$
⟨*proof*⟩

thm *List.append-eq-conv-conj*
lemma *i-append-eq-conv-conj*:
$((xs \frown f) = g) = (xs = g \Downarrow length\ xs \wedge f = g \Uparrow length\ xs)$
⟨*proof*⟩

thm *List.take-add*
lemma *i-take-add*: $f \Downarrow (i + j) = (f \Downarrow i)\ @\ (f \Uparrow i \Downarrow j)$
⟨*proof*⟩

thm *List.append-eq-append-conv-if*
lemma *i-append-eq-i-append-conv-if-aux*:
$length\ xs \leq length\ ys \Longrightarrow$
$(xs \frown f = ys \frown g) = (xs = ys \downarrow length\ xs \wedge f = (ys \uparrow length\ xs) \frown g)$
⟨*proof*⟩
lemma *i-append-eq-i-append-conv-if*:
$(xs \frown f = ys \frown g) =$
$(if\ length\ xs \leq length\ ys$
$\ then\ xs = ys \downarrow length\ xs \wedge f = (ys \uparrow length\ xs) \frown g$
$\ else\ xs \downarrow length\ ys = ys \wedge (xs \uparrow length\ ys) \frown f = g)$
⟨*proof*⟩

thm *List.take-hd-drop*
lemma *i-take-hd-i-drop*: $(f \Downarrow n)\ @\ [(f \Uparrow n)\ 0] = f \Downarrow Suc\ n$
⟨*proof*⟩

thm *List.id-take-nth-drop*
lemma *id-i-take-nth-i-drop*: $f = (f \Downarrow n) \frown (([f\ n] \frown f) \Uparrow Suc\ n)$
⟨*proof*⟩

thm *List.upd-conv-take-nth-drop*
lemma *upd-conv-i-take-nth-i-drop*:
$f\ (n := x) = (f \Downarrow n) \frown ([x] \frown (f \Uparrow Suc\ n))$
⟨*proof*⟩

thm *nat.induct*[*of* $\lambda n.\ P\ (f \downarrow n)\ n$]
theorem *i-take-induct*:
$[\![\ P\ (f \Downarrow 0);\ \bigwedge n.\ P\ (f \Downarrow n) \Longrightarrow P\ (\ f \Downarrow Suc\ n)\ ]\!] \Longrightarrow P\ (\ f \Downarrow n)$
⟨*proof*⟩

thm *i-take-induct*
theorem *take-induct*[*rule-format*]:
$[\![\ P\ (s \downarrow 0);$
$\bigwedge n.\ [\![\ Suc\ n < length\ s;\ P\ (s \downarrow n)\ ]\!] \Longrightarrow P\ (\ s \downarrow Suc\ n);$
$i < length\ s]\!]$
$\Longrightarrow P\ (s \downarrow i)$
⟨*proof*⟩

theorem *i-drop-induct*:
$[\![\ P\ (f \Uparrow 0);\ \bigwedge n.\ P\ (f \Uparrow n) \Longrightarrow P\ (\ f \Uparrow Suc\ n)\ ]\!] \Longrightarrow P\ (\ f \Uparrow n)$
⟨*proof*⟩
thm *i-drop-induct*
theorem *f-drop-induct*[*rule-format*]:
$[\![\ P\ (s \uparrow 0);$
$\bigwedge n.\ [\![\ Suc\ n < length\ s;\ P\ (s \uparrow n)\ ]\!] \Longrightarrow P\ (\ s \uparrow Suc\ n);$
$i < length\ s]\!]$
$\Longrightarrow P\ (s \uparrow i)$
⟨*proof*⟩

lemma *i-take-drop-eq-map*: $f \Uparrow m \Downarrow n = map\ f\ [m..<m+n]$
⟨*proof*⟩

thm *List.map-eq-Cons-conv*
lemma *o-eq-i-append-imp*:
$f \circ g = ys \frown i \Longrightarrow$
$\exists xs\ h.\ g = xs \frown h \wedge map\ f\ xs = ys \wedge f \circ h = i$
⟨*proof*⟩

corollary *o-eq-i-append-conv*:
$(f \circ g = ys \frown i) =$
$(\exists xs\ h.\ g = xs \frown h \wedge map\ f\ xs = ys \wedge f \circ h = i)$
⟨*proof*⟩
corollary *i-append-eq-o-conv*:
$(ys \frown i = f \circ g) =$
$(\exists xs\ h.\ g = xs \frown h \wedge map\ f\ xs = ys \wedge f \circ h = i)$
⟨*proof*⟩

11.1.3 *zip* for infinite lists

term *zip*
definition
i-zip :: *'a ilist* ⇒ *'b ilist* ⇒ (*'a* × *'b*) *ilist*
where

i-zip f g ≡ λ*n*. (*f n*, *g n*)

lemma *i-zip-nth*: (*i-zip f g*) *n* = (*f n*, *g n*)
⟨*proof*⟩

thm *zip-swap*
lemma *i-zip-swap*: (λ(*y*, *x*). (*x*, *y*)) ∘ *i-zip g f* = *i-zip f g*
⟨*proof*⟩

lemma *i-zip-i-take*: (*i-zip f g*) ⇓ *n* = *zip* (*f* ⇓ *n*) (*g* ⇓ *n*)
⟨*proof*⟩

lemma *i-zip-i-drop*: (*i-zip f g*) ⇑ *n* = *i-zip* (*f* ⇑ *n*) (*g* ⇑ *n*)
⟨*proof*⟩

thm *List.map-fst-zip*
lemma *fst-o-izip*: *fst* ∘ (*i-zip f g*) = *f*
⟨*proof*⟩
lemma *snd-o-i-zip*: *snd* ∘ (*i-zip f g*) = *g*
⟨*proof*⟩

thm *List.update-zip*
lemma *update-i-zip*:
(*i-zip f g*)(*n* := *xy*) = *i-zip* (*f*(*n* := *fst xy*)) (*g*(*n* := *snd xy*))
⟨*proof*⟩

thm *List.zip-Cons-Cons*
lemma *i-zip-Cons-Cons*:
i-zip ([*x*] ⌢ *f*) ([*y*] ⌢ *g*) = [(*x*, *y*)] ⌢ (*i-zip f g*)
⟨*proof*⟩

thm *List.zip-append1*
lemma *i-zip-i-append1*:
i-zip (*xs* ⌢ *f*) *g* = *zip xs* (*g* ⇓ *length xs*) ⌢ (*i-zip f* (*g* ⇑ *length xs*))
⟨*proof*⟩

thm *List.zip-append2*
lemma *i-zip-i-append2*:
i-zip f (*ys* ⌢ *g*) = *zip* (*f* ⇓ *length ys*) *ys* ⌢ (*i-zip* (*f* ⇑ *length ys*) *g*)
⟨*proof*⟩

thm *List.zip-append*
lemma *i-zip-append*:
length xs = *length ys* ⟹
i-zip (*xs* ⌢ *f*) (*ys* ⌢ *g*) = *zip xs ys* ⌢ (*i-zip f g*)
⟨*proof*⟩

thm *List.set-zip*

lemma *i-zip-range*: *range* (*i-zip f g*) = { (*f n*, *g n*)| *n*. *True* }
⟨*proof*⟩

thm *List.zip-update*
lemma *i-zip-update*:
i-zip (*f*(*n* := *x*)) (*g*(*n* := *y*)) = (*i-zip f g*)(*n* := (*x*, *y*))
⟨*proof*⟩

lemma *i-zip-const*: *i-zip* (λ*n*. *x*) (λ*n*. *y*) = (λ*n*. (*x*, *y*))
⟨*proof*⟩

11.1.4 Mapping functions with two arguments to infinite lists

thm *map2.simps*
definition *i-map2* ::
(∗ *Function taking two parameters* ∗)
('*a* ⇒ '*b* ⇒ '*c*) ⇒
(∗ *Lists of parameters* ∗)
'*a ilist* ⇒ '*b ilist* ⇒
'*c ilist*
where
i-map2 f xs ys ≡ λ*n*. *f* (*xs n*) (*ys n*)

lemma *i-map2-nth*: (*i-map2 f xs ys*) *n* = *f* (*xs n*) (*ys n*)
⟨*proof*⟩

lemma *i-map2-Cons-Cons*:
i-map2 f ([*x*] ⌢ *xs*) ([*y*] ⌢ *ys*) =
[*f x y*] ⌢ (*i-map2 f xs ys*)
⟨*proof*⟩

lemma *i-map2-take-ge*:
n ≤ *n1* ⟹
i-map2 f xs ys ⇓ *n* =
map2 f (*xs* ⇓ *n*) (*ys* ⇓ *n1*)
⟨*proof*⟩
lemma *i-map2-take-take*:
i-map2 f xs ys ⇓ *n* =
map2 f (*xs* ⇓ *n*) (*ys* ⇓ *n*)
⟨*proof*⟩

lemma *i-map2-drop*:
(*i-map2 f xs ys*) ⇑ *n* =
(*i-map2 f* (*xs* ⇑ *n*) (*ys* ⇑ *n*))
⟨*proof*⟩

lemma *i-map2-append-append*:
length xs1 = *length ys1* ⟹
i-map2 f (*xs1* ⌢ *xs*) (*ys1* ⌢ *ys*) =

map2 f xs1 ys1 ⌢ i-map2 f xs ys
⟨*proof*⟩

lemma *i-map2-Cons-left*:
i-map2 f ([x] ⌢ xs) ys =
[f x (ys 0)] ⌢ i-map2 f xs (ys ⇑ Suc 0)
⟨*proof*⟩
lemma *i-map2-Cons-right*:
i-map2 f xs ([y] ⌢ ys) =
[f (xs 0) y] ⌢ i-map2 f (xs ⇑ Suc 0) ys
⟨*proof*⟩

lemma *i-map2-append-take-drop-left*:
i-map2 f (xs1 ⌢ xs) ys =
map2 f xs1 (ys ⇓ length xs1) ⌢
i-map2 f xs (ys ⇑ length xs1)
⟨*proof*⟩
lemma *i-map2-append-take-drop-right*:
i-map2 f xs (ys1 ⌢ ys) =
map2 f (xs ⇓ length ys1) ys1 ⌢
i-map2 f (xs ⇑ length ys1) ys
⟨*proof*⟩

thm *o-cong*
lemma *i-map2-cong*:
⟦ xs1 = xs2; ys1 = ys2;
⋀x y. ⟦ x ∈ range xs2; y ∈ range ys2 ⟧ ⟹ f x y = g x y ⟧ ⟹
i-map2 f xs1 ys1 = i-map2 g xs2 ys2
⟨*proof*⟩

thm *o-eq-conv*
lemma *i-map2-eq-conv*:
(i-map2 f xs ys = i-map2 g xs ys) = (∀ i. f (xs i) (ys i) = g (xs i) (ys i))
⟨*proof*⟩

lemma *i-map2-replicate*: *i-map2 f (λn. x) (λn. y) = (λn. f x y)*
⟨*proof*⟩

lemma *i-map2-i-zip-conv*:
i-map2 f xs ys = (λ(x,y). f x y) ∘ (i-zip xs ys)
⟨*proof*⟩

11.2 Generalised lists as combination of finite and infinite lists

11.2.1 Basic definitions

datatype *'a glist = FL 'a list | IL 'a ilist*

thm *list.simps*
term *nth*

definition
glength :: *'a glist* ⇒ *inat*
where
glength a ≡ *case a of*
FL xs ⇒ *Fin* (*length xs*) |
IL f ⇒ ∞
definition
gCons :: *'a* ⇒ *'a glist* ⇒ *'a glist* (**infixr** #$_g$ *65*)
where
x #$_g$ *a* ≡ *case a of*
FL xs ⇒ *FL* (*x* # *xs*) |
IL g ⇒ *IL* ([*x*] ⌢ *g*)
definition
gappend :: *'a glist* ⇒ *'a glist* ⇒ *'a glist* (**infixr** @$_g$ *65*)
where
gappend a b ≡ *case a of*
FL xs ⇒ (*case b of FL ys* ⇒ *FL* (*xs* @ *ys*) | *IL f* ⇒ *IL* (*xs* ⌢ *f*)) |
IL f ⇒ *IL f*
definition
gmap :: (*'a* ⇒ *'b*) ⇒ *'a glist* ⇒ *'b glist*
where
gmap f a ≡ *case a of*
FL xs ⇒ *FL* (*map f xs*) |
IL g ⇒ *IL* (*f* ∘ *g*)
definition
gtake :: *inat* ⇒ *'a glist* ⇒ *'a glist*
where
gtake n a ≡ *case n of*
Fin m ⇒ *FL* (*case a of*
FL xs ⇒ *xs* ↓ *m* |
IL f ⇒ *f* ⇓ *m*) |
∞ ⇒ *a*
definition
gdrop :: *inat* ⇒ *'a glist* ⇒ *'a glist*
where
gdrop n a ≡ *case n of*
Fin m ⇒ (*case a of*
FL xs ⇒ *FL* (*xs* ↑ *m*) |
IL f ⇒ *IL* (*f* ⇑ *m*)) |
∞ ⇒ *FL* []
definition
gset :: *'a glist* ⇒ *'a set*
where
gset a ≡ *case a of*
FL xs ⇒ *set xs* |
IL f ⇒ *range f*

definition
  gnth :: *'a glist ⇒ nat ⇒ 'a* (**infixl** $!_g$ *100*)
where
  $a\ !_g\ n \equiv$ *case a of*
    FL xs ⇒ xs ! n |
    IL f ⇒ f n

abbreviation (*xsymbols*)
  g-take' :: *'a glist ⇒ inat ⇒ 'a glist* (**infixl** $\downarrow_g$ *100*)
where
  $a \downarrow_g n \equiv$ *gtake n a*
abbreviation (*xsymbols*)
  g-drop' :: *'a glist ⇒ inat ⇒ 'a glist* (**infixl** $\uparrow_g$ *100*)
where
  $a \uparrow_g n \equiv$ *gdrop n a*
syntax (*HTML* **output**)
  g-take' :: *'a glist ⇒ inat ⇒ 'a glist* (**infixl** $\downarrow_g$ *100*)
  g-drop' :: *'a glist ⇒ inat ⇒ 'a glist* (**infixl** $\uparrow_g$ *100*)

11.2.2 *glength*

lemma *glength-fin*[*simp*]: *glength* (*FL xs*) = *Fin* (*length xs*)
⟨*proof*⟩

lemma *glength-infin*[*simp*]: *glength* (*IL f*) = ∞
⟨*proof*⟩

lemma *gappend-glength*[*simp*]: *glength* (a @$_g$ b) = *glength a* + *glength b*
⟨*proof*⟩

lemma *gmap-glength*[*simp*]: *glength* (*gmap f a*) = *glength a*
⟨*proof*⟩

lemma *glength-0-conv*[*simp*]: (*glength a* = *0*) = (a = *FL* [])
⟨*proof*⟩

lemma *glength-greater-0-conv*[*simp*]: (*0* < *glength a*) = ($a \neq$ *FL* [])
⟨*proof*⟩

lemma *glength-gSuc-conv*:
  (*glength a* = *iSuc n*) =
  ($\exists x\ b.\ a = x\ \#_g\ b \wedge$ *glength b* = *n*)
⟨*proof*⟩
 thm *i-take-first*
 ⟨*proof*⟩

lemma *gSuc-glength-conv*:
  (*iSuc n* = *glength a*) =
  ($\exists x\ b.\ a = x\ \#_g\ b \wedge$ *glength b* = *n*)

⟨*proof*⟩

11.2.3 $@_g$ − gappend

thm *append-Nil*
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*]: $(a \ @_g\ b = a \ @_g\ c) = (glength\ a = \infty \vee b = c)$
⟨*proof*⟩
lemma *same-gappend-eq*:
$glength\ a < \infty \Longrightarrow (a \ @_g\ b = a \ @_g\ c) = (b = c)$
⟨*proof*⟩

11.2.4 *gmap*

lemma *gmap-gappend*[*simp*]: $gmap\ f\ (a \ @_g\ b) = gmap\ f\ a \ @_g\ gmap\ f\ b$
⟨*proof*⟩

thm *map-map*
lemma *gmap-gmap*[*simp*]: $gmap\ f\ (gmap\ g\ a) = gmap\ (f \circ g)\ a$
⟨*proof*⟩

thm *map-eq-conv*
lemma *gmap-eq-conv*[*simp*]: $(gmap\ f\ a = gmap\ g\ a) = (\forall x{\in}gset\ a.\ f\ x = g\ x)$
⟨*proof*⟩

thm *map-cong*
lemma *gmap-cong*:
$[\![\ a = b; \bigwedge x.\ x \in gset\ b \Longrightarrow f\ x = g\ x\ ]\!] \Longrightarrow gmap\ f\ a = gmap\ g\ b$
⟨*proof*⟩

thm *map-is-Nil-conv*
lemma *gmap-is-Nil-conv*: $(gmap\ f\ a = FL\ []) = (a = FL\ [])$

⟨*proof*⟩

lemma *gmap-eq-imp-glength-eq*:
gmap f a = gmap f b ⟹ glength a = glength b
⟨*proof*⟩

11.2.5 *gset*

thm *set-append*
lemma *gset-gappend*[*simp*]:
gset (a @$_g$ b) =
(case a of FL a′ ⇒ set a′ ∪ gset b | IL a′ ⇒ range a′)
⟨*proof*⟩
lemma *gset-gappend-if*:
gset (a @$_g$ b) =
(if glength a < ∞ then gset a ∪ gset b else gset a)
⟨*proof*⟩

thm *set-empty*
lemma *gset-empty*[*simp*]: *(gset a = {}) = (a = FL [])*
⟨*proof*⟩

thm *set-map*
lemma *gset-gmap*[*simp*]: *gset (gmap f a) = f ‘ gset a*
⟨*proof*⟩

thm *card-length*
lemma *icard-glength*: *icard (gset a) ≤ glength a*
⟨*proof*⟩

11.2.6 $!_g$ − gnth

thm *nth-Cons-0*
lemma *gnth-gCons-0*[*simp*]: *(x #$_g$ a) !$_g$ 0 = x*
⟨*proof*⟩

thm *nth-Cons-Suc*
lemma *gnth-gCons-Suc*[*simp*]: *(x #$_g$ a) !$_g$ Suc n = a !$_g$ n*
⟨*proof*⟩

thm *nth-append*
lemma *gnth-gappend*:
(a @$_g$ b) !$_g$ n =
(if Fin n < glength a then a !$_g$ n
else b !$_g$ (n − the-Fin (glength a)))
⟨*proof*⟩

thm *nth-append-length-plus*
lemma *gnth-gappend-length-plus*[*simp*]: *(FL xs @$_g$ b) !$_g$ (length xs + n) = b !$_g$ n*
⟨*proof*⟩

thm *nth-map*
lemma *gmap-gnth*[*simp*]: $Fin\ n < glength\ a \Longrightarrow gmap\ f\ a\ !_g\ n = f\ (a\ !_g\ n)$
⟨*proof*⟩

thm *in-set-conv-nth*
lemma *in-gset-cong-gnth*: $(x \in gset\ a) = (\exists\, i.\ Fin\ i < glength\ a \wedge a\ !_g\ i = x)$
⟨*proof*⟩

11.2.7 *gtake* and *gdrop*

thm *take-0*
lemma *gtake-0*[*simp*]: $a \downarrow_g 0 = FL\ []$
⟨*proof*⟩

thm *drop-0*
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*⟩

thm *take-all*
lemma *gtake-all*[*simp*]: $glength\ a \leq n \Longrightarrow a \downarrow_g n = a$
⟨*proof*⟩

thm *drop-all*
lemma *gdrop-all*[*simp*]: $glength\ a \leq n \Longrightarrow a \uparrow_g n = FL\ []$
⟨*proof*⟩

thm *take-Suc-Cons*
lemma *gtake-iSuc-gCons*[*simp*]: $(x\ \#_g\ a) \downarrow_g (iSuc\ n) = x\ \#_g\ a \downarrow_g n$
⟨*proof*⟩

thm *drop-Suc-Cons*
lemma *gdrop-iSuc-gCons*[*simp*]: $(x\ \#_g\ a) \uparrow_g (iSuc\ n) = a \uparrow_g n$
⟨*proof*⟩

thm *take-Suc*
lemma *gtake-iSuc*: $a \neq FL\ [] \Longrightarrow a \downarrow_g (iSuc\ n) = a\ !_g\ 0\ \#_g\ (a \uparrow_g (iSuc\ 0) \downarrow_g n)$
⟨*proof*⟩

thm *drop-Suc*
lemma *gdrop-iSuc*: $a \uparrow_g (iSuc\ n) = a \uparrow_g (iSuc\ 0) \uparrow_g n$
⟨*proof*⟩

thm *nth-via-drop*
lemma *gnth-via-grop*: $a \uparrow_g (Fin\ n) = x\ \#_g\ b \Longrightarrow a\ !_g\ n = x$
⟨*proof*⟩

thm *take-Suc-conv-app-nth*[*no-vars*]
thm *i-take-Suc-conv-app-nth*
lemma *gtake-iSuc-conv-gapp-gnth*:
$Fin\ n < glength\ a \Longrightarrow a \downarrow_g Fin\ (Suc\ n) = a \downarrow_g (Fin\ n)\ @_g\ FL\ [a\ !_g\ n]$
⟨*proof*⟩

thm *drop-Suc-conv-tl*
lemma *gdrop-iSuc-conv-tl*:
$Fin\ n < glength\ a \Longrightarrow a\ !_g\ n\ \#_g\ a \uparrow_g Fin\ (Suc\ n) = a \uparrow_g Fin\ n$
⟨*proof*⟩

thm *length-take*
lemma *glength-gtake*[*simp*]: $glength\ (a \downarrow_g n) = min\ (glength\ a)\ n$
⟨*proof*⟩

thm *length-drop*
lemma *glength-drop*[*simp*]: $glength\ (a \uparrow_g (Fin\ n)) = glength\ a - (Fin\ n)$
⟨*proof*⟩

end

12 ListInf-Prefix: Prefices on finite and infinite lists

theory *ListInf-Prefix*
imports $ISABELLE-HOME/src/HOL/Library/List-Prefix ListInf
begin

12.1 Additional list prefix results

lemma *prefix-eq-prefix-take-ex*: $(xs \leq ys) = (\exists n.\ ys \downarrow n = xs)$
⟨*proof*⟩

lemma *prefix-take-eq-prefix-take-ex*: $(ys \downarrow (length\ xs) = xs) = (\exists n.\ ys \downarrow n = xs)$
⟨*proof*⟩

lemma *prefix-eq-prefix-take*: $(xs \leq ys) = (ys \downarrow (length\ xs) = xs)$
⟨*proof*⟩

lemma *strict-prefix-take-eq-strict-prefix-take-ex*:
$(ys \downarrow (length\ xs) = xs \wedge xs \neq ys) =$

$((\exists\, n.\ ys \downarrow n = xs) \wedge xs \neq ys)$
⟨*proof*⟩

lemma *strict-prefix-eq-strict-prefix-take-ex*: $(xs < ys) = ((\exists\, n.\ ys \downarrow n = xs) \wedge xs \neq ys)$
⟨*proof*⟩
lemma *strict-prefix-eq-strict-prefix-take*: $(xs < ys) = (ys \downarrow (length\ xs) = xs \wedge xs \neq ys)$
⟨*proof*⟩

lemma *take-imp-prefix*: $xs \downarrow n \leq xs$
⟨*proof*⟩

lemma *eq-imp-prefix*: $xs = (ys::'a\ list) \Longrightarrow xs \leq ys$
⟨*proof*⟩

lemma *le-take-imp-prefix*: $a \leq b \Longrightarrow xs \downarrow a \leq xs \downarrow b$
⟨*proof*⟩

lemma *take-prefix-imp-le*:
$[\![\ a \leq length\ xs;\ xs \downarrow a \leq xs \downarrow b\ ]\!] \Longrightarrow a \leq b$
⟨*proof*⟩

lemma *take-prefix-le-conv*:
$a \leq length\ xs \Longrightarrow (xs \downarrow a \leq xs \downarrow b) = (a \leq b)$
⟨*proof*⟩
lemma *append-imp-prefix*[*simp*, *intro*]: $a \leq a\ @\ b$
⟨*proof*⟩

lemma *prefix-imp-take-eq*:
$[\![\ n \leq length\ xs;\ xs \leq ys\ ]\!] \Longrightarrow xs \downarrow n = ys \downarrow n$
⟨*proof*⟩

lemma *prefix-length-le-eq-conv*: $(xs \leq ys \wedge length\ ys \leq length\ xs) = (xs = ys)$
⟨*proof*⟩

lemma *take-length-prefix-conv*:
$length\ xs \leq length\ ys \Longrightarrow (ys \downarrow length\ xs \leq xs) = (xs \leq ys)$
⟨*proof*⟩

lemma *append-eq-imp-take*:
$[\![\ k \leq length\ xs;\ length\ r1 = k;\ r1\ @\ r2 = xs\ ]\!] \Longrightarrow r1 = xs \downarrow k$
⟨*proof*⟩

lemma *take-the-conv*:
$xs \downarrow k = (if\ length\ xs \leq k\ then\ xs\ else\ (THE\ r.\ length\ r = k \wedge (\exists\, r2.\ r\ @\ r2 = xs)))$

⟨*proof*⟩

lemma *prefix-refl*: $xs \leq (xs::'a\ list)$
⟨*proof*⟩

lemma *prefix-trans*: $[\![\ xs \leq ys;\ (ys::'a\ list) \leq zs\ ]\!] \Longrightarrow xs \leq zs$
⟨*proof*⟩

lemma *prefix-antisym*: $[\![\ xs \leq ys;\ (ys::'a\ list) \leq xs\ ]\!] \Longrightarrow xs = ys$
⟨*proof*⟩

12.2 Counting equal pairs

Counting number of equal elements in two lists

definition
  mirror-pair :: $('a \times 'b) \Rightarrow ('b \times 'a)$
where
  mirror-pair p $\equiv$ (*snd p*, *fst p*)

lemma *zip-mirror*[*rule-format*]:
  $[\![\ i < min\ (length\ xs)\ (length\ ys);$
    $p1 = (zip\ xs\ ys)\ !\ i;\ p2 = (zip\ ys\ xs)\ !\ i\ ]\!] \Longrightarrow$
  mirror-pair p1 = *p2*
⟨*proof*⟩

definition
  equal-pair :: $('a \times 'a) \Rightarrow bool$
where
  equal-pair p $\equiv$ (*fst p* = *snd p*)

lemma *mirror-pair-equal*: *equal-pair* (*mirror-pair p*) = (*equal-pair p*)
⟨*proof*⟩

primrec
  equal-pair-count :: $('a \times 'a)\ list \Rightarrow nat$
where
  equal-pair-count [] = *0*
| *equal-pair-count* (*p* # *ps*) = (
    if (*fst p* = *snd p*)
      then Suc (*equal-pair-count ps*)
      else 0)

lemma *equal-pair-count-le*: *equal-pair-count xs* $\leq$ *length xs*
⟨*proof*⟩

lemma *equal-pair-count-0*:
  fst (*hd ps*) $\neq$ *snd* (*hd ps*) $\Longrightarrow$ *equal-pair-count ps* = *0*
⟨*proof*⟩

lemma *equal-pair-count-Suc*:
equal-pair-count ((*a*, *a*) # *ps*) = *Suc* (*equal-pair-count ps*)
⟨*proof*⟩

lemma *equal-pair-count-eq-pairwise*[*rule-format*]:
⟦ *length ps1* = *length ps2*;
∀ *i*<*length ps2*. *equal-pair* (*ps1* ! *i*) = *equal-pair*(*ps2* ! *i*) ⟧
⟹ *equal-pair-count ps1* = *equal-pair-count ps2*
⟨*proof*⟩

lemma *equal-pair-count-mirror-pairwise*[*rule-format*]:
⟦ *length ps1* = *length ps2*;
∀ *i*<*length ps2*. *ps1* ! *i* = *mirror-pair* (*ps2* ! *i*) ⟧
⟹ *equal-pair-count ps1* = *equal-pair-count ps2*
⟨*proof*⟩
thm *mirror-pair-equal*
⟨*proof*⟩

lemma *equal-pair-count-correct*:
⋀*i*. *i* < *equal-pair-count ps* ⟹ *equal-pair* (*ps* ! *i*)
⟨*proof*⟩
thm *equal-pair-count-correct*

lemma *equal-pair-count-maximality-aux*[*rule-format*]: ⋀*i*.
i = *equal-pair-count ps* ⟹ *length ps* = *i* ∨ ¬ *equal-pair* (*ps* ! *i*)
⟨*proof*⟩
thm *equal-pair-count-maximality-aux*
corollary *equal-pair-count-maximality1a*[*rule-format*]:
equal-pair-count ps = *length ps* ∨ ¬ *equal-pair* (*ps*!*equal-pair-count ps*)
thm *equal-pair-count-maximality-aux*[*of equal-pair-count ps ps*, *simplified*]
⟨*proof*⟩

corollary *equal-pair-count-maximality1b*[*rule-format*]:
equal-pair-count ps ≠ *length ps* ⟹
¬ *equal-pair* (*ps*!*equal-pair-count ps*)
⟨*proof*⟩

lemma *equal-pair-count-maximality2a*[*rule-format*]:
equal-pair-count ps = *length ps* ∨ (∗ *either all pairs are equal* ∗)
(∀ *i*≥*equal-pair-count ps*.(∃ *j*≤*i*. ¬*equal-pair* (*ps* ! *j*)))
⟨*proof*⟩
thm *equal-pair-count-maximality1b equal-pair-count-le*
⟨*proof*⟩
corollary *equal-pair-count-maximality2b*[*rule-format*]:
equal-pair-count ps ≠ *length ps* ⟹

$\forall i{\geq}$*equal-pair-count ps*.($\exists j{\leq}i$. $\neg$*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
thm
equal-pair-count-correct[*no-vars*]
equal-pair-count-maximality[*no-vars*]

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*]:
inf-prefix-length xs ys = *inf-prefix-length ys xs*
⟨*proof*⟩
thm *equal-pair-count-mirror-pairwise*
⟨*proof*⟩

lemma *inf-prefix-length-leL*[*intro*]:
inf-prefix-length xs ys ≤ *length xs*
⟨*proof*⟩
corollary *inf-prefix-length-leR*[*intro*]:
inf-prefix-length xs ys ≤ *length ys*
thm *inf-prefix-length-commute*[*of xs ys*]
⟨*proof*⟩
lemmas *inf-prefix-length-le* =
inf-prefix-length-leL
inf-prefix-length-leR

lemma *inf-prefix-length-le-min*[*rule-format*]:
inf-prefix-length xs ys ≤ *min* (*length xs*) (*length ys*)
⟨*proof*⟩

thm *equal-pair-count-0*
lemma *hd-inf-prefix-length-0*:
hd xs ≠ *hd ys* ⟹ *inf-prefix-length xs ys* = *0*
⟨*proof*⟩

lemma *inf-prefix-length-NilL*[*simp*]: *inf-prefix-length* [] *ys* = *0*
⟨*proof*⟩
lemma *inf-prefix-length-NilR*[*simp*]: *inf-prefix-length xs* [] = *0*
⟨*proof*⟩

thm *equal-pair-count-Suc*
lemma *inf-prefix-length-Suc*[*simp*]:
inf-prefix-length (*a* # *xs*) (*a* # *ys*) = *Suc* (*inf-prefix-length xs ys*)
⟨*proof*⟩

thm *equal-pair-count-correct*
lemma *inf-prefix-length-correct*:
i < *inf-prefix-length xs ys* ⟹ *xs* ! *i* = *ys* ! *i*
thm *order-less-le-trans*[*OF - inf-prefix-length-leL*]
⟨*proof*⟩

corollary *nth-neq-imp-inf-prefix-length-le*:
xs ! *i* ≠ *ys* ! *i* ⟹ *inf-prefix-length xs ys* ≤ *i*
⟨*proof*⟩

thm *equal-pair-count-maximality1b*
lemma *inf-prefix-length-maximality1*[*rule-format*]:
inf-prefix-length xs ys ≠ *min* (*length xs*) (*length ys*) ⟹
xs ! (*inf-prefix-length xs ys*) ≠ *ys* ! (*inf-prefix-length xs ys*)
thm *equal-pair-count-maximality1b*[*of zip xs ys*]
thm *equal-pair-count-maximality1b*[*of zip xs ys*, *folded inf-prefix-length-def*]
⟨*proof*⟩

thm *equal-pair-count-maximality2b*
corollary *inf-prefix-length-maximality2*[*rule-format*]:
⟦ *inf-prefix-length xs ys* ≠ *min* (*length xs*) (*length ys*);
inf-prefix-length xs ys ≤ *i* ⟧ ⟹
∃*j*≤*i*. *xs* ! *j* ≠ *ys* ! *j*
⟨*proof*⟩

lemmas *inf-prefix-length-maximality* =
inf-prefix-length-maximality1 inf-prefix-length-maximality2

lemma *inf-prefix-length-append*[*simp*]:
inf-prefix-length (*zs* @ *xs*) (*zs* @ *ys*) =
length zs + *inf-prefix-length xs ys*
⟨*proof*⟩

lemma *inf-prefix-length-take-correct*:
n ≤ *inf-prefix-length xs ys* ⟹ *xs* ↓ *n* = *ys* ↓ *n*
⟨*proof*⟩

lemma *inf-prefix-length-0-imp-hd-neq*:

⟦ *xs* ≠ []; *ys* ≠ []; *inf-prefix-length xs ys* = *0* ⟧ ⟹ *hd xs* ≠ *hd ys*
⟨*proof*⟩
thm *inf-prefix-length-maximality2*[*of xs ys 0*]
⟨*proof*⟩

12.4 Prefix infimum

definition
inf-prefix :: *'a list* ⇒ *'a list* ⇒ *'a list* (**infixl** ⊓ *70*)
where
xs ⊓ *ys* ≡ *xs* ↓ (*inf-prefix-length xs ys*)

lemma *length-inf-prefix*: *length* (*xs* ⊓ *ys*) = *inf-prefix-length xs ys*
⟨*proof*⟩

lemma *inf-prefix-commute*: *xs* ⊓ *ys* = *ys* ⊓ *xs*
⟨*proof*⟩

lemma *inf-prefix-takeL*: *xs* ⊓ *ys* = *xs* ↓ (*inf-prefix-length xs ys*)
⟨*proof*⟩
lemma *inf-prefix-takeR*: *xs* ⊓ *ys* = *ys* ↓ (*inf-prefix-length xs ys*)
⟨*proof*⟩

thm *inf-prefix-length-correct*
lemma *inf-prefix-correct*: *i* < *length* (*xs* ⊓ *ys*) ⟹ *xs* ! *i* = *ys* ! *i*
⟨*proof*⟩
corollary *inf-prefix-correctL*:
i < *length* (*xs* ⊓ *ys*) ⟹ (*xs* ⊓ *ys*) ! *i* = *xs* ! *i*
⟨*proof*⟩
corollary *inf-prefix-correctR*:
i < *length* (*xs* ⊓ *ys*) ⟹ (*xs* ⊓ *ys*) ! *i* = *ys* ! *i*
⟨*proof*⟩

thm *inf-prefix-length-take-correct*
lemma *inf-prefix-take-correct*:
n ≤ *length* (*xs* ⊓ *ys*) ⟹ *xs* ↓ *n* = *ys* ↓ *n*
⟨*proof*⟩

lemma *is-inf-prefix*[*rule-format*]:
⟦ *length zs* = *length* (*xs* ⊓ *ys*);
⋀*i*. *i* < *length* (*xs* ⊓ *ys*) ⟹ *zs* ! *i* = *xs* ! *i* ∧ *zs* ! *i* = *ys* ! *i* ⟧ ⟹
zs = *xs* ⊓ *ys*
⟨*proof*⟩

thm *hd-inf-prefix-length-0*
lemma *hd-inf-prefix-Nil*: *hd xs* ≠ *hd ys* ⟹ *xs* ⊓ *ys* = []
⟨*proof*⟩

thm *inf-prefix-length-0-imp-hd-neq*

lemma *inf-prefix-Nil-imp-hd-neq*:
⟦ $xs \neq []$; $ys \neq []$; $xs \sqcap ys = []$ ⟧ ⟹ $hd\ xs \neq hd\ ys$
⟨*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*⟩
thm *inf-prefix-correctL inf-prefix-correctR*
⟨*proof*⟩

lemma *hd-neq-inf-prefix-append*:
$hd\ xs \neq hd\ ys \Longrightarrow (zs\ @\ xs) \sqcap (zs\ @\ ys) = zs$
⟨*proof*⟩

thm *inf-prefix-length-NilL*
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*
thm *inf-prefix-Nil*

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*: $xs \sqcap ys \leq xs$
⟨*proof*⟩
lemma *inf-prefix-le2*: $xs \sqcap ys \leq ys$
⟨*proof*⟩

thm *min-le-iff-conj*
lemma *le-inf-prefix-iff*: $(x \leq y \sqcap z) = (x \leq y \wedge x \leq z)$
⟨*proof*⟩
lemma *le-imp-le-inf-prefix*: ⟦ $x \leq y$; $x \leq z$ ⟧ ⟹ $x \leq y \sqcap z$
thm *le-inf-prefix-iff*[*THEN iffD2*]
⟨*proof*⟩

interpretation *prefix*:
semilattice-inf
op $\leq$ :: *'a list* ⇒ *'a list* ⇒ *bool*

op < :: *'a list* ⇒ *'a list* ⇒ *bool*
op ⊓ :: *'a list* ⇒ *'a list* ⇒ *'a list*
⟨*proof*⟩

12.5 Prefices for infinite lists

thm *prefix-def*
term λ*x y*. *x* ≤ (*y*::*'a list*)
definition
iprefix :: *'a list* ⇒ *'a ilist* ⇒ *bool* (**infixl** ⊑ *50*)
where
xs ⊑ *f* ≡ ∃ *g*. *f* = *xs* ⌢ *g*

thm *prefix-eq-prefix-take*
lemma *iprefix-eq-iprefix-take*: (*xs* ⊑ *f*) = (*f* ⇓ *length xs* = *xs*)
⟨*proof*⟩
thm *i-append-i-take-i-drop-id*
⟨*proof*⟩

thm *prefix-take-eq-prefix-take-ex*
lemma *iprefix-take-eq-iprefix-take-ex*:
(*f* ⇓ *length xs* = *xs*) = (∃ *n*. *f* ⇓ *n* = *xs*)
⟨*proof*⟩

thm *prefix-eq-prefix-take-ex*
lemma *iprefix-eq-iprefix-take-ex*: (*xs* ⊑ *f*) = (∃ *n*. *f* ⇓ *n* = *xs*)
⟨*proof*⟩

thm *take-imp-prefix*
lemma *i-take-imp-iprefix*[*intro*]: *f* ⇓ *n* ⊑ *f*
⟨*proof*⟩

thm *take-prefix-le-conv*
lemma *i-take-prefix-le-conv*: (*f* ⇓ *a* ≤ *f* ⇓ *b*) = (*a* ≤ *b*)
⟨*proof*⟩

thm *append-imp-prefix*
lemma *i-append-imp-iprefix*[*simp*,*intro*]: *xs* ⊑ *xs* ⌢ *f*
⟨*proof*⟩

thm *prefix-imp-take-eq*
lemma *iprefix-imp-take-eq*:
⟦ *n* ≤ *length xs*; *xs* ⊑ *f* ⟧ ⟹ *xs* ↓ *n* = *f* ⇓ *n*
⟨*proof*⟩

thm *prefix-refl*
thm *prefix-trans*
thm *prefix-antisym*
lemma *prefix-iprefix-trans*: ⟦ *xs* ≤ *ys*; *ys* ⊑ *f* ⟧ ⟹ *xs* ⊑ *f*

⟨*proof*⟩

thm *take-length-prefix-conv*
lemma *i-take-length-prefix-conv*: $(f \Downarrow length\ xs \leq xs) = (xs \sqsubseteq f)$
thm *prefix-length-le-eq-conv*
⟨*proof*⟩

thm *List-Prefix.prefixI*
lemma *iprefixI*[*intro?*]: $f = xs \frown g \Longrightarrow xs \sqsubseteq f$
⟨*proof*⟩
thm *List-Prefix.prefixE*
lemma *iprefixE*[*elim?*]: $[\![\ xs \sqsubseteq f;\ \bigwedge g.\ f = xs \frown g \Longrightarrow C\ ]\!] \Longrightarrow C$
⟨*proof*⟩

thm *List-Prefix.Nil-prefix*
lemma *Nil-iprefix*[*iff*]: $[] \sqsubseteq f$
⟨*proof*⟩

thm *List-Prefix.same-prefix-prefix*
lemma *same-prefix-iprefix*[*simp*]: $(xs\ @\ ys \sqsubseteq xs \frown f) = (ys \sqsubseteq f)$
⟨*proof*⟩

thm *List-Prefix.prefix-prefix*
lemma *prefix-iprefix*[*simp*]: $xs \leq ys \Longrightarrow xs \sqsubseteq ys \frown f$
⟨*proof*⟩

thm *List-Prefix.append-prefixD*
lemma *append-iprefixD*: $xs\ @\ ys \sqsubseteq f \Longrightarrow xs \sqsubseteq f$
⟨*proof*⟩

lemma *iprefix-length-le-imp-prefix*:
$[\![\ xs \sqsubseteq ys \frown f;\ length\ xs \leq length\ ys\ ]\!] \Longrightarrow xs \leq ys$
⟨*proof*⟩

thm *List-Prefix.prefix-append*
lemma *iprefix-i-append*:
$(xs \sqsubseteq ys \frown f) = (xs \leq ys \vee (\exists\, zs.\ xs = ys\ @\ zs \wedge zs \sqsubseteq f))$
⟨*proof*⟩

lemma *i-append-one-iprefix*:
$xs \sqsubseteq f \Longrightarrow xs\ @\ [f\ (length\ xs)] \sqsubseteq f$
⟨*proof*⟩

lemma *iprefix-same-length-le*:
$[\![\ xs \sqsubseteq f;\ ys \sqsubseteq f;\ length\ xs \leq length\ ys\ ]\!] \Longrightarrow xs \leq ys$
⟨*proof*⟩

thm *List-Prefix.prefix-same-cases*

lemma *iprefix-same-cases*:
$[\![xs \sqsubseteq f; ys \sqsubseteq f]\!] \Longrightarrow xs \leq ys \vee ys \leq xs$
⟨*proof*⟩

thm *List-Prefix.set-mono-prefix*
lemma *set-mono-iprefix*: $xs \sqsubseteq f \Longrightarrow set\ xs \subseteq range\ f$
⟨*proof*⟩

end
theory *ListInfinite*
imports
CommonSet/SetIntervalStep
ListInf/ListInf-Prefix
begin
end