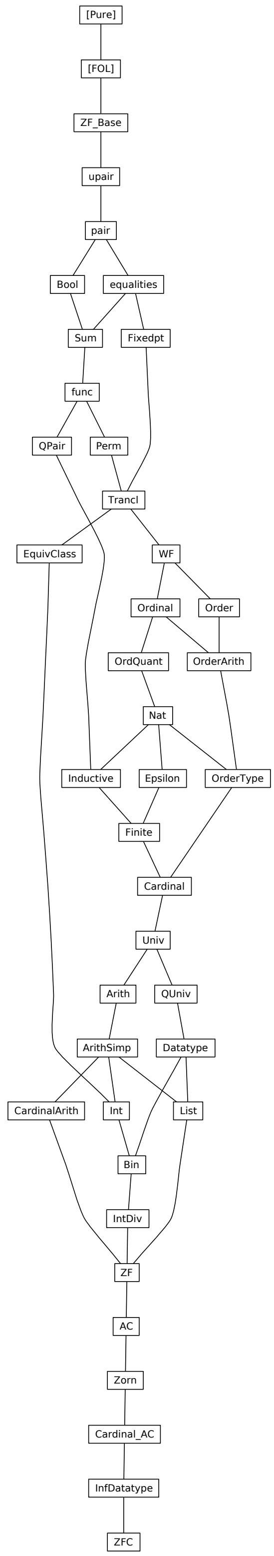
ZF

Lawrence C Paulson and others

January 18, 2026

Contents


[Pure]
[FOL]
ZF_Base
upair
pair
Bool
equalities
Sum
Fixedpt
func
QPair
Perm
Trancl
EquivClass
WF
Ordinal
Order
OrdQuant
OrderArith
Nat
Inductive
Epsilon
OrderType
Finite
Cardinal
Univ
Arith
QUniv
ArithSimp
Datatype
CardinalArith
Int
List
Bin
IntDiv
ZF
AC
Zorn
Cardinal_AC
InfDatatype
ZFC

1 Base of Zermelo-Fraenkel Set Theory

theory *ZF-Base*
imports *FOL*
begin

1.1 Signature

declare [[*eta-contract* = *false*]]

typedecl *i*
instance *i* :: *term* ⟨*proof*⟩

axiomatization *mem* :: [*i*, *i*] ⇒ *o* (**infixl** ‹∈› *50*) — membership relation
  and *zero* :: *i* (‹*0*›) — the empty set
  and *Pow* :: *i* ⇒ *i* — power sets
  and *Inf* :: *i* — infinite set
  and *Union* :: *i* ⇒ *i* (‹(‹*open-block notation*=‹*prefix* ⋃››⋃ -)› [*90*] *90*)
  and *PrimReplace* :: [*i*, [*i*, *i*] ⇒ *o*] ⇒ *i*

abbreviation *not-mem* :: [*i*, *i*] ⇒ *o* (**infixl** ‹∉› *50*) — negated membership relation
  where *x* ∉ *y* ≡ ¬ (*x* ∈ *y*)

1.2 Bounded Quantifiers

definition *Ball* :: [*i*, *i* ⇒ *o*] ⇒ *o*
  where *Ball*(*A*, *P*) ≡ ∀ *x*. *x*∈*A* ⟶ *P*(*x*)

definition *Bex* :: [*i*, *i* ⇒ *o*] ⇒ *o*
  where *Bex*(*A*, *P*) ≡ ∃ *x*. *x*∈*A* ∧ *P*(*x*)

syntax
  -Ball :: [*pttrn*, *i*, *o*] ⇒ *o* (‹(‹*indent*=*3 notation*=‹*binder* ∀∈››∀ -∈-./ -)› *10*)
  -Bex :: [*pttrn*, *i*, *o*] ⇒ *o* (‹(‹*indent*=*3 notation*=‹*binder* ∃∈››∃ -∈-./ -)› *10*)
syntax-consts
  -Ball ⇌ *Ball* **and**
  -Bex ⇌ *Bex*
translations
  ∀ *x*∈*A*. *P* ⇌ *CONST Ball*(*A*, λ*x*. *P*)
  ∃ *x*∈*A*. *P* ⇌ *CONST Bex*(*A*, λ*x*. *P*)

1.3 Variations on Replacement

definition *Replace* :: [*i*, [*i*, *i*] ⇒ *o*] ⇒ *i*
  where *Replace*(*A*,*P*) ≡ *PrimReplace*(*A*, λ*x y*. (∃!*z*. *P*(*x*,*z*)) ∧ *P*(*x*,*y*))

syntax
  -Replace :: [*pttrn*, *pttrn*, *i*, *o*] ⇒ *i* (‹(‹*indent*=*1 notation*=‹*mixfix relational replacement*››{- ./ - ∈ -, -})›)

syntax-consts
-Replace ⇌ *Replace*
translations
$\{y.\ x{\in}A,\ Q\}$ ⇌ *CONST Replace*$(A,\ \lambda x\ y.\ Q)$

definition *RepFun* :: $[i,\ i \Rightarrow i] \Rightarrow i$
where *RepFun*$(A,f) \equiv \{y\ .\ x{\in}A,\ y{=}f(x)\}$

syntax
-RepFun :: $[i,\ pttrn,\ i] \Rightarrow i$ (‹(‹indent=1 notation=‹mixfix functional replacement››{- ./ - ∈ -})› [51,0,51])
syntax-consts
-RepFun ⇌ *RepFun*
translations
$\{b.\ x{\in}A\}$ ⇌ *CONST RepFun*$(A,\ \lambda x.\ b)$

definition *Collect* :: $[i,\ i \Rightarrow o] \Rightarrow i$
where *Collect*$(A,P) \equiv \{y\ .\ x{\in}A,\ x{=}y \wedge P(x)\}$

syntax
-Collect :: $[pttrn,\ i,\ o] \Rightarrow i$ (‹(‹indent=1 notation=‹mixfix set comprehension››{- ∈ - ./ -})›)
syntax-consts
-Collect ⇌ *Collect*
translations
$\{x{\in}A.\ P\}$ ⇌ *CONST Collect*$(A,\ \lambda x.\ P)$

1.4 General union and intersection

definition *Inter* :: $i \Rightarrow i$ (‹(‹open-block notation=‹prefix ⋂››⋂ -)› [90] 90)
where $\bigcap(A) \equiv \{\ x{\in}\bigcup(A)\ .\ \forall y{\in}A.\ x{\in}y\}$

syntax
-UNION :: $[pttrn,\ i,\ i] \Rightarrow i$ (‹(‹indent=3 notation=‹binder ⋃∈››⋃ -∈-./ -)› 10)
-INTER :: $[pttrn,\ i,\ i] \Rightarrow i$ (‹(‹indent=3 notation=‹binder ⋂∈››⋂ -∈-./ -)› 10)
syntax-consts
-UNION == *Union* **and**
-INTER == *Inter*
translations
$\bigcup x{\in}A.\ B$ == *CONST Union*$(\{B.\ x{\in}A\})$
$\bigcap x{\in}A.\ B$ == *CONST Inter*$(\{B.\ x{\in}A\})$

1.5 Finite sets and binary operations

definition *Upair* :: $[i,\ i] \Rightarrow i$
where *Upair*$(a,b) \equiv \{y.\ x{\in}Pow(Pow(0)),\ (x{=}0 \wedge y{=}a)\ |\ (x{=}Pow(0) \wedge y{=}b)\}$

definition *Subset* :: $[i, i] \Rightarrow o$ (**infixl** ‹⊆› *50*) — subset relation
  where *subset-def*: $A \subseteq B \equiv \forall x{\in}A.\ x{\in}B$

definition *Diff* :: $[i, i] \Rightarrow i$ (**infixl** ‹−› *65*) — set difference
  where $A - B \equiv \{\ x{\in}A\ .\ \neg(x{\in}B)\ \}$

definition *Un* :: $[i, i] \Rightarrow i$ (**infixl** ‹∪› *65*) — binary union
  where $A \cup B \equiv \bigcup(\mathit{Upair}(A,B))$

definition *Int* :: $[i, i] \Rightarrow i$ (**infixl** ‹∩› *70*) — binary intersection
  where $A \cap B \equiv \bigcap(\mathit{Upair}(A,B))$

definition *cons* :: $[i, i] \Rightarrow i$
  where $\mathit{cons}(a,A) \equiv \mathit{Upair}(a,a) \cup A$

definition *succ* :: $i \Rightarrow i$
  where $\mathit{succ}(i) \equiv \mathit{cons}(i, i)$

nonterminal *is*
syntax
  :: *i* ⇒ *is* (‹-›)
  -Enum :: [*i*, *is*] ⇒ *is* (‹-,/ -›)
  -Finset :: *is* ⇒ *i* (‹(‹*indent=1 notation=*‹*mixfix set enumeration*››{-})›)
translations
  {*x*, *xs*} == *CONST cons*(*x*, {*xs*})
  {*x*} == *CONST cons*(*x*, *0*)

1.6 Axioms

axiomatization
where
  extension: $A = B \longleftrightarrow A \subseteq B \wedge B \subseteq A$ **and**
  Union-iff: $A \in \bigcup(C) \longleftrightarrow (\exists B{\in}C.\ A{\in}B)$ **and**
  Pow-iff: $A \in \mathit{Pow}(B) \longleftrightarrow A \subseteq B$ **and**

  infinity: $0 \in \mathit{Inf} \wedge (\forall y{\in}\mathit{Inf}.\ \mathit{succ}(y) \in \mathit{Inf})$ **and**

  foundation: $A = 0 \vee (\exists x{\in}A.\ \forall y{\in}x.\ y{\notin}A)$ **and**

  replacement: $(\forall x{\in}A.\ \forall y\ z.\ P(x,y) \wedge P(x,z) \longrightarrow y = z) \Longrightarrow$
    $b \in \mathit{PrimReplace}(A,P) \longleftrightarrow (\exists x{\in}A.\ P(x,b))$

1.7 Definite descriptions – via Replace over the set "1"

definition *The* :: $(i \Rightarrow o) \Rightarrow i$ (**binder** ‹*THE* › *10*)
  where *the-def*: $\mathit{The}(P) \equiv \bigcup(\{y\ .\ x \in \{0\},\ P(y)\})$

definition *If* :: [*o*, *i*, *i*] ⇒ *i* (‹(‹*notation*=‹*mixfix if then else*››*if* (-)/ *then* (-)/ *else* (-))› [*10*] *10*)
  where *if-def*: *if P then a else b* ≡ *THE z*. *P* ∧ *z*=*a* | ¬*P* ∧ *z*=*b*

abbreviation (*input*)
  old-if :: [*o*, *i*, *i*] ⇒ *i* (‹*if* ′(-,-,-′)›)
  where *if*(*P*,*a*,*b*) ≡ *If*(*P*,*a*,*b*)

1.8 Ordered Pairing

definition *Pair* :: [*i*, *i*] ⇒ *i*
  where *Pair*(*a*,*b*) ≡ {{*a*,*a*}, {*a*,*b*}}

definition *fst* :: *i* ⇒ *i*
  where *fst*(*p*) ≡ *THE a*. ∃ *b*. *p* = *Pair*(*a*, *b*)

definition *snd* :: *i* ⇒ *i*
  where *snd*(*p*) ≡ *THE b*. ∃ *a*. *p* = *Pair*(*a*, *b*)

definition *split* :: [[*i*, *i*] ⇒ ′*a*, *i*] ⇒ ′*a*::{} — for pattern-matching
  where *split*(*c*) ≡ λ*p*. *c*(*fst*(*p*), *snd*(*p*))

nonterminal *tuple-args*
syntax
  :: *i* ⇒ *tuple-args* (‹-›)
  -Tuple-args :: [*i*, *tuple-args*] ⇒ *tuple-args* (‹-,/ -›)
  -Tuple :: [*i*, *tuple-args*] ⇒ *i* (‹(‹*indent*=*1* *notation*=‹*mixfix tuple enumeration*››⟨-,/ -⟩)›)
translations
  ⟨*x*, *y*, *z*⟩ == ⟨*x*, ⟨*y*, *z*⟩⟩
  ⟨*x*, *y*⟩ == *CONST Pair*(*x*, *y*)

nonterminal *patterns*
syntax
  -pattern :: *patterns* ⇒ *pttrn* (‹(‹*open-block notation*=‹*pattern tuple*››⟨-⟩)›)
  :: *pttrn* ⇒ *patterns* (‹-›)
  -patterns :: [*pttrn*, *patterns*] ⇒ *patterns* (‹-,/-›)
syntax-consts
  -pattern -patterns == *split*
translations
  λ⟨*x*,*y*,*zs*⟩.*b* == *CONST split*(λ*x* ⟨*y*,*zs*⟩.*b*)
  λ⟨*x*,*y*⟩.*b* == *CONST split*(λ*x y*. *b*)

definition *Sigma* :: [*i*, *i* ⇒ *i*] ⇒ *i*
  where *Sigma*(*A*,*B*) ≡ ⋃ *x*∈*A*. ⋃ *y*∈*B*(*x*). {⟨*x*,*y*⟩}

abbreviation *cart-prod* :: [*i*, *i*] ⇒ *i* (**infixr** ‹×› *80*) — Cartesian product

where $A \times B \equiv Sigma(A, \lambda\text{-}. B)$

1.9 Relations and Functions

definition *converse* :: $i \Rightarrow i$
where $converse(r) \equiv \{z.\ w{\in}r,\ \exists\, x\ y.\ w{=}\langle x{,}y\rangle \wedge z{=}\langle y{,}x\rangle\}$

definition *domain* :: $i \Rightarrow i$
where $domain(r) \equiv \{x.\ w{\in}r,\ \exists\, y.\ w{=}\langle x{,}y\rangle\}$

definition *range* :: $i \Rightarrow i$
where $range(r) \equiv domain(converse(r))$

definition *field* :: $i \Rightarrow i$
where $field(r) \equiv domain(r) \cup range(r)$

definition *relation* :: $i \Rightarrow o$ — recognizes sets of pairs
where $relation(r) \equiv \forall\, z{\in}r.\ \exists\, x\ y.\ z = \langle x{,}y\rangle$

definition *function* :: $i \Rightarrow o$ — recognizes functions; can have non-pairs
where $function(r) \equiv \forall\, x\ y.\ \langle x{,}y\rangle \in r \longrightarrow (\forall\, y'.\ \langle x{,}y'\rangle \in r \longrightarrow y = y')$

definition *Image* :: $[i, i] \Rightarrow i$ (**infixl** ‹‘‘› *90*) — image
where *image-def*: $r\ \text{‘‘}\ A \equiv \{y \in range(r).\ \exists\, x{\in}A.\ \langle x{,}y\rangle \in r\}$

definition *vimage* :: $[i, i] \Rightarrow i$ (**infixl** ‹−‘‘› *90*) — inverse image
where *vimage-def*: $r\ {-}\text{‘‘}\ A \equiv converse(r)\text{‘‘}A$

definition *restrict* :: $[i, i] \Rightarrow i$
where $restrict(r{,}A) \equiv \{z \in r.\ \exists\, x{\in}A.\ \exists\, y.\ z = \langle x{,}y\rangle\}$

definition *Lambda* :: $[i, i \Rightarrow i] \Rightarrow i$
where *lam-def*: $Lambda(A{,}b) \equiv \{\langle x{,}b(x)\rangle.\ x{\in}A\}$

definition *apply* :: $[i, i] \Rightarrow i$ (**infixl** ‹‘› *90*) — function application
where $f\text{‘}a \equiv \bigcup(f\text{‘‘}\{a\})$

definition *Pi* :: $[i, i \Rightarrow i] \Rightarrow i$
where $Pi(A{,}B) \equiv \{f{\in}Pow(Sigma(A{,}B)).\ A{\subseteq}domain(f) \wedge function(f)\}$

abbreviation *function-space* :: $[i, i] \Rightarrow i$ (**infixr** ‹→› *60*) — function space
where $A \to B \equiv Pi(A, \lambda\text{-}. B)$

syntax
-PROD :: [*pttrn*, *i*, *i*] ⇒ *i* (‹(‹*indent=3 notation=*‹*mixfix* ∏∈››∏ -∈-./ -)› *10*)
-SUM :: [*pttrn*, *i*, *i*] ⇒ *i* (‹(‹*indent=3 notation=*‹*mixfix* ∑∈››∑ -∈-./ -)› *10*)
-lam :: [*pttrn*, *i*, *i*] ⇒ *i* (‹(‹*indent=3 notation=*‹*mixfix* λ∈››λ-∈-./ -)› *10*)
syntax-consts
-PROD == *Pi* **and**
-SUM == *Sigma* **and**
-lam == *Lambda*
translations
∏*x*∈*A*. *B* == *CONST Pi*(*A*, λ*x*. *B*)
∑*x*∈*A*. *B* == *CONST Sigma*(*A*, λ*x*. *B*)
λ*x*∈*A*. *f* == *CONST Lambda*(*A*, λ*x*. *f*)

1.10 ASCII syntax

notation (*ASCII*)
cart-prod (**infixr** ‹∗› *80*) **and**
Int (**infixl** ‹*Int*› *70*) **and**
Un (**infixl** ‹*Un*› *65*) **and**
function-space (**infixr** ‹−>› *60*) **and**
Subset (**infixl** ‹<=› *50*) **and**
mem (**infixl** ‹:› *50*) **and**
not-mem (**infixl** ‹¬:› *50*)

syntax (*ASCII*)
-Ball :: [*pttrn*, *i*, *o*] ⇒ *o* (‹(‹*indent=3 notation=*‹*binder ALL*:››*ALL* -:-./ -)› *10*)
-Bex :: [*pttrn*, *i*, *o*] ⇒ *o* (‹(‹*indent=3 notation=*‹*binder EX*:››*EX* -:-./ -)› *10*)
-Collect :: [*pttrn*, *i*, *o*] ⇒ *i* (‹(‹*indent=1 notation=*‹*mixfix set comprehension*››{-: - ./ -})›)
-Replace :: [*pttrn*, *pttrn*, *i*, *o*] ⇒ *i* (‹(‹*indent=1 notation=*‹*mixfix relational replacement*››{- ./ -: -, -})›)
-RepFun :: [*i*, *pttrn*, *i*] ⇒ *i* (‹(‹*indent=1 notation=*‹*mixfix functional replacement*››{- ./ -: -})› [*51*,*0*,*51*])
-UNION :: [*pttrn*, *i*, *i*] ⇒ *i* (‹(‹*indent=3 notation=*‹*binder UN*:››*UN* -:-./ -)› *10*)
-INTER :: [*pttrn*, *i*, *i*] ⇒ *i* (‹(‹*indent=3 notation=*‹*binder INT*:››*INT* -:-./ -)› *10*)
-PROD :: [*pttrn*, *i*, *i*] ⇒ *i* (‹(‹*indent=3 notation=*‹*binder PROD*:››*PROD* -:-./ -)› *10*)
-SUM :: [*pttrn*, *i*, *i*] ⇒ *i* (‹(‹*indent=3 notation=*‹*binder SUM*:››*SUM* -:-./ -)› *10*)
-lam :: [*pttrn*, *i*, *i*] ⇒ *i* (‹(‹*indent=3 notation=*‹*binder lam*:››*lam* -:-./ -)› *10*)

-Tuple :: [*i*, *tuple-args*] ⇒ *i* (‹(‹*indent=1 notation=*‹*mixfix tuple enumeration*››<-,/ ->)›)
-pattern :: *patterns* ⇒ *pttrn* (‹<->›)

1.11 Substitution

lemma *subst-elem*: $[\![b{\in}A;\ \ a{=}b]\!] \Longrightarrow a{\in}A$
⟨*proof*⟩

1.12 Bounded universal quantifier

lemma *ballI* [*intro*!]: $[\![\bigwedge x.\ x{\in}A \Longrightarrow P(x)]\!] \Longrightarrow \forall x{\in}A.\ P(x)$
⟨*proof*⟩

lemmas *strip* = *impI allI ballI*

lemma *bspec* [*dest?*]: $[\![\forall x{\in}A.\ P(x);\ \ x{:}\ A]\!] \Longrightarrow P(x)$
⟨*proof*⟩

lemma *rev-ballE* [*elim*]:
$[\![\forall x{\in}A.\ P(x);\ \ x{\notin}A \Longrightarrow Q;\ \ P(x) \Longrightarrow Q]\!] \Longrightarrow Q$
⟨*proof*⟩

lemma *ballE*: $[\![\forall x{\in}A.\ P(x);\ \ P(x) \Longrightarrow Q;\ \ x{\notin}A \Longrightarrow Q]\!] \Longrightarrow Q$
⟨*proof*⟩

lemma *rev-bspec*: $[\![x{:}\ A;\ \ \forall x{\in}A.\ P(x)]\!] \Longrightarrow P(x)$
⟨*proof*⟩

lemma *ball-triv* [*simp*]: $(\forall x{\in}A.\ P) \longleftrightarrow ((\exists x.\ x{\in}A) \longrightarrow P)$
⟨*proof*⟩

lemma *ball-cong* [*cong*]:
$[\![A{=}A';\ \ \bigwedge x.\ x{\in}A' \Longrightarrow P(x) \longleftrightarrow P'(x)]\!] \Longrightarrow (\forall x{\in}A.\ P(x)) \longleftrightarrow (\forall x{\in}A'.\ P'(x))$
⟨*proof*⟩

lemma *atomize-ball*:
$(\bigwedge x.\ x \in A \Longrightarrow P(x)) \equiv \mathit{Trueprop}\ (\forall x{\in}A.\ P(x))$
⟨*proof*⟩

lemmas [*symmetric*, *rulify*] = *atomize-ball*
and [*symmetric*, *defn*] = *atomize-ball*

1.13 Bounded existential quantifier

lemma *bexI* [*intro*]: $[\![P(x);\ \ x{:}\ A]\!] \Longrightarrow \exists x{\in}A.\ P(x)$

⟨*proof*⟩

lemma *rev-bexI*: $[\![x{\in}A;\ \ P(x)]\!] \Longrightarrow \exists\, x{\in}A.\ P(x)$
⟨*proof*⟩

lemma *bexCI*: $[\![\forall\, x{\in}A.\ \neg P(x) \Longrightarrow P(a);\ \ a{:}\ A]\!] \Longrightarrow \exists\, x{\in}A.\ P(x)$
⟨*proof*⟩

lemma *bexE* [*elim*!]: $[\![\exists\, x{\in}A.\ P(x);\ \ \bigwedge x.\ [\![x{\in}A;\ P(x)]\!] \Longrightarrow Q]\!] \Longrightarrow Q$
⟨*proof*⟩

lemma *bex-triv* [*simp*]: $(\exists\, x{\in}A.\ P) \longleftrightarrow ((\exists\, x.\ x{\in}A) \wedge P)$
⟨*proof*⟩

lemma *bex-cong* [*cong*]:
$[\![A{=}A';\ \ \bigwedge x.\ x{\in}A' \Longrightarrow P(x) \longleftrightarrow P'(x)]\!]$
$\Longrightarrow (\exists\, x{\in}A.\ P(x)) \longleftrightarrow (\exists\, x{\in}A'.\ P'(x))$
⟨*proof*⟩

1.14 Rules for subsets

lemma *subsetI* [*intro*!]:
$(\bigwedge x.\ x{\in}A \Longrightarrow x{\in}B) \Longrightarrow A \subseteq B$
⟨*proof*⟩

lemma *subsetD* [*elim*]: $[\![A \subseteq B;\ \ c{\in}A]\!] \Longrightarrow c{\in}B$
⟨*proof*⟩

lemma *subsetCE* [*elim*]:
$[\![A \subseteq B;\ \ c{\notin}A \Longrightarrow P;\ \ c{\in}B \Longrightarrow P]\!] \Longrightarrow P$
⟨*proof*⟩

lemma *rev-subsetD*: $[\![c{\in}A;\ A{\subseteq}B]\!] \Longrightarrow c{\in}B$
⟨*proof*⟩

lemma *contra-subsetD*: $[\![A \subseteq B;\ c \notin B]\!] \Longrightarrow c \notin A$
⟨*proof*⟩

lemma *rev-contra-subsetD*: $[\![c \notin B;\ \ A \subseteq B]\!] \Longrightarrow c \notin A$
⟨*proof*⟩

lemma *subset-refl* [*simp*]: $A \subseteq A$
⟨*proof*⟩

lemma *subset-trans*: $[\![A\subseteq B;\;\; B\subseteq C]\!] \Longrightarrow A\subseteq C$
⟨*proof*⟩

lemma *subset-iff*:
$A\subseteq B \longleftrightarrow (\forall x.\; x\in A \longrightarrow x\in B)$
⟨*proof*⟩

For calculations

declare *subsetD* [*trans*] *rev-subsetD* [*trans*] *subset-trans* [*trans*]

1.15 Rules for equality

lemma *equalityI* [*intro*]: $[\![A \subseteq B;\;\; B \subseteq A]\!] \Longrightarrow A = B$
⟨*proof*⟩

lemma *equality-iffI*: $(\bigwedge x.\; x\in A \longleftrightarrow x\in B) \Longrightarrow A = B$
⟨*proof*⟩

lemmas *equalityD1* = *extension* [*THEN iffD1*, *THEN conjunct1*]
lemmas *equalityD2* = *extension* [*THEN iffD1*, *THEN conjunct2*]

lemma *equalityE*: $[\![A = B;\;\; [\![A\subseteq B;\; B\subseteq A]\!] \Longrightarrow P]\!] \;\Longrightarrow\; P$
⟨*proof*⟩

lemma *equalityCE*:
$[\![A = B;\;\; [\![c\in A;\; c\in B]\!] \Longrightarrow P;\;\; [\![c\notin A;\; c\notin B]\!] \Longrightarrow P]\!] \;\Longrightarrow\; P$
⟨*proof*⟩

lemma *equality-iffD*:
$A = B \Longrightarrow (\bigwedge x.\; x \in A \longleftrightarrow x \in B)$
⟨*proof*⟩

1.16 Rules for Replace – the derived form of replacement

lemma *Replace-iff*:
$b \in \{y.\; x\in A,\; P(x,y)\} \;\longleftrightarrow\; (\exists x\in A.\; P(x,b) \wedge (\forall y.\; P(x,y) \longrightarrow y=b))$
⟨*proof*⟩

lemma *ReplaceI* [*intro*]:
$[\![P(x,b);\;\; x\colon A;\;\; \bigwedge y.\; P(x,y) \Longrightarrow y=b]\!] \Longrightarrow$
$b \in \{y.\; x\in A,\; P(x,y)\}$
⟨*proof*⟩

lemma *ReplaceE*:

$[\![b \in \{y.\ x{\in}A,\ P(x,y)\};$
$\quad \bigwedge x.\ [\![x{:}\ A;\ \ P(x,b);\ \ \forall\, y.\ P(x,y) {\longrightarrow} y{=}b]\!] \Longrightarrow R$
$]\!] \Longrightarrow R$
⟨*proof*⟩

lemma *ReplaceE2* [*elim*!]:
$[\![b \in \{y.\ x{\in}A,\ P(x,y)\};$
$\quad \bigwedge x.\ [\![x{:}\ A;\ \ P(x,b)]\!] \Longrightarrow R$
$]\!] \Longrightarrow R$
⟨*proof*⟩

lemma *Replace-cong* [*cong*]:
$[\![A{=}B;\ \ \bigwedge x\ y.\ x{\in}B \Longrightarrow P(x,y) \longleftrightarrow Q(x,y)]\!] \Longrightarrow Replace(A,P) = Replace(B,Q)$
⟨*proof*⟩

1.17 Rules for RepFun

lemma *RepFunI*: $a \in A \Longrightarrow f(a) \in \{f(x).\ x{\in}A\}$
⟨*proof*⟩

lemma *RepFun-eqI* [*intro*]: $[\![b{=}f(a);\ \ a \in A]\!] \Longrightarrow b \in \{f(x).\ x{\in}A\}$
⟨*proof*⟩

lemma *RepFunE* [*elim*!]:
$[\![b \in \{f(x).\ x{\in}A\};$
$\quad \bigwedge x.[\![x{\in}A;\ \ b{=}f(x)]\!] \Longrightarrow P]\!] \Longrightarrow$
P
⟨*proof*⟩

lemma *RepFun-cong* [*cong*]:
$[\![A{=}B;\ \ \bigwedge x.\ x{\in}B \Longrightarrow f(x){=}g(x)]\!] \Longrightarrow RepFun(A,f) = RepFun(B,g)$
⟨*proof*⟩

lemma *RepFun-iff* [*simp*]: $b \in \{f(x).\ x{\in}A\} \longleftrightarrow (\exists\, x{\in}A.\ b{=}f(x))$
⟨*proof*⟩

lemma *triv-RepFun* [*simp*]: $\{x.\ x{\in}A\} = A$
⟨*proof*⟩

1.18 Rules for Collect – forming a subset by separation

lemma *separation* [*simp*]: $a \in \{x{\in}A.\ P(x)\} \longleftrightarrow a{\in}A \wedge P(a)$
⟨*proof*⟩

lemma *CollectI* [*intro*!]: $[\![a{\in}A;\ \ P(a)]\!] \Longrightarrow a \in \{x{\in}A.\ P(x)\}$
⟨*proof*⟩

lemma *CollectE* [*elim*!]: $[\![a \in \{x{\in}A.\ P(x)\};\ \ [\![a{\in}A;\ P(a)]\!] \Longrightarrow R]\!] \Longrightarrow R$

$\langle proof \rangle$

lemma *CollectD1*: $a \in \{x{\in}A.\ P(x)\} \Longrightarrow a{\in}A$ **and** *CollectD2*: $a \in \{x{\in}A.\ P(x)\} \Longrightarrow P(a)$
$\langle proof \rangle$

lemma *Collect-cong* [*cong*]:
$[\![A=B;\ \bigwedge x.\ x{\in}B \Longrightarrow P(x) \longleftrightarrow Q(x)]\!]$
$\Longrightarrow Collect(A,\ \lambda x.\ P(x)) = Collect(B,\ \lambda x.\ Q(x))$
$\langle proof \rangle$

1.19 Rules for Unions

declare *Union-iff* [*simp*]

lemma *UnionI* [*intro*]: $[\![B\colon C;\ \ A\colon B]\!] \Longrightarrow A\colon \bigcup(C)$
$\langle proof \rangle$

lemma *UnionE* [*elim!*]: $[\![A \in \bigcup(C);\ \ \bigwedge B.[\![A\colon B;\ \ B\colon C]\!] \Longrightarrow R]\!] \Longrightarrow R$
$\langle proof \rangle$

1.20 Rules for Unions of families

lemma *UN-iff* [*simp*]: $b \in (\bigcup x{\in}A.\ B(x)) \longleftrightarrow (\exists x{\in}A.\ b \in B(x))$
$\langle proof \rangle$

lemma *UN-I*: $[\![a\colon A;\ \ b\colon B(a)]\!] \Longrightarrow b\colon (\bigcup x{\in}A.\ B(x))$
$\langle proof \rangle$

lemma *UN-E* [*elim!*]:
$[\![b \in (\bigcup x{\in}A.\ B(x));\ \ \bigwedge x.[\![x\colon A;\ \ b\colon B(x)]\!] \Longrightarrow R]\!] \Longrightarrow R$
$\langle proof \rangle$

lemma *UN-cong*:
$[\![A=B;\ \bigwedge x.\ x{\in}B \Longrightarrow C(x)=D(x)]\!] \Longrightarrow (\bigcup x{\in}A.\ C(x)) = (\bigcup x{\in}B.\ D(x))$
$\langle proof \rangle$

1.21 Rules for the empty set

lemma *not-mem-empty* [*simp*]: $a \notin 0$
$\langle proof \rangle$

lemmas *emptyE* [*elim!*] = *not-mem-empty* [*THEN notE*]

lemma *empty-subsetI* [*simp*]: $0 \subseteq A$
$\langle proof \rangle$

lemma *equals0I*: $[\![\bigwedge y.\ y{\in}A \Longrightarrow False]\!] \Longrightarrow A{=}0$
⟨*proof*⟩

lemma *equals0D* [*dest*]: $A{=}0 \Longrightarrow a \notin A$
⟨*proof*⟩

declare *sym* [*THEN equals0D*, *dest*]

lemma *not-emptyI*: $a{\in}A \Longrightarrow A \neq 0$
⟨*proof*⟩

lemma *not-emptyE*: $[\![A \neq 0;\ \ \bigwedge x.\ x{\in}A \Longrightarrow R]\!] \Longrightarrow R$
⟨*proof*⟩

1.22 Rules for Inter

lemma *Inter-iff*: $A \in \bigcap(C) \longleftrightarrow (\forall x{\in}C.\ A{:}\ x) \wedge C{\neq}0$
⟨*proof*⟩

lemma *InterI* [*intro*!]:
$[\![\bigwedge x.\ x{:}\ C \Longrightarrow A{:}\ x;\ \ C{\neq}0]\!] \Longrightarrow A \in \bigcap(C)$
⟨*proof*⟩

lemma *InterD* [*elim*, *Pure.elim*]: $[\![A \in \bigcap(C);\ \ B \in C]\!] \Longrightarrow A \in B$
⟨*proof*⟩

lemma *InterE* [*elim*]:
$[\![A \in \bigcap(C);\ \ B{\notin}C \Longrightarrow R;\ \ A{\in}B \Longrightarrow R]\!] \Longrightarrow R$
⟨*proof*⟩

1.23 Rules for Intersections of families

lemma *INT-iff*: $b \in (\bigcap x{\in}A.\ B(x)) \longleftrightarrow (\forall x{\in}A.\ b \in B(x)) \wedge A{\neq}0$
⟨*proof*⟩

lemma *INT-I*: $[\![\bigwedge x.\ x{:}\ A \Longrightarrow b{:}\ B(x);\ \ A{\neq}0]\!] \Longrightarrow b{:}\ (\bigcap x{\in}A.\ B(x))$
⟨*proof*⟩

lemma *INT-E*: $[\![b \in (\bigcap x{\in}A.\ B(x));\ \ a{:}\ A]\!] \Longrightarrow b \in B(a)$
⟨*proof*⟩

lemma *INT-cong*:
$[\![A{=}B;\ \ \bigwedge x.\ x{\in}B \Longrightarrow C(x){=}D(x)]\!] \Longrightarrow (\bigcap x{\in}A.\ C(x)) = (\bigcap x{\in}B.\ D(x))$
⟨*proof*⟩

1.24 Rules for Powersets

lemma *PowI*: $A \subseteq B \Longrightarrow A \in Pow(B)$
⟨*proof*⟩

lemma *PowD*: $A \in Pow(B) \Longrightarrow A \subseteq B$
⟨*proof*⟩

declare *Pow-iff* [*iff*]

lemmas *Pow-bottom* = *empty-subsetI* [*THEN PowI*] — $0 \in Pow(B)$
lemmas *Pow-top* = *subset-refl* [*THEN PowI*] — $A \in Pow(A)$

1.25 Cantor's Theorem: There is no surjection from a set to its powerset.

lemma *cantor*: $\exists S \in Pow(A).\ \forall x{\in}A.\ b(x) \neq S$
⟨*proof*⟩

end

2 Unordered Pairs

theory *upair*
imports *ZF-Base*
keywords *print-tcset* :: *diag*
begin

⟨*ML*⟩

2.1 Unordered Pairs: constant *Upair*

lemma *Upair-iff* [*simp*]: $c \in Upair(a,b) \longleftrightarrow (c=a \mid c=b)$
⟨*proof*⟩

lemma *UpairI1*: $a \in Upair(a,b)$
⟨*proof*⟩

lemma *UpairI2*: $b \in Upair(a,b)$
⟨*proof*⟩

lemma *UpairE*: $[\![a \in Upair(b,c);\ a=b \Longrightarrow P;\ a=c \Longrightarrow P]\!] \Longrightarrow P$
⟨*proof*⟩

2.2 Rules for Binary Union, Defined via *Upair*

lemma *Un-iff* [*simp*]: $c \in A \cup B \longleftrightarrow (c \in A \mid c \in B)$
⟨*proof*⟩

lemma *UnI1*: $c \in A \Longrightarrow c \in A \cup B$
⟨*proof*⟩

lemma *UnI2*: $c \in B \Longrightarrow c \in A \cup B$
⟨*proof*⟩

declare *UnI1* [*elim?*] *UnI2* [*elim?*]

lemma *UnE* [*elim*!]: $[\![c \in A \cup B;\ \ c \in A \Longrightarrow P;\ \ c \in B \Longrightarrow P]\!] \Longrightarrow P$
⟨*proof*⟩

lemma *UnE'*: $[\![c \in A \cup B;\ \ c \in A \Longrightarrow P;\ \ [\![c \in B;\ \ c \notin A]\!] \Longrightarrow P]\!] \Longrightarrow P$
⟨*proof*⟩

lemma *UnCI* [*intro*!]: $(c \notin B \Longrightarrow c \in A) \Longrightarrow c \in A \cup B$
⟨*proof*⟩

2.3 Rules for Binary Intersection, Defined via *Upair*

lemma *Int-iff* [*simp*]: $c \in A \cap B \longleftrightarrow (c \in A \wedge c \in B)$
⟨*proof*⟩

lemma *IntI* [*intro*!]: $[\![c \in A;\ \ c \in B]\!] \Longrightarrow c \in A \cap B$
⟨*proof*⟩

lemma *IntD1*: $c \in A \cap B \Longrightarrow c \in A$
⟨*proof*⟩

lemma *IntD2*: $c \in A \cap B \Longrightarrow c \in B$
⟨*proof*⟩

lemma *IntE* [*elim*!]: $[\![c \in A \cap B;\ \ [\![c \in A;\ c \in B]\!] \Longrightarrow P]\!] \Longrightarrow P$
⟨*proof*⟩

2.4 Rules for Set Difference, Defined via *Upair*

lemma *Diff-iff* [*simp*]: $c \in A-B \longleftrightarrow (c \in A \wedge c \notin B)$
⟨*proof*⟩

lemma *DiffI* [*intro*!]: $[\![c \in A;\ \ c \notin B]\!] \Longrightarrow c \in A - B$
⟨*proof*⟩

lemma *DiffD1*: $c \in A - B \Longrightarrow c \in A$
⟨*proof*⟩

lemma *DiffD2*: $c \in A - B \Longrightarrow c \notin B$
⟨*proof*⟩

lemma *DiffE* [*elim*!]: $[\![c \in A - B;\ [\![c \in A;\ c \notin B]\!] \Longrightarrow P]\!] \Longrightarrow P$
⟨*proof*⟩

2.5 Rules for *cons*

lemma *cons-iff* [*simp*]: $a \in cons(b,A) \longleftrightarrow (a=b \mid a \in A)$
⟨*proof*⟩

lemma *consI1* [*simp*, *TC*]: $a \in cons(a,B)$
⟨*proof*⟩

lemma *consI2*: $a \in B \Longrightarrow a \in cons(b,B)$
⟨*proof*⟩

lemma *consE* [*elim*!]: $[\![a \in cons(b,A);\ a=b \Longrightarrow P;\ a \in A \Longrightarrow P]\!] \Longrightarrow P$
⟨*proof*⟩

lemma *consE'*:
$[\![a \in cons(b,A);\ a=b \Longrightarrow P;\ [\![a \in A;\ a\neq b]\!] \Longrightarrow P]\!] \Longrightarrow P$
⟨*proof*⟩

lemma *consCI* [*intro*!]: $(a \notin B \Longrightarrow a=b) \Longrightarrow a \in cons(b,B)$
⟨*proof*⟩

lemma *cons-not-0* [*simp*]: $cons(a,B) \neq 0$
⟨*proof*⟩

lemmas *cons-neq-0* = *cons-not-0* [*THEN notE*]

declare *cons-not-0* [*THEN not-sym*, *simp*]

2.6 Singletons

lemma *singleton-iff*: $a \in \{b\} \longleftrightarrow a=b$
⟨*proof*⟩

lemma *singletonI* [*intro*!]: $a \in \{a\}$
⟨*proof*⟩

lemmas *singletonE* = *singleton-iff* [*THEN iffD1*, *elim-format*, *elim*!]

2.7 Descriptions

lemma *the-equality* [*intro*]:
$[\![P(a);\ \bigwedge x.\ P(x) \Longrightarrow x=a]\!] \Longrightarrow (THE\ x.\ P(x)) = a$
⟨*proof*⟩

lemma *the-equality2*: $[\![\exists!x.\ P(x);\ \ P(a)]\!] \Longrightarrow (THE\ x.\ P(x)) = a$
⟨*proof*⟩

lemma *theI*: $\exists!x.\ P(x) \Longrightarrow P(THE\ x.\ P(x))$
⟨*proof*⟩

lemma *the-0*: $\neg\ (\exists!x.\ P(x)) \Longrightarrow (THE\ x.\ P(x)){=}0$
⟨*proof*⟩

lemma *theI2*:
assumes *p1*: $\neg\ Q(0) \Longrightarrow \exists!x.\ P(x)$
and *p2*: $\bigwedge x.\ P(x) \Longrightarrow Q(x)$
shows $Q(THE\ x.\ P(x))$
⟨*proof*⟩

lemma *the-eq-trivial* [*simp*]: $(THE\ x.\ x = a) = a$
⟨*proof*⟩

lemma *the-eq-trivial2* [*simp*]: $(THE\ x.\ a = x) = a$
⟨*proof*⟩

2.8 Conditional Terms: *if−then−else*

lemma *if-true* [*simp*]: (*if True then a else b*) = *a*
⟨*proof*⟩

lemma *if-false* [*simp*]: (*if False then a else b*) = *b*
⟨*proof*⟩

lemma *if-cong*:
$[\![P \longleftrightarrow Q;\ \ Q \Longrightarrow a{=}c;\ \ \neg Q \Longrightarrow b{=}d]\!]$
$\Longrightarrow$ (*if P then a else b*) = (*if Q then c else d*)
⟨*proof*⟩

lemma *if-weak-cong*: $P \longleftrightarrow Q \Longrightarrow$ (*if P then x else y*) = (*if Q then x else y*)
⟨*proof*⟩

lemma *if-P*: $P \Longrightarrow$ (*if P then a else b*) = *a*
⟨*proof*⟩

lemma *if-not-P*: $\neg P \Longrightarrow (\textit{if } P \textit{ then } a \textit{ else } b) = b$
$\langle proof \rangle$

lemma *split-if* [*split*]:
$P(\textit{if } Q \textit{ then } x \textit{ else } y) \longleftrightarrow ((Q \longrightarrow P(x)) \wedge (\neg Q \longrightarrow P(y)))$
$\langle proof \rangle$

lemmas *split-if-eq1* = *split-if* [*of* $\lambda x.\ x = b$] **for** b
lemmas *split-if-eq2* = *split-if* [*of* $\lambda x.\ a = x$] **for** a

lemmas *split-if-mem1* = *split-if* [*of* $\lambda x.\ x \in b$] **for** b
lemmas *split-if-mem2* = *split-if* [*of* $\lambda x.\ a \in x$] **for** a

lemmas *split-ifs* = *split-if-eq1 split-if-eq2 split-if-mem1 split-if-mem2*

lemma *if-iff*: a: $(\textit{if } P \textit{ then } x \textit{ else } y) \longleftrightarrow P \wedge a \in x \mid \neg P \wedge a \in y$
$\langle proof \rangle$

lemma *if-type* [*TC*]:
$[\![P \Longrightarrow a \in A;\ \neg P \Longrightarrow b \in A]\!] \Longrightarrow (\textit{if } P \textit{ then } a \textit{ else } b)$: A
$\langle proof \rangle$

lemma *split-if-asm*: $P(\textit{if } Q \textit{ then } x \textit{ else } y) \longleftrightarrow (\neg((Q \wedge \neg P(x)) \mid (\neg Q \wedge \neg P(y))))$
$\langle proof \rangle$

lemmas *if-splits* = *split-if split-if-asm*

2.9 Consequences of Foundation

lemma *mem-asym*: $[\![a \in b;\ \neg P \Longrightarrow b \in a]\!] \Longrightarrow P$
$\langle proof \rangle$

lemma *mem-irrefl*: $a \in a \Longrightarrow P$
$\langle proof \rangle$

lemma *mem-not-refl*: $a \notin a$
$\langle proof \rangle$

lemma *mem-imp-not-eq*: $a \in A \Longrightarrow a \neq A$

⟨*proof*⟩

lemma *eq-imp-not-mem*: $a{=}A \Longrightarrow a \notin A$
⟨*proof*⟩

2.10 Rules for Successor

lemma *succ-iff*: $i \in succ(j) \longleftrightarrow i{=}j \mid i \in j$
⟨*proof*⟩

lemma *succI1* [*simp*]: $i \in succ(i)$
⟨*proof*⟩

lemma *succI2*: $i \in j \Longrightarrow i \in succ(j)$
⟨*proof*⟩

lemma *succE* [*elim*!]:
$[\![i \in succ(j);\ \ i{=}j \Longrightarrow P;\ \ i \in j \Longrightarrow P]\!] \Longrightarrow P$
⟨*proof*⟩

lemma *succCI* [*intro*!]: $(i{\notin}j \Longrightarrow i{=}j) \Longrightarrow i \in succ(j)$
⟨*proof*⟩

lemma *succ-not-0* [*simp*]: $succ(n) \neq 0$
⟨*proof*⟩

lemmas *succ-neq-0* = *succ-not-0* [*THEN notE*, *elim*!]

declare *succ-not-0* [*THEN not-sym*, *simp*]
declare *sym* [*THEN succ-neq-0*, *elim*!]

lemmas *succ-subsetD* = *succI1* [*THEN* [*2*] *subsetD*]

lemmas *succ-neq-self* = *succI1* [*THEN mem-imp-not-eq*, *THEN not-sym*]

lemma *succ-inject-iff* [*simp*]: $succ(m) = succ(n) \longleftrightarrow m{=}n$
⟨*proof*⟩

lemmas *succ-inject* = *succ-inject-iff* [*THEN iffD1*, *dest*!]

2.11 Miniscoping of the Bounded Universal Quantifier

lemma *ball-simps1*:
$(\forall x{\in}A.\ P(x) \land Q) \ \ \longleftrightarrow (\forall x{\in}A.\ P(x)) \land (A{=}0 \mid Q)$
$(\forall x{\in}A.\ P(x) \mid Q) \ \ \longleftrightarrow ((\forall x{\in}A.\ P(x)) \mid Q)$
$(\forall x{\in}A.\ P(x) \longrightarrow Q) \longleftrightarrow ((\exists x{\in}A.\ P(x)) \longrightarrow Q)$
$(\neg(\forall x{\in}A.\ P(x))) \longleftrightarrow (\exists x{\in}A.\ \neg P(x))$

$(\forall x{\in}0. P(x)) \longleftrightarrow True$
$(\forall x{\in}succ(i).P(x)) \longleftrightarrow P(i) \land (\forall x{\in}i.\ P(x))$
$(\forall x{\in}cons(a,B).P(x)) \longleftrightarrow P(a) \land (\forall x{\in}B.\ P(x))$
$(\forall x{\in}RepFun(A,f).\ P(x)) \longleftrightarrow (\forall y{\in}A.\ P(f(y)))$
$(\forall x{\in}\bigcup(A).P(x)) \longleftrightarrow (\forall y{\in}A.\ \forall x{\in}y.\ P(x))$
⟨*proof*⟩

lemma *ball-simps2*:
$(\forall x{\in}A.\ P \land Q(x)) \quad \longleftrightarrow (A{=}0 \mid P) \land (\forall x{\in}A.\ Q(x))$
$(\forall x{\in}A.\ P \mid Q(x)) \quad \longleftrightarrow (P \mid (\forall x{\in}A.\ Q(x)))$
$(\forall x{\in}A.\ P \longrightarrow Q(x)) \longleftrightarrow (P \longrightarrow (\forall x{\in}A.\ Q(x)))$
⟨*proof*⟩

lemma *ball-simps3*:
$(\forall x{\in}Collect(A,Q).P(x)) \longleftrightarrow (\forall x{\in}A.\ Q(x) \longrightarrow P(x))$
⟨*proof*⟩

lemmas *ball-simps* [*simp*] = *ball-simps1 ball-simps2 ball-simps3*

lemma *ball-conj-distrib*:
$(\forall x{\in}A.\ P(x) \land Q(x)) \longleftrightarrow ((\forall x{\in}A.\ P(x)) \land (\forall x{\in}A.\ Q(x)))$
⟨*proof*⟩

2.12 Miniscoping of the Bounded Existential Quantifier

lemma *bex-simps1*:
$(\exists x{\in}A.\ P(x) \land Q) \longleftrightarrow ((\exists x{\in}A.\ P(x)) \land Q)$
$(\exists x{\in}A.\ P(x) \mid Q) \longleftrightarrow (\exists x{\in}A.\ P(x)) \mid (A{\neq}0 \land Q)$
$(\exists x{\in}A.\ P(x) \longrightarrow Q) \longleftrightarrow ((\forall x{\in}A.\ P(x)) \longrightarrow (A{\neq}0 \land Q))$
$(\exists x{\in}0.P(x)) \longleftrightarrow False$
$(\exists x{\in}succ(i).P(x)) \longleftrightarrow P(i) \mid (\exists x{\in}i.\ P(x))$
$(\exists x{\in}cons(a,B).P(x)) \longleftrightarrow P(a) \mid (\exists x{\in}B.\ P(x))$
$(\exists x{\in}RepFun(A,f).\ P(x)) \longleftrightarrow (\exists y{\in}A.\ P(f(y)))$
$(\exists x{\in}\bigcup(A).P(x)) \longleftrightarrow (\exists y{\in}A.\ \exists x{\in}y.\ P(x))$
$(\neg(\exists x{\in}A.\ P(x))) \longleftrightarrow (\forall x{\in}A.\ \neg P(x))$
⟨*proof*⟩

lemma *bex-simps2*:
$(\exists x{\in}A.\ P \land Q(x)) \longleftrightarrow (P \land (\exists x{\in}A.\ Q(x)))$
$(\exists x{\in}A.\ P \mid Q(x)) \longleftrightarrow (A{\neq}0 \land P) \mid (\exists x{\in}A.\ Q(x))$
$(\exists x{\in}A.\ P \longrightarrow Q(x)) \longleftrightarrow ((A{=}0 \mid P) \longrightarrow (\exists x{\in}A.\ Q(x)))$
⟨*proof*⟩

lemma *bex-simps3*:
$(\exists x{\in}Collect(A,Q).P(x)) \longleftrightarrow (\exists x{\in}A.\ Q(x) \land P(x))$
⟨*proof*⟩

lemmas *bex-simps* [*simp*] = *bex-simps1 bex-simps2 bex-simps3*

lemma *bex-disj-distrib*:
$(\exists x{\in}A.\ P(x) \mid Q(x)) \longleftrightarrow ((\exists x{\in}A.\ P(x)) \mid (\exists x{\in}A.\ Q(x)))$
⟨*proof*⟩

lemma *bex-triv-one-point1* [*simp*]: $(\exists x{\in}A.\ x{=}a) \longleftrightarrow (a \in A)$
⟨*proof*⟩

lemma *bex-triv-one-point2* [*simp*]: $(\exists x{\in}A.\ a{=}x) \longleftrightarrow (a \in A)$
⟨*proof*⟩

lemma *bex-one-point1* [*simp*]: $(\exists x{\in}A.\ x{=}a \wedge P(x)) \longleftrightarrow (a \in A \wedge P(a))$
⟨*proof*⟩

lemma *bex-one-point2* [*simp*]: $(\exists x{\in}A.\ a{=}x \wedge P(x)) \longleftrightarrow (a \in A \wedge P(a))$
⟨*proof*⟩

lemma *ball-one-point1* [*simp*]: $(\forall x{\in}A.\ x{=}a \longrightarrow P(x)) \longleftrightarrow (a \in A \longrightarrow P(a))$
⟨*proof*⟩

lemma *ball-one-point2* [*simp*]: $(\forall x{\in}A.\ a{=}x \longrightarrow P(x)) \longleftrightarrow (a \in A \longrightarrow P(a))$
⟨*proof*⟩

2.13 Miniscoping of the Replacement Operator

These cover both *Replace* and *Collect*

lemma *Rep-simps* [*simp*]:
$\{x.\ y \in 0,\ R(x,y)\} = 0$
$\{x \in 0.\ P(x)\} = 0$
$\{x \in A.\ Q\} = (\textit{if } Q \textit{ then } A \textit{ else } 0)$
$RepFun(0,f) = 0$
$RepFun(succ(i),f) = cons(f(i),\ RepFun(i,f))$
$RepFun(cons(a,B),f) = cons(f(a),\ RepFun(B,f))$
⟨*proof*⟩

2.14 Miniscoping of Unions

lemma *UN-simps1*:
$(\bigcup x{\in}C.\ cons(a,\ B(x))) = (\textit{if } C{=}0 \textit{ then } 0 \textit{ else } cons(a,\ \bigcup x{\in}C.\ B(x)))$
$(\bigcup x{\in}C.\ A(x) \cup B') = (\textit{if } C{=}0 \textit{ then } 0 \textit{ else } (\bigcup x{\in}C.\ A(x)) \cup B')$
$(\bigcup x{\in}C.\ A' \cup B(x)) = (\textit{if } C{=}0 \textit{ then } 0 \textit{ else } A' \cup (\bigcup x{\in}C.\ B(x)))$
$(\bigcup x{\in}C.\ A(x) \cap B') = ((\bigcup x{\in}C.\ A(x)) \cap B')$
$(\bigcup x{\in}C.\ A' \cap B(x)) = (A' \cap (\bigcup x{\in}C.\ B(x)))$
$(\bigcup x{\in}C.\ A(x) - B') = ((\bigcup x{\in}C.\ A(x)) - B')$
$(\bigcup x{\in}C.\ A' - B(x)) = (\textit{if } C{=}0 \textit{ then } 0 \textit{ else } A' - (\bigcap x{\in}C.\ B(x)))$
⟨*proof*⟩

lemma *UN-simps2*:
$(\bigcup x\in\bigcup(A).\ B(x)) = (\bigcup y\in A.\ \bigcup x\in y.\ B(x))$
$(\bigcup z\in(\bigcup x\in A.\ B(x)).\ C(z)) = (\bigcup x\in A.\ \bigcup z\in B(x).\ C(z))$
$(\bigcup x\in RepFun(A,f).\ B(x)) = (\bigcup a\in A.\ B(f(a)))$
⟨*proof*⟩

lemmas *UN-simps* [*simp*] = *UN-simps1 UN-simps2*

Opposite of miniscoping: pull the operator out

lemma *UN-extend-simps1*:
$(\bigcup x\in C.\ A(x)) \cup B = (\textit{if } C=0 \textit{ then } B \textit{ else } (\bigcup x\in C.\ A(x) \cup B))$
$((\bigcup x\in C.\ A(x)) \cap B) = (\bigcup x\in C.\ A(x) \cap B)$
$((\bigcup x\in C.\ A(x)) - B) = (\bigcup x\in C.\ A(x) - B)$
⟨*proof*⟩

lemma *UN-extend-simps2*:
$cons(a, \bigcup x\in C.\ B(x)) = (\textit{if } C=0 \textit{ then } \{a\} \textit{ else } (\bigcup x\in C.\ cons(a, B(x))))$
$A \cup (\bigcup x\in C.\ B(x)) = (\textit{if } C=0 \textit{ then } A \textit{ else } (\bigcup x\in C.\ A \cup B(x)))$
$(A \cap (\bigcup x\in C.\ B(x))) = (\bigcup x\in C.\ A \cap B(x))$
$A - (\bigcap x\in C.\ B(x)) = (\textit{if } C=0 \textit{ then } A \textit{ else } (\bigcup x\in C.\ A - B(x)))$
$(\bigcup y\in A.\ \bigcup x\in y.\ B(x)) = (\bigcup x\in\bigcup(A).\ B(x))$
$(\bigcup a\in A.\ B(f(a))) = (\bigcup x\in RepFun(A,f).\ B(x))$
⟨*proof*⟩

lemma *UN-UN-extend*:
$(\bigcup x\in A.\ \bigcup z\in B(x).\ C(z)) = (\bigcup z\in(\bigcup x\in A.\ B(x)).\ C(z))$
⟨*proof*⟩

lemmas *UN-extend-simps* = *UN-extend-simps1 UN-extend-simps2 UN-UN-extend*

2.15 Miniscoping of Intersections

lemma *INT-simps1*:
$(\bigcap x\in C.\ A(x) \cap B) = (\bigcap x\in C.\ A(x)) \cap B$
$(\bigcap x\in C.\ A(x) - B) = (\bigcap x\in C.\ A(x)) - B$
$(\bigcap x\in C.\ A(x) \cup B) = (\textit{if } C=0 \textit{ then } 0 \textit{ else } (\bigcap x\in C.\ A(x)) \cup B)$
⟨*proof*⟩

lemma *INT-simps2*:
$(\bigcap x\in C.\ A \cap B(x)) = A \cap (\bigcap x\in C.\ B(x))$
$(\bigcap x\in C.\ A - B(x)) = (\textit{if } C=0 \textit{ then } 0 \textit{ else } A - (\bigcup x\in C.\ B(x)))$
$(\bigcap x\in C.\ cons(a, B(x))) = (\textit{if } C=0 \textit{ then } 0 \textit{ else } cons(a, \bigcap x\in C.\ B(x)))$
$(\bigcap x\in C.\ A \cup B(x)) = (\textit{if } C=0 \textit{ then } 0 \textit{ else } A \cup (\bigcap x\in C.\ B(x)))$
⟨*proof*⟩

lemmas *INT-simps* [*simp*] = *INT-simps1 INT-simps2*

Opposite of miniscoping: pull the operator out

lemma *INT-extend-simps1*:

$(\bigcap x{\in}C.\ A(x)) \cap B = (\bigcap x{\in}C.\ A(x) \cap B)$
$(\bigcap x{\in}C.\ A(x)) - B = (\bigcap x{\in}C.\ A(x) - B)$
$(\bigcap x{\in}C.\ A(x)) \cup B = (\mathit{if}\ C{=}0\ \mathit{then}\ B\ \mathit{else}\ (\bigcap x{\in}C.\ A(x) \cup B))$
⟨*proof*⟩

lemma *INT-extend-simps2*:
$A \cap (\bigcap x{\in}C.\ B(x)) = (\bigcap x{\in}C.\ A \cap B(x))$
$A - (\bigcup x{\in}C.\ B(x)) = (\mathit{if}\ C{=}0\ \mathit{then}\ A\ \mathit{else}\ (\bigcap x{\in}C.\ A - B(x)))$
$\mathit{cons}(a, \bigcap x{\in}C.\ B(x)) = (\mathit{if}\ C{=}0\ \mathit{then}\ \{a\}\ \mathit{else}\ (\bigcap x{\in}C.\ \mathit{cons}(a, B(x))))$
$A \cup (\bigcap x{\in}C.\ B(x)) = (\mathit{if}\ C{=}0\ \mathit{then}\ A\ \mathit{else}\ (\bigcap x{\in}C.\ A \cup B(x)))$
⟨*proof*⟩

lemmas *INT-extend-simps* = *INT-extend-simps1 INT-extend-simps2*

2.16 Other simprules

lemma *misc-simps* [*simp*]:
$0 \cup A = A$
$A \cup 0 = A$
$0 \cap A = 0$
$A \cap 0 = 0$
$0 - A = 0$
$A - 0 = A$
$\bigcup(0) = 0$
$\bigcup(\mathit{cons}(b,A)) = b \cup \bigcup(A)$
$\bigcap(\{b\}) = b$
⟨*proof*⟩

end

3 Ordered Pairs

theory *pair* **imports** *upair*
begin

⟨*ML*⟩

lemma *singleton-eq-iff* [*iff*]: $\{a\} = \{b\} \longleftrightarrow a{=}b$
⟨*proof*⟩

lemma *doubleton-eq-iff*: $\{a,b\} = \{c,d\} \longleftrightarrow (a{=}c \land b{=}d) \mid (a{=}d \land b{=}c)$
⟨*proof*⟩

lemma *Pair-iff* [*simp*]: $\langle a,b\rangle = \langle c,d\rangle \longleftrightarrow a{=}c \land b{=}d$
⟨*proof*⟩

lemmas *Pair-inject* = *Pair-iff* [*THEN iffD1*, *THEN conjE*, *elim*!]

lemmas *Pair-inject1* = *Pair-iff* [*THEN iffD1*, *THEN conjunct1*]
lemmas *Pair-inject2* = *Pair-iff* [*THEN iffD1*, *THEN conjunct2*]

lemma *Pair-not-0*: $\langle a,b \rangle \neq 0$
⟨*proof*⟩

lemmas *Pair-neq-0* = *Pair-not-0* [*THEN notE*, *elim*!]

declare *sym* [*THEN Pair-neq-0*, *elim*!]

lemma *Pair-neq-fst*: $\langle a,b \rangle = a \Longrightarrow P$
⟨*proof*⟩

lemma *Pair-neq-snd*: $\langle a,b \rangle = b \Longrightarrow P$
⟨*proof*⟩

3.1 Sigma: Disjoint Union of a Family of Sets

Generalizes Cartesian product

lemma *Sigma-iff* [*simp*]: $\langle a,b \rangle$: *Sigma*$(A,B) \longleftrightarrow a \in A \wedge b \in B(a)$
⟨*proof*⟩

lemma *SigmaI* [*TC*,*intro*!]: $[\![a \in A;\ \ b \in B(a)]\!] \Longrightarrow \langle a,b \rangle \in$ *Sigma*(A,B)
⟨*proof*⟩

lemmas *SigmaD1* = *Sigma-iff* [*THEN iffD1*, *THEN conjunct1*]
lemmas *SigmaD2* = *Sigma-iff* [*THEN iffD1*, *THEN conjunct2*]

lemma *SigmaE* [*elim*!]:
$[\![c \in$ *Sigma*(A,B);
$\bigwedge x\ y. [\![x \in A;\ \ y \in B(x);\ \ c = \langle x,y \rangle]\!] \Longrightarrow P$
$]\!] \Longrightarrow P$
⟨*proof*⟩

lemma *SigmaE2* [*elim*!]:
$[\![\langle a,b \rangle \in$ *Sigma*(A,B);
$[\![a \in A;\ \ b \in B(a)]\!] \Longrightarrow P$
$]\!] \Longrightarrow P$
⟨*proof*⟩

lemma *Sigma-cong*:
$[\![A=A';\ \ \bigwedge x.\ x \in A' \Longrightarrow B(x)=B'(x)]\!] \Longrightarrow$
Sigma(A,B) = *Sigma*(A',B')
⟨*proof*⟩

lemma *Sigma-empty1* [*simp*]: *Sigma*(*0*,*B*) = *0*
⟨*proof*⟩

lemma *Sigma-empty2* [*simp*]: *A*∗*0* = *0*
⟨*proof*⟩

lemma *Sigma-empty-iff*: *A*∗*B*=*0* ⟷ *A*=*0* | *B*=*0*
⟨*proof*⟩

3.2 Projections *fst* and *snd*

lemma *fst-conv* [*simp*]: *fst*(⟨*a*,*b*⟩) = *a*
⟨*proof*⟩

lemma *snd-conv* [*simp*]: *snd*(⟨*a*,*b*⟩) = *b*
⟨*proof*⟩

lemma *fst-type* [*TC*]: *p* ∈ *Sigma*(*A*,*B*) ⟹ *fst*(*p*) ∈ *A*
⟨*proof*⟩

lemma *snd-type* [*TC*]: *p* ∈ *Sigma*(*A*,*B*) ⟹ *snd*(*p*) ∈ *B*(*fst*(*p*))
⟨*proof*⟩

lemma *Pair-fst-snd-eq*: *a* ∈ *Sigma*(*A*,*B*) ⟹ <*fst*(*a*),*snd*(*a*)> = *a*
⟨*proof*⟩

3.3 The Eliminator, *split*

lemma *split* [*simp*]: *split*(λ*x y*. *c*(*x*,*y*), ⟨*a*,*b*⟩) ≡ *c*(*a*,*b*)
⟨*proof*⟩

lemma *split-type* [*TC*]:
⟦*p* ∈ *Sigma*(*A*,*B*);
⋀*x y*.⟦*x* ∈ *A*; *y* ∈ *B*(*x*)⟧ ⟹ *c*(*x*,*y*):*C*(⟨*x*,*y*⟩)
⟧ ⟹ *split*(λ*x y*. *c*(*x*,*y*), *p*) ∈ *C*(*p*)
⟨*proof*⟩

lemma *expand-split*:
u ∈ *A*∗*B* ⟹
R(*split*(*c*,*u*)) ⟷ (∀ *x*∈*A*. ∀ *y*∈*B*. *u* = ⟨*x*,*y*⟩ ⟶ *R*(*c*(*x*,*y*)))
⟨*proof*⟩

3.4 A version of *split* for Formulae: Result Type *o*

lemma *splitI*: *R*(*a*,*b*) ⟹ *split*(*R*, ⟨*a*,*b*⟩)
⟨*proof*⟩

lemma *splitE*:
⟦*split*(*R*,*z*); *z* ∈ *Sigma*(*A*,*B*);

$\bigwedge x\ y.\ [\![z = \langle x,y\rangle;\ \ R(x,y)]\!] \Longrightarrow P$
$]\!] \Longrightarrow P$
⟨*proof*⟩

lemma *splitD*: $split(R,\langle a,b\rangle) \Longrightarrow R(a,b)$
⟨*proof*⟩

Complex rules for Sigma.

lemma *split-paired-Bex-Sigma* [*simp*]:
$(\exists z \in Sigma(A,B).\ P(z)) \longleftrightarrow (\exists x \in A.\ \exists y \in B(x).\ P(\langle x,y\rangle))$
⟨*proof*⟩

lemma *split-paired-Ball-Sigma* [*simp*]:
$(\forall z \in Sigma(A,B).\ P(z)) \longleftrightarrow (\forall x \in A.\ \forall y \in B(x).\ P(\langle x,y\rangle))$
⟨*proof*⟩

end

4 Basic Equalities and Inclusions

theory *equalities* **imports** *pair* **begin**

These cover union, intersection, converse, domain, range, etc. Philippe de Groote proved many of the inclusions.

lemma *in-mono*: $A \subseteq B \Longrightarrow x \in A \longrightarrow x \in B$
⟨*proof*⟩

lemma *the-eq-0* [*simp*]: $(THE\ x.\ False) = 0$
⟨*proof*⟩

4.1 Bounded Quantifiers

The following are not added to the default simpset because (a) they duplicate the body and (b) there are no similar rules for *Int*.

lemma *ball-Un*: $(\forall x \in A \cup B.\ P(x)) \longleftrightarrow (\forall x \in A.\ P(x)) \wedge (\forall x \in B.\ P(x))$
⟨*proof*⟩

lemma *bex-Un*: $(\exists x \in A \cup B.\ P(x)) \longleftrightarrow (\exists x \in A.\ P(x)) \mid (\exists x \in B.\ P(x))$
⟨*proof*⟩

lemma *ball-UN*: $(\forall z \in (\bigcup x \in A.\ B(x)).\ P(z)) \longleftrightarrow (\forall x \in A.\ \forall z \in B(x).\ P(z))$
⟨*proof*⟩

lemma *bex-UN*: $(\exists z \in (\bigcup x \in A.\ B(x)).\ P(z)) \longleftrightarrow (\exists x \in A.\ \exists z \in B(x).\ P(z))$
⟨*proof*⟩

4.2 Converse of a Relation

lemma *converse-iff* [*simp*]: $\langle a,b\rangle \in$ *converse*$(r) \longleftrightarrow \langle b,a\rangle \in r$
⟨*proof*⟩

lemma *converseI* [*intro*!]: $\langle a,b\rangle \in r \Longrightarrow \langle b,a\rangle \in$ *converse*(r)
⟨*proof*⟩

lemma *converseD*: $\langle a,b\rangle \in$ *converse*$(r) \Longrightarrow \langle b,a\rangle \in r$
⟨*proof*⟩

lemma *converseE* [*elim*!]:
$\llbracket yx \in$ *converse*(r);
$\bigwedge x\ y.\ \llbracket yx=\langle y,x\rangle;\ \ \langle x,y\rangle \in r\rrbracket \Longrightarrow P\rrbracket$
$\Longrightarrow P$
⟨*proof*⟩

lemma *converse-converse*: $r \subseteq Sigma(A,B) \Longrightarrow$ *converse*(*converse*$(r)) = r$
⟨*proof*⟩

lemma *converse-type*: $r \subseteq A*B \Longrightarrow$ *converse*$(r) \subseteq B*A$
⟨*proof*⟩

lemma *converse-prod* [*simp*]: *converse*$(A*B) = B*A$
⟨*proof*⟩

lemma *converse-empty* [*simp*]: *converse*$(0) = 0$
⟨*proof*⟩

lemma *converse-subset-iff*:
$A \subseteq Sigma(X,Y) \Longrightarrow$ *converse*$(A) \subseteq$ *converse*$(B) \longleftrightarrow A \subseteq B$
⟨*proof*⟩

4.3 Finite Set Constructions Using *cons*

lemma *cons-subsetI*: $\llbracket a \in C;\ B \subseteq C\rrbracket \Longrightarrow cons(a,B) \subseteq C$
⟨*proof*⟩

lemma *subset-consI*: $B \subseteq cons(a,B)$
⟨*proof*⟩

lemma *cons-subset-iff* [*iff*]: $cons(a,B) \subseteq C \longleftrightarrow a \in C \land B \subseteq C$
⟨*proof*⟩

lemmas *cons-subsetE* = *cons-subset-iff* [*THEN iffD1*, *THEN conjE*]

lemma *subset-empty-iff*: $A \subseteq 0 \longleftrightarrow A=0$
⟨*proof*⟩

lemma *subset-cons-iff*: $C \subseteq cons(a,B) \longleftrightarrow C \subseteq B \mid (a \in C \wedge C - \{a\} \subseteq B)$
⟨*proof*⟩

lemma *cons-eq*: $\{a\} \cup B = cons(a,B)$
⟨*proof*⟩

lemma *cons-commute*: $cons(a,\ cons(b,\ C)) = cons(b,\ cons(a,\ C))$
⟨*proof*⟩

lemma *cons-absorb*: $a\colon B \Longrightarrow cons(a,B) = B$
⟨*proof*⟩

lemma *cons-Diff*: $a\colon B \Longrightarrow cons(a,\ B - \{a\}) = B$
⟨*proof*⟩

lemma *Diff-cons-eq*: $cons(a,B) - C = (\textit{if } a \in C \textit{ then } B - C \textit{ else } cons(a,B-C))$
⟨*proof*⟩

lemma *equal-singleton*: $[\![a\colon C;\ \bigwedge y.\ y \in C \Longrightarrow y=b]\!] \Longrightarrow C = \{b\}$
⟨*proof*⟩

lemma [*simp*]: $cons(a,cons(a,B)) = cons(a,B)$
⟨*proof*⟩

lemma *singleton-subsetI*: $a \in C \Longrightarrow \{a\} \subseteq C$
⟨*proof*⟩

lemma *singleton-subsetD*: $\{a\} \subseteq C \Longrightarrow a \in C$
⟨*proof*⟩

lemma *subset-succI*: $i \subseteq succ(i)$
⟨*proof*⟩

lemma *succ-subsetI*: $[\![i \in j;\ i \subseteq j]\!] \Longrightarrow succ(i) \subseteq j$
⟨*proof*⟩

lemma *succ-subsetE*:
$[\![succ(i) \subseteq j;\ [\![i \in j;\ i \subseteq j]\!] \Longrightarrow P]\!] \Longrightarrow P$
⟨*proof*⟩

lemma *succ-subset-iff*: $succ(a) \subseteq B \longleftrightarrow (a \subseteq B \wedge a \in B)$
⟨*proof*⟩

4.4 Binary Intersection

lemma *Int-subset-iff*: $C \subseteq A \cap B \longleftrightarrow C \subseteq A \wedge C \subseteq B$
⟨*proof*⟩

lemma *Int-lower1*: $A \cap B \subseteq A$
⟨*proof*⟩

lemma *Int-lower2*: $A \cap B \subseteq B$
⟨*proof*⟩

lemma *Int-greatest*: $[\![C \subseteq A;\ \ C \subseteq B]\!] \Longrightarrow C \subseteq A \cap B$
⟨*proof*⟩

lemma *Int-cons*: $cons(a,B) \cap C \subseteq cons(a,\ B \cap C)$
⟨*proof*⟩

lemma *Int-absorb* [*simp*]: $A \cap A = A$
⟨*proof*⟩

lemma *Int-left-absorb*: $A \cap (A \cap B) = A \cap B$
⟨*proof*⟩

lemma *Int-commute*: $A \cap B = B \cap A$
⟨*proof*⟩

lemma *Int-left-commute*: $A \cap (B \cap C) = B \cap (A \cap C)$
⟨*proof*⟩

lemma *Int-assoc*: $(A \cap B) \cap C\ =\ A \cap (B \cap C)$
⟨*proof*⟩

lemmas *Int-ac*= *Int-assoc Int-left-absorb Int-commute Int-left-commute*

lemma *Int-absorb1*: $B \subseteq A \Longrightarrow A \cap B = B$
⟨*proof*⟩

lemma *Int-absorb2*: $A \subseteq B \Longrightarrow A \cap B = A$
⟨*proof*⟩

lemma *Int-Un-distrib*: $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$
⟨*proof*⟩

lemma *Int-Un-distrib2*: $(B \cup C) \cap A = (B \cap A) \cup (C \cap A)$
⟨*proof*⟩

lemma *subset-Int-iff*: $A \subseteq B \longleftrightarrow A \cap B = A$
⟨*proof*⟩

lemma *subset-Int-iff2*: $A{\subseteq}B \longleftrightarrow B \cap A = A$
⟨*proof*⟩

lemma *Int-Diff-eq*: $C{\subseteq}A \Longrightarrow (A{-}B) \cap C = C{-}B$
⟨*proof*⟩

lemma *Int-cons-left*:
$cons(a,A) \cap B = (\mathit{if}\ a \in B\ \mathit{then}\ cons(a, A \cap B)\ \mathit{else}\ A \cap B)$
⟨*proof*⟩

lemma *Int-cons-right*:
$A \cap cons(a, B) = (\mathit{if}\ a \in A\ \mathit{then}\ cons(a, A \cap B)\ \mathit{else}\ A \cap B)$
⟨*proof*⟩

lemma *cons-Int-distrib*: $cons(x, A \cap B) = cons(x, A) \cap cons(x, B)$
⟨*proof*⟩

4.5 Binary Union

lemma *Un-subset-iff*: $A \cup B \subseteq C \longleftrightarrow A \subseteq C \land B \subseteq C$
⟨*proof*⟩

lemma *Un-upper1*: $A \subseteq A \cup B$
⟨*proof*⟩

lemma *Un-upper2*: $B \subseteq A \cup B$
⟨*proof*⟩

lemma *Un-least*: $[\![A{\subseteq}C;\ B{\subseteq}C]\!] \Longrightarrow A \cup B \subseteq C$
⟨*proof*⟩

lemma *Un-cons*: $cons(a,B) \cup C = cons(a, B \cup C)$
⟨*proof*⟩

lemma *Un-absorb* [*simp*]: $A \cup A = A$
⟨*proof*⟩

lemma *Un-left-absorb*: $A \cup (A \cup B) = A \cup B$
⟨*proof*⟩

lemma *Un-commute*: $A \cup B = B \cup A$
⟨*proof*⟩

lemma *Un-left-commute*: $A \cup (B \cup C) = B \cup (A \cup C)$
⟨*proof*⟩

lemma *Un-assoc*: $(A \cup B) \cup C = A \cup (B \cup C)$
⟨*proof*⟩

lemmas *Un-ac* = *Un-assoc Un-left-absorb Un-commute Un-left-commute*

lemma *Un-absorb1*: $A \subseteq B \Longrightarrow A \cup B = B$
⟨*proof*⟩

lemma *Un-absorb2*: $B \subseteq A \Longrightarrow A \cup B = A$
⟨*proof*⟩

lemma *Un-Int-distrib*: $(A \cap B) \cup C = (A \cup C) \cap (B \cup C)$
⟨*proof*⟩

lemma *subset-Un-iff*: $A \subseteq B \longleftrightarrow A \cup B = B$
⟨*proof*⟩

lemma *subset-Un-iff2*: $A \subseteq B \longleftrightarrow B \cup A = B$
⟨*proof*⟩

lemma *Un-empty* [*iff*]: $(A \cup B = 0) \longleftrightarrow (A = 0 \wedge B = 0)$
⟨*proof*⟩

lemma *Un-eq-Union*: $A \cup B = \bigcup(\{A, B\})$
⟨*proof*⟩

4.6 Set Difference

lemma *Diff-subset*: $A-B \subseteq A$
⟨*proof*⟩

lemma *Diff-contains*: $[\![C \subseteq A; \ C \cap B = 0]\!] \Longrightarrow C \subseteq A-B$
⟨*proof*⟩

lemma *subset-Diff-cons-iff*: $B \subseteq A - cons(c,C) \longleftrightarrow B \subseteq A-C \wedge c \notin B$
⟨*proof*⟩

lemma *Diff-cancel*: $A - A = 0$
⟨*proof*⟩

lemma *Diff-triv*: $A \cap B = 0 \Longrightarrow A - B = A$
⟨*proof*⟩

lemma *empty-Diff* [*simp*]: $0 - A = 0$
⟨*proof*⟩

lemma *Diff-0* [*simp*]: $A - 0 = A$
⟨*proof*⟩

lemma *Diff-eq-0-iff*: $A - B = 0 \longleftrightarrow A \subseteq B$
⟨*proof*⟩

lemma *Diff-cons*: $A - cons(a,B) = A - B - \{a\}$
⟨*proof*⟩

lemma *Diff-cons2*: $A - cons(a,B) = A - \{a\} - B$
⟨*proof*⟩

lemma *Diff-disjoint*: $A \cap (B-A) = 0$
⟨*proof*⟩

lemma *Diff-partition*: $A\subseteq B \Longrightarrow A \cup (B-A) = B$
⟨*proof*⟩

lemma *subset-Un-Diff*: $A \subseteq B \cup (A - B)$
⟨*proof*⟩

lemma *double-complement*: $[\![A\subseteq B;\ B\subseteq C]\!] \Longrightarrow B-(C-A) = A$
⟨*proof*⟩

lemma *double-complement-Un*: $(A \cup B) - (B-A) = A$
⟨*proof*⟩

lemma *Un-Int-crazy*:
$(A \cap B) \cup (B \cap C) \cup (C \cap A) = (A \cup B) \cap (B \cup C) \cap (C \cup A)$
⟨*proof*⟩

lemma *Diff-Un*: $A - (B \cup C) = (A-B) \cap (A-C)$
⟨*proof*⟩

lemma *Diff-Int*: $A - (B \cap C) = (A-B) \cup (A-C)$
⟨*proof*⟩

lemma *Un-Diff*: $(A \cup B) - C = (A - C) \cup (B - C)$
⟨*proof*⟩

lemma *Int-Diff*: $(A \cap B) - C = A \cap (B - C)$
⟨*proof*⟩

lemma *Diff-Int-distrib*: $C \cap (A-B) = (C \cap A) - (C \cap B)$
⟨*proof*⟩

lemma *Diff-Int-distrib2*: $(A-B) \cap C = (A \cap C) - (B \cap C)$
⟨*proof*⟩

lemma *Un-Int-assoc-iff*: $(A \cap B) \cup C = A \cap (B \cup C) \longleftrightarrow C\subseteq A$
⟨*proof*⟩

4.7 Big Union and Intersection

lemma *Union-subset-iff*: $\bigcup(A) \subseteq C \longleftrightarrow (\forall x{\in}A.\ x \subseteq C)$
⟨*proof*⟩

lemma *Union-upper*: $B{\in}A \Longrightarrow B \subseteq \bigcup(A)$
⟨*proof*⟩

lemma *Union-least*: $[\![\bigwedge x.\ x{\in}A \Longrightarrow x{\subseteq}C]\!] \Longrightarrow \bigcup(A) \subseteq C$
⟨*proof*⟩

lemma *Union-cons* [*simp*]: $\bigcup(cons(a,B)) = a \cup \bigcup(B)$
⟨*proof*⟩

lemma *Union-Un-distrib*: $\bigcup(A \cup B) = \bigcup(A) \cup \bigcup(B)$
⟨*proof*⟩

lemma *Union-Int-subset*: $\bigcup(A \cap B) \subseteq \bigcup(A) \cap \bigcup(B)$
⟨*proof*⟩

lemma *Union-disjoint*: $\bigcup(C) \cap A = 0 \longleftrightarrow (\forall B{\in}C.\ B \cap A = 0)$
⟨*proof*⟩

lemma *Union-empty-iff*: $\bigcup(A) = 0 \longleftrightarrow (\forall B{\in}A.\ B{=}0)$
⟨*proof*⟩

lemma *Int-Union2*: $\bigcup(B) \cap A = (\bigcup C{\in}B.\ C \cap A)$
⟨*proof*⟩

lemma *Inter-subset-iff*: $A{\neq}0 \Longrightarrow C \subseteq \bigcap(A) \longleftrightarrow (\forall x{\in}A.\ C \subseteq x)$
⟨*proof*⟩

lemma *Inter-lower*: $B{\in}A \Longrightarrow \bigcap(A) \subseteq B$
⟨*proof*⟩

lemma *Inter-greatest*: $[\![A{\neq}0;\ \bigwedge x.\ x{\in}A \Longrightarrow C{\subseteq}x]\!] \Longrightarrow C \subseteq \bigcap(A)$
⟨*proof*⟩

lemma *INT-lower*: $x{\in}A \Longrightarrow (\bigcap x{\in}A.\ B(x)) \subseteq B(x)$
⟨*proof*⟩

lemma *INT-greatest*: $[\![A{\neq}0;\ \bigwedge x.\ x{\in}A \Longrightarrow C{\subseteq}B(x)]\!] \Longrightarrow C \subseteq (\bigcap x{\in}A.\ B(x))$
⟨*proof*⟩

lemma *Inter-0* [*simp*]: $\bigcap(0) = 0$
⟨*proof*⟩

lemma *Inter-Un-subset*:
$[\![z{\in}A;\ z{\in}B]\!] \Longrightarrow \bigcap(A) \cup \bigcap(B) \subseteq \bigcap(A \cap B)$
⟨*proof*⟩

lemma *Inter-Un-distrib*:
$[\![A{\neq}0;\ \ B{\neq}0]\!] \Longrightarrow \bigcap(A \cup B) = \bigcap(A) \cap \bigcap(B)$
⟨*proof*⟩

lemma *Union-singleton*: $\bigcup(\{b\}) = b$
⟨*proof*⟩

lemma *Inter-singleton*: $\bigcap(\{b\}) = b$
⟨*proof*⟩

lemma *Inter-cons* [*simp*]:
$\bigcap(cons(a,B)) = (if\ B{=}0\ then\ a\ else\ a \cap \bigcap(B))$
⟨*proof*⟩

4.8 Unions and Intersections of Families

lemma *subset-UN-iff-eq*: $A \subseteq (\bigcup i{\in}I.\ B(i)) \longleftrightarrow A = (\bigcup i{\in}I.\ A \cap B(i))$
⟨*proof*⟩

lemma *UN-subset-iff*: $(\bigcup x{\in}A.\ B(x)) \subseteq C \longleftrightarrow (\forall x{\in}A.\ B(x) \subseteq C)$
⟨*proof*⟩

lemma *UN-upper*: $x{\in}A \Longrightarrow B(x) \subseteq (\bigcup x{\in}A.\ B(x))$
⟨*proof*⟩

lemma *UN-least*: $[\![\bigwedge x.\ x{\in}A \Longrightarrow B(x){\subseteq}C]\!] \Longrightarrow (\bigcup x{\in}A.\ B(x)) \subseteq C$
⟨*proof*⟩

lemma *Union-eq-UN*: $\bigcup(A) = (\bigcup x{\in}A.\ x)$
⟨*proof*⟩

lemma *Inter-eq-INT*: $\bigcap(A) = (\bigcap x{\in}A.\ x)$
⟨*proof*⟩

lemma *UN-0* [*simp*]: $(\bigcup i{\in}0.\ A(i)) = 0$
⟨*proof*⟩

lemma *UN-singleton*: $(\bigcup x{\in}A.\ \{x\}) = A$
⟨*proof*⟩

lemma *UN-Un*: $(\bigcup i{\in}\ A \cup B.\ C(i)) = (\bigcup i{\in}\ A.\ C(i)) \cup (\bigcup i{\in}B.\ C(i))$
⟨*proof*⟩

lemma *INT-Un*: $(\bigcap i{\in}I \cup J.\ A(i)) =$
$(\mathit{if}\ I{=}0\ \mathit{then}\ \bigcap j{\in}J.\ A(j)$
$\mathit{else\ if}\ J{=}0\ \mathit{then}\ \bigcap i{\in}I.\ A(i)$
$\mathit{else}\ ((\bigcap i{\in}I.\ A(i)) \cap\ (\bigcap j{\in}J.\ A(j))))$
⟨*proof*⟩

lemma *UN-UN-flatten*: $(\bigcup x \in (\bigcup y{\in}A.\ B(y)).\ C(x)) = (\bigcup y{\in}A.\ \bigcup x{\in}\ B(y).\ C(x))$
⟨*proof*⟩

lemma *Int-UN-distrib*: $B \cap (\bigcup i{\in}I.\ A(i)) = (\bigcup i{\in}I.\ B \cap A(i))$
⟨*proof*⟩

lemma *Un-INT-distrib*: $I{\neq}0 \Longrightarrow B \cup (\bigcap i{\in}I.\ A(i)) = (\bigcap i{\in}I.\ B \cup A(i))$
⟨*proof*⟩

lemma *Int-UN-distrib2*:
$(\bigcup i{\in}I.\ A(i)) \cap (\bigcup j{\in}J.\ B(j)) = (\bigcup i{\in}I.\ \bigcup j{\in}J.\ A(i) \cap B(j))$
⟨*proof*⟩

lemma *Un-INT-distrib2*: $[\![I{\neq}0;\ \ J{\neq}0]\!] \Longrightarrow$
$(\bigcap i{\in}I.\ A(i)) \cup (\bigcap j{\in}J.\ B(j)) = (\bigcap i{\in}I.\ \bigcap j{\in}J.\ A(i) \cup B(j))$
⟨*proof*⟩

lemma *UN-constant* [*simp*]: $(\bigcup y{\in}A.\ c) = (\mathit{if}\ A{=}0\ \mathit{then}\ 0\ \mathit{else}\ c)$
⟨*proof*⟩

lemma *INT-constant* [*simp*]: $(\bigcap y{\in}A.\ c) = (\mathit{if}\ A{=}0\ \mathit{then}\ 0\ \mathit{else}\ c)$
⟨*proof*⟩

lemma *UN-RepFun* [*simp*]: $(\bigcup y{\in}\ \mathit{RepFun}(A,f).\ B(y)) = (\bigcup x{\in}A.\ B(f(x)))$
⟨*proof*⟩

lemma *INT-RepFun* [*simp*]: $(\bigcap x{\in}\mathit{RepFun}(A,f).\ B(x)) \quad = (\bigcap a{\in}A.\ B(f(a)))$
⟨*proof*⟩

lemma *INT-Union-eq*:
$0 \notin A \Longrightarrow (\bigcap x{\in}\ \bigcup(A).\ B(x)) = (\bigcap y{\in}A.\ \bigcap x{\in}y.\ B(x))$
⟨*proof*⟩

lemma *INT-UN-eq*:
$(\forall x{\in}A.\ B(x) \neq 0)$
$\Longrightarrow (\bigcap z{\in}\ (\bigcup x{\in}A.\ B(x)).\ C(z)) = (\bigcap x{\in}A.\ \bigcap z{\in}\ B(x).\ C(z))$
⟨*proof*⟩

lemma *UN-Un-distrib*:

$(\bigcup i{\in}I.\ A(i) \cup B(i)) = (\bigcup i{\in}I.\ A(i)) \ \cup \ (\bigcup i{\in}I.\ B(i))$
⟨*proof*⟩

lemma *INT-Int-distrib*:
$I{\neq}0 \Longrightarrow (\bigcap i{\in}I.\ A(i) \cap B(i)) = (\bigcap i{\in}I.\ A(i)) \cap (\bigcap i{\in}I.\ B(i))$
⟨*proof*⟩

lemma *UN-Int-subset*:
$(\bigcup z{\in}I \cap J.\ A(z)) \subseteq (\bigcup z{\in}I.\ A(z)) \cap (\bigcup z{\in}J.\ A(z))$
⟨*proof*⟩

lemma *Diff-UN*: $I{\neq}0 \Longrightarrow B - (\bigcup i{\in}I.\ A(i)) = (\bigcap i{\in}I.\ B - A(i))$
⟨*proof*⟩

lemma *Diff-INT*: $I{\neq}0 \Longrightarrow B - (\bigcap i{\in}I.\ A(i)) = (\bigcup i{\in}I.\ B - A(i))$
⟨*proof*⟩

lemma *Sigma-cons1*: $Sigma(cons(a,B),\ C) = (\{a\}{*}C(a)) \cup Sigma(B,C)$
⟨*proof*⟩

lemma *Sigma-cons2*: $A * cons(b,B) = A{*}\{b\} \cup A{*}B$
⟨*proof*⟩

lemma *Sigma-succ1*: $Sigma(succ(A),\ B) = (\{A\}{*}B(A)) \cup Sigma(A,B)$
⟨*proof*⟩

lemma *Sigma-succ2*: $A * succ(B) = A{*}\{B\} \cup A{*}B$
⟨*proof*⟩

lemma *SUM-UN-distrib1*:
$(\sum x \in (\bigcup y{\in}A.\ C(y)).\ B(x)) = (\bigcup y{\in}A.\ \sum x{\in}C(y).\ B(x))$
⟨*proof*⟩

lemma *SUM-UN-distrib2*:
$(\sum i{\in}I.\ \bigcup j{\in}J.\ C(i,j)) = (\bigcup j{\in}J.\ \sum i{\in}I.\ C(i,j))$
⟨*proof*⟩

lemma *SUM-Un-distrib1*:
$(\sum i{\in}I \cup J.\ C(i)) = (\sum i{\in}I.\ C(i)) \cup (\sum j{\in}J.\ C(j))$
⟨*proof*⟩

lemma *SUM-Un-distrib2*:

$(\sum i \in I.\ A(i) \cup B(i)) = (\sum i \in I.\ A(i)) \cup (\sum i \in I.\ B(i))$
⟨*proof*⟩

lemma *prod-Un-distrib2*: $I * (A \cup B) = I*A \cup I*B$
⟨*proof*⟩

lemma *SUM-Int-distrib1*:
$(\sum i \in I \cap J.\ C(i)) = (\sum i \in I.\ C(i)) \cap (\sum j \in J.\ C(j))$
⟨*proof*⟩

lemma *SUM-Int-distrib2*:
$(\sum i \in I.\ A(i) \cap B(i)) = (\sum i \in I.\ A(i)) \cap (\sum i \in I.\ B(i))$
⟨*proof*⟩

lemma *prod-Int-distrib2*: $I * (A \cap B) = I*A \cap I*B$
⟨*proof*⟩

lemma *SUM-eq-UN*: $(\sum i \in I.\ A(i)) = (\bigcup i \in I.\ \{i\} * A(i))$
⟨*proof*⟩

lemma *times-subset-iff*:
$(A'*B' \subseteq A*B) \longleftrightarrow (A' = 0 \mid B' = 0 \mid (A' \subseteq A) \wedge (B' \subseteq B))$
⟨*proof*⟩

lemma *Int-Sigma-eq*:
$(\sum x \in A'.\ B'(x)) \cap (\sum x \in A.\ B(x)) = (\sum x \in A' \cap A.\ B'(x) \cap B(x))$
⟨*proof*⟩

lemma *domain-iff*: a: $domain(r) \longleftrightarrow (\exists y.\ \langle a,y \rangle \in r)$
⟨*proof*⟩

lemma *domainI* [*intro*]: $\langle a,b \rangle \in r \Longrightarrow a$: $domain(r)$
⟨*proof*⟩

lemma *domainE* [*elim*!]:
$[\![a \in domain(r);\ \bigwedge y.\ \langle a,y \rangle \in r \Longrightarrow P]\!] \Longrightarrow P$
⟨*proof*⟩

lemma *domain-subset*: $domain(Sigma(A,B)) \subseteq A$
⟨*proof*⟩

lemma *domain-of-prod*: $b \in B \Longrightarrow domain(A*B) = A$
⟨*proof*⟩

lemma *domain-0* [*simp*]: $domain(0) = 0$
⟨*proof*⟩

lemma *domain-cons* [*simp*]: $domain(cons(\langle a,b\rangle,r)) = cons(a, domain(r))$
⟨*proof*⟩

lemma *domain-Un-eq* [*simp*]: $domain(A \cup B) = domain(A) \cup domain(B)$
⟨*proof*⟩

lemma *domain-Int-subset*: $domain(A \cap B) \subseteq domain(A) \cap domain(B)$
⟨*proof*⟩

lemma *domain-Diff-subset*: $domain(A) - domain(B) \subseteq domain(A - B)$
⟨*proof*⟩

lemma *domain-UN*: $domain(\bigcup x\in A. B(x)) = (\bigcup x\in A. domain(B(x)))$
⟨*proof*⟩

lemma *domain-Union*: $domain(\bigcup(A)) = (\bigcup x\in A. domain(x))$
⟨*proof*⟩

lemma *rangeI* [*intro*]: $\langle a,b\rangle\in r \Longrightarrow b \in range(r)$
⟨*proof*⟩

lemma *rangeE* [*elim*!]: $[\![b \in range(r); \bigwedge x. \langle x,b\rangle\in r \Longrightarrow P]\!] \Longrightarrow P$
⟨*proof*⟩

lemma *range-subset*: $range(A*B) \subseteq B$
⟨*proof*⟩

lemma *range-of-prod*: $a\in A \Longrightarrow range(A*B) = B$
⟨*proof*⟩

lemma *range-0* [*simp*]: $range(0) = 0$
⟨*proof*⟩

lemma *range-cons* [*simp*]: $range(cons(\langle a,b\rangle,r)) = cons(b, range(r))$
⟨*proof*⟩

lemma *range-Un-eq* [*simp*]: $range(A \cup B) = range(A) \cup range(B)$
⟨*proof*⟩

lemma *range-Int-subset*: $range(A \cap B) \subseteq range(A) \cap range(B)$
⟨*proof*⟩

lemma *range-Diff-subset*: $range(A) - range(B) \subseteq range(A - B)$

⟨*proof*⟩

lemma *domain-converse* [*simp*]: $domain(converse(r)) = range(r)$
⟨*proof*⟩

lemma *range-converse* [*simp*]: $range(converse(r)) = domain(r)$
⟨*proof*⟩

lemma *fieldI1*: $\langle a,b\rangle \in r \Longrightarrow a \in field(r)$
⟨*proof*⟩

lemma *fieldI2*: $\langle a,b\rangle \in r \Longrightarrow b \in field(r)$
⟨*proof*⟩

lemma *fieldCI* [*intro*]:
$(\neg \langle c,a\rangle \in r \Longrightarrow \langle a,b\rangle \in r) \Longrightarrow a \in field(r)$
⟨*proof*⟩

lemma *fieldE* [*elim*!]:
$[\![a \in field(r);$
$\bigwedge x. \langle a,x\rangle \in r \Longrightarrow P;$
$\bigwedge x. \langle x,a\rangle \in r \Longrightarrow P]\!] \Longrightarrow P$
⟨*proof*⟩

lemma *field-subset*: $field(A*B) \subseteq A \cup B$
⟨*proof*⟩

lemma *domain-subset-field*: $domain(r) \subseteq field(r)$
⟨*proof*⟩

lemma *range-subset-field*: $range(r) \subseteq field(r)$
⟨*proof*⟩

lemma *domain-times-range*: $r \subseteq Sigma(A,B) \Longrightarrow r \subseteq domain(r)*range(r)$
⟨*proof*⟩

lemma *field-times-field*: $r \subseteq Sigma(A,B) \Longrightarrow r \subseteq field(r)*field(r)$
⟨*proof*⟩

lemma *relation-field-times-field*: $relation(r) \Longrightarrow r \subseteq field(r)*field(r)$
⟨*proof*⟩

lemma *field-of-prod*: $field(A*A) = A$
⟨*proof*⟩

lemma *field-0* [*simp*]: $field(0) = 0$

⟨*proof*⟩

lemma *field-cons* [*simp*]: $field(cons(\langle a,b\rangle,r)) = cons(a, cons(b, field(r)))$
⟨*proof*⟩

lemma *field-Un-eq* [*simp*]: $field(A \cup B) = field(A) \cup field(B)$
⟨*proof*⟩

lemma *field-Int-subset*: $field(A \cap B) \subseteq field(A) \cap field(B)$
⟨*proof*⟩

lemma *field-Diff-subset*: $field(A) - field(B) \subseteq field(A - B)$
⟨*proof*⟩

lemma *field-converse* [*simp*]: $field(converse(r)) = field(r)$
⟨*proof*⟩

lemma *rel-Union*: $(\forall x\in S. \exists A\ B. x \subseteq A*B) \Longrightarrow$
$\bigcup(S) \subseteq domain(\bigcup(S)) * range(\bigcup(S))$
⟨*proof*⟩

lemma *rel-Un*: $[\![r \subseteq A*B; \ s \subseteq C*D]\!] \Longrightarrow (r \cup s) \subseteq (A \cup C) * (B \cup D)$
⟨*proof*⟩

lemma *domain-Diff-eq*: $[\![\langle a,c\rangle \in r; c\neq b]\!] \Longrightarrow domain(r-\{\langle a,b\rangle\}) = domain(r)$
⟨*proof*⟩

lemma *range-Diff-eq*: $[\![\langle c,b\rangle \in r; c\neq a]\!] \Longrightarrow range(r-\{\langle a,b\rangle\}) = range(r)$
⟨*proof*⟩

4.9 Image of a Set under a Function or Relation

lemma *image-iff*: $b \in r``A \longleftrightarrow (\exists x\in A. \langle x,b\rangle\in r)$
⟨*proof*⟩

lemma *image-singleton-iff*: $b \in r``\{a\} \longleftrightarrow \langle a,b\rangle\in r$
⟨*proof*⟩

lemma *imageI* [*intro*]: $[\![\langle a,b\rangle\in r; \ a\in A]\!] \Longrightarrow b \in r``A$
⟨*proof*⟩

lemma *imageE* [*elim*!]:
$[\![b: r``A; \ \bigwedge x.[\![\langle x,b\rangle\in r; \ x\in A]\!] \Longrightarrow P]\!] \Longrightarrow P$
⟨*proof*⟩

lemma *image-subset*: $r \subseteq A*B \Longrightarrow r``C \subseteq B$
⟨*proof*⟩

lemma *image-0* [*simp*]: $r``0 = 0$
⟨*proof*⟩

lemma *image-Un* [*simp*]: $r``(A \cup B) = (r``A) \cup (r``B)$
⟨*proof*⟩

lemma *image-UN*: $r `` (\bigcup x\in A.\ B(x)) = (\bigcup x\in A.\ r `` B(x))$
⟨*proof*⟩

lemma *Collect-image-eq*:
$\{z \in Sigma(A,B).\ P(z)\} `` C = (\bigcup x \in A.\ \{y \in B(x).\ x \in C \wedge P(\langle x,y\rangle)\})$
⟨*proof*⟩

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

lemma *image-Int-square-subset*: $(r \cap A*A)``B \subseteq (r``B) \cap A$
⟨*proof*⟩

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

lemma *image-0-left* [*simp*]: $0``A = 0$
⟨*proof*⟩

lemma *image-Un-left*: $(r \cup s)``A = (r``A) \cup (s``A)$
⟨*proof*⟩

lemma *image-Int-subset-left*: $(r \cap s)``A \subseteq (r``A) \cap (s``A)$
⟨*proof*⟩

4.10 Inverse Image of a Set under a Function or Relation

lemma *vimage-iff*:
$a \in r-``B \longleftrightarrow (\exists y\in B.\ \langle a,y\rangle\in r)$
⟨*proof*⟩

lemma *vimage-singleton-iff*: $a \in r-``\{b\} \longleftrightarrow \langle a,b\rangle\in r$
⟨*proof*⟩

lemma *vimageI* [*intro*]: $[\![\langle a,b\rangle\in r;\ \ b\in B]\!] \Longrightarrow a \in r-``B$
⟨*proof*⟩

lemma *vimageE* [*elim*!]:
$[\![a: r-``B;\ \ \bigwedge x.[\![\langle a,x\rangle\in r;\ \ x\in B]\!] \Longrightarrow P]\!] \Longrightarrow P$
⟨*proof*⟩

lemma *vimage-subset*: $r \subseteq A*B \Longrightarrow r-``C \subseteq A$
⟨*proof*⟩

lemma *vimage-0* [*simp*]: $r-``0 = 0$
⟨*proof*⟩

lemma *vimage-Un* [*simp*]: $r-``(A \cup B) = (r-``A) \cup (r-``B)$
⟨*proof*⟩

lemma *vimage-Int-subset*: $r-``(A \cap B) \subseteq (r-``A) \cap (r-``B)$
⟨*proof*⟩

lemma *vimage-eq-UN*: $f -``B = (\bigcup y{\in}B. f-``\{y\})$
⟨*proof*⟩

lemma *function-vimage-Int*:
$function(f) \Longrightarrow f-``(A \cap B) = (f-``A) \cap (f-``B)$
⟨*proof*⟩

lemma *function-vimage-Diff*: $function(f) \Longrightarrow f-``(A-B) = (f-``A) - (f-``B)$
⟨*proof*⟩

lemma *function-image-vimage*: $function(f) \Longrightarrow f `` (f-`` A) \subseteq A$
⟨*proof*⟩

lemma *vimage-Int-square-subset*: $(r \cap A*A)-``B \subseteq (r-``B) \cap A$
⟨*proof*⟩

lemma *vimage-Int-square*: $B{\subseteq}A \Longrightarrow (r \cap A*A)-``B = (r-``B) \cap A$
⟨*proof*⟩

lemma *vimage-0-left* [*simp*]: $0-``A = 0$
⟨*proof*⟩

lemma *vimage-Un-left*: $(r \cup s)-``A = (r-``A) \cup (s-``A)$
⟨*proof*⟩

lemma *vimage-Int-subset-left*: $(r \cap s)-``A \subseteq (r-``A) \cap (s-``A)$
⟨*proof*⟩

lemma *converse-Un* [*simp*]: $converse(A \cup B) = converse(A) \cup converse(B)$

⟨*proof*⟩

lemma *converse-Int* [*simp*]: $converse(A \cap B) = converse(A) \cap converse(B)$
⟨*proof*⟩

lemma *converse-Diff* [*simp*]: $converse(A - B) = converse(A) - converse(B)$
⟨*proof*⟩

lemma *converse-UN* [*simp*]: $converse(\bigcup x{\in}A.\ B(x)) = (\bigcup x{\in}A.\ converse(B(x)))$
⟨*proof*⟩

lemma *converse-INT* [*simp*]:
$converse(\bigcap x{\in}A.\ B(x)) = (\bigcap x{\in}A.\ converse(B(x)))$
⟨*proof*⟩

4.11 Powerset Operator

lemma *Pow-0* [*simp*]: $Pow(0) = \{0\}$
⟨*proof*⟩

lemma *Pow-insert*: $Pow\ (cons(a,A)) = Pow(A) \cup \{cons(a,X)\ .\ X{:}\ Pow(A)\}$
⟨*proof*⟩

lemma *Un-Pow-subset*: $Pow(A) \cup Pow(B) \subseteq Pow(A \cup B)$
⟨*proof*⟩

lemma *UN-Pow-subset*: $(\bigcup x{\in}A.\ Pow(B(x))) \subseteq Pow(\bigcup x{\in}A.\ B(x))$
⟨*proof*⟩

lemma *subset-Pow-Union*: $A \subseteq Pow(\bigcup(A))$
⟨*proof*⟩

lemma *Union-Pow-eq* [*simp*]: $\bigcup(Pow(A)) = A$
⟨*proof*⟩

lemma *Union-Pow-iff*: $\bigcup(A) \in Pow(B) \longleftrightarrow A \in Pow(Pow(B))$
⟨*proof*⟩

lemma *Pow-Int-eq* [*simp*]: $Pow(A \cap B) = Pow(A) \cap Pow(B)$
⟨*proof*⟩

lemma *Pow-INT-eq*: $A{\neq}0 \Longrightarrow Pow(\bigcap x{\in}A.\ B(x)) = (\bigcap x{\in}A.\ Pow(B(x)))$
⟨*proof*⟩

4.12 RepFun

lemma *RepFun-subset*: $[\![\bigwedge x.\ x{\in}A \Longrightarrow f(x) \in B]\!] \Longrightarrow \{f(x).\ x{\in}A\} \subseteq B$
⟨*proof*⟩

lemma *RepFun-eq-0-iff* [*simp*]: $\{f(x).x \in A\}=0 \longleftrightarrow A=0$
⟨*proof*⟩

lemma *RepFun-constant* [*simp*]: $\{c.\ x \in A\} = (\textit{if } A=0 \textit{ then } 0 \textit{ else } \{c\})$
⟨*proof*⟩

4.13 Collect

lemma *Collect-subset*: $Collect(A,P) \subseteq A$
⟨*proof*⟩

lemma *Collect-Un*: $Collect(A \cup B,\ P) = Collect(A,P) \cup Collect(B,P)$
⟨*proof*⟩

lemma *Collect-Int*: $Collect(A \cap B,\ P) = Collect(A,P) \cap Collect(B,P)$
⟨*proof*⟩

lemma *Collect-Diff*: $Collect(A - B,\ P) = Collect(A,P) - Collect(B,P)$
⟨*proof*⟩

lemma *Collect-cons*: $\{x \in cons(a,B).\ P(x)\} =$
$(\textit{if } P(a) \textit{ then } cons(a,\ \{x \in B.\ P(x)\}) \textit{ else } \{x \in B.\ P(x)\})$
⟨*proof*⟩

lemma *Int-Collect-self-eq*: $A \cap Collect(A,P) = Collect(A,P)$
⟨*proof*⟩

lemma *Collect-Collect-eq* [*simp*]:
$Collect(Collect(A,P),\ Q) = Collect(A,\ \lambda x.\ P(x) \wedge Q(x))$
⟨*proof*⟩

lemma *Collect-Int-Collect-eq*:
$Collect(A,P) \cap Collect(A,Q) = Collect(A,\ \lambda x.\ P(x) \wedge Q(x))$
⟨*proof*⟩

lemma *Collect-Union-eq* [*simp*]:
$Collect(\bigcup x \in A.\ B(x),\ P) = (\bigcup x \in A.\ Collect(B(x),\ P))$
⟨*proof*⟩

lemma *Collect-Int-left*: $\{x \in A.\ P(x)\} \cap B = \{x \in A \cap B.\ P(x)\}$
⟨*proof*⟩

lemma *Collect-Int-right*: $A \cap \{x \in B.\ P(x)\} = \{x \in A \cap B.\ P(x)\}$
⟨*proof*⟩

lemma *Collect-disj-eq*: $\{x \in A.\ P(x) \mid Q(x)\} = Collect(A,\ P) \cup Collect(A,\ Q)$
⟨*proof*⟩

lemma *Collect-conj-eq*: $\{x \in A.\ P(x) \wedge Q(x)\} = Collect(A,\ P) \cap Collect(A,\ Q)$

⟨*proof*⟩

lemmas *subset-SIs* = *subset-refl cons-subsetI subset-consI*
Union-least UN-least Un-least
Inter-greatest Int-greatest RepFun-subset
Un-upper1 Un-upper2 Int-lower1 Int-lower2

⟨*ML*⟩

end

5 Least and Greatest Fixed Points; the Knaster-Tarski Theorem

theory *Fixedpt* **imports** *equalities* **begin**

definition

bnd-mono :: $[i,i\Rightarrow i]\Rightarrow o$ **where**
$bnd\text{-}mono(D,h) \equiv h(D)<=D \wedge (\forall W\ X.\ W<=X \longrightarrow X<=D \longrightarrow h(W) \subseteq h(X))$

definition
lfp :: $[i,i\Rightarrow i]\Rightarrow i$ **where**
$lfp(D,h) \equiv \bigcap(\{X: Pow(D).\ h(X) \subseteq X\})$

definition
gfp :: $[i,i\Rightarrow i]\Rightarrow i$ **where**
$gfp(D,h) \equiv \bigcup(\{X: Pow(D).\ X \subseteq h(X)\})$

The theorem is proved in the lattice of subsets of D, namely $Pow(D)$, with Inter as the greatest lower bound.

5.1 Monotone Operators

lemma *bnd-monoI*:
$[\![h(D)<=D;$
$\bigwedge W\ X.\ [\![W<=D;\ \ X<=D;\ \ W<=X]\!] \Longrightarrow h(W) \subseteq h(X)$
$]\!] \Longrightarrow bnd\text{-}mono(D,h)$
⟨*proof*⟩

lemma *bnd-monoD1*: $bnd\text{-}mono(D,h) \Longrightarrow h(D) \subseteq D$
⟨*proof*⟩

lemma *bnd-monoD2*: $[\![bnd\text{-}mono(D,h);\ \ W<=X;\ \ X<=D]\!] \Longrightarrow h(W) \subseteq h(X)$
⟨*proof*⟩

lemma *bnd-mono-subset*:

$[\![bnd\text{-}mono(D,h);\ \ X{<=}D]\!] \Longrightarrow h(X) \subseteq D$
⟨*proof*⟩

lemma *bnd-mono-Un*:
$[\![bnd\text{-}mono(D,h);\ \ A \subseteq D;\ \ B \subseteq D]\!] \Longrightarrow h(A) \cup h(B) \subseteq h(A \cup B)$
⟨*proof*⟩

lemma *bnd-mono-UN*:
$[\![bnd\text{-}mono(D,h);\ \ \forall i{\in}I.\ A(i) \subseteq D]\!]$
$\Longrightarrow (\bigcup i{\in}I.\ h(A(i))) \subseteq h((\bigcup i{\in}I.\ A(i)))$
⟨*proof*⟩

lemma *bnd-mono-Int*:
$[\![bnd\text{-}mono(D,h);\ \ A \subseteq D;\ \ B \subseteq D]\!] \Longrightarrow h(A \cap B) \subseteq h(A) \cap h(B)$
⟨*proof*⟩

5.2 Proof of Knaster-Tarski Theorem using *lfp*

lemma *lfp-lowerbound*:
$[\![h(A) \subseteq A;\ \ A{<=}D]\!] \Longrightarrow lfp(D,h) \subseteq A$
⟨*proof*⟩

lemma *lfp-subset*: $lfp(D,h) \subseteq D$
⟨*proof*⟩

lemma *def-lfp-subset*: $A \equiv lfp(D,h) \Longrightarrow A \subseteq D$
⟨*proof*⟩

lemma *lfp-greatest*:
$[\![h(D) \subseteq D;\ \ \bigwedge X.\ [\![h(X) \subseteq X;\ \ X{<=}D]\!] \Longrightarrow A{<=}X]\!] \Longrightarrow A \subseteq lfp(D,h)$
⟨*proof*⟩

lemma *lfp-lemma1*:
$[\![bnd\text{-}mono(D,h);\ \ h(A){<=}A;\ \ A{<=}D]\!] \Longrightarrow h(lfp(D,h)) \subseteq A$
⟨*proof*⟩

lemma *lfp-lemma2*: $bnd\text{-}mono(D,h) \Longrightarrow h(lfp(D,h)) \subseteq lfp(D,h)$
⟨*proof*⟩

lemma *lfp-lemma3*:
$bnd\text{-}mono(D,h) \Longrightarrow lfp(D,h) \subseteq h(lfp(D,h))$
⟨*proof*⟩

lemma *lfp-unfold*: $bnd\text{-}mono(D,h) \Longrightarrow lfp(D,h) = h(lfp(D,h))$
⟨*proof*⟩

lemma *def-lfp-unfold*:
$[\![A \equiv lfp(D,h);\ \ bnd\text{-}mono(D,h)]\!] \Longrightarrow A = h(A)$
⟨*proof*⟩

5.3 General Induction Rule for Least Fixedpoints

lemma *Collect-is-pre-fixedpt*:
$[\![bnd\text{-}mono(D,h);\ \ \bigwedge x.\ x \in h(Collect(lfp(D,h),P)) \Longrightarrow P(x)]\!]$
$\Longrightarrow h(Collect(lfp(D,h),P)) \subseteq Collect(lfp(D,h),P)$
⟨*proof*⟩

lemma *induct*:
$[\![bnd\text{-}mono(D,h);\ \ a \in lfp(D,h);$
$\bigwedge x.\ x \in h(Collect(lfp(D,h),P)) \Longrightarrow P(x)$
$]\!] \Longrightarrow P(a)$
⟨*proof*⟩

lemma *def-induct*:
$[\![A \equiv lfp(D,h);\ \ bnd\text{-}mono(D,h);\ \ a{:}A;$
$\bigwedge x.\ x \in h(Collect(A,P)) \Longrightarrow P(x)$
$]\!] \Longrightarrow P(a)$
⟨*proof*⟩

lemma *lfp-Int-lowerbound*:
$[\![h(D \cap A) \subseteq A;\ \ bnd\text{-}mono(D,h)]\!] \Longrightarrow lfp(D,h) \subseteq A$
⟨*proof*⟩

lemma *lfp-mono*:
assumes *hmono*: $bnd\text{-}mono(D,h)$
and *imono*: $bnd\text{-}mono(E,i)$
and *subhi*: $\bigwedge X.\ X{<=}D \Longrightarrow h(X) \subseteq i(X)$
shows $lfp(D,h) \subseteq lfp(E,i)$
⟨*proof*⟩

lemma *lfp-mono2*:
$[\![i(D) \subseteq D;\ \ \bigwedge X.\ X{<=}D \Longrightarrow h(X) \subseteq i(X)]\!] \Longrightarrow lfp(D,h) \subseteq lfp(D,i)$
⟨*proof*⟩

lemma *lfp-cong*:
$[\![D{=}D';\ \bigwedge X.\ X \subseteq D' \Longrightarrow h(X) = h'(X)]\!] \Longrightarrow lfp(D,h) = lfp(D',h')$
⟨*proof*⟩

5.4 Proof of Knaster-Tarski Theorem using *gfp*

lemma *gfp-upperbound*: $[\![A \subseteq h(A);\ \ A<=D]\!] \Longrightarrow A \subseteq gfp(D,h)$
⟨*proof*⟩

lemma *gfp-subset*: $gfp(D,h) \subseteq D$
⟨*proof*⟩

lemma *def-gfp-subset*: $A \equiv gfp(D,h) \Longrightarrow A \subseteq D$
⟨*proof*⟩

lemma *gfp-least*:
$[\![bnd\text{-}mono(D,h);\ \ \bigwedge X.\ [\![X \subseteq h(X);\ \ X<=D]\!] \Longrightarrow X<=A]\!] \Longrightarrow$
$gfp(D,h) \subseteq A$
⟨*proof*⟩

lemma *gfp-lemma1*:
$[\![bnd\text{-}mono(D,h);\ \ A<=h(A);\ \ A<=D]\!] \Longrightarrow A \subseteq h(gfp(D,h))$
⟨*proof*⟩

lemma *gfp-lemma2*: $bnd\text{-}mono(D,h) \Longrightarrow gfp(D,h) \subseteq h(gfp(D,h))$
⟨*proof*⟩

lemma *gfp-lemma3*:
$bnd\text{-}mono(D,h) \Longrightarrow h(gfp(D,h)) \subseteq gfp(D,h)$
⟨*proof*⟩

lemma *gfp-unfold*: $bnd\text{-}mono(D,h) \Longrightarrow gfp(D,h) = h(gfp(D,h))$
⟨*proof*⟩

lemma *def-gfp-unfold*:
$[\![A \equiv gfp(D,h);\ \ bnd\text{-}mono(D,h)]\!] \Longrightarrow A = h(A)$
⟨*proof*⟩

5.5 Coinduction Rules for Greatest Fixed Points

lemma *weak-coinduct*: $[\![a\colon X;\ \ X \subseteq h(X);\ \ X \subseteq D]\!] \Longrightarrow a \in gfp(D,h)$
⟨*proof*⟩

lemma *coinduct-lemma*:
$[\![X \subseteq h(X \cup gfp(D,h));\ \ X \subseteq D;\ \ bnd\text{-}mono(D,h)]\!] \Longrightarrow$
$X \cup gfp(D,h) \subseteq h(X \cup gfp(D,h))$
⟨*proof*⟩

lemma *coinduct*:
$[\![bnd\text{-}mono(D,h);\ \ a\colon X;\ \ X \subseteq h(X \cup gfp(D,h));\ \ X \subseteq D]\!]$
$\Longrightarrow a \in gfp(D,h)$

⟨*proof*⟩

lemma *def-coinduct*:
  $[\![A \equiv gfp(D,h);\ \ bnd\text{-}mono(D,h);\ \ a\colon X;\ \ X \subseteq h(X \cup A);\ \ X \subseteq D]\!] \Longrightarrow$
  $a \in A$
⟨*proof*⟩

lemma *def-Collect-coinduct*:
  $[\![A \equiv gfp(D,\ \lambda w.\ Collect(D,P(w)));\ \ bnd\text{-}mono(D,\ \lambda w.\ Collect(D,P(w)));$
  $a\colon X;\ \ X \subseteq D;\ \ \bigwedge z.\ z\colon X \Longrightarrow P(X \cup A,\ z)]\!] \Longrightarrow$
  $a \in A$
⟨*proof*⟩

lemma *gfp-mono*:
  $[\![bnd\text{-}mono(D,h);\ \ D \subseteq E;$
  $\bigwedge X.\ X{<=}D \Longrightarrow h(X) \subseteq i(X)]\!] \Longrightarrow gfp(D,h) \subseteq gfp(E,i)$
⟨*proof*⟩

end

6 Booleans in Zermelo-Fraenkel Set Theory

theory *Bool* **imports** *pair* **begin**

abbreviation
  one (‹*1*›) **where**
  1 ≡ *succ*(*0*)

abbreviation
  two (‹*2*›) **where**
  2 ≡ *succ*(*1*)

2 is equal to bool, but is used as a number rather than a type.

definition *bool* ≡ {*0*,*1*}

definition *cond*(*b*,*c*,*d*) ≡ *if*(*b*=*1*,*c*,*d*)

definition *not*(*b*) ≡ *cond*(*b*,*0*,*1*)

definition
  and :: [*i*,*i*]⇒*i* (**infixl** ‹*and*› *70*) **where**
  a and b ≡ *cond*(*a*,*b*,*0*)

definition
  or :: [*i*,*i*]⇒*i* (**infixl** ‹*or*› *65*) **where**
  a or b ≡ *cond*(*a*,*1*,*b*)

definition
  xor :: $[i,i]\Rightarrow i$ (**infixl** ‹*xor*› *65*) **where**
    a *xor* $b \equiv cond(a,not(b),b)$

lemmas *bool-defs* = *bool-def cond-def*

lemma *singleton-0*: $\{0\} = 1$
⟨*proof*⟩

lemma *bool-1I* [*simp*,*TC*]: $1 \in bool$
⟨*proof*⟩

lemma *bool-0I* [*simp*,*TC*]: $0 \in bool$
⟨*proof*⟩

lemma *one-not-0*: $1\neq 0$
⟨*proof*⟩

lemmas *one-neq-0* = *one-not-0* [*THEN notE*]

lemma *boolE*:
    $[\![c: bool;\ \ c=1 \Longrightarrow P;\ \ c=0 \Longrightarrow P]\!] \Longrightarrow P$
⟨*proof*⟩

lemma *cond-1* [*simp*]: $cond(1,c,d) = c$
⟨*proof*⟩

lemma *cond-0* [*simp*]: $cond(0,c,d) = d$
⟨*proof*⟩

lemma *cond-type* [*TC*]: $[\![b: bool;\ \ c: A(1);\ \ d: A(0)]\!] \Longrightarrow cond(b,c,d): A(b)$
⟨*proof*⟩

lemma *cond-simple-type*: $[\![b: bool;\ \ c: A;\ \ d: A]\!] \Longrightarrow cond(b,c,d): A$
⟨*proof*⟩

lemma *def-cond-1*: $[\![\bigwedge b.\ j(b)\equiv cond(b,c,d)]\!] \Longrightarrow j(1) = c$
⟨*proof*⟩

lemma *def-cond-0*: $[\![\bigwedge b.\ j(b) \equiv cond(b,c,d)]\!] \Longrightarrow j(0) = d$
⟨*proof*⟩

lemmas *not-1* = *not-def* [*THEN def-cond-1*, *simp*]
lemmas *not-0* = *not-def* [*THEN def-cond-0*, *simp*]

lemmas *and-1* = *and-def* [*THEN def-cond-1*, *simp*]
lemmas *and-0* = *and-def* [*THEN def-cond-0*, *simp*]

lemmas *or-1* = *or-def* [*THEN def-cond-1*, *simp*]
lemmas *or-0* = *or-def* [*THEN def-cond-0*, *simp*]

lemmas *xor-1* = *xor-def* [*THEN def-cond-1*, *simp*]
lemmas *xor-0* = *xor-def* [*THEN def-cond-0*, *simp*]

lemma *not-type* [*TC*]: $a{:}bool \Longrightarrow not(a) \in bool$
⟨*proof*⟩

lemma *and-type* [*TC*]: $[\![a{:}bool;\ \ b{:}bool]\!] \Longrightarrow a\ and\ b \in bool$
⟨*proof*⟩

lemma *or-type* [*TC*]: $[\![a{:}bool;\ \ b{:}bool]\!] \Longrightarrow a\ or\ b \in bool$
⟨*proof*⟩

lemma *xor-type* [*TC*]: $[\![a{:}bool;\ \ b{:}bool]\!] \Longrightarrow a\ xor\ b \in bool$
⟨*proof*⟩

lemmas *bool-typechecks* = *bool-1I bool-0I cond-type not-type and-type or-type xor-type*

6.1 Laws About 'not'

lemma *not-not* [*simp*]: $a{:}bool \Longrightarrow not(not(a)) = a$
⟨*proof*⟩

lemma *not-and* [*simp*]: $a{:}bool \Longrightarrow not(a\ and\ b) = not(a)\ or\ not(b)$
⟨*proof*⟩

lemma *not-or* [*simp*]: $a{:}bool \Longrightarrow not(a\ or\ b) = not(a)\ and\ not(b)$
⟨*proof*⟩

6.2 Laws About 'and'

lemma *and-absorb* [*simp*]: $a{:}\ bool \Longrightarrow a\ and\ a = a$
⟨*proof*⟩

lemma *and-commute*: $[\![a{:}\ bool;\ b{:}bool]\!] \Longrightarrow a\ and\ b = b\ and\ a$
⟨*proof*⟩

lemma *and-assoc*: $a{:}\ bool \Longrightarrow (a\ and\ b)\ and\ c\ =\ a\ and\ (b\ and\ c)$

⟨*proof*⟩

lemma *and-or-distrib*: $[\![a\colon bool;\ b{:}bool;\ c{:}bool]\!] \Longrightarrow$
  $(a\ or\ b)\ and\ c\ =\ (a\ and\ c)\ or\ (b\ and\ c)$
⟨*proof*⟩

6.3 Laws About 'or'

lemma *or-absorb* [*simp*]: $a\colon bool \Longrightarrow a\ or\ a = a$
⟨*proof*⟩

lemma *or-commute*: $[\![a\colon bool;\ b{:}bool]\!] \Longrightarrow a\ or\ b = b\ or\ a$
⟨*proof*⟩

lemma *or-assoc*: $a\colon bool \Longrightarrow (a\ or\ b)\ or\ c\ =\ a\ or\ (b\ or\ c)$
⟨*proof*⟩

lemma *or-and-distrib*: $[\![a\colon bool;\ b\colon bool;\ c\colon bool]\!] \Longrightarrow$
  $(a\ and\ b)\ or\ c\ =\ (a\ or\ c)\ and\ (b\ or\ c)$
⟨*proof*⟩

definition
  bool-of-o :: $o \Rightarrow i$ **where**
  bool-of-o$(P) \equiv (if\ P\ then\ 1\ else\ 0)$

lemma [*simp*]: *bool-of-o*$(True) = 1$
⟨*proof*⟩

lemma [*simp*]: *bool-of-o*$(False) = 0$
⟨*proof*⟩

lemma [*simp,TC*]: *bool-of-o*$(P) \in bool$
⟨*proof*⟩

lemma [*simp*]: $($*bool-of-o*$(P) = 1) \longleftrightarrow P$
⟨*proof*⟩

lemma [*simp*]: $($*bool-of-o*$(P) = 0) \longleftrightarrow \neg P$
⟨*proof*⟩

end

7 Disjoint Sums

theory *Sum* **imports** *Bool equalities* **begin**

And the "Part" primitive for simultaneous recursive type definitions

definition *sum* :: $[i,i] \Rightarrow i$ (**infixr** ‹+› *65*) **where**

$A+B \equiv \{0\}*A \cup \{1\}*B$

definition *Inl* :: $i \Rightarrow i$ **where**
$Inl(a) \equiv \langle 0,a\rangle$

definition *Inr* :: $i \Rightarrow i$ **where**
$Inr(b) \equiv \langle 1,b\rangle$

definition *case* :: $[i \Rightarrow i,\ i \Rightarrow i,\ i] \Rightarrow i$ **where**
$case(c,d) \equiv (\lambda\langle y,z\rangle.\ cond(y,\ d(z),\ c(z)))$

definition *Part* :: $[i,i \Rightarrow i] \Rightarrow i$ **where**
$Part(A,h) \equiv \{x \in A.\ \exists z.\ x = h(z)\}$

7.1 Rules for the *Part* Primitive

lemma *Part-iff*:
$a \in Part(A,h) \longleftrightarrow a \in A \wedge (\exists y.\ a=h(y))$
⟨*proof*⟩

lemma *Part-eqI* [*intro*]:
$[\![a \in A;\ \ a=h(b)]\!] \Longrightarrow a \in Part(A,h)$
⟨*proof*⟩

lemmas *PartI* = *refl* [*THEN* [2] *Part-eqI*]

lemma *PartE* [*elim*!]:
$[\![a \in Part(A,h);\ \ \bigwedge z.\ [\![a \in A;\ \ a=h(z)]\!] \Longrightarrow P$
$]\!] \Longrightarrow P$
⟨*proof*⟩

lemma *Part-subset*: $Part(A,h) \subseteq A$
⟨*proof*⟩

7.2 Rules for Disjoint Sums

lemmas *sum-defs* = *sum-def Inl-def Inr-def case-def*

lemma *Sigma-bool*: $Sigma(bool,C) = C(0) + C(1)$
⟨*proof*⟩

lemma *InlI* [*intro*!,*simp*,*TC*]: $a \in A \Longrightarrow Inl(a) \in A+B$
⟨*proof*⟩

lemma *InrI* [*intro*!,*simp*,*TC*]: $b \in B \Longrightarrow Inr(b) \in A+B$
⟨*proof*⟩

lemma *sumE* [*elim*!]:
$[\![u \in A+B;$
$\bigwedge x.\ [\![x \in A;\ \ u=Inl(x)]\!] \Longrightarrow P;$
$\bigwedge y.\ [\![y \in B;\ \ u=Inr(y)]\!] \Longrightarrow P$
$]\!] \Longrightarrow P$
⟨*proof*⟩

lemma *Inl-iff* [*iff*]: $Inl(a)=Inl(b) \longleftrightarrow a=b$
⟨*proof*⟩

lemma *Inr-iff* [*iff*]: $Inr(a)=Inr(b) \longleftrightarrow a=b$
⟨*proof*⟩

lemma *Inl-Inr-iff* [*simp*]: $Inl(a)=Inr(b) \longleftrightarrow False$
⟨*proof*⟩

lemma *Inr-Inl-iff* [*simp*]: $Inr(b)=Inl(a) \longleftrightarrow False$
⟨*proof*⟩

lemma *sum-empty* [*simp*]: $0+0 = 0$
⟨*proof*⟩

lemmas *Inl-inject* = *Inl-iff* [*THEN iffD1*]
lemmas *Inr-inject* = *Inr-iff* [*THEN iffD1*]
lemmas *Inl-neq-Inr* = *Inl-Inr-iff* [*THEN iffD1*, *THEN FalseE*, *elim*!]
lemmas *Inr-neq-Inl* = *Inr-Inl-iff* [*THEN iffD1*, *THEN FalseE*, *elim*!]

lemma *InlD*: $Inl(a): A+B \Longrightarrow a \in A$
⟨*proof*⟩

lemma *InrD*: $Inr(b): A+B \Longrightarrow b \in B$
⟨*proof*⟩

lemma *sum-iff*: $u \in A+B \longleftrightarrow (\exists\, x.\ x \in A \land u=Inl(x)) \mid (\exists\, y.\ y \in B \land u=Inr(y))$
⟨*proof*⟩

lemma *Inl-in-sum-iff* [*simp*]: $(Inl(x) \in A+B) \longleftrightarrow (x \in A)$
⟨*proof*⟩

lemma *Inr-in-sum-iff* [*simp*]: $(Inr(y) \in A+B) \longleftrightarrow (y \in B)$
⟨*proof*⟩

lemma *sum-subset-iff*: $A+B \subseteq C+D \longleftrightarrow A<=C \land B<=D$
⟨*proof*⟩

lemma *sum-equal-iff*: $A+B = C+D \longleftrightarrow A=C \land B=D$
⟨*proof*⟩

lemma *sum-eq-2-times*: $A+A = 2*A$
⟨*proof*⟩

7.3 The Eliminator: *case*

lemma *case-Inl* [*simp*]: $case(c, d, Inl(a)) = c(a)$
⟨*proof*⟩

lemma *case-Inr* [*simp*]: $case(c, d, Inr(b)) = d(b)$
⟨*proof*⟩

lemma *case-type* [*TC*]:
$[\![u \in A+B;$
$\bigwedge x.\ x \in A \Longrightarrow c(x): C(Inl(x));$
$\bigwedge y.\ y \in B \Longrightarrow d(y): C(Inr(y))$
$]\!] \Longrightarrow case(c,d,u) \in C(u)$
⟨*proof*⟩

lemma *expand-case*: $u \in A+B \Longrightarrow$
$R(case(c,d,u)) \longleftrightarrow$
$((\forall x{\in}A.\ u = Inl(x) \longrightarrow R(c(x))) \land$
$(\forall y{\in}B.\ u = Inr(y) \longrightarrow R(d(y))))$
⟨*proof*⟩

lemma *case-cong*:
$[\![z \in A+B;$
$\bigwedge x.\ x \in A \Longrightarrow c(x)=c'(x);$
$\bigwedge y.\ y \in B \Longrightarrow d(y)=d'(y)$
$]\!] \Longrightarrow case(c,d,z) = case(c',d',z)$
⟨*proof*⟩

lemma *case-case*: $z \in A+B \Longrightarrow$
$case(c, d, case(\lambda x.\ Inl(c'(x)), \lambda y.\ Inr(d'(y)), z)) =$
$case(\lambda x.\ c(c'(x)), \lambda y.\ d(d'(y)), z)$
⟨*proof*⟩

7.4 More Rules for $Part(A, h)$

lemma *Part-mono*: $A<=B \Longrightarrow Part(A,h)<=Part(B,h)$
⟨*proof*⟩

lemma *Part-Collect*: $Part(Collect(A,P), h) = Collect(Part(A,h), P)$
⟨*proof*⟩

lemmas *Part-CollectE* =
  Part-Collect [*THEN equalityD1*, *THEN subsetD*, *THEN CollectE*]

lemma *Part-Inl*: $Part(A+B,Inl) = \{Inl(x).\ x \in A\}$
⟨*proof*⟩

lemma *Part-Inr*: $Part(A+B,Inr) = \{Inr(y).\ y \in B\}$
⟨*proof*⟩

lemma *PartD1*: $a \in Part(A,h) \Longrightarrow a \in A$
⟨*proof*⟩

lemma *Part-id*: $Part(A,\lambda x.\ x) = A$
⟨*proof*⟩

lemma *Part-Inr2*: $Part(A+B,\ \lambda x.\ Inr(h(x))) = \{Inr(y).\ y \in Part(B,h)\}$
⟨*proof*⟩

lemma *Part-sum-equality*: $C \subseteq A+B \Longrightarrow Part(C,Inl) \cup Part(C,Inr) = C$
⟨*proof*⟩

end

8 Functions, Function Spaces, Lambda-Abstraction

theory *func* **imports** *equalities Sum* **begin**

8.1 The Pi Operator: Dependent Function Space

lemma *subset-Sigma-imp-relation*: $r \subseteq Sigma(A,B) \Longrightarrow relation(r)$
⟨*proof*⟩

lemma *relation-converse-converse* [*simp*]:
  $relation(r) \Longrightarrow converse(converse(r)) = r$
⟨*proof*⟩

lemma *relation-restrict* [*simp*]: $relation(restrict(r,A))$
⟨*proof*⟩

lemma *Pi-iff*:
  $f \in Pi(A,B) \longleftrightarrow function(f) \wedge f<=Sigma(A,B) \wedge A<=domain(f)$
⟨*proof*⟩

lemma *Pi-iff-old*:
  $f \in Pi(A,B) \longleftrightarrow f<=Sigma(A,B) \wedge (\forall x\in A.\ \exists!y.\ \langle x,y\rangle: f)$
⟨*proof*⟩

lemma *fun-is-function*: $f \in Pi(A,B) \Longrightarrow function(f)$

⟨*proof*⟩

lemma *function-imp-Pi*:
$[\![function(f);\ relation(f)]\!] \Longrightarrow f \in domain(f) \ {-}{>}\ range(f)$
⟨*proof*⟩

lemma *functionI*:
$[\![\bigwedge x\ y\ y'.\ [\![\langle x,y\rangle{:}r;\ {<}x,y'{>}{:}r]\!] \Longrightarrow y{=}y']\!] \Longrightarrow function(r)$
⟨*proof*⟩

lemma *fun-is-rel*: $f \in Pi(A,B) \Longrightarrow f \subseteq Sigma(A,B)$
⟨*proof*⟩

lemma *Pi-cong*:
$[\![A{=}A';\ \ \bigwedge x.\ x \in A' \Longrightarrow B(x){=}B'(x)]\!] \Longrightarrow Pi(A,B) = Pi(A',B')$
⟨*proof*⟩

lemma *fun-weaken-type*: $[\![f \in A{-}{>}B;\ \ B{<}{=}D]\!] \Longrightarrow f \in A{-}{>}D$
⟨*proof*⟩

8.2 Function Application

lemma *apply-equality2*: $[\![\langle a,b\rangle{:}\ f;\ \ \langle a,c\rangle{:}\ f;\ \ f \in Pi(A,B)]\!] \Longrightarrow b{=}c$
⟨*proof*⟩

lemma *function-apply-equality*: $[\![\langle a,b\rangle{:}\ f;\ \ function(f)]\!] \Longrightarrow f`a = b$
⟨*proof*⟩

lemma *apply-equality*: $[\![\langle a,b\rangle{:}\ f;\ \ f \in Pi(A,B)]\!] \Longrightarrow f`a = b$
⟨*proof*⟩

lemma *apply-0*: $a \notin domain(f) \Longrightarrow f`a = 0$
⟨*proof*⟩

lemma *Pi-memberD*: $[\![f \in Pi(A,B);\ \ c \in f]\!] \Longrightarrow \exists x{\in}A.\ \ c = {<}x,f`x{>}$
⟨*proof*⟩

lemma *function-apply-Pair*: $[\![function(f);\ \ a \in domain(f)]\!] \Longrightarrow {<}a,f`a{>}{:}\ f$
⟨*proof*⟩

lemma *apply-Pair*: $[\![f \in Pi(A,B);\ \ a \in A]\!] \Longrightarrow {<}a,f`a{>}{:}\ f$
⟨*proof*⟩

lemma *apply-type* [*TC*]: $[\![f \in Pi(A,B);\ \ a \in A]\!] \Longrightarrow f`a \in B(a)$
$\langle proof \rangle$

lemma *apply-funtype*: $[\![f \in A{-}{>}B;\ \ a \in A]\!] \Longrightarrow f`a \in B$
$\langle proof \rangle$

lemma *apply-iff*: $f \in Pi(A,B) \Longrightarrow \langle a,b\rangle: f \longleftrightarrow a \in A \wedge f`a = b$
$\langle proof \rangle$

lemma *Pi-type*: $[\![f \in Pi(A,C);\ \ \bigwedge x.\ x \in A \Longrightarrow f`x \in B(x)]\!] \Longrightarrow f \in Pi(A,B)$
$\langle proof \rangle$

lemma *Pi-Collect-iff*:
$(f \in Pi(A,\ \lambda x.\ \{y \in B(x).\ P(x,y)\}))$
$\longleftrightarrow\ f \in Pi(A,B) \wedge (\forall x{\in}A.\ P(x,\ f`x))$
$\langle proof \rangle$

lemma *Pi-weaken-type*:
$[\![f \in Pi(A,B);\ \ \bigwedge x.\ x \in A \Longrightarrow B(x){<=}C(x)]\!] \Longrightarrow f \in Pi(A,C)$
$\langle proof \rangle$

lemma *domain-type*: $[\![\langle a,b\rangle \in f;\ \ f \in Pi(A,B)]\!] \Longrightarrow a \in A$
$\langle proof \rangle$

lemma *range-type*: $[\![\langle a,b\rangle \in f;\ \ f \in Pi(A,B)]\!] \Longrightarrow b \in B(a)$
$\langle proof \rangle$

lemma *Pair-mem-PiD*: $[\![\langle a,b\rangle: f;\ \ f \in Pi(A,B)]\!] \Longrightarrow a \in A \wedge b \in B(a) \wedge f`a = b$
$\langle proof \rangle$

8.3 Lambda Abstraction

lemma *lamI*: $a \in A \Longrightarrow <a,b(a)> \in (\lambda x{\in}A.\ b(x))$
$\langle proof \rangle$

lemma *lamE*:
$[\![p: (\lambda x{\in}A.\ b(x));\ \ \bigwedge x.[\![x \in A;\ p{=}<x,b(x)>]\!] \Longrightarrow P$
$]\!] \Longrightarrow\ P$
$\langle proof \rangle$

lemma *lamD*: $[\![\langle a,c\rangle: (\lambda x{\in}A.\ b(x))]\!] \Longrightarrow c = b(a)$
$\langle proof \rangle$

lemma *lam-type* [*TC*]:
$[\![\bigwedge x.\ x \in A \Longrightarrow b(x): B(x)]\!] \Longrightarrow (\lambda x{\in}A.\ b(x)) \in Pi(A,B)$
⟨*proof*⟩

lemma *lam-funtype*: $(\lambda x{\in}A.\ b(x)) \in A \text{ −> } \{b(x).\ x \in A\}$
⟨*proof*⟩

lemma *function-lam*: *function* $(\lambda x{\in}A.\ b(x))$
⟨*proof*⟩

lemma *relation-lam*: *relation* $(\lambda x{\in}A.\ b(x))$
⟨*proof*⟩

lemma *beta-if* [*simp*]: $(\lambda x{\in}A.\ b(x))$ ‘ $a = (\mathit{if}\ a \in A\ \mathit{then}\ b(a)\ \mathit{else}\ 0)$
⟨*proof*⟩

lemma *beta*: $a \in A \Longrightarrow (\lambda x{\in}A.\ b(x))$ ‘ $a = b(a)$
⟨*proof*⟩

lemma *lam-empty* [*simp*]: $(\lambda x{\in}0.\ b(x)) = 0$
⟨*proof*⟩

lemma *domain-lam* [*simp*]: $domain(Lambda(A,b)) = A$
⟨*proof*⟩

lemma *lam-cong* [*cong*]:
$[\![A{=}A';\ \ \bigwedge x.\ x \in A' \Longrightarrow b(x){=}b'(x)]\!] \Longrightarrow Lambda(A,b) = Lambda(A',b')$
⟨*proof*⟩

lemma *lam-theI*:
$(\bigwedge x.\ x \in A \Longrightarrow \exists!y.\ Q(x,y)) \Longrightarrow \exists f.\ \forall x{\in}A.\ Q(x, f`x)$
⟨*proof*⟩

lemma *lam-eqE*: $[\![(\lambda x{\in}A.\ f(x)) = (\lambda x{\in}A.\ g(x));\ \ a \in A]\!] \Longrightarrow f(a){=}g(a)$
⟨*proof*⟩

lemma *Pi-empty1* [*simp*]: $Pi(0,A) = \{0\}$
⟨*proof*⟩

lemma *singleton-fun* [*simp*]: $\{\langle a,b\rangle\} \in \{a\} \text{ −> } \{b\}$
⟨*proof*⟩

lemma *Pi-empty2* [*simp*]: $(A\text{−>}0) = (\mathit{if}\ A{=}0\ \mathit{then}\ \{0\}\ \mathit{else}\ 0)$
⟨*proof*⟩

lemma *fun-space-empty-iff* [*iff*]: $(A\text{-}{>}X)=0 \longleftrightarrow X=0 \wedge (A \neq 0)$
⟨*proof*⟩

8.4 Extensionality

lemma *fun-subset*:
$[\![f \in Pi(A,B);\ \ g \in Pi(C,D);\ \ A<=C;$
$\bigwedge x.\ x \in A \Longrightarrow f`x = g`x]\!] \Longrightarrow f<=g$
⟨*proof*⟩

lemma *fun-extension*:
$[\![f \in Pi(A,B);\ \ g \in Pi(A,D);$
$\bigwedge x.\ x \in A \Longrightarrow f`x = g`x]\!] \Longrightarrow f=g$
⟨*proof*⟩

lemma *eta* [*simp*]: $f \in Pi(A,B) \Longrightarrow (\lambda x{\in}A.\ f`x) = f$
⟨*proof*⟩

lemma *fun-extension-iff*:
$[\![f \in Pi(A,B);\ g \in Pi(A,C)]\!] \Longrightarrow (\forall a{\in}A.\ f`a = g`a) \longleftrightarrow f=g$
⟨*proof*⟩

lemma *fun-subset-eq*: $[\![f \in Pi(A,B);\ g \in Pi(A,C)]\!] \Longrightarrow f \subseteq g \longleftrightarrow (f = g)$
⟨*proof*⟩

lemma *Pi-lamE*:
assumes *major*: $f \in Pi(A,B)$
and *minor*: $\bigwedge b.\ [\![\forall x{\in}A.\ b(x){:}B(x);\ \ f = (\lambda x{\in}A.\ b(x))]\!] \Longrightarrow P$
shows P
⟨*proof*⟩

8.5 Images of Functions

lemma *image-lam*: $C \subseteq A \Longrightarrow (\lambda x{\in}A.\ b(x))\ ``\ C = \{b(x).\ x \in C\}$
⟨*proof*⟩

lemma *Repfun-function-if*:
$function(f)$
$\Longrightarrow \{f`x.\ x \in C\} = (if\ C \subseteq domain(f)\ then\ f``C\ else\ cons(0,f``C))$
⟨*proof*⟩

lemma *image-function*:
$[\![function(f);\ \ C \subseteq domain(f)]\!] \Longrightarrow f``C = \{f`x.\ x \in C\}$
⟨*proof*⟩

lemma *image-fun*: $[\![f \in Pi(A,B);\ \ C \subseteq A]\!] \Longrightarrow f``C = \{f`x.\ x \in C\}$

⟨*proof*⟩

lemma *image-eq-UN*:
assumes *f*: $f \in Pi(A,B)$ $C \subseteq A$ **shows** $f``C = (\bigcup x{\in}C.\ \{f \text{ ` } x\})$
⟨*proof*⟩

lemma *Pi-image-cons*:
$[\![f \in Pi(A,B);\ \ x \in A]\!] \Longrightarrow f \text{ `` } cons(x,y) = cons(f`x,\ f``y)$
⟨*proof*⟩

8.6 Properties of *restrict*(*f*, *A*)

lemma *restrict-subset*: $restrict(f,A) \subseteq f$
⟨*proof*⟩

lemma *function-restrictI*:
$function(f) \Longrightarrow function(restrict(f,A))$
⟨*proof*⟩

lemma *restrict-type2*: $[\![f \in Pi(C,B);\ \ A<=C]\!] \Longrightarrow restrict(f,A) \in Pi(A,B)$
⟨*proof*⟩

lemma *restrict*: $restrict(f,A) \text{ ` } a = (if\ a \in A\ then\ f`a\ else\ 0)$
⟨*proof*⟩

lemma *restrict-empty* [*simp*]: $restrict(f,0) = 0$
⟨*proof*⟩

lemma *restrict-iff*: $z \in restrict(r,A) \longleftrightarrow z \in r \wedge (\exists x{\in}A.\ \exists y.\ z = \langle x,\ y\rangle)$
⟨*proof*⟩

lemma *restrict-restrict* [*simp*]:
$restrict(restrict(r,A),B) = restrict(r,\ A \cap B)$
⟨*proof*⟩

lemma *domain-restrict* [*simp*]: $domain(restrict(f,C)) = domain(f) \cap C$
⟨*proof*⟩

lemma *restrict-idem*: $f \subseteq Sigma(A,B) \Longrightarrow restrict(f,A) = f$
⟨*proof*⟩

lemma *domain-restrict-idem*:
$[\![domain(r) \subseteq A;\ relation(r)]\!] \Longrightarrow restrict(r,A) = r$
⟨*proof*⟩

lemma *domain-restrict-lam* [*simp*]: $domain(restrict(Lambda(A,f),C)) = A \cap C$
⟨*proof*⟩

lemma *restrict-if* [*simp*]: *restrict*(*f*,*A*) ‘ *a* = (*if a* ∈ *A then f*‘*a else 0*)
⟨*proof*⟩

lemma *restrict-lam-eq*:
  A<=*C* ⟹ *restrict*(λ*x*∈*C*. *b*(*x*), *A*) = (λ*x*∈*A*. *b*(*x*))
⟨*proof*⟩

lemma *fun-cons-restrict-eq*:
  f ∈ *cons*(*a*, *b*) −> *B* ⟹ *f* = *cons*(<*a*, *f* ‘ *a*>, *restrict*(*f*, *b*))
⟨*proof*⟩

8.7 Unions of Functions

lemma *function-Union*:
  ⟦∀ *x*∈*S*. *function*(*x*);
    ∀ *x*∈*S*. ∀ *y*∈*S*. *x*<=*y* | *y*<=*x*⟧
  ⟹ *function*(⋃(*S*))
⟨*proof*⟩

lemma *fun-Union*:
  ⟦∀ *f*∈*S*. ∃ *C D*. *f* ∈ *C*−>*D*;
      ∀ *f*∈*S*. ∀ *y*∈*S*. *f*<=*y* | *y*<=*f*⟧ ⟹
    ⋃(*S*) ∈ *domain*(⋃(*S*)) −> *range*(⋃(*S*))
⟨*proof*⟩

lemma *gen-relation-Union*:
  (⋀*f*. *f*∈*F* ⟹ *relation*(*f*)) ⟹ *relation*(⋃(*F*))
⟨*proof*⟩

lemmas *Un-rls* = *Un-subset-iff SUM-Un-distrib1 prod-Un-distrib2*
    subset-trans [*OF* - *Un-upper1*]
    subset-trans [*OF* - *Un-upper2*]

lemma *fun-disjoint-Un*:
  ⟦*f* ∈ *A*−>*B*; *g* ∈ *C*−>*D*; *A* ∩ *C* = *0*⟧
  ⟹ (*f* ∪ *g*) ∈ (*A* ∪ *C*) −> (*B* ∪ *D*)

⟨*proof*⟩

lemma *fun-disjoint-apply1*: *a* ∉ *domain*(*g*) ⟹ (*f* ∪ *g*)‘*a* = *f*‘*a*
⟨*proof*⟩

lemma *fun-disjoint-apply2*: *c* ∉ *domain*(*f*) ⟹ (*f* ∪ *g*)‘*c* = *g*‘*c*
⟨*proof*⟩

8.8 Domain and Range of a Function or Relation

lemma *domain-of-fun*: $f \in Pi(A,B) \Longrightarrow domain(f)=A$
⟨*proof*⟩

lemma *apply-rangeI*: $[\![f \in Pi(A,B);\ \ a \in A]\!] \Longrightarrow f`a \in range(f)$
⟨*proof*⟩

lemma *range-of-fun*: $f \in Pi(A,B) \Longrightarrow f \in A{-}{>}range(f)$
⟨*proof*⟩

8.9 Extensions of Functions

lemma *fun-extend*:
$[\![f \in A{-}{>}B;\ \ c \notin A]\!] \Longrightarrow cons(\langle c,b\rangle,f) \in cons(c,A)\ {-}{>}\ cons(b,B)$
⟨*proof*⟩

lemma *fun-extend3*:
$[\![f \in A{-}{>}B;\ \ c \notin A;\ \ b \in B]\!] \Longrightarrow cons(\langle c,b\rangle,f) \in cons(c,A)\ {-}{>}\ B$
⟨*proof*⟩

lemma *extend-apply*:
$c \notin domain(f) \Longrightarrow cons(\langle c,b\rangle,f)`a = (if\ a{=}c\ then\ b\ else\ f`a)$
⟨*proof*⟩

lemma *fun-extend-apply* [*simp*]:
$[\![f \in A{-}{>}B;\ \ c \notin A]\!] \Longrightarrow cons(\langle c,b\rangle,f)`a = (if\ a{=}c\ then\ b\ else\ f`a)$
⟨*proof*⟩

lemmas *singleton-apply* = *apply-equality* [*OF singletonI singleton-fun*, *simp*]

lemma *cons-fun-eq*:
$c \notin A \Longrightarrow cons(c,A)\ {-}{>}\ B = (\bigcup f \in A{-}{>}B.\ \bigcup b{\in}B.\ \{cons(\langle c,b\rangle,\ f)\})$
⟨*proof*⟩

lemma *succ-fun-eq*: $succ(n)\ {-}{>}\ B = (\bigcup f \in n{-}{>}B.\ \bigcup b{\in}B.\ \{cons(\langle n,b\rangle,\ f)\})$
⟨*proof*⟩

8.10 Function Updates

definition
update :: $[i,i,i] \Rightarrow i$ **where**
$update(f,a,b) \equiv \lambda x{\in}cons(a,\ domain(f)).\ if(x{=}a,\ b,\ f`x)$

nonterminal *updbinds* **and** *updbind*

syntax
-updbind :: [*i*, *i*] ⇒ *updbind* (‹(‹indent=2 notation=‹infix update››- :=/ -)›)
:: *updbind* ⇒ *updbinds* (‹-›)

-updbinds :: [*updbind*, *updbinds*] ⇒ *updbinds* (‹-,/ -›)
-Update :: [*i*, *updbinds*] ⇒ *i* (‹(‹*open-block notation*=‹*mixfix function update*››-/'((-)'))› [*900*,*0*] *900*)
syntax-consts
-Update ⇌ *update*
translations
-Update (*f*, *-updbinds*(*b*,*bs*)) == *-Update* (*-Update*(*f*,*b*), *bs*)
f(*x*:=*y*) == *CONST update*(*f*,*x*,*y*)

lemma *update-apply* [*simp*]: $f(x{:=}y)$ ‘ $z = (if\ z{=}x\ then\ y\ else\ f`z)$
⟨*proof*⟩

lemma *update-idem*: $[\![f`x = y;\ f \in Pi(A,B);\ x \in A]\!] \Longrightarrow f(x{:=}y) = f$
⟨*proof*⟩

declare *refl* [*THEN update-idem*, *simp*]

lemma *domain-update* [*simp*]: $domain(f(x{:=}y)) = cons(x,\ domain(f))$
⟨*proof*⟩

lemma *update-type*: $[\![f \in Pi(A,B);\ x \in A;\ y \in B(x)]\!] \Longrightarrow f(x{:=}y) \in Pi(A,\ B)$
⟨*proof*⟩

8.11 Monotonicity Theorems

8.11.1 Replacement in its Various Forms

lemma *Replace-mono*: $A{<=}B \Longrightarrow Replace(A,P) \subseteq Replace(B,P)$
⟨*proof*⟩

lemma *RepFun-mono*: $A{<=}B \Longrightarrow \{f(x).\ x \in A\} \subseteq \{f(x).\ x \in B\}$
⟨*proof*⟩

lemma *Pow-mono*: $A{<=}B \Longrightarrow Pow(A) \subseteq Pow(B)$
⟨*proof*⟩

lemma *Union-mono*: $A{<=}B \Longrightarrow \bigcup(A) \subseteq \bigcup(B)$
⟨*proof*⟩

lemma *UN-mono*:
$[\![A{<=}C;\ \bigwedge x.\ x \in A \Longrightarrow B(x){<=}D(x)]\!] \Longrightarrow (\bigcup x{\in}A.\ B(x)) \subseteq (\bigcup x{\in}C.\ D(x))$
⟨*proof*⟩

lemma *Inter-anti-mono*: $[\![A{<=}B;\ A{\neq}0]\!] \Longrightarrow \bigcap(B) \subseteq \bigcap(A)$
⟨*proof*⟩

lemma *cons-mono*: $C{<=}D \Longrightarrow cons(a,C) \subseteq cons(a,D)$

⟨*proof*⟩

lemma *Un-mono*: $[\![A<=C;\ \ B<=D]\!] \Longrightarrow A \cup B \subseteq C \cup D$
⟨*proof*⟩

lemma *Int-mono*: $[\![A<=C;\ \ B<=D]\!] \Longrightarrow A \cap B \subseteq C \cap D$
⟨*proof*⟩

lemma *Diff-mono*: $[\![A<=C;\ \ D<=B]\!] \Longrightarrow A-B \subseteq C-D$
⟨*proof*⟩

8.11.2 Standard Products, Sums and Function Spaces

lemma *Sigma-mono* [*rule-format*]:
$[\![A<=C;\ \ \bigwedge x.\ x \in A \longrightarrow B(x) \subseteq D(x)]\!] \Longrightarrow Sigma(A,B) \subseteq Sigma(C,D)$
⟨*proof*⟩

lemma *sum-mono*: $[\![A<=C;\ \ B<=D]\!] \Longrightarrow A+B \subseteq C+D$
⟨*proof*⟩

lemma *Pi-mono*: $B<=C \Longrightarrow A->B \subseteq A->C$
⟨*proof*⟩

lemma *lam-mono*: $A<=B \Longrightarrow Lambda(A,c) \subseteq Lambda(B,c)$
⟨*proof*⟩

8.11.3 Converse, Domain, Range, Field

lemma *converse-mono*: $r<=s \Longrightarrow converse(r) \subseteq converse(s)$
⟨*proof*⟩

lemma *domain-mono*: $r<=s \Longrightarrow domain(r)<=domain(s)$
⟨*proof*⟩

lemmas *domain-rel-subset* = *subset-trans* [*OF domain-mono domain-subset*]

lemma *range-mono*: $r<=s \Longrightarrow range(r)<=range(s)$
⟨*proof*⟩

lemmas *range-rel-subset* = *subset-trans* [*OF range-mono range-subset*]

lemma *field-mono*: $r<=s \Longrightarrow field(r)<=field(s)$
⟨*proof*⟩

lemma *field-rel-subset*: $r \subseteq A*A \Longrightarrow field(r) \subseteq A$
⟨*proof*⟩

8.11.4 Images

lemma *image-pair-mono*:
  $[\![\bigwedge x\ y.\ \langle x,y\rangle{:}r \Longrightarrow \langle x,y\rangle{:}s;\ \ A{<=}B]\!] \Longrightarrow r``A \subseteq s``B$
⟨*proof*⟩

lemma *vimage-pair-mono*:
  $[\![\bigwedge x\ y.\ \langle x,y\rangle{:}r \Longrightarrow \langle x,y\rangle{:}s;\ \ A{<=}B]\!] \Longrightarrow r-``A \subseteq s-``B$
⟨*proof*⟩

lemma *image-mono*: $[\![r{<=}s;\ \ A{<=}B]\!] \Longrightarrow r``A \subseteq s``B$
⟨*proof*⟩

lemma *vimage-mono*: $[\![r{<=}s;\ \ A{<=}B]\!] \Longrightarrow r-``A \subseteq s-``B$
⟨*proof*⟩

lemma *Collect-mono*:
  $[\![A{<=}B;\ \ \bigwedge x.\ x \in A \Longrightarrow P(x) \longrightarrow Q(x)]\!] \Longrightarrow Collect(A,P) \subseteq Collect(B,Q)$
⟨*proof*⟩

lemmas *basic-monos* = *subset-refl imp-refl disj-mono conj-mono ex-mono*
  Collect-mono Part-mono in-mono

lemma *bex-image-simp*:
  $[\![f \in Pi(X,\ Y);\ A \subseteq X]\!] \Longrightarrow (\exists x{\in}f``A.\ P(x)) \longleftrightarrow (\exists x{\in}A.\ P(f`x))$
  ⟨*proof*⟩

lemma *ball-image-simp*:
  $[\![f \in Pi(X,\ Y);\ A \subseteq X]\!] \Longrightarrow (\forall x{\in}f``A.\ P(x)) \longleftrightarrow (\forall x{\in}A.\ P(f`x))$
  ⟨*proof*⟩

end

9 Quine-Inspired Ordered Pairs and Disjoint Sums

theory *QPair* **imports** *Sum func* **begin**

For non-well-founded data structures in ZF. Does not precisely follow Quine's construction. Thanks to Thomas Forster for suggesting this approach!

W. V. Quine, On Ordered Pairs and Relations, in Selected Logic Papers, 1966.

definition
  QPair :: [*i*, *i*] ⇒ *i* (‹(‹*indent=1 notation=*‹*mixfix Quine pair*››<-;/ ->)›)
  where <*a*;*b*> ≡ *a*+*b*

definition

qfst :: *i* ⇒ *i* **where**
qfst(*p*) ≡ *THE a*. ∃ *b*. *p*=<*a*;*b*>

definition
qsnd :: *i* ⇒ *i* **where**
qsnd(*p*) ≡ *THE b*. ∃ *a*. *p*=<*a*;*b*>

definition
qsplit :: [[*i*, *i*] ⇒ ′*a*, *i*] ⇒ ′*a*::{} **where**
qsplit(*c*,*p*) ≡ *c*(*qfst*(*p*), *qsnd*(*p*))

definition
qconverse :: *i* ⇒ *i* **where**
qconverse(*r*) ≡ {*z*. *w* ∈ *r*, ∃ *x y*. *w*=<*x*;*y*> ∧ *z*=<*y*;*x*>}

definition
QSigma :: [*i*, *i* ⇒ *i*] ⇒ *i* **where**
QSigma(*A*,*B*) ≡ ⋃*x*∈*A*. ⋃*y*∈*B*(*x*). {<*x*;*y*>}

syntax
-*QSUM* :: [*idt*, *i*, *i*] ⇒ *i* (‹(‹*indent=3 notation*=‹*binder QSUM*∈››*QSUM* - ∈ -./ -)› *10*)
syntax-consts
-*QSUM* ⇌ *QSigma*
translations
QSUM x ∈ *A*. *B* => *CONST QSigma*(*A*, λ*x*. *B*)

abbreviation
qprod (**infixr** ‹<∗>› *80*) **where**
A <∗> *B* ≡ *QSigma*(*A*, λ-. *B*)

definition
qsum :: [*i*,*i*]⇒*i* (**infixr** ‹<+>› *65*) **where**
A <+> *B* ≡ ({*0*} <∗> *A*) ∪ ({*1*} <∗> *B*)

definition
QInl :: *i*⇒*i* **where**
QInl(*a*) ≡ <*0*;*a*>

definition
QInr :: *i*⇒*i* **where**
QInr(*b*) ≡ <*1*;*b*>

definition
qcase :: [*i*⇒*i*, *i*⇒*i*, *i*]⇒*i* **where**
qcase(*c*,*d*) ≡ *qsplit*(λ*y z*. *cond*(*y*, *d*(*z*), *c*(*z*)))

9.1 Quine ordered pairing

lemma *QPair-empty* [*simp*]: *<0;0> = 0*
⟨*proof*⟩

lemma *QPair-iff* [*simp*]: *<a;b> = <c;d>* ⟷ *a=c* ∧ *b=d*
⟨*proof*⟩

lemmas *QPair-inject* = *QPair-iff* [*THEN iffD1*, *THEN conjE*, *elim*!]

lemma *QPair-inject1*: *<a;b> = <c;d>* ⟹ *a=c*
⟨*proof*⟩

lemma *QPair-inject2*: *<a;b> = <c;d>* ⟹ *b=d*
⟨*proof*⟩

9.1.1 QSigma: Disjoint union of a family of sets Generalizes Cartesian product

lemma *QSigmaI* [*intro*!]: ⟦*a* ∈ *A*; *b* ∈ *B*(*a*)⟧ ⟹ *<a;b>* ∈ *QSigma*(*A*,*B*)
⟨*proof*⟩

lemma *QSigmaE* [*elim*!]:
⟦*c* ∈ *QSigma*(*A*,*B*);
⋀*x y*.⟦*x* ∈ *A*; *y* ∈ *B*(*x*); *c=<x;y>*⟧ ⟹ *P*
⟧ ⟹ *P*
⟨*proof*⟩

lemma *QSigmaE2* [*elim*!]:
⟦*<a;b>*: *QSigma*(*A*,*B*); ⟦*a* ∈ *A*; *b* ∈ *B*(*a*)⟧ ⟹ *P*⟧ ⟹ *P*
⟨*proof*⟩

lemma *QSigmaD1*: *<a;b>* ∈ *QSigma*(*A*,*B*) ⟹ *a* ∈ *A*
⟨*proof*⟩

lemma *QSigmaD2*: *<a;b>* ∈ *QSigma*(*A*,*B*) ⟹ *b* ∈ *B*(*a*)
⟨*proof*⟩

lemma *QSigma-cong*:
⟦*A=A′*; ⋀*x*. *x* ∈ *A′* ⟹ *B*(*x*)=*B′*(*x*)⟧ ⟹
QSigma(*A*,*B*) = *QSigma*(*A′*,*B′*)
⟨*proof*⟩

lemma *QSigma-empty1* [*simp*]: *QSigma*(*0*,*B*) = *0*
⟨*proof*⟩

lemma *QSigma-empty2* [*simp*]: *A <∗> 0 = 0*

⟨*proof*⟩

9.1.2 Projections: qfst, qsnd

lemma *qfst-conv* [*simp*]: *qfst*(<*a*;*b*>) = *a*
⟨*proof*⟩

lemma *qsnd-conv* [*simp*]: *qsnd*(<*a*;*b*>) = *b*
⟨*proof*⟩

lemma *qfst-type* [*TC*]: *p* ∈ *QSigma*(*A*,*B*) ⟹ *qfst*(*p*) ∈ *A*
⟨*proof*⟩

lemma *qsnd-type* [*TC*]: *p* ∈ *QSigma*(*A*,*B*) ⟹ *qsnd*(*p*) ∈ *B*(*qfst*(*p*))
⟨*proof*⟩

lemma *QPair-qfst-qsnd-eq*: *a* ∈ *QSigma*(*A*,*B*) ⟹ <*qfst*(*a*); *qsnd*(*a*)> = *a*
⟨*proof*⟩

9.1.3 Eliminator: qsplit

lemma *qsplit* [*simp*]: *qsplit*(λ*x y*. *c*(*x*,*y*), <*a*;*b*>) ≡ *c*(*a*,*b*)
⟨*proof*⟩

lemma *qsplit-type* [*elim*!]:
⟦*p* ∈ *QSigma*(*A*,*B*);
⋀*x y*.⟦*x* ∈ *A*; *y* ∈ *B*(*x*)⟧ ⟹ *c*(*x*,*y*):*C*(<*x*;*y*>)
⟧ ⟹ *qsplit*(λ*x y*. *c*(*x*,*y*), *p*) ∈ *C*(*p*)
⟨*proof*⟩

lemma *expand-qsplit*:
u ∈ *A*<∗>*B* ⟹ *R*(*qsplit*(*c*,*u*)) ⟷ (∀ *x*∈*A*. ∀ *y*∈*B*. *u* = <*x*;*y*> ⟶ *R*(*c*(*x*,*y*)))
⟨*proof*⟩

9.1.4 qsplit for predicates: result type o

lemma *qsplitI*: *R*(*a*,*b*) ⟹ *qsplit*(*R*, <*a*;*b*>)
⟨*proof*⟩

lemma *qsplitE*:
⟦*qsplit*(*R*,*z*); *z* ∈ *QSigma*(*A*,*B*);
⋀*x y*. ⟦*z* = <*x*;*y*>; *R*(*x*,*y*)⟧ ⟹ *P*
⟧ ⟹ *P*
⟨*proof*⟩

lemma *qsplitD*: *qsplit*(*R*,<*a*;*b*>) ⟹ *R*(*a*,*b*)
⟨*proof*⟩

9.1.5 qconverse

lemma *qconverseI* [*intro*!]: <*a*;*b*>:*r* ⟹ <*b*;*a*>:*qconverse*(*r*)
⟨*proof*⟩

lemma *qconverseD* [*elim*!]: <*a*;*b*> ∈ *qconverse*(*r*) ⟹ <*b*;*a*> ∈ *r*
⟨*proof*⟩

lemma *qconverseE* [*elim*!]:
  ⟦*yx* ∈ *qconverse*(*r*);
    ⋀*x y*. ⟦*yx*=<*y*;*x*>; <*x*;*y*>:*r*⟧ ⟹ *P*
⟧ ⟹ *P*
⟨*proof*⟩

lemma *qconverse-qconverse*: *r*<=*QSigma*(*A*,*B*) ⟹ *qconverse*(*qconverse*(*r*)) = *r*
⟨*proof*⟩

lemma *qconverse-type*: *r* ⊆ *A* <∗> *B* ⟹ *qconverse*(*r*) ⊆ *B* <∗> *A*
⟨*proof*⟩

lemma *qconverse-prod*: *qconverse*(*A* <∗> *B*) = *B* <∗> *A*
⟨*proof*⟩

lemma *qconverse-empty*: *qconverse*(*0*) = *0*
⟨*proof*⟩

9.2 The Quine-inspired notion of disjoint sum

lemmas *qsum-defs* = *qsum-def QInl-def QInr-def qcase-def*

lemma *QInlI* [*intro*!]: *a* ∈ *A* ⟹ *QInl*(*a*) ∈ *A* <+> *B*
⟨*proof*⟩

lemma *QInrI* [*intro*!]: *b* ∈ *B* ⟹ *QInr*(*b*) ∈ *A* <+> *B*
⟨*proof*⟩

lemma *qsumE* [*elim*!]:
  ⟦*u* ∈ *A* <+> *B*;
    ⋀*x*. ⟦*x* ∈ *A*; *u*=*QInl*(*x*)⟧ ⟹ *P*;
    ⋀*y*. ⟦*y* ∈ *B*; *u*=*QInr*(*y*)⟧ ⟹ *P*
⟧ ⟹ *P*
⟨*proof*⟩

lemma *QInl-iff* [*iff*]: $QInl(a){=}QInl(b) \longleftrightarrow a{=}b$
⟨*proof*⟩

lemma *QInr-iff* [*iff*]: $QInr(a){=}QInr(b) \longleftrightarrow a{=}b$
⟨*proof*⟩

lemma *QInl-QInr-iff* [*simp*]: $QInl(a){=}QInr(b) \longleftrightarrow False$
⟨*proof*⟩

lemma *QInr-QInl-iff* [*simp*]: $QInr(b){=}QInl(a) \longleftrightarrow False$
⟨*proof*⟩

lemma *qsum-empty* [*simp*]: $0{<}{+}{>}0 = 0$
⟨*proof*⟩

lemmas *QInl-inject* = *QInl-iff* [*THEN iffD1*]
lemmas *QInr-inject* = *QInr-iff* [*THEN iffD1*]
lemmas *QInl-neq-QInr* = *QInl-QInr-iff* [*THEN iffD1*, *THEN FalseE*, *elim*!]
lemmas *QInr-neq-QInl* = *QInr-QInl-iff* [*THEN iffD1*, *THEN FalseE*, *elim*!]

lemma *QInlD*: $QInl(a)\colon A{<}{+}{>}B \Longrightarrow a \in A$
⟨*proof*⟩

lemma *QInrD*: $QInr(b)\colon A{<}{+}{>}B \Longrightarrow b \in B$
⟨*proof*⟩

lemma *qsum-iff*:
$u \in A <+> B \longleftrightarrow (\exists x.\ x \in A \wedge u{=}QInl(x)) \mid (\exists y.\ y \in B \wedge u{=}QInr(y))$
⟨*proof*⟩

lemma *qsum-subset-iff*: $A <+> B \subseteq C <+> D \longleftrightarrow A{<=}C \wedge B{<=}D$
⟨*proof*⟩

lemma *qsum-equal-iff*: $A <+> B = C <+> D \longleftrightarrow A{=}C \wedge B{=}D$
⟨*proof*⟩

9.2.1 Eliminator – qcase

lemma *qcase-QInl* [*simp*]: $qcase(c, d, QInl(a)) = c(a)$
⟨*proof*⟩

lemma *qcase-QInr* [*simp*]: $qcase(c, d, QInr(b)) = d(b)$
⟨*proof*⟩

lemma *qcase-type*:
$[\![u \in A <+> B;$
$\bigwedge x.\ x \in A \Longrightarrow c(x): C(QInl(x));$
$\bigwedge y.\ y \in B \Longrightarrow d(y): C(QInr(y))$
$]\!] \Longrightarrow qcase(c,d,u) \in C(u)$
⟨*proof*⟩

lemma *Part-QInl*: $Part(A <+> B,QInl) = \{QInl(x).\ x \in A\}$
⟨*proof*⟩

lemma *Part-QInr*: $Part(A <+> B,QInr) = \{QInr(y).\ y \in B\}$
⟨*proof*⟩

lemma *Part-QInr2*: $Part(A <+> B, \lambda x.\ QInr(h(x))) = \{QInr(y).\ y \in Part(B,h)\}$
⟨*proof*⟩

lemma *Part-qsum-equality*: $C \subseteq A <+> B \Longrightarrow Part(C,QInl) \cup Part(C,QInr) = C$
⟨*proof*⟩

9.2.2 Monotonicity

lemma *QPair-mono*: $[\![a<=c;\ \ b<=d]\!] \Longrightarrow <a;b> \subseteq <c;d>$
⟨*proof*⟩

lemma *QSigma-mono* [*rule-format*]:
$[\![A<=C;\ \ \forall x\in A.\ B(x) \subseteq D(x)]\!] \Longrightarrow QSigma(A,B) \subseteq QSigma(C,D)$
⟨*proof*⟩

lemma *QInl-mono*: $a<=b \Longrightarrow QInl(a) \subseteq QInl(b)$
⟨*proof*⟩

lemma *QInr-mono*: $a<=b \Longrightarrow QInr(a) \subseteq QInr(b)$
⟨*proof*⟩

lemma *qsum-mono*: $[\![A<=C;\ \ B<=D]\!] \Longrightarrow A <+> B \subseteq C <+> D$
⟨*proof*⟩

end

10 Injections, Surjections, Bijections, Composition

theory *Perm* **imports** *func* **begin**

definition

comp :: $[i,i]\Rightarrow i$ (**infixr** ‹*O*› *60*) **where**

$r\ O\ s \equiv \{xz \in domain(s)*range(r)\ .$
$\quad \exists\, x\ y\ z.\ xz{=}\langle x{,}z\rangle \wedge \langle x{,}y\rangle{:}s \wedge \langle y{,}z\rangle{:}r\}$

definition

$id\ \ ::\ i{\Rightarrow}i$ **where**
$id(A) \equiv (\lambda x{\in}A.\ x)$

definition

$inj\ \ ::\ [i{,}i]{\Rightarrow}i$ **where**
$inj(A{,}B) \equiv \{\ f \in A{-}{>}B.\ \forall\, w{\in}A.\ \forall\, x{\in}A.\ f`w{=}f`x \longrightarrow w{=}x\}$

definition

$surj\ \ ::\ [i{,}i]{\Rightarrow}i$ **where**
$surj(A{,}B) \equiv \{\ f \in A{-}{>}B\ .\ \forall\, y{\in}B.\ \exists\, x{\in}A.\ f`x{=}y\}$

definition

$bij\ \ ::\ [i{,}i]{\Rightarrow}i$ **where**
$bij(A{,}B) \equiv inj(A{,}B) \cap surj(A{,}B)$

10.1 Surjective Function Space

lemma *surj-is-fun*: $f \in surj(A{,}B) \Longrightarrow f \in A{-}{>}B$
⟨*proof*⟩

lemma *fun-is-surj*: $f \in Pi(A{,}B) \Longrightarrow f \in surj(A{,}range(f))$
⟨*proof*⟩

lemma *surj-range*: $f \in surj(A{,}B) \Longrightarrow range(f){=}B$
⟨*proof*⟩

A function with a right inverse is a surjection

lemma *f-imp-surjective*:
$[\![f \in A{-}{>}B;\ \bigwedge y.\ y \in B \Longrightarrow d(y){:}\ A;\ \bigwedge y.\ y \in B \Longrightarrow f`d(y) = y]\!]$
$\Longrightarrow f \in surj(A{,}B)$
⟨*proof*⟩

lemma *lam-surjective*:
$[\![\bigwedge x.\ x \in A \Longrightarrow c(x){:}\ B;$
$\bigwedge y.\ y \in B \Longrightarrow d(y){:}\ A;$
$\bigwedge y.\ y \in B \Longrightarrow c(d(y)) = y$
$]\!] \Longrightarrow (\lambda x{\in}A.\ c(x)) \in surj(A{,}B)$
⟨*proof*⟩

Cantor's theorem revisited

lemma *cantor-surj*: $f \notin surj(A{,}Pow(A))$

⟨*proof*⟩

10.2 Injective Function Space

lemma *inj-is-fun*: $f \in inj(A,B) \Longrightarrow f \in A{-}{>}B$
⟨*proof*⟩

Good for dealing with sets of pairs, but a bit ugly in use [used in AC]

lemma *inj-equality*:
$[\![\langle a,b\rangle{:}f;\ \ \langle c,b\rangle{:}f;\ \ f \in inj(A,B)]\!] \Longrightarrow a{=}c$
⟨*proof*⟩

lemma *inj-apply-equality*: $[\![f \in inj(A,B);\ \ f`a{=}f`b;\ \ a \in A;\ \ b \in A]\!] \Longrightarrow a{=}b$
⟨*proof*⟩

A function with a left inverse is an injection

lemma *f-imp-injective*: $[\![f \in A{-}{>}B;\ \ \forall x{\in}A.\ d(f`x){=}x]\!] \Longrightarrow f \in inj(A,B)$
⟨*proof*⟩

lemma *lam-injective*:
$[\![\bigwedge x.\ x \in A \Longrightarrow c(x){:}\ B;$
$\bigwedge x.\ x \in A \Longrightarrow d(c(x)) = x]\!]$
$\Longrightarrow (\lambda x{\in}A.\ c(x)) \in inj(A,B)$
⟨*proof*⟩

10.3 Bijections

lemma *bij-is-inj*: $f \in bij(A,B) \Longrightarrow f \in inj(A,B)$
⟨*proof*⟩

lemma *bij-is-surj*: $f \in bij(A,B) \Longrightarrow f \in surj(A,B)$
⟨*proof*⟩

lemma *bij-is-fun*: $f \in bij(A,B) \Longrightarrow f \in A{-}{>}B$
⟨*proof*⟩

lemma *lam-bijective*:
$[\![\bigwedge x.\ x \in A \Longrightarrow c(x){:}\ B;$
$\bigwedge y.\ y \in B \Longrightarrow d(y){:}\ A;$
$\bigwedge x.\ x \in A \Longrightarrow d(c(x)) = x;$
$\bigwedge y.\ y \in B \Longrightarrow c(d(y)) = y$
$]\!] \Longrightarrow (\lambda x{\in}A.\ c(x)) \in bij(A,B)$
⟨*proof*⟩

lemma *RepFun-bijective*: $(\forall y{\in}x.\ \exists!y'.\ f(y') = f(y))$
$\Longrightarrow (\lambda z{\in}\{f(y).\ y \in x\}.\ THE\ y.\ f(y) = z) \in bij(\{f(y).\ y \in x\},\ x)$
⟨*proof*⟩

10.4 Identity Function

lemma *idI* [*intro*!]: $a \in A \Longrightarrow \langle a,a \rangle \in id(A)$
⟨*proof*⟩

lemma *idE* [*elim*!]: $[\![p \in id(A);\ \bigwedge x.[\![x \in A;\ p=\langle x,x \rangle]\!] \Longrightarrow P]\!] \Longrightarrow P$
⟨*proof*⟩

lemma *id-type*: $id(A) \in A->A$
⟨*proof*⟩

lemma *id-conv* [*simp*]: $x \in A \Longrightarrow id(A)`x = x$
⟨*proof*⟩

lemma *id-mono*: $A<=B \Longrightarrow id(A) \subseteq id(B)$
⟨*proof*⟩

lemma *id-subset-inj*: $A<=B \Longrightarrow id(A)\colon inj(A,B)$
⟨*proof*⟩

lemmas *id-inj* = *subset-refl* [*THEN id-subset-inj*]

lemma *id-surj*: $id(A)\colon surj(A,A)$
⟨*proof*⟩

lemma *id-bij*: $id(A)\colon bij(A,A)$
⟨*proof*⟩

lemma *subset-iff-id*: $A \subseteq B \longleftrightarrow id(A) \in A->B$
⟨*proof*⟩

id as the identity relation

lemma *id-iff* [*simp*]: $\langle x,y \rangle \in id(A) \longleftrightarrow x=y \wedge y \in A$
⟨*proof*⟩

10.5 Converse of a Function

lemma *inj-converse-fun*: $f \in inj(A,B) \Longrightarrow converse(f) \in range(f)->A$
⟨*proof*⟩

Equations for converse(f)

The premises are equivalent to saying that f is injective...

lemma *left-inverse-lemma*:
$[\![f \in A->B;\ converse(f)\colon C->A;\ a \in A]\!] \Longrightarrow converse(f)`(f`a) = a$
⟨*proof*⟩

lemma *left-inverse* [*simp*]: $[\![f \in inj(A,B);\ a \in A]\!] \Longrightarrow converse(f)`(f`a) = a$
⟨*proof*⟩

lemma *left-inverse-eq*:
$[\![f \in inj(A,B);\ f\ `\ x = y;\ x \in A]\!] \Longrightarrow converse(f)\ `\ y = x$
⟨*proof*⟩

lemmas *left-inverse-bij* = *bij-is-inj* [*THEN left-inverse*]

lemma *right-inverse-lemma*:
$[\![f \in A{-}{>}B;\ \ converse(f)\colon C{-}{>}A;\ \ b \in C]\!] \Longrightarrow f`(converse(f)`b) = b$
⟨*proof*⟩

lemma *right-inverse* [*simp*]:
$[\![f \in inj(A,B);\ \ b \in range(f)]\!] \Longrightarrow f`(converse(f)`b) = b$
⟨*proof*⟩

lemma *right-inverse-bij*: $[\![f \in bij(A,B);\ \ b \in B]\!] \Longrightarrow f`(converse(f)`b) = b$
⟨*proof*⟩

10.6 Converses of Injections, Surjections, Bijections

lemma *inj-converse-inj*: $f \in inj(A,B) \Longrightarrow converse(f)\colon inj(range(f),\ A)$
⟨*proof*⟩

lemma *inj-converse-surj*: $f \in inj(A,B) \Longrightarrow converse(f)\colon surj(range(f),\ A)$
⟨*proof*⟩

Adding this as an intro! rule seems to cause looping

lemma *bij-converse-bij* [*TC*]: $f \in bij(A,B) \Longrightarrow converse(f)\colon bij(B,A)$
⟨*proof*⟩

10.7 Composition of Two Relations

The inductive definition package could derive these theorems for $r\ O\ s$

lemma *compI* [*intro*]: $[\![\langle a,b\rangle{:}s;\ \langle b,c\rangle{:}r]\!] \Longrightarrow \langle a,c\rangle \in r\ O\ s$
⟨*proof*⟩

lemma *compE* [*elim*!]:
$[\![xz \in r\ O\ s;$
$\bigwedge x\ y\ z.\ [\![xz{=}\langle x,z\rangle;\ \ \langle x,y\rangle{:}s;\ \ \langle y,z\rangle{:}r]\!] \Longrightarrow P]\!]$
$\Longrightarrow P$
⟨*proof*⟩

lemma *compEpair*:
$[\![\langle a,c\rangle \in r\ O\ s;$
$\bigwedge y.\ [\![\langle a,y\rangle{:}s;\ \ \langle y,c\rangle{:}r]\!] \Longrightarrow P]\!]$
$\Longrightarrow P$
⟨*proof*⟩

lemma *converse-comp*: $converse(R\ O\ S) = converse(S)\ O\ converse(R)$

$\langle proof \rangle$

10.8 Domain and Range – see Suppes, Section 3.1

Boyer et al., Set Theory in First-Order Logic, JAR 2 (1986), 287-327

lemma *range-comp*: $range(r\ O\ s) \subseteq range(r)$
$\langle proof \rangle$

lemma *range-comp-eq*: $domain(r) \subseteq range(s) \Longrightarrow range(r\ O\ s) = range(r)$
$\langle proof \rangle$

lemma *domain-comp*: $domain(r\ O\ s) \subseteq domain(s)$
$\langle proof \rangle$

lemma *domain-comp-eq*: $range(s) \subseteq domain(r) \Longrightarrow domain(r\ O\ s) = domain(s)$
$\langle proof \rangle$

lemma *image-comp*: $(r\ O\ s)``A = r``(s``A)$
$\langle proof \rangle$

lemma *inj-inj-range*: $f \in inj(A,B) \Longrightarrow f \in inj(A,range(f))$
$\langle proof \rangle$

lemma *inj-bij-range*: $f \in inj(A,B) \Longrightarrow f \in bij(A,range(f))$
$\langle proof \rangle$

10.9 Other Results

lemma *comp-mono*: $[\![r'<=r;\ s'<=s]\!] \Longrightarrow (r'\ O\ s') \subseteq (r\ O\ s)$
$\langle proof \rangle$

composition preserves relations

lemma *comp-rel*: $[\![s<=A*B;\ \ r<=B*C]\!] \Longrightarrow (r\ O\ s) \subseteq A*C$
$\langle proof \rangle$

associative law for composition

lemma *comp-assoc*: $(r\ O\ s)\ O\ t = r\ O\ (s\ O\ t)$
$\langle proof \rangle$

lemma *left-comp-id*: $r<=A*B \Longrightarrow id(B)\ O\ r = r$
$\langle proof \rangle$

lemma *right-comp-id*: $r<=A*B \Longrightarrow r\ O\ id(A) = r$
$\langle proof \rangle$

10.10 Composition Preserves Functions, Injections, and Surjections

lemma *comp-function*: $[\![function(g);\ \ function(f)]\!] \Longrightarrow function(f\ O\ g)$
⟨*proof*⟩

Don't think the premises can be weakened much

lemma *comp-fun*: $[\![g \in A{-}{>}B;\ \ f \in B{-}{>}C]\!] \Longrightarrow (f\ O\ g) \in A{-}{>}C$
⟨*proof*⟩

lemma *comp-fun-apply* [*simp*]:
$[\![g \in A{-}{>}B;\ \ a \in A]\!] \Longrightarrow (f\ O\ g)`a = f`(g`a)$
⟨*proof*⟩

Simplifies compositions of lambda-abstractions

lemma *comp-lam*:
$[\![\bigwedge x.\ x \in A \Longrightarrow b(x){:}\ B]\!]$
$\Longrightarrow (\lambda y{\in}B.\ c(y))\ O\ (\lambda x{\in}A.\ b(x)) = (\lambda x{\in}A.\ c(b(x)))$
⟨*proof*⟩

lemma *comp-inj*:
$[\![g \in inj(A,B);\ \ f \in inj(B,C)]\!] \Longrightarrow (f\ O\ g) \in inj(A,C)$
⟨*proof*⟩

lemma *comp-surj*:
$[\![g \in surj(A,B);\ \ f \in surj(B,C)]\!] \Longrightarrow (f\ O\ g) \in surj(A,C)$
⟨*proof*⟩

lemma *comp-bij*:
$[\![g \in bij(A,B);\ \ f \in bij(B,C)]\!] \Longrightarrow (f\ O\ g) \in bij(A,C)$
⟨*proof*⟩

10.11 Dual Properties of *inj* and *surj*

Useful for proofs from D Pastre. Automatic theorem proving in set theory. Artificial Intelligence, 10:1–27, 1978.

lemma *comp-mem-injD1*:
$[\![(f\ O\ g){:}\ inj(A,C);\ \ g \in A{-}{>}B;\ \ f \in B{-}{>}C]\!] \Longrightarrow g \in inj(A,B)$
⟨*proof*⟩

lemma *comp-mem-injD2*:
$[\![(f\ O\ g){:}\ inj(A,C);\ \ g \in surj(A,B);\ \ f \in B{-}{>}C]\!] \Longrightarrow f \in inj(B,C)$
⟨*proof*⟩

lemma *comp-mem-surjD1*:
$[\![(f\ O\ g){:}\ surj(A,C);\ \ g \in A{-}{>}B;\ \ f \in B{-}{>}C]\!] \Longrightarrow f \in surj(B,C)$
⟨*proof*⟩

lemma *comp-mem-surjD2*:
$[\![(f\ O\ g): surj(A,C);\ \ g \in A{-}{>}B;\ \ f \in inj(B,C)]\!] \Longrightarrow g \in surj(A,B)$
⟨*proof*⟩

10.11.1 Inverses of Composition

left inverse of composition; one inclusion is $f \in A \to B \Longrightarrow id(A) \subseteq converse(f)\ O\ f$

lemma *left-comp-inverse*: $f \in inj(A,B) \Longrightarrow converse(f)\ O\ f = id(A)$
⟨*proof*⟩

right inverse of composition; one inclusion is $f \in A \to B \Longrightarrow f\ O\ converse(f) \subseteq id(B)$

lemma *right-comp-inverse*:
$f \in surj(A,B) \Longrightarrow f\ O\ converse(f) = id(B)$
⟨*proof*⟩

10.11.2 Proving that a Function is a Bijection

lemma *comp-eq-id-iff*:
$[\![f \in A{-}{>}B;\ \ g \in B{-}{>}A]\!] \Longrightarrow f\ O\ g = id(B) \longleftrightarrow (\forall y{\in}B.\ f`(g`y){=}y)$
⟨*proof*⟩

lemma *fg-imp-bijective*:
$[\![f \in A{-}{>}B;\ \ g \in B{-}{>}A;\ \ f\ O\ g = id(B);\ \ g\ O\ f = id(A)]\!] \Longrightarrow f \in bij(A,B)$
⟨*proof*⟩

lemma *nilpotent-imp-bijective*: $[\![f \in A{-}{>}A;\ \ f\ O\ f = id(A)]\!] \Longrightarrow f \in bij(A,A)$
⟨*proof*⟩

lemma *invertible-imp-bijective*:
$[\![converse(f): B{-}{>}A;\ \ f \in A{-}{>}B]\!] \Longrightarrow f \in bij(A,B)$
⟨*proof*⟩

10.11.3 Unions of Functions

See similar theorems in func.thy

Theorem by KG, proof by LCP

lemma *inj-disjoint-Un*:
$[\![f \in inj(A,B);\ \ g \in inj(C,D);\ \ B \cap D = 0]\!]$
$\Longrightarrow (\lambda a{\in}A \cup C.\ if\ a \in A\ then\ f`a\ else\ g`a) \in inj(A \cup C,\ B \cup D)$
⟨*proof*⟩

lemma *surj-disjoint-Un*:
$[\![f \in surj(A,B);\ \ g \in surj(C,D);\ \ A \cap C = 0]\!]$

$\Longrightarrow (f \cup g) \in surj(A \cup C,\ B \cup D)$
⟨*proof*⟩

A simple, high-level proof; the version for injections follows from it, using $f \in inj(A,\ B) \longleftrightarrow f \in bij(A,\ range(f))$

lemma *bij-disjoint-Un*:
$[\![f \in bij(A,B);\ \ g \in bij(C,D);\ \ A \cap C = 0;\ \ B \cap D = 0]\!]$
$\Longrightarrow (f \cup g) \in bij(A \cup C,\ B \cup D)$
⟨*proof*⟩

10.11.4 Restrictions as Surjections and Bijections

lemma *surj-image*:
$f \in Pi(A,B) \Longrightarrow f \in surj(A,\ f``A)$
⟨*proof*⟩

lemma *surj-image-eq*: $f \in surj(A,\ B) \Longrightarrow f``A = B$
⟨*proof*⟩

lemma *restrict-image* [*simp*]: $restrict(f,A)\ ``\ B = f\ ``\ (A \cap B)$
⟨*proof*⟩

lemma *restrict-inj*:
$[\![f \in inj(A,B);\ \ C<=A]\!] \Longrightarrow restrict(f,C): inj(C,B)$
⟨*proof*⟩

lemma *restrict-surj*: $[\![f \in Pi(A,B);\ \ C<=A]\!] \Longrightarrow restrict(f,C): surj(C,\ f``C)$
⟨*proof*⟩

lemma *restrict-bij*:
$[\![f \in inj(A,B);\ \ C<=A]\!] \Longrightarrow restrict(f,C): bij(C,\ f``C)$
⟨*proof*⟩

10.11.5 Lemmas for Ramsey's Theorem

lemma *inj-weaken-type*: $[\![f \in inj(A,B);\ \ B<=D]\!] \Longrightarrow f \in inj(A,D)$
⟨*proof*⟩

lemma *inj-succ-restrict*:
$[\![f \in inj(succ(m),\ A)]\!] \Longrightarrow restrict(f,m) \in inj(m,\ A-\{f`m\})$
⟨*proof*⟩

lemma *inj-extend*:
$[\![f \in inj(A,B);\ \ a \notin A;\ \ b \notin B]\!]$
$\Longrightarrow cons(\langle a,b\rangle,f) \in inj(cons(a,A),\ cons(b,B))$
⟨*proof*⟩

end

11 Relations: Their General Properties and Transitive Closure

theory *Trancl* **imports** *Fixedpt Perm* **begin**

definition
refl :: $[i,i]\Rightarrow o$ **where**
$refl(A,r) \equiv (\forall x\in A.\ \langle x,x\rangle \in r)$

definition
irrefl :: $[i,i]\Rightarrow o$ **where**
$irrefl(A,r) \equiv \forall x\in A.\ \langle x,x\rangle \notin r$

definition
sym :: $i\Rightarrow o$ **where**
$sym(r) \equiv \forall x\ y.\ \langle x,y\rangle: r \longrightarrow \langle y,x\rangle: r$

definition
asym :: $i\Rightarrow o$ **where**
$asym(r) \equiv \forall x\ y.\ \langle x,y\rangle:r \longrightarrow \neg\ \langle y,x\rangle:r$

definition
antisym :: $i\Rightarrow o$ **where**
$antisym(r) \equiv \forall x\ y.\langle x,y\rangle:r \longrightarrow \langle y,x\rangle:r \longrightarrow x=y$

definition
trans :: $i\Rightarrow o$ **where**
$trans(r) \equiv \forall x\ y\ z.\ \langle x,y\rangle: r \longrightarrow \langle y,z\rangle: r \longrightarrow \langle x,z\rangle: r$

definition
trans-on :: $[i,i]\Rightarrow o$ (‹(‹open-block notation=‹mixfix trans-on››trans[-]'(-'))›) **where**
$trans[A](r) \equiv \forall x\in A.\ \forall y\in A.\ \forall z\in A.$
$\langle x,y\rangle: r \longrightarrow \langle y,z\rangle: r \longrightarrow \langle x,z\rangle: r$

definition
rtrancl :: $i\Rightarrow i$ (‹(‹notation=‹postfix ^*››-^*)› [100] 100) **where**
$r\hat{}* \equiv lfp(field(r)*field(r),\ \lambda s.\ id(field(r)) \cup (r\ O\ s))$

definition
trancl :: $i\Rightarrow i$ (‹(‹notation=‹postfix ^+››-^+)› [100] 100) **where**
$r\hat{}+ \equiv r\ O\ r\hat{}*$

definition
equiv :: $[i,i]\Rightarrow o$ **where**
$equiv(A,r) \equiv r \subseteq A*A \wedge refl(A,r) \wedge sym(r) \wedge trans(r)$

11.1 General properties of relations

11.1.1 irreflexivity

lemma *irreflI*:
  $[\![\bigwedge x.\ x \in A \Longrightarrow \langle x,x\rangle \notin r]\!] \Longrightarrow irrefl(A,r)$
⟨*proof*⟩

lemma *irreflE*: $[\![irrefl(A,r);\ \ x \in A]\!] \Longrightarrow\ \langle x,x\rangle \notin r$
⟨*proof*⟩

11.1.2 symmetry

lemma *symI*:
  $[\![\bigwedge x\ y.\langle x,y\rangle: r \Longrightarrow \langle y,x\rangle: r]\!] \Longrightarrow sym(r)$
⟨*proof*⟩

lemma *symE*: $[\![sym(r);\ \langle x,y\rangle: r]\!]\ \Longrightarrow\ \langle y,x\rangle: r$
⟨*proof*⟩

11.1.3 antisymmetry

lemma *antisymI*:
  $[\![\bigwedge x\ y.[\![\langle x,y\rangle: r;\ \ \langle y,x\rangle: r]\!] \Longrightarrow x=y]\!] \Longrightarrow antisym(r)$
⟨*proof*⟩

lemma *antisymE*: $[\![antisym(r);\ \langle x,y\rangle: r;\ \ \langle y,x\rangle: r]\!]\ \Longrightarrow\ x=y$
⟨*proof*⟩

11.1.4 transitivity

lemma *transD*: $[\![trans(r);\ \ \langle a,b\rangle:r;\ \ \langle b,c\rangle:r]\!] \Longrightarrow \langle a,c\rangle:r$
⟨*proof*⟩

lemma *trans-onD*:
  $[\![trans[A](r);\ \ \langle a,b\rangle:r;\ \ \langle b,c\rangle:r;\ \ a \in A;\ \ b \in A;\ \ c \in A]\!] \Longrightarrow \langle a,c\rangle:r$
⟨*proof*⟩

lemma *trans-imp-trans-on*: $trans(r) \Longrightarrow trans[A](r)$
⟨*proof*⟩

lemma *trans-on-imp-trans*: $[\![trans[A](r);\ r \subseteq A*A]\!] \Longrightarrow trans(r)$
⟨*proof*⟩

11.2 Transitive closure of a relation

lemma *rtrancl-bnd-mono*:
  $bnd\text{-}mono(field(r)*field(r),\ \lambda s.\ id(field(r)) \cup (r\ O\ s))$
⟨*proof*⟩

lemma *rtrancl-mono*: $r<=s \Longrightarrow r\hat{}* \subseteq s\hat{}*$

⟨*proof*⟩

lemmas *rtrancl-unfold* =
rtrancl-bnd-mono [*THEN* *rtrancl-def* [*THEN* *def-lfp-unfold*]]

lemmas *rtrancl-type* = *rtrancl-def* [*THEN* *def-lfp-subset*]

lemma *relation-rtrancl*: *relation*(r^∗)
⟨*proof*⟩

lemma *rtrancl-refl*: ⟦$a \in$ *field*(r)⟧ $\Longrightarrow$ ⟨a,a⟩ $\in$ r^∗
⟨*proof*⟩

lemma *rtrancl-into-rtrancl*: ⟦⟨a,b⟩ $\in$ r^∗; ⟨b,c⟩ $\in$ r⟧ $\Longrightarrow$ ⟨a,c⟩ $\in$ r^∗
⟨*proof*⟩

lemma *r-into-rtrancl*: ⟨a,b⟩ $\in$ r $\Longrightarrow$ ⟨a,b⟩ $\in$ r^∗
⟨*proof*⟩

lemma *r-subset-rtrancl*: *relation*(r) $\Longrightarrow$ $r \subseteq r$^∗
⟨*proof*⟩

lemma *rtrancl-field*: *field*(r^∗) = *field*(r)
⟨*proof*⟩

lemma *rtrancl-full-induct* [*case-names initial step*, *consumes 1*]:
⟦⟨a,b⟩ $\in$ r^∗;
⋀x. $x \in$ *field*(r) $\Longrightarrow$ P(⟨x,x⟩);
⋀x y z.⟦P(⟨x,y⟩); ⟨x,y⟩: r^∗; ⟨y,z⟩: r⟧ $\Longrightarrow$ P(⟨x,z⟩)⟧
$\Longrightarrow$ P(⟨a,b⟩)
⟨*proof*⟩

lemma *rtrancl-induct* [*case-names initial step*, *induct set*: *rtrancl*]:
⟦⟨a,b⟩ $\in$ r^∗;
$P(a)$;
⋀y z.⟦⟨a,y⟩ $\in$ r^∗; ⟨y,z⟩ $\in$ r; $P(y)$⟧ $\Longrightarrow$ $P(z)$
⟧ $\Longrightarrow$ $P(b)$

⟨*proof*⟩

lemma *trans-rtrancl*: $trans(r\hat{\ }*)$
⟨*proof*⟩

lemmas *rtrancl-trans* = *trans-rtrancl* [*THEN transD*]

lemma *rtranclE*:
$[\![\langle a,b\rangle \in r\hat{\ }*;\ (a=b) \Longrightarrow P;$
$\bigwedge y.[\![\langle a,y\rangle \in r\hat{\ }*;\ \ \langle y,b\rangle \in r]\!] \Longrightarrow P]\!]$
$\Longrightarrow P$
⟨*proof*⟩

lemma *trans-trancl*: $trans(r\hat{\ }+)$
⟨*proof*⟩

lemmas *trans-on-trancl* = *trans-trancl* [*THEN trans-imp-trans-on*]

lemmas *trancl-trans* = *trans-trancl* [*THEN transD*]

lemma *trancl-into-rtrancl*: $\langle a,b\rangle \in r\hat{\ }+ \Longrightarrow \langle a,b\rangle \in r\hat{\ }*$
⟨*proof*⟩

lemma *r-into-trancl*: $\langle a,b\rangle \in r \Longrightarrow \langle a,b\rangle \in r\hat{\ }+$
⟨*proof*⟩

lemma *r-subset-trancl*: $relation(r) \Longrightarrow r \subseteq r\hat{\ }+$
⟨*proof*⟩

lemma *rtrancl-into-trancl1*: $[\![\langle a,b\rangle \in r\hat{\ }*;\ \langle b,c\rangle \in r]\!] \Longrightarrow \langle a,c\rangle \in r\hat{\ }+$
⟨*proof*⟩

lemma *rtrancl-into-trancl2*:
$[\![\langle a,b\rangle \in r;\ \langle b,c\rangle \in r\hat{\ }*]\!] \Longrightarrow \langle a,c\rangle \in r\hat{\ }+$
⟨*proof*⟩

lemma *trancl-induct* [*case-names initial step*, *induct set*: *trancl*]:
$[\![\langle a,b\rangle \in r\hat{\ }+;$
$\bigwedge y.\ [\![\langle a,y\rangle \in r]\!] \Longrightarrow P(y);$
$\bigwedge y\ z.[\![\langle a,y\rangle \in r\hat{\ }+;\ \langle y,z\rangle \in r;\ P(y)]\!] \Longrightarrow P(z)$
$]\!] \Longrightarrow P(b)$
⟨*proof*⟩

lemma *tranclE*:
$[\![\langle a,b\rangle \in r\hat{\ }+;$
$\langle a,b\rangle \in r \Longrightarrow P;$
$\bigwedge y.[\![\langle a,y\rangle \in r\hat{\ }+;\ \langle y,b\rangle \in r]\!] \Longrightarrow P$
$]\!] \Longrightarrow P$
⟨*proof*⟩

lemma *trancl-type*: $r\hat{\ }+ \subseteq field(r)*field(r)$
⟨*proof*⟩

lemma *relation-trancl*: $relation(r\hat{\ }+)$
⟨*proof*⟩

lemma *trancl-subset-times*: $r \subseteq A * A \Longrightarrow r\hat{\ }+ \subseteq A * A$
⟨*proof*⟩

lemma *trancl-mono*: $r{<=}s \Longrightarrow r\hat{\ }+ \subseteq s\hat{\ }+$
⟨*proof*⟩

lemma *trancl-eq-r*: $[\![relation(r);\ trans(r)]\!] \Longrightarrow r\hat{\ }+ = r$
⟨*proof*⟩

lemma *rtrancl-idemp* [*simp*]: $(r\hat{\ }*)\hat{\ }* = r\hat{\ }*$
⟨*proof*⟩

lemma *rtrancl-subset*: $[\![R \subseteq S;\ S \subseteq R\hat{\ }*]\!] \Longrightarrow S\hat{\ }* = R\hat{\ }*$
⟨*proof*⟩

lemma *rtrancl-Un-rtrancl*:
$[\![relation(r);\ relation(s)]\!] \Longrightarrow (r\hat{\ }* \cup s\hat{\ }*)\hat{\ }* = (r \cup s)\hat{\ }*$
⟨*proof*⟩

lemma *rtrancl-converseD*: $\langle x,y\rangle$:*converse*(r)^* $\Longrightarrow$ $\langle x,y\rangle$:*converse*$(r*)$
⟨*proof*⟩

lemma *rtrancl-converseI*: $\langle x,y\rangle$:*converse*$(r*)$ $\Longrightarrow$ $\langle x,y\rangle$:*converse*(r)^*
⟨*proof*⟩

lemma *rtrancl-converse*: *converse*(r)^* = *converse*$(r*)$
⟨*proof*⟩

lemma *trancl-converseD*: $\langle a,\ b\rangle$:*converse*(r)^+ $\Longrightarrow$ $\langle a,\ b\rangle$:*converse*$(r$^+$)$
⟨*proof*⟩

lemma *trancl-converseI*: $\langle x,y\rangle$:*converse*$(r$^+$)$ $\Longrightarrow$ $\langle x,y\rangle$:*converse*(r)^+
⟨*proof*⟩

lemma *trancl-converse*: *converse*(r)^+ = *converse*$(r$^+$)$
⟨*proof*⟩

lemma *converse-trancl-induct* [*case-names initial step*, *consumes 1*]:
$[\![\langle a,\ b\rangle{:}r$^+; $\bigwedge y.\ \langle y,\ b\rangle\ {:}r \Longrightarrow P(y)$;
$\quad\bigwedge y\ z.\ [\![\langle y,\ z\rangle \in r;\ \langle z,\ b\rangle \in r$^+; $P(z)]\!] \Longrightarrow P(y)]\!]$
$\quad\Longrightarrow P(a)$
⟨*proof*⟩

end

12 Well-Founded Recursion

theory *WF* **imports** *Trancl* **begin**

definition
wf :: $i \Rightarrow o$ **where**

wf$(r) \equiv \forall Z.\ Z{=}0 \mid (\exists x{\in}Z.\ \forall y.\ \langle y,x\rangle{:}r \longrightarrow \neg\ y \in Z)$

definition
wf-on :: $[i,i]\Rightarrow o$ (‹(‹open-block notation=‹mixfix wf-on››wf[-]'(-'))›) **where**

wf-on$(A,r) \equiv$ *wf*$(r \cap A{*}A)$

definition
is-recfun :: $[i,\ i,\ [i,i]\Rightarrow i,\ i] \Rightarrow o$ **where**
is-recfun$(r,a,H,f) \equiv (f = (\lambda x{\in}r{-}``\{a\}.\ H(x,$ *restrict*$(f,\ r{-}``\{x\}))))$

definition
the-recfun :: $[i,\ i,\ [i,i]\Rightarrow i] \Rightarrow i$ **where**
the-recfun$(r,a,H) \equiv (THE\ f.$ *is-recfun*$(r,a,H,f))$

definition
wftrec :: $[i,\ i,\ [i,i]\Rightarrow i]\ \Rightarrow i$ **where**
$wftrec(r,a,H) \equiv H(a,\ the\text{-}recfun(r,a,H))$

definition
wfrec :: $[i,\ i,\ [i,i]\Rightarrow i]\ \Rightarrow i$ **where**

$wfrec(r,a,H) \equiv wftrec(r\hat{}+,\ a,\ \lambda x\, f.\ H(x,\ restrict(f,r-``\{x\})))$

definition
wfrec-on :: $[i, i, i, [i,i]\Rightarrow i]\Rightarrow i$ (‹(‹*open-block notation*=‹*mixfix wfrec-on*››*wfrec*[-]′(-,-,-′))›)
where $wfrec[A](r,a,H) \equiv wfrec(r \cap A{*}A,\ a,\ H)$

12.1 Well-Founded Relations

12.1.1 Equivalences between *wf* and *wf-on*

lemma *wf-imp-wf-on*: $wf(r) \Longrightarrow wf[A](r)$
⟨*proof*⟩

lemma *wf-on-imp-wf*: $[\![wf[A](r);\ r \subseteq A{*}A]\!] \Longrightarrow wf(r)$
⟨*proof*⟩

lemma *wf-on-field-imp-wf*: $wf[field(r)](r) \Longrightarrow wf(r)$
⟨*proof*⟩

lemma *wf-iff-wf-on-field*: $wf(r) \longleftrightarrow wf[field(r)](r)$
⟨*proof*⟩

lemma *wf-on-subset-A*: $[\![wf[A](r);\ \ B{<=}A]\!] \Longrightarrow wf[B](r)$
⟨*proof*⟩

lemma *wf-on-subset-r*: $[\![wf[A](r);\ s{<=}r]\!] \Longrightarrow wf[A](s)$
⟨*proof*⟩

lemma *wf-subset*: $[\![wf(s);\ r{<=}s]\!] \Longrightarrow wf(r)$
⟨*proof*⟩

12.1.2 Introduction Rules for *wf-on*

If every non-empty subset of A has an r-minimal element then we have $wf[A](r)$.

lemma *wf-onI*:
assumes *prem*: $\bigwedge Z\ u.\ [\![Z{<=}A;\ \ u \in Z;\ \ \forall x{\in}Z.\ \exists y{\in}Z.\ \langle y,x\rangle{:}r]\!] \Longrightarrow False$
shows $wf[A](r)$
⟨*proof*⟩

If r allows well-founded induction over A then we have $wf[A](r)$. Premise is equivalent to $\bigwedge B.\ \forall x{\in}A.\ (\forall y.\ \langle y,\ x\rangle \in r \longrightarrow y \in B) \longrightarrow x \in B \Longrightarrow A \subseteq B$

lemma *wf-onI2*:
assumes *prem*: $\bigwedge y\ B.\ [\![\forall x{\in}A.\ (\forall y{\in}A.\ \langle y,x\rangle{:}r \longrightarrow y \in B) \longrightarrow x \in B;\quad y \in A]\!] \Longrightarrow y \in B$
shows $wf[A](r)$
⟨*proof*⟩

12.1.3 Well-founded Induction

Consider the least z in $domain(r)$ such that $P(z)$ does not hold...

lemma *wf-induct-raw*:
$[\![wf(r);$
$\bigwedge x.[\![\forall y.\ \langle y,x\rangle{:}\ r \longrightarrow P(y)]\!] \Longrightarrow P(x)]\!]$
$\Longrightarrow P(a)$
⟨*proof*⟩

lemmas *wf-induct* = *wf-induct-raw* [*rule-format*, *consumes 1*, *case-names step*, *induct set*: *wf*]

The form of this rule is designed to match *wfI*

lemma *wf-induct2*:
$[\![wf(r);\ \ a \in A;\ \ field(r){<=}A;$
$\bigwedge x.[\![x \in A;\ \ \forall y.\ \langle y,x\rangle{:}\ r \longrightarrow P(y)]\!] \Longrightarrow P(x)]\!]$
$\Longrightarrow\ P(a)$
⟨*proof*⟩

lemma *field-Int-square*: $field(r \cap A{*}A) \subseteq A$
⟨*proof*⟩

lemma *wf-on-induct-raw* [*consumes 2*, *induct set*: *wf-on*]:
$[\![wf[A](r);\ \ a \in A;$
$\bigwedge x.[\![x \in A;\ \ \forall y{\in}A.\ \langle y,x\rangle{:}\ r \longrightarrow P(y)]\!] \Longrightarrow P(x)$
$]\!]\ \Longrightarrow\ P(a)$
⟨*proof*⟩

lemma *wf-on-induct* [*consumes 2*, *case-names step*, *induct set*: *wf-on*]:
$wf[A](r) \Longrightarrow a \in A \Longrightarrow (\bigwedge x.\ x \in A \Longrightarrow (\bigwedge y.\ y \in A \Longrightarrow \langle y,\ x\rangle \in r \Longrightarrow P(y)) \Longrightarrow P(x)) \Longrightarrow P(a)$
⟨*proof*⟩

If r allows well-founded induction then we have $wf(r)$.

lemma *wfI*:
$[\![field(r){<=}A;$
$\bigwedge y\ B.\ [\![\forall x{\in}A.\ (\forall y{\in}A.\ \langle y,x\rangle{:}r \longrightarrow y \in B) \longrightarrow x \in B;\ \ y \in A]\!]$
$\Longrightarrow y \in B]\!]$
$\Longrightarrow\ wf(r)$
⟨*proof*⟩

12.2 Basic Properties of Well-Founded Relations

lemma *wf-not-refl*: $wf(r) \Longrightarrow \langle a,a\rangle \notin r$
⟨*proof*⟩

lemma *wf-not-sym* [*rule-format*]: $wf(r) \Longrightarrow \forall x.\ \langle a,x\rangle{:}r \longrightarrow \langle x,a\rangle \notin r$
⟨*proof*⟩

lemmas *wf-asym* = *wf-not-sym* [*THEN swap*]

lemma *wf-on-not-refl*: $[\![wf[A](r);\ a \in A]\!] \Longrightarrow \langle a,a\rangle \notin r$
⟨*proof*⟩

lemma *wf-on-not-sym*:
$[\![wf[A](r);\ \ a \in A]\!] \Longrightarrow (\bigwedge b.\ b{\in}A \Longrightarrow \langle a,b\rangle{:}r \Longrightarrow \langle b,a\rangle{\notin}r)$
⟨*proof*⟩

lemma *wf-on-asym*:
$[\![wf[A](r);\ \ \neg Z \Longrightarrow \langle a,b\rangle \in r;$
$\langle b,a\rangle \notin r \Longrightarrow Z;\ \neg Z \Longrightarrow a \in A;\ \neg Z \Longrightarrow b \in A]\!] \Longrightarrow Z$
⟨*proof*⟩

lemma *wf-on-chain3*:
$[\![wf[A](r);\ \langle a,b\rangle{:}r;\ \langle b,c\rangle{:}r;\ \langle c,a\rangle{:}r;\ a \in A;\ b \in A;\ c \in A]\!] \Longrightarrow P$
⟨*proof*⟩

transitive closure of a WF relation is WF provided A is downward closed

lemma *wf-on-trancl*:
$[\![wf[A](r);\ \ r-``A \subseteq A]\!] \Longrightarrow wf[A](r\hat{}+)$
⟨*proof*⟩

lemma *wf-trancl*: $wf(r) \Longrightarrow wf(r\hat{}+)$
⟨*proof*⟩

$r -``\ \{a\}$ is the set of everything under a in r

lemmas *underI* = *vimage-singleton-iff* [*THEN iffD2*]
lemmas *underD* = *vimage-singleton-iff* [*THEN iffD1*]

12.3 The Predicate *is-recfun*

lemma *is-recfun-type*: $is\text{-}recfun(r,a,H,f) \Longrightarrow f \in r-``\{a\} \ {-}{>}\ range(f)$
⟨*proof*⟩

lemmas *is-recfun-imp-function* = *is-recfun-type* [*THEN fun-is-function*]

lemma *apply-recfun*:
$[\![is\text{-}recfun(r,a,H,f);\ \langle x,a\rangle{:}r]\!] \Longrightarrow f`x = H(x,\ restrict(f,r-``\{x\}))$

$\langle proof \rangle$

lemma *is-recfun-equal* [*rule-format*]:
$[\![wf(r);\ \ trans(r);\ \ is\text{-}recfun(r,a,H,f);\ \ is\text{-}recfun(r,b,H,g)]\!]$
$\Longrightarrow \langle x,a\rangle{:}r \longrightarrow \langle x,b\rangle{:}r \longrightarrow f`x{=}g`x$
$\langle proof \rangle$

lemma *is-recfun-cut*:
$[\![wf(r);\ \ trans(r);$
$is\text{-}recfun(r,a,H,f);\ \ is\text{-}recfun(r,b,H,g);\ \ \langle b,a\rangle{:}r]\!]$
$\Longrightarrow restrict(f,\ r-``\{b\}) = g$
$\langle proof \rangle$

12.4 Recursion: Main Existence Lemma

lemma *is-recfun-functional*:
$[\![wf(r);\ trans(r);\ is\text{-}recfun(r,a,H,f);\ is\text{-}recfun(r,a,H,g)]\!] \Longrightarrow f{=}g$
$\langle proof \rangle$

lemma *the-recfun-eq*:
$[\![is\text{-}recfun(r,a,H,f);\ \ wf(r);\ \ trans(r)]\!] \Longrightarrow the\text{-}recfun(r,a,H) = f$
$\langle proof \rangle$

lemma *is-the-recfun*:
$[\![is\text{-}recfun(r,a,H,f);\ \ wf(r);\ \ trans(r)]\!]$
$\Longrightarrow is\text{-}recfun(r,\ a,\ H,\ the\text{-}recfun(r,a,H))$
$\langle proof \rangle$

lemma *unfold-the-recfun*:
$[\![wf(r);\ \ trans(r)]\!] \Longrightarrow is\text{-}recfun(r,\ a,\ H,\ the\text{-}recfun(r,a,H))$
$\langle proof \rangle$

12.5 Unfolding $wftrec(r,\ a,\ H)$

lemma *the-recfun-cut*:
$[\![wf(r);\ \ trans(r);\ \ \langle b,a\rangle{:}r]\!]$
$\Longrightarrow restrict(the\text{-}recfun(r,a,H),\ r-``\{b\}) = the\text{-}recfun(r,b,H)$
$\langle proof \rangle$

lemma *wftrec*:
$[\![wf(r);\ \ trans(r)]\!] \Longrightarrow$
$wftrec(r,a,H) = H(a,\ \lambda x{\in}r-``\{a\}.\ wftrec(r,x,H))$
$\langle proof \rangle$

12.5.1 Removal of the Premise $trans(r)$

lemma *wfrec*:
$wf(r) \Longrightarrow wfrec(r,a,H) = H(a,\ \lambda x{\in}r-``\{a\}.\ wfrec(r,x,H))$

⟨*proof*⟩

lemma *def-wfrec*:
$[\![\bigwedge x.\ h(x) \equiv wfrec(r,x,H);\ \ wf(r)]\!] \Longrightarrow$
$h(a) = H(a,\ \lambda x{\in}r-``\{a\}.\ h(x))$
⟨*proof*⟩

lemma *wfrec-type*:
$[\![wf(r);\ \ a \in A;\ \ field(r) <= A;$
$\bigwedge x\ u.\ [\![x \in A;\ \ u \in Pi(r-``\{x\},\ B)]\!] \Longrightarrow H(x,u) \in B(x)$
$]\!] \Longrightarrow wfrec(r,a,H) \in B(a)$
⟨*proof*⟩

lemma *wfrec-on*:
$[\![wf[A](r);\ \ a \in A]\!] \Longrightarrow$
$wfrec[A](r,a,H) = H(a,\ \lambda x{\in}(r-``\{a\}) \cap A.\ wfrec[A](r,x,H))$
⟨*proof*⟩

Minimal-element characterization of well-foundedness

lemma *wf-eq-minimal*: $wf(r) \longleftrightarrow (\forall\, Q\ x.\ x \in Q \longrightarrow (\exists\, z{\in}Q.\ \forall\, y.\ \langle y,z\rangle{:}r \longrightarrow y{\notin}Q))$
⟨*proof*⟩

end

13 Transitive Sets and Ordinals

theory *Ordinal* **imports** *WF Bool equalities* **begin**

definition
Memrel :: $i{\Rightarrow}i$ **where**
$Memrel(A) \equiv \{z{\in}A{*}A\ .\ \exists\, x\ y.\ z{=}\langle x,y\rangle \wedge x{\in}y\ \}$

definition
Transset :: $i{\Rightarrow}o$ **where**
$Transset(i) \equiv \forall\, x{\in}i.\ x{<=}i$

definition
Ord :: $i{\Rightarrow}o$ **where**
$Ord(i) \equiv Transset(i) \wedge (\forall\, x{\in}i.\ Transset(x))$

definition
lt :: $[i,i] \Rightarrow o$ (**infixl** ‹<› *50*) **where**
$i{<}j \equiv i{\in}j \wedge Ord(j)$

definition
Limit :: $i{\Rightarrow}o$ **where**

$Limit(i) \quad \equiv Ord(i) \wedge 0<i \wedge (\forall y.\ y<i \longrightarrow succ(y)<i)$

abbreviation
le (**infixl** ‹≤› *50*) **where**
$x \leq y \equiv x < succ(y)$

13.1 Rules for Transset

13.1.1 Three Neat Characterisations of Transset

lemma *Transset-iff-Pow*: $Transset(A) <-> A<=Pow(A)$
⟨*proof*⟩

lemma *Transset-iff-Union-succ*: $Transset(A) <-> \bigcup(succ(A)) = A$
⟨*proof*⟩

lemma *Transset-iff-Union-subset*: $Transset(A) <-> \bigcup(A) \subseteq A$
⟨*proof*⟩

13.1.2 Consequences of Downwards Closure

lemma *Transset-doubleton-D*:
$[\![Transset(C);\ \{a,b\}:\ C]\!] \Longrightarrow a\in C \wedge b\in C$
⟨*proof*⟩

lemma *Transset-Pair-D*:
$[\![Transset(C);\ \langle a,b\rangle\in C]\!] \Longrightarrow a\in C \wedge b\in C$
⟨*proof*⟩

lemma *Transset-includes-domain*:
$[\![Transset(C);\ A*B \subseteq C;\ b \in B]\!] \Longrightarrow A \subseteq C$
⟨*proof*⟩

lemma *Transset-includes-range*:
$[\![Transset(C);\ A*B \subseteq C;\ a \in A]\!] \Longrightarrow B \subseteq C$
⟨*proof*⟩

13.1.3 Closure Properties

lemma *Transset-0*: $Transset(0)$
⟨*proof*⟩

lemma *Transset-Un*:
$[\![Transset(i);\ \ Transset(j)]\!] \Longrightarrow Transset(i \cup j)$
⟨*proof*⟩

lemma *Transset-Int*:
$[\![Transset(i);\ \ Transset(j)]\!] \Longrightarrow Transset(i \cap j)$
⟨*proof*⟩

lemma *Transset-succ*: $Transset(i) \Longrightarrow Transset(succ(i))$
⟨*proof*⟩

lemma *Transset-Pow*: $Transset(i) \Longrightarrow Transset(Pow(i))$
⟨*proof*⟩

lemma *Transset-Union*: $Transset(A) \Longrightarrow Transset(\bigcup(A))$
⟨*proof*⟩

lemma *Transset-Union-family*:
$[\![\bigwedge i.\ i{\in}A \Longrightarrow Transset(i)]\!] \Longrightarrow Transset(\bigcup(A))$
⟨*proof*⟩

lemma *Transset-Inter-family*:
$[\![\bigwedge i.\ i{\in}A \Longrightarrow Transset(i)]\!] \Longrightarrow Transset(\bigcap(A))$
⟨*proof*⟩

lemma *Transset-UN*:
$(\bigwedge x.\ x \in A \Longrightarrow Transset(B(x))) \Longrightarrow Transset\ (\bigcup x{\in}A.\ B(x))$
⟨*proof*⟩

lemma *Transset-INT*:
$(\bigwedge x.\ x \in A \Longrightarrow Transset(B(x))) \Longrightarrow Transset\ (\bigcap x{\in}A.\ B(x))$
⟨*proof*⟩

13.2 Lemmas for Ordinals

lemma *OrdI*:
$[\![Transset(i);\ \bigwedge x.\ x{\in}i \Longrightarrow Transset(x)]\!] \Longrightarrow Ord(i)$
⟨*proof*⟩

lemma *Ord-is-Transset*: $Ord(i) \Longrightarrow Transset(i)$
⟨*proof*⟩

lemma *Ord-contains-Transset*:
$[\![Ord(i);\ j{\in}i]\!] \Longrightarrow Transset(j)$
⟨*proof*⟩

lemma *Ord-in-Ord*: $[\![Ord(i);\ j{\in}i]\!] \Longrightarrow Ord(j)$
⟨*proof*⟩

lemma *Ord-in-Ord'*: $[\![j{\in}i;\ Ord(i)]\!] \Longrightarrow Ord(j)$
⟨*proof*⟩

lemmas *Ord-succD* = *Ord-in-Ord* [*OF - succI1*]

lemma *Ord-subset-Ord*: $[\![Ord(i); \ \ Transset(j); \ \ j<=i]\!] \Longrightarrow Ord(j)$
⟨*proof*⟩

lemma *OrdmemD*: $[\![j \in i; \ \ Ord(i)]\!] \Longrightarrow j<=i$
⟨*proof*⟩

lemma *Ord-trans*: $[\![i \in j; \ \ j \in k; \ \ Ord(k)]\!] \Longrightarrow i \in k$
⟨*proof*⟩

lemma *Ord-succ-subsetI*: $[\![i \in j; \ \ Ord(j)]\!] \Longrightarrow succ(i) \subseteq j$
⟨*proof*⟩

13.3 The Construction of Ordinals: 0, succ, Union

lemma *Ord-0* [*iff*,*TC*]: $Ord(0)$
⟨*proof*⟩

lemma *Ord-succ* [*TC*]: $Ord(i) \Longrightarrow Ord(succ(i))$
⟨*proof*⟩

lemmas *Ord-1* = *Ord-0* [*THEN Ord-succ*]

lemma *Ord-succ-iff* [*iff*]: $Ord(succ(i)) <-> Ord(i)$
⟨*proof*⟩

lemma *Ord-Un* [*intro*,*simp*,*TC*]: $[\![Ord(i); Ord(j)]\!] \Longrightarrow Ord(i \cup j)$
⟨*proof*⟩

lemma *Ord-Int* [*TC*]: $[\![Ord(i); Ord(j)]\!] \Longrightarrow Ord(i \cap j)$
⟨*proof*⟩

There is no set of all ordinals, for then it would contain itself

lemma *ON-class*: $\neg (\forall i. \ i \in X <-> Ord(i))$
⟨*proof*⟩

13.4 < is 'less Than' for Ordinals

lemma *ltI*: $[\![i \in j; \ \ Ord(j)]\!] \Longrightarrow i<j$
⟨*proof*⟩

lemma *ltE*:
$[\![i<j; \ \ [\![i \in j; \ \ Ord(i); \ \ Ord(j)]\!] \Longrightarrow P]\!] \Longrightarrow P$
⟨*proof*⟩

lemma *ltD*: $i<j \Longrightarrow i \in j$
⟨*proof*⟩

lemma *not-lt0* [*simp*]: $\neg \ i<0$
⟨*proof*⟩

lemma *lt-Ord*: $j<i \Longrightarrow Ord(j)$
⟨*proof*⟩

lemma *lt-Ord2*: $j<i \Longrightarrow Ord(i)$
⟨*proof*⟩

lemmas *le-Ord2* = *lt-Ord2* [*THEN Ord-succD*]

lemmas *lt0E* = *not-lt0* [*THEN notE*, *elim*!]

lemma *lt-trans* [*trans*]: $[\![i<j; \ j<k]\!] \Longrightarrow i<k$
⟨*proof*⟩

lemma *lt-not-sym*: $i<j \Longrightarrow \neg\ (j<i)$
⟨*proof*⟩

lemmas *lt-asym* = *lt-not-sym* [*THEN swap*]

lemma *lt-irrefl* [*elim*!]: $i<i \Longrightarrow P$
⟨*proof*⟩

lemma *lt-not-refl*: $\neg\ i<i$
⟨*proof*⟩

Recall that $i \leq j$ abbreviates $i \leq j$!

lemma *le-iff*: $i \leq j$ <-> $i<j \mid (i=j \wedge Ord(j))$
⟨*proof*⟩

lemma *leI*: $i<j \Longrightarrow i \leq j$
⟨*proof*⟩

lemma *le-eqI*: $[\![i=j; \ Ord(j)]\!] \Longrightarrow i \leq j$
⟨*proof*⟩

lemmas *le-refl* = *refl* [*THEN le-eqI*]

lemma *le-refl-iff* [*iff*]: $i \leq i$ <-> $Ord(i)$
⟨*proof*⟩

lemma *leCI*: $(\neg\ (i=j \wedge Ord(j)) \Longrightarrow i<j) \Longrightarrow i \leq j$
⟨*proof*⟩

lemma *leE*:
$[\![i \leq j; \ i<j \Longrightarrow P; \ [\![i=j; \ Ord(j)]\!] \Longrightarrow P]\!] \Longrightarrow P$

$\langle proof \rangle$

lemma *le-anti-sym*: $[\![i \le j;\ \ j \le i]\!] \Longrightarrow i{=}j$
$\langle proof \rangle$

lemma *le0-iff* [*simp*]: $i \le 0 <-> i{=}0$
$\langle proof \rangle$

lemmas *le0D* = *le0-iff* [*THEN iffD1*, *dest*!]

13.5 Natural Deduction Rules for Memrel

lemma *Memrel-iff* [*simp*]: $\langle a,b\rangle \in Memrel(A) <-> a{\in}b \wedge a{\in}A \wedge b{\in}A$
$\langle proof \rangle$

lemma *MemrelI* [*intro*!]: $[\![a \in b;\ \ a \in A;\ \ b \in A]\!] \Longrightarrow \langle a,b\rangle \in Memrel(A)$
$\langle proof \rangle$

lemma *MemrelE* [*elim*!]:
$[\![\langle a,b\rangle \in Memrel(A);$
$[\![a \in A;\ \ b \in A;\ \ a{\in}b]\!] \ \Longrightarrow P]\!]$
$\Longrightarrow P$
$\langle proof \rangle$

lemma *Memrel-type*: $Memrel(A) \subseteq A{*}A$
$\langle proof \rangle$

lemma *Memrel-mono*: $A{<=}B \Longrightarrow Memrel(A) \subseteq Memrel(B)$
$\langle proof \rangle$

lemma *Memrel-0* [*simp*]: $Memrel(0) = 0$
$\langle proof \rangle$

lemma *Memrel-1* [*simp*]: $Memrel(1) = 0$
$\langle proof \rangle$

lemma *relation-Memrel*: $relation(Memrel(A))$
$\langle proof \rangle$

lemma *wf-Memrel*: $wf(Memrel(A))$
$\langle proof \rangle$

The premise $Ord(i)$ does not suffice.

lemma *trans-Memrel*:
$Ord(i) \Longrightarrow trans(Memrel(i))$
$\langle proof \rangle$

However, the following premise is strong enough.

lemma *Transset-trans-Memrel*:
$\forall j{\in}i.\ Transset(j) \Longrightarrow trans(Memrel(i))$
$\langle proof \rangle$

lemma *Transset-Memrel-iff*:
$Transset(A) \Longrightarrow \langle a,b\rangle \in Memrel(A) <-> a{\in}b \wedge b{\in}A$
$\langle proof \rangle$

13.6 Transfinite Induction

lemma *Transset-induct*:
$[\![i \in k;\ \ Transset(k);$
$\bigwedge x.[\![x \in k;\ \ \forall y{\in}x.\ P(y)]\!] \Longrightarrow P(x)]\!]$
$\Longrightarrow\ P(i)$
$\langle proof \rangle$

lemma *Ord-induct* [*consumes 2*]:
$i \in k \Longrightarrow Ord(k) \Longrightarrow (\bigwedge x.\ x \in k \Longrightarrow (\bigwedge y.\ y \in x \Longrightarrow P(y)) \Longrightarrow P(x)) \Longrightarrow P(i)$
$\langle proof \rangle$

lemma *trans-induct* [*consumes 1*, *case-names step*]:
$Ord(i) \Longrightarrow (\bigwedge x.\ Ord(x) \Longrightarrow (\bigwedge y.\ y \in x \Longrightarrow P(y)) \Longrightarrow P(x)) \Longrightarrow P(i)$
$\langle proof \rangle$

14 Fundamental properties of the epsilon ordering (< on ordinals)

14.0.1 Proving That < is a Linear Ordering on the Ordinals

lemma *Ord-linear*:
$Ord(i) \Longrightarrow Ord(j) \Longrightarrow i{\in}j \mid i{=}j \mid j{\in}i$
$\langle proof \rangle$

The trichotomy law for ordinals

lemma *Ord-linear-lt*:
assumes *o*: $Ord(i)\ Ord(j)$
obtains (*lt*) $i{<}j$ | (*eq*) $i{=}j$ | (*gt*) $j{<}i$
$\langle proof \rangle$

lemma *Ord-linear2*:
assumes *o*: $Ord(i)\ Ord(j)$
obtains (*lt*) $i{<}j$ | (*ge*) $j \le i$
$\langle proof \rangle$

lemma *Ord-linear-le*:
assumes *o*: $Ord(i)\ Ord(j)$

obtains (*le*) $i \le j$ | (*ge*) $j \le i$
⟨*proof*⟩

lemma *le-imp-not-lt*: $j \le i \Longrightarrow \neg\ i<j$
⟨*proof*⟩

lemma *not-lt-imp-le*: $[\![\neg\ i<j;\ \ Ord(i);\ \ Ord(j)]\!] \Longrightarrow j \le i$
⟨*proof*⟩

14.0.2 Some Rewrite Rules for <, ≤

lemma *Ord-mem-iff-lt*: $Ord(j) \Longrightarrow i{\in}j <-> i<j$
⟨*proof*⟩

lemma *not-lt-iff-le*: $[\![Ord(i);\ \ Ord(j)]\!] \Longrightarrow \neg\ i<j <-> j \le i$
⟨*proof*⟩

lemma *not-le-iff-lt*: $[\![Ord(i);\ \ Ord(j)]\!] \Longrightarrow \neg\ i \le j <-> j<i$
⟨*proof*⟩

lemma *Ord-0-le*: $Ord(i) \Longrightarrow 0 \le i$
⟨*proof*⟩

lemma *Ord-0-lt*: $[\![Ord(i);\ \ i{\neq}0]\!] \Longrightarrow 0<i$
⟨*proof*⟩

lemma *Ord-0-lt-iff*: $Ord(i) \Longrightarrow i{\neq}0 <-> 0<i$
⟨*proof*⟩

14.1 Results about Less-Than or Equals

lemma *zero-le-succ-iff* [*iff*]: $0 \le succ(x) <-> Ord(x)$
⟨*proof*⟩

lemma *subset-imp-le*: $[\![j<=i;\ \ Ord(i);\ \ Ord(j)]\!] \Longrightarrow j \le i$
⟨*proof*⟩

lemma *le-imp-subset*: $i \le j \Longrightarrow i<=j$
⟨*proof*⟩

lemma *le-subset-iff*: $j \le i <-> j<=i \wedge Ord(i) \wedge Ord(j)$
⟨*proof*⟩

lemma *le-succ-iff*: $i \le succ(j) <-> i \le j \mid i{=}succ(j) \wedge Ord(i)$
⟨*proof*⟩

lemma *all-lt-imp-le*: $[\![Ord(i);\ \ Ord(j);\ \ \bigwedge x.\ x<j \Longrightarrow x<i]\!] \Longrightarrow j \le i$
⟨*proof*⟩

14.1.1 Transitivity Laws

lemma *lt-trans1*: $[\![i \le j;\ \ j{<}k]\!] \Longrightarrow i{<}k$
⟨*proof*⟩

lemma *lt-trans2*: $[\![i{<}j;\ \ j \le k]\!] \Longrightarrow i{<}k$
⟨*proof*⟩

lemma *le-trans*: $[\![i \le j;\ \ j \le k]\!] \Longrightarrow i \le k$
⟨*proof*⟩

lemma *succ-leI*: $i{<}j \Longrightarrow succ(i) \le j$
⟨*proof*⟩

lemma *succ-leE*: $succ(i) \le j \Longrightarrow i{<}j$
⟨*proof*⟩

lemma *succ-le-iff* [*iff*]: $succ(i) \le j <-> i{<}j$
⟨*proof*⟩

lemma *succ-le-imp-le*: $succ(i) \le succ(j) \Longrightarrow i \le j$
⟨*proof*⟩

lemma *lt-subset-trans*: $[\![i \subseteq j;\ \ j{<}k;\ \ Ord(i)]\!] \Longrightarrow i{<}k$
⟨*proof*⟩

lemma *lt-imp-0-lt*: $j{<}i \Longrightarrow 0{<}i$
⟨*proof*⟩

lemma *succ-lt-iff*: $succ(i) < j <-> i{<}j \wedge succ(i) \neq j$
⟨*proof*⟩

lemma *Ord-succ-mem-iff*: $Ord(j) \Longrightarrow succ(i) \in succ(j) <-> i{\in}j$
⟨*proof*⟩

14.1.2 Union and Intersection

lemma *Un-upper1-le*: $[\![Ord(i);\ Ord(j)]\!] \Longrightarrow i \le i \cup j$
⟨*proof*⟩

lemma *Un-upper2-le*: $[\![Ord(i);\ Ord(j)]\!] \Longrightarrow j \le i \cup j$
⟨*proof*⟩

lemma *Un-least-lt*: $[\![i{<}k;\ \ j{<}k]\!] \Longrightarrow i \cup j < k$
⟨*proof*⟩

lemma *Un-least-lt-iff*: $[\![Ord(i);\ Ord(j)]\!] \Longrightarrow i \cup j < k\ \ <->\ \ i{<}k \wedge j{<}k$
⟨*proof*⟩

lemma *Un-least-mem-iff*:
  $[\![Ord(i);\ Ord(j);\ Ord(k)]\!] \Longrightarrow i \cup j \in k\ \ <-> \ \ i{\in}k \wedge j{\in}k$
⟨*proof*⟩

lemma *Int-greatest-lt*: $[\![i{<}k;\ \ j{<}k]\!] \Longrightarrow i \cap j < k$
⟨*proof*⟩

lemma *Ord-Un-if*:
  $[\![Ord(i);\ Ord(j)]\!] \Longrightarrow i \cup j = (\textit{if}\ j{<}i\ \textit{then}\ i\ \textit{else}\ j)$
⟨*proof*⟩

lemma *succ-Un-distrib*:
  $[\![Ord(i);\ Ord(j)]\!] \Longrightarrow succ(i \cup j) = succ(i) \cup succ(j)$
⟨*proof*⟩

lemma *lt-Un-iff*:
  $[\![Ord(i);\ Ord(j)]\!] \Longrightarrow k < i \cup j <-> k < i \mid k < j$
⟨*proof*⟩

lemma *le-Un-iff*:
  $[\![Ord(i);\ Ord(j)]\!] \Longrightarrow k \le i \cup j <-> k \le i \mid k \le j$
⟨*proof*⟩

lemma *Un-upper1-lt*: $[\![k < i;\ Ord(j)]\!] \Longrightarrow k < i \cup j$
⟨*proof*⟩

lemma *Un-upper2-lt*: $[\![k < j;\ Ord(i)]\!] \Longrightarrow k < i \cup j$
⟨*proof*⟩

lemma *Ord-Union-succ-eq*: $Ord(i) \Longrightarrow \bigcup(succ(i)) = i$
⟨*proof*⟩

14.2 Results about Limits

lemma *Ord-Union* [*intro*,*simp*,*TC*]: $[\![\bigwedge i.\ i{\in}A \Longrightarrow Ord(i)]\!] \Longrightarrow Ord(\bigcup(A))$
⟨*proof*⟩

lemma *Ord-UN* [*intro*,*simp*,*TC*]:
  $[\![\bigwedge x.\ x{\in}A \Longrightarrow Ord(B(x))]\!] \Longrightarrow Ord(\bigcup x{\in}A.\ B(x))$
⟨*proof*⟩

lemma *Ord-Inter* [*intro*,*simp*,*TC*]:
  $[\![\bigwedge i.\ i{\in}A \Longrightarrow Ord(i)]\!] \Longrightarrow Ord(\bigcap(A))$
⟨*proof*⟩

lemma *Ord-INT* [*intro*,*simp*,*TC*]:

$[\![\bigwedge x.\ x{\in}A \Longrightarrow Ord(B(x))]\!] \Longrightarrow Ord(\bigcap x{\in}A.\ B(x))$
⟨*proof*⟩

lemma *UN-least-le*:
$[\![Ord(i);\ \bigwedge x.\ x{\in}A \Longrightarrow b(x) \le i]\!] \Longrightarrow (\bigcup x{\in}A.\ b(x)) \le i$
⟨*proof*⟩

lemma *UN-succ-least-lt*:
$[\![j<i;\ \bigwedge x.\ x{\in}A \Longrightarrow b(x)<j]\!] \Longrightarrow (\bigcup x{\in}A.\ succ(b(x))) < i$
⟨*proof*⟩

lemma *UN-upper-lt*:
$[\![a{\in}A;\ i < b(a);\ Ord(\bigcup x{\in}A.\ b(x))]\!] \Longrightarrow i < (\bigcup x{\in}A.\ b(x))$
⟨*proof*⟩

lemma *UN-upper-le*:
$[\![a \in A;\ i \le b(a);\ Ord(\bigcup x{\in}A.\ b(x))]\!] \Longrightarrow i \le (\bigcup x{\in}A.\ b(x))$
⟨*proof*⟩

lemma *lt-Union-iff*: $\forall i{\in}A.\ Ord(i) \Longrightarrow (j < \bigcup(A))$ <−> $(\exists i{\in}A.\ j<i)$
⟨*proof*⟩

lemma *Union-upper-le*:
$[\![j \in J;\ i{\le}j;\ Ord(\bigcup(J))]\!] \Longrightarrow i \le \bigcup J$
⟨*proof*⟩

lemma *le-implies-UN-le-UN*:
$[\![\bigwedge x.\ x{\in}A \Longrightarrow c(x) \le d(x)]\!] \Longrightarrow (\bigcup x{\in}A.\ c(x)) \le (\bigcup x{\in}A.\ d(x))$
⟨*proof*⟩

lemma *Ord-equality*: $Ord(i) \Longrightarrow (\bigcup y{\in}i.\ succ(y)) = i$
⟨*proof*⟩

lemma *Ord-Union-subset*: $Ord(i) \Longrightarrow \bigcup(i) \subseteq i$
⟨*proof*⟩

14.3 Limit Ordinals – General Properties

lemma *Limit-Union-eq*: $Limit(i) \Longrightarrow \bigcup(i) = i$
⟨*proof*⟩

lemma *Limit-is-Ord*: $Limit(i) \Longrightarrow Ord(i)$
⟨*proof*⟩

lemma *Limit-has-0*: $Limit(i) \Longrightarrow 0 < i$
⟨*proof*⟩

lemma *Limit-nonzero*: $Limit(i) \Longrightarrow i \neq 0$
⟨*proof*⟩

lemma *Limit-has-succ*: $[\![Limit(i);\ \ j<i]\!] \Longrightarrow succ(j) < i$
⟨*proof*⟩

lemma *Limit-succ-lt-iff* [*simp*]: $Limit(i) \Longrightarrow succ(j) < i <-> (j<i)$
⟨*proof*⟩

lemma *zero-not-Limit* [*iff*]: $\neg\ Limit(0)$
⟨*proof*⟩

lemma *Limit-has-1*: $Limit(i) \Longrightarrow 1 < i$
⟨*proof*⟩

lemma *increasing-LimitI*: $[\![0<l;\ \forall\, x\in l.\ \exists\, y\in l.\ x<y]\!] \Longrightarrow Limit(l)$
⟨*proof*⟩

lemma *non-succ-LimitI*:
  assumes *i*: $0<i$ **and** *nsucc*: $\bigwedge y.\ succ(y) \neq i$
  shows $Limit(i)$
⟨*proof*⟩

lemma *succ-LimitE* [*elim!*]: $Limit(succ(i)) \Longrightarrow P$
⟨*proof*⟩

lemma *not-succ-Limit* [*simp*]: $\neg\ Limit(succ(i))$
⟨*proof*⟩

lemma *Limit-le-succD*: $[\![Limit(i);\ \ i \leq succ(j)]\!] \Longrightarrow i \leq j$
⟨*proof*⟩

14.3.1 Traditional 3-Way Case Analysis on Ordinals

lemma *Ord-cases-disj*: $Ord(i) \Longrightarrow i=0 \mid (\exists\, j.\ Ord(j) \wedge i=succ(j)) \mid Limit(i)$
⟨*proof*⟩

lemma *Ord-cases*:
 assumes *i*: $Ord(i)$
 obtains (*0*) $i=0$ | (*succ*) *j* **where** $Ord(j)$ $i=succ(j)$ | (*limit*) $Limit(i)$
⟨*proof*⟩

lemma *trans-induct3-raw*:

$$[\![Ord(i);\ P(0);\ \bigwedge x.\ [\![Ord(x);\ \ P(x)]\!] \Longrightarrow P(succ(x));\ \bigwedge x.\ [\![Limit(x);\ \ \forall\, y\in x.\ P(y)]\!] \Longrightarrow P(x)\ ]\!] \Longrightarrow P(i)$$

$\langle proof \rangle$

lemma *trans-induct3* [*case-names 0 succ limit*, *consumes 1*]:
$Ord(i) \Longrightarrow P(0) \Longrightarrow (\bigwedge x.\ Ord(x) \Longrightarrow P(x) \Longrightarrow P(succ(x))) \Longrightarrow (\bigwedge x.\ Limit(x) \Longrightarrow (\bigwedge y.\ y \in x \Longrightarrow P(y)) \Longrightarrow P(x)) \Longrightarrow P(i)$
$\langle proof \rangle$

A set of ordinals is either empty, contains its own union, or its union is a limit ordinal.

lemma *Union-le*: $[\![\bigwedge x.\ x{\in}I \Longrightarrow x{\le}j;\ Ord(j)]\!] \Longrightarrow \bigcup(I) \le j$
$\langle proof \rangle$

lemma *Ord-set-cases*:
assumes I: $\forall i{\in}I.\ Ord(i)$
shows $I{=}0 \vee \bigcup(I) \in I \vee (\bigcup(I) \notin I \wedge Limit(\bigcup(I)))$
$\langle proof \rangle$

If the union of a set of ordinals is a successor, then it is an element of that set.

lemma *Ord-Union-eq-succD*: $[\![\forall x{\in}X.\ Ord(x);\ \bigcup X = succ(j)]\!] \Longrightarrow succ(j) \in X$
$\langle proof \rangle$

lemma *Limit-Union* [*rule-format*]: $[\![I \neq 0;\ (\bigwedge i.\ i{\in}I \Longrightarrow Limit(i))]\!] \Longrightarrow Limit(\bigcup I)$
$\langle proof \rangle$

end

15 Special quantifiers

theory *OrdQuant* **imports** *Ordinal* **begin**

15.1 Quantifiers and union operator for ordinals

definition

$oall :: [i,\ i \Rightarrow o] \Rightarrow o$ **where**
$oall(A,\ P) \equiv \forall x.\ x{<}A \longrightarrow P(x)$

definition
$oex :: [i,\ i \Rightarrow o] \Rightarrow o$ **where**
$oex(A,\ P) \equiv \exists x.\ x{<}A \wedge P(x)$

definition

$OUnion :: [i,\ i \Rightarrow i] \Rightarrow i$ **where**
$OUnion(i,B) \equiv \{z{:}\ \bigcup x{\in}i.\ B(x).\ Ord(i)\}$

syntax

-oall :: [*idt*, *i*, *o*] ⇒ *o* (‹(‹*indent=3 notation=*‹*binder* ∀ <››∀ -<-./ -)› *10*)
-oex :: [*idt*, *i*, *o*] ⇒ *o* (‹(‹*indent=3 notation=*‹*binder* ∃ <››∃ -<-./ -)› *10*)
-OUNION :: [*idt*, *i*, *i*] ⇒ *i* (‹(‹*indent=3 notation=*‹*binder* ⋃ <››⋃ -<-./ -)› *10*)

syntax-consts
-oall ⇌ *oall* **and**
-oex ⇌ *oex* **and**
-OUNION ⇌ *OUnion*

translations
∀ *x*<*a*. *P* ⇌ *CONST oall*(*a*, λ*x*. *P*)
∃ *x*<*a*. *P* ⇌ *CONST oex*(*a*, λ*x*. *P*)
⋃ *x*<*a*. *B* ⇌ *CONST OUnion*(*a*, λ*x*. *B*)

15.1.1 simplification of the new quantifiers

lemma [*simp*]: (∀ *x*<*0*. *P*(*x*))
⟨*proof*⟩

lemma [*simp*]: ¬(∃ *x*<*0*. *P*(*x*))
⟨*proof*⟩

lemma [*simp*]: (∀ *x*<*succ*(*i*). *P*(*x*)) <−> (*Ord*(*i*) ⟶ *P*(*i*) ∧ (∀ *x*<*i*. *P*(*x*)))
⟨*proof*⟩

lemma [*simp*]: (∃ *x*<*succ*(*i*). *P*(*x*)) <−> (*Ord*(*i*) ∧ (*P*(*i*) | (∃ *x*<*i*. *P*(*x*))))
⟨*proof*⟩

15.1.2 Union over ordinals

lemma *Ord-OUN* [*intro*,*simp*]:
⟦⋀*x*. *x*<*A* ⟹ *Ord*(*B*(*x*))⟧ ⟹ *Ord*(⋃ *x*<*A*. *B*(*x*))
⟨*proof*⟩

lemma *OUN-upper-lt*:
⟦*a*<*A*; *i* < *b*(*a*); *Ord*(⋃ *x*<*A*. *b*(*x*))⟧ ⟹ *i* < (⋃ *x*<*A*. *b*(*x*))
⟨*proof*⟩

lemma *OUN-upper-le*:
⟦*a*<*A*; *i*≤*b*(*a*); *Ord*(⋃ *x*<*A*. *b*(*x*))⟧ ⟹ *i* ≤ (⋃ *x*<*A*. *b*(*x*))
⟨*proof*⟩

lemma *Limit-OUN-eq*: *Limit*(*i*) ⟹ (⋃ *x*<*i*. *x*) = *i*
⟨*proof*⟩

lemma *OUN-least*:
(⋀*x*. *x*<*A* ⟹ *B*(*x*) ⊆ *C*) ⟹ (⋃ *x*<*A*. *B*(*x*)) ⊆ *C*
⟨*proof*⟩

lemma *OUN-least-le*:

$[\![Ord(i); \ \bigwedge x.\ x{<}A \Longrightarrow b(x) \le i]\!] \Longrightarrow (\bigcup x{<}A.\ b(x)) \le i$
⟨*proof*⟩

lemma *le-implies-OUN-le-OUN*:
$[\![\bigwedge x.\ x{<}A \Longrightarrow c(x) \le d(x)]\!] \Longrightarrow (\bigcup x{<}A.\ c(x)) \le (\bigcup x{<}A.\ d(x))$
⟨*proof*⟩

lemma *OUN-UN-eq*:
$(\bigwedge x.\ x \in A \Longrightarrow Ord(B(x)))$
$\Longrightarrow (\bigcup z < (\bigcup x{\in}A.\ B(x)).\ C(z)) = (\bigcup x{\in}A.\ \bigcup z < B(x).\ C(z))$
⟨*proof*⟩

lemma *OUN-Union-eq*:
$(\bigwedge x.\ x \in X \Longrightarrow Ord(x))$
$\Longrightarrow (\bigcup z < \bigcup(X).\ C(z)) = (\bigcup x{\in}X.\ \bigcup z < x.\ C(z))$
⟨*proof*⟩

lemma *atomize-oall* [*symmetric*, *rulify*]:
$(\bigwedge x.\ x{<}A \Longrightarrow P(x)) \equiv Trueprop\ (\forall x{<}A.\ P(x))$
⟨*proof*⟩

15.1.3 universal quantifier for ordinals

lemma *oallI* [*intro*!]:
$[\![\bigwedge x.\ x{<}A \Longrightarrow P(x)]\!] \Longrightarrow \forall x{<}A.\ P(x)$
⟨*proof*⟩

lemma *ospec*: $[\![\forall x{<}A.\ P(x); \ \ x{<}A]\!] \Longrightarrow P(x)$
⟨*proof*⟩

lemma *oallE*:
$[\![\forall x{<}A.\ P(x); \ \ P(x) \Longrightarrow Q; \ \ \neg x{<}A \Longrightarrow Q]\!] \Longrightarrow Q$
⟨*proof*⟩

lemma *rev-oallE* [*elim*]:
$[\![\forall x{<}A.\ P(x); \ \ \neg x{<}A \Longrightarrow Q; \ \ P(x) \Longrightarrow Q]\!] \Longrightarrow Q$
⟨*proof*⟩

lemma *oall-simp* [*simp*]: $(\forall x{<}a.\ True) <-> True$
⟨*proof*⟩

lemma *oall-cong* [*cong*]:
$[\![a{=}a'; \ \ \bigwedge x.\ x{<}a' \Longrightarrow P(x) <-> P'(x)]\!]$
$\Longrightarrow oall(a, \lambda x.\ P(x)) <-> oall(a', \lambda x.\ P'(x))$
⟨*proof*⟩

15.1.4 existential quantifier for ordinals

lemma *oexI* [*intro*]:
$[\![P(x);\ \ x<A]\!] \Longrightarrow \exists\, x<A.\ P(x)$
⟨*proof*⟩

lemma *oexCI*:
$[\![\forall\, x<A.\ \neg P(x) \Longrightarrow P(a);\ \ a<A]\!] \Longrightarrow \exists\, x<A.\ P(x)$
⟨*proof*⟩

lemma *oexE* [*elim*!]:
$[\![\exists\, x<A.\ P(x);\ \ \bigwedge x.\ [\![x<A;\ P(x)]\!] \Longrightarrow Q]\!] \Longrightarrow Q$
⟨*proof*⟩

lemma *oex-cong* [*cong*]:
$[\![a=a';\ \ \bigwedge x.\ x<a' \Longrightarrow P(x) <-> P'(x)]\!]$
$\Longrightarrow oex(a,\ \lambda x.\ P(x)) <-> oex(a',\ \lambda x.\ P'(x))$
⟨*proof*⟩

15.1.5 Rules for Ordinal-Indexed Unions

lemma *OUN-I* [*intro*]: $[\![a<i;\ \ b \in B(a)]\!] \Longrightarrow b\colon (\bigcup z<i.\ B(z))$
⟨*proof*⟩

lemma *OUN-E* [*elim*!]:
$[\![b \in (\bigcup z<i.\ B(z));\ \ \bigwedge a.[\![b \in B(a);\ \ a<i]\!] \Longrightarrow R]\!] \Longrightarrow R$
⟨*proof*⟩

lemma *OUN-iff*: $b \in (\bigcup x<i.\ B(x)) <-> (\exists\, x<i.\ b \in B(x))$
⟨*proof*⟩

lemma *OUN-cong* [*cong*]:
$[\![i=j;\ \ \bigwedge x.\ x<j \Longrightarrow C(x)=D(x)]\!] \Longrightarrow (\bigcup x<i.\ C(x)) = (\bigcup x<j.\ D(x))$
⟨*proof*⟩

lemma *lt-induct*:
$[\![i<k;\ \ \bigwedge x.[\![x<k;\ \ \forall\, y<x.\ P(y)]\!] \Longrightarrow P(x)]\!] \ \Longrightarrow\ P(i)$
⟨*proof*⟩

15.2 Quantification over a class

definition
rall :: $[i \Rightarrow o,\ i \Rightarrow o] \Rightarrow o$ **where**
$rall(M,\ P) \equiv \forall\, x.\ M(x) \longrightarrow P(x)$

definition
rex :: $[i \Rightarrow o,\ i \Rightarrow o] \Rightarrow o$ **where**
$rex(M,\ P) \equiv \exists\, x.\ M(x) \wedge P(x)$

syntax
  -rall :: [*pttrn*, *i*⇒*o*, *o*] ⇒ *o* (‹(‹*indent=3 notation=*‹*binder* ∀ []››∀ -[-]./ -)› *10*)
  -rex :: [*pttrn*, *i*⇒*o*, *o*] ⇒ *o* (‹(‹*indent=3 notation=*‹*binder* ∃ []››∃ -[-]./ -)› *10*)
syntax-consts
  -rall ⇌ *rall* **and**
  -rex ⇌ *rex*
translations
  $\forall\, x[M].\ P \rightleftharpoons CONST\ rall(M,\ \lambda x.\ P)$
  $\exists\, x[M].\ P \rightleftharpoons CONST\ rex(M,\ \lambda x.\ P)$

15.2.1 Relativized universal quantifier

lemma *rallI* [*intro*!]: $[\![\bigwedge x.\ M(x) \Longrightarrow P(x)]\!] \Longrightarrow \forall\, x[M].\ P(x)$
⟨*proof*⟩

lemma *rspec*: $[\![\forall\, x[M].\ P(x);\ M(x)]\!] \Longrightarrow P(x)$
⟨*proof*⟩

lemma *rev-rallE* [*elim*]:
  $[\![\forall\, x[M].\ P(x);\ \ \neg\ M(x) \Longrightarrow Q;\ \ P(x) \Longrightarrow Q]\!] \Longrightarrow Q$
⟨*proof*⟩

lemma *rallE*: $[\![\forall\, x[M].\ P(x);\ \ P(x) \Longrightarrow Q;\ \ \neg\ M(x) \Longrightarrow Q]\!] \Longrightarrow Q$
⟨*proof*⟩

lemma *rall-triv* [*simp*]: $(\forall\, x[M].\ P) \longleftrightarrow ((\exists\, x.\ M(x)) \longrightarrow P)$
⟨*proof*⟩

lemma *rall-cong* [*cong*]:
  $(\bigwedge x.\ M(x) \Longrightarrow P(x) <-> P'(x)) \Longrightarrow (\forall\, x[M].\ P(x)) <-> (\forall\, x[M].\ P'(x))$
⟨*proof*⟩

15.2.2 Relativized existential quantifier

lemma *rexI* [*intro*]: $[\![P(x);\ M(x)]\!] \Longrightarrow \exists\, x[M].\ P(x)$
⟨*proof*⟩

lemma *rev-rexI*: $[\![M(x);\ \ P(x)]\!] \Longrightarrow \exists\, x[M].\ P(x)$
⟨*proof*⟩

lemma *rexCI*: $[\![\forall\, x[M].\ \neg P(x) \Longrightarrow P(a);\ M(a)]\!] \Longrightarrow \exists\, x[M].\ P(x)$
⟨*proof*⟩

lemma *rexE* [*elim*!]: $[\![\exists\, x[M].\ P(x);\ \ \bigwedge x.\ [\![M(x);\ P(x)]\!] \Longrightarrow Q]\!] \Longrightarrow Q$
⟨*proof*⟩

lemma *rex-triv* [*simp*]: $(\exists\, x[M].\ P) \longleftrightarrow ((\exists\, x.\ M(x)) \wedge P)$
⟨*proof*⟩

lemma *rex-cong* [*cong*]:
$(\bigwedge x.\ M(x) \Longrightarrow P(x) <-> P'(x)) \Longrightarrow (\exists\, x[M].\ P(x)) <-> (\exists\, x[M].\ P'(x))$
⟨*proof*⟩

lemma *rall-is-ball* [*simp*]: $(\forall\, x[\lambda z.\ z{\in}A].\ P(x)) <-> (\forall\, x{\in}A.\ P(x))$
⟨*proof*⟩

lemma *rex-is-bex* [*simp*]: $(\exists\, x[\lambda z.\ z{\in}A].\ P(x)) <-> (\exists\, x{\in}A.\ P(x))$
⟨*proof*⟩

lemma *atomize-rall*: $(\bigwedge x.\ M(x) \Longrightarrow P(x)) \equiv \mathit{Trueprop}\ (\forall\, x[M].\ P(x))$
⟨*proof*⟩

declare *atomize-rall* [*symmetric*, *rulify*]

lemma *rall-simps1*:
$(\forall\, x[M].\ P(x) \wedge Q) \quad <-> (\forall\, x[M].\ P(x)) \wedge ((\forall\, x[M].\ \mathit{False}) \mid Q)$
$(\forall\, x[M].\ P(x) \mid Q) \quad <-> ((\forall\, x[M].\ P(x)) \mid Q)$
$(\forall\, x[M].\ P(x) \longrightarrow Q) <-> ((\exists\, x[M].\ P(x)) \longrightarrow Q)$
$(\neg(\forall\, x[M].\ P(x))) <-> (\exists\, x[M].\ \neg P(x))$
⟨*proof*⟩

lemma *rall-simps2*:
$(\forall\, x[M].\ P \wedge Q(x)) \quad <-> ((\forall\, x[M].\ \mathit{False}) \mid P) \wedge (\forall\, x[M].\ Q(x))$
$(\forall\, x[M].\ P \mid Q(x)) \quad <-> (P \mid (\forall\, x[M].\ Q(x)))$
$(\forall\, x[M].\ P \longrightarrow Q(x)) <-> (P \longrightarrow (\forall\, x[M].\ Q(x)))$
⟨*proof*⟩

lemmas *rall-simps* [*simp*] = *rall-simps1 rall-simps2*

lemma *rall-conj-distrib*:
$(\forall\, x[M].\ P(x) \wedge Q(x)) <-> ((\forall\, x[M].\ P(x)) \wedge (\forall\, x[M].\ Q(x)))$
⟨*proof*⟩

lemma *rex-simps1*:
$(\exists\, x[M].\ P(x) \wedge Q) <-> ((\exists\, x[M].\ P(x)) \wedge Q)$
$(\exists\, x[M].\ P(x) \mid Q) <-> (\exists\, x[M].\ P(x)) \mid ((\exists\, x[M].\ \mathit{True}) \wedge Q)$
$(\exists\, x[M].\ P(x) \longrightarrow Q) <-> ((\forall\, x[M].\ P(x)) \longrightarrow ((\exists\, x[M].\ \mathit{True}) \wedge Q))$
$(\neg(\exists\, x[M].\ P(x))) <-> (\forall\, x[M].\ \neg P(x))$
⟨*proof*⟩

lemma *rex-simps2*:

(∃x[M]. P ∧ Q(x)) <-> (P ∧ (∃x[M]. Q(x)))
(∃x[M]. P | Q(x)) <-> ((∃x[M]. *True*) ∧ P) | (∃x[M]. Q(x))
(∃x[M]. P ⟶ Q(x)) <-> (((∀x[M]. *False*) | P) ⟶ (∃x[M]. Q(x)))
⟨*proof*⟩

lemmas *rex-simps* [*simp*] = *rex-simps1 rex-simps2*

lemma *rex-disj-distrib*:
(∃x[M]. P(x) | Q(x)) <-> ((∃x[M]. P(x)) | (∃x[M]. Q(x)))
⟨*proof*⟩

15.2.3 One-point rule for bounded quantifiers

lemma *rex-triv-one-point1* [*simp*]: (∃x[M]. x=a) <-> (M(a))
⟨*proof*⟩

lemma *rex-triv-one-point2* [*simp*]: (∃x[M]. a=x) <-> (M(a))
⟨*proof*⟩

lemma *rex-one-point1* [*simp*]: (∃x[M]. x=a ∧ P(x)) <-> (M(a) ∧ P(a))
⟨*proof*⟩

lemma *rex-one-point2* [*simp*]: (∃x[M]. a=x ∧ P(x)) <-> (M(a) ∧ P(a))
⟨*proof*⟩

lemma *rall-one-point1* [*simp*]: (∀x[M]. x=a ⟶ P(x)) <-> (M(a) ⟶ P(a))
⟨*proof*⟩

lemma *rall-one-point2* [*simp*]: (∀x[M]. a=x ⟶ P(x)) <-> (M(a) ⟶ P(a))
⟨*proof*⟩

15.2.4 Sets as Classes

definition
setclass :: [*i*,*i*] ⇒ *o* (‹(‹*open-block notation*=‹*prefix setclass*››##-)› [*40*] *40*)
where
setclass(A) ≡ λx. x ∈ A

lemma *setclass-iff* [*simp*]: *setclass*(A,x) <-> x ∈ A
⟨*proof*⟩

lemma *rall-setclass-is-ball* [*simp*]: (∀x[##A]. P(x)) <-> (∀x∈A. P(x))
⟨*proof*⟩

lemma *rex-setclass-is-bex* [*simp*]: (∃x[##A]. P(x)) <-> (∃x∈A. P(x))
⟨*proof*⟩

⟨*ML*⟩

Setting up the one-point-rule simproc

⟨*ML*⟩

end

16 The Natural numbers As a Least Fixed Point

theory *Nat* **imports** *OrdQuant Bool* **begin**

definition
  $nat :: i$ **where**
    $nat \equiv lfp(Inf, \lambda X. \{0\} \cup \{succ(i). i \in X\})$

definition
  $quasinat :: i \Rightarrow o$ **where**
    $quasinat(n) \equiv n=0 \mid (\exists m. n = succ(m))$

definition

  $nat\text{-}case :: [i, i\Rightarrow i, i]\Rightarrow i$ **where**
    $nat\text{-}case(a,b,k) \equiv THE\ y. k=0 \wedge y=a \mid (\exists x. k=succ(x) \wedge y=b(x))$

definition
  $nat\text{-}rec :: [i, i, [i,i]\Rightarrow i]\Rightarrow i$ **where**
    $nat\text{-}rec(k,a,b) \equiv$
      $wfrec(Memrel(nat), k, \lambda n\ f. nat\text{-}case(a, \lambda m. b(m, f`m), n))$

definition
  $Le :: i$ **where**
    $Le \equiv \{\langle x,y\rangle:nat*nat. x \le y\}$

definition
  $Lt :: i$ **where**
    $Lt \equiv \{\langle x, y\rangle:nat*nat. x < y\}$

definition
  $Ge :: i$ **where**
    $Ge \equiv \{\langle x,y\rangle:nat*nat. y \le x\}$

definition
  $Gt :: i$ **where**
    $Gt \equiv \{\langle x,y\rangle:nat*nat. y < x\}$

definition
  $greater\text{-}than :: i\Rightarrow i$ **where**
    $greater\text{-}than(n) \equiv \{i \in nat. n < i\}$

No need for a less-than operator: a natural number is its list of predecessors!

lemma *nat-bnd-mono*: $bnd\text{-}mono(Inf, \lambda X.\ \{0\} \cup \{succ(i).\ i \in X\})$
⟨*proof*⟩

lemmas *nat-unfold* = *nat-bnd-mono* [*THEN nat-def* [*THEN def-lfp-unfold*]]

lemma *nat-0I* [*iff*,*TC*]: $0 \in nat$
⟨*proof*⟩

lemma *nat-succI* [*intro*!,*TC*]: $n \in nat \Longrightarrow succ(n) \in nat$
⟨*proof*⟩

lemma *nat-1I* [*iff*,*TC*]: $1 \in nat$
⟨*proof*⟩

lemma *nat-2I* [*iff*,*TC*]: $2 \in nat$
⟨*proof*⟩

lemma *bool-subset-nat*: $bool \subseteq nat$
⟨*proof*⟩

lemmas *bool-into-nat* = *bool-subset-nat* [*THEN subsetD*]

16.1 Injectivity Properties and Induction

lemma *nat-induct* [*case-names 0 succ*, *induct set*: *nat*]:
$[\![n \in nat;\ P(0);\ \bigwedge x.\ [\![x \in nat;\ P(x)]\!] \Longrightarrow P(succ(x))]\!] \Longrightarrow P(n)$
⟨*proof*⟩

lemma *natE*:
assumes $n \in nat$
obtains (*0*) $n=0$ | (*succ*) x **where** $x \in nat$ $n=succ(x)$
⟨*proof*⟩

lemma *nat-into-Ord* [*simp*]: $n \in nat \Longrightarrow Ord(n)$
⟨*proof*⟩

lemmas *nat-0-le* = *nat-into-Ord* [*THEN Ord-0-le*]

lemmas *nat-le-refl* = *nat-into-Ord* [*THEN le-refl*]

lemma *Ord-nat* [*iff*]: $Ord(nat)$
⟨*proof*⟩

lemma *Limit-nat* [*iff*]: $Limit(nat)$
⟨*proof*⟩

lemma *naturals-not-limit*: $a \in nat \Longrightarrow \neg\ Limit(a)$
⟨*proof*⟩

lemma *succ-natD*: $succ(i)$: $nat \Longrightarrow i \in nat$
⟨*proof*⟩

lemma *nat-succ-iff* [*iff*]: $succ(n)$: $nat \longleftrightarrow n \in nat$
⟨*proof*⟩

lemma *nat-le-Limit*: $Limit(i) \Longrightarrow nat \le i$
⟨*proof*⟩

lemmas *succ-in-naturalD* = *Ord-trans* [*OF succI1 - nat-into-Ord*]

lemma *lt-nat-in-nat*: $[\![m<n;\ \ n \in nat]\!] \Longrightarrow m \in nat$
⟨*proof*⟩

lemma *le-in-nat*: $[\![m \le n;\ n \in nat]\!] \Longrightarrow m \in nat$
⟨*proof*⟩

16.2 Variations on Mathematical Induction

lemmas *complete-induct* = *Ord-induct* [*OF - Ord-nat*, *case-names less*, *consumes 1*]

lemma *complete-induct-rule* [*case-names less*, *consumes 1*]:
$i \in nat \Longrightarrow (\bigwedge x.\ x \in nat \Longrightarrow (\bigwedge y.\ y \in x \Longrightarrow P(y)) \Longrightarrow P(x)) \Longrightarrow P(i)$
⟨*proof*⟩

lemma *nat-induct-from*:
assumes $m \le n$ $m \in nat$ $n \in nat$
and $P(m)$
and $\bigwedge x.\ [\![x \in nat;\ \ m \le x;\ \ P(x)]\!] \Longrightarrow P(succ(x))$
shows $P(n)$
⟨*proof*⟩

lemma *diff-induct* [*case-names 0 0-succ succ-succ*, *consumes 2*]:
$[\![m \in nat;\ \ n \in nat;$
$\bigwedge x.\ x \in nat \Longrightarrow P(x,0);$
$\bigwedge y.\ y \in nat \Longrightarrow P(0,succ(y));$
$\bigwedge x\ y.\ [\![x \in nat;\ \ y \in nat;\ \ P(x,y)]\!] \Longrightarrow P(succ(x),succ(y))]\!]$
$\Longrightarrow P(m,n)$
⟨*proof*⟩

lemma *succ-lt-induct-lemma* [*rule-format*]:
$m \in nat \Longrightarrow P(m,succ(m)) \longrightarrow (\forall x{\in}nat.\ P(m,x) \longrightarrow P(m,succ(x))) \longrightarrow$
$(\forall n{\in}nat.\ m{<}n \longrightarrow P(m,n))$
⟨*proof*⟩

lemma *succ-lt-induct*:
$[\![m{<}n;\ \ n \in nat;$
$P(m,succ(m));$
$\bigwedge x.\ [\![x \in nat;\ \ P(m,x)]\!] \Longrightarrow P(m,succ(x))]\!]$
$\Longrightarrow P(m,n)$
⟨*proof*⟩

16.3 quasinat: to allow a case-split rule for *nat-case*

True if the argument is zero or any successor

lemma [*iff*]: $quasinat(0)$
⟨*proof*⟩

lemma [*iff*]: $quasinat(succ(x))$
⟨*proof*⟩

lemma *nat-imp-quasinat*: $n \in nat \Longrightarrow quasinat(n)$
⟨*proof*⟩

lemma *non-nat-case*: $\neg\ quasinat(x) \Longrightarrow nat\text{-}case(a,b,x) = 0$
⟨*proof*⟩

lemma *nat-cases-disj*: $k{=}0 \mid (\exists\, y.\ k = succ(y)) \mid \neg\ quasinat(k)$
⟨*proof*⟩

lemma *nat-cases*:
$[\![k{=}0 \Longrightarrow P;\ \ \bigwedge y.\ k = succ(y) \Longrightarrow P;\ \neg\ quasinat(k) \Longrightarrow P]\!] \Longrightarrow P$
⟨*proof*⟩

lemma *nat-case-0* [*simp*]: $nat\text{-}case(a,b,0) = a$
⟨*proof*⟩

lemma *nat-case-succ* [*simp*]: $nat\text{-}case(a,b,succ(n)) = b(n)$
⟨*proof*⟩

lemma *nat-case-type* [*TC*]:
$[\![n \in nat;\ \ a \in C(0);\ \ \bigwedge m.\ m \in nat \Longrightarrow b(m){:}\ C(succ(m))]\!]$
$\Longrightarrow nat\text{-}case(a,b,n) \in C(n)$

⟨*proof*⟩

lemma *split-nat-case*:
  $P(nat\text{-}case(a,b,k)) \longleftrightarrow$
   $((k=0 \longrightarrow P(a)) \land (\forall x.\ k=succ(x) \longrightarrow P(b(x))) \land (\neg\ quasinat(k) \longrightarrow P(0)))$
⟨*proof*⟩

16.4 Recursion on the Natural Numbers

lemma *nat-rec-0*: $nat\text{-}rec(0,a,b) = a$
⟨*proof*⟩

lemma *nat-rec-succ*: $m \in nat \Longrightarrow nat\text{-}rec(succ(m),a,b) = b(m,\ nat\text{-}rec(m,a,b))$
⟨*proof*⟩

lemma *Un-nat-type* [*TC*]: $[\![i \in nat;\ j \in nat]\!] \Longrightarrow i \cup j \in nat$
⟨*proof*⟩

lemma *Int-nat-type* [*TC*]: $[\![i \in nat;\ j \in nat]\!] \Longrightarrow i \cap j \in nat$
⟨*proof*⟩

lemma *nat-nonempty* [*simp*]: $nat \neq 0$
⟨*proof*⟩

A natural number is the set of its predecessors

lemma *nat-eq-Collect-lt*: $i \in nat \Longrightarrow \{j{\in}nat.\ j<i\} = i$
⟨*proof*⟩

lemma *Le-iff* [*iff*]: $\langle x,y\rangle \in Le \longleftrightarrow x \leq y \land x \in nat \land y \in nat$
⟨*proof*⟩

end

17 Inductive and Coinductive Definitions

theory *Inductive*
imports *Fixedpt QPair Nat*
keywords
  inductive coinductive inductive-cases rep-datatype primrec :: *thy-decl* **and**
  domains intros monos con-defs type-intros type-elims
   elimination induction case-eqns recursor-eqns :: *quasi-command*
begin

lemma *def-swap-iff*: $a \equiv b \Longrightarrow a = c \longleftrightarrow c = b$
  ⟨*proof*⟩

lemma *def-trans*: $f \equiv g \Longrightarrow g(a) = b \Longrightarrow f(a) = b$
⟨*proof*⟩

lemma *refl-thin*: $\bigwedge P.\ a = a \Longrightarrow P \Longrightarrow P$ ⟨*proof*⟩

⟨*ML*⟩

end

18 Epsilon Induction and Recursion

theory *Epsilon* **imports** *Nat* **begin**

definition
eclose :: $i \Rightarrow i$ **where**
$eclose(A) \equiv \bigcup n \in nat.\ nat\text{-}rec(n, A, \lambda m\ r. \bigcup(r))$

definition
transrec :: $[i, [i,i] \Rightarrow i] \Rightarrow i$ **where**
$transrec(a,H) \equiv wfrec(Memrel(eclose(\{a\})), a, H)$

definition
rank :: $i \Rightarrow i$ **where**
$rank(a) \equiv transrec(a, \lambda x\ f. \bigcup y \in x.\ succ(f`y))$

definition
transrec2 :: $[i, i, [i,i] \Rightarrow i] \Rightarrow i$ **where**
$transrec2(k, a, b) \equiv$
$transrec(k,$
$\lambda i\ r.\ if(i=0, a,$
$if(\exists j.\ i=succ(j),$
$b(THE\ j.\ i=succ(j), r`(THE\ j.\ i=succ(j))),$
$\bigcup j<i.\ r`j)))$

definition
recursor :: $[i, [i,i] \Rightarrow i, i] \Rightarrow i$ **where**
$recursor(a,b,k) \equiv transrec(k, \lambda n\ f.\ nat\text{-}case(a, \lambda m.\ b(m, f`m), n))$

definition
rec :: $[i, i, [i,i] \Rightarrow i] \Rightarrow i$ **where**
$rec(k,a,b) \equiv recursor(a,b,k)$

18.1 Basic Closure Properties

lemma *arg-subset-eclose*: $A \subseteq eclose(A)$
⟨*proof*⟩

lemmas *arg-into-eclose* = *arg-subset-eclose* [*THEN subsetD*]

lemma *Transset-eclose*: $Transset(eclose(A))$
⟨*proof*⟩

lemmas *eclose-subset* =
Transset-eclose [*unfolded Transset-def*, *THEN bspec*]

lemmas *ecloseD* = *eclose-subset* [*THEN subsetD*]

lemmas *arg-in-eclose-sing* = *arg-subset-eclose* [*THEN singleton-subsetD*]
lemmas *arg-into-eclose-sing* = *arg-in-eclose-sing* [*THEN ecloseD*]

lemmas *eclose-induct* =
Transset-induct [*OF - Transset-eclose*, *induct set*: *eclose*]

lemma *eps-induct*:
$[\![\bigwedge x.\ \forall y{\in}x.\ P(y) \Longrightarrow P(x)]\!] \Longrightarrow P(a)$
⟨*proof*⟩

18.2 Leastness of *eclose*

lemma *eclose-least-lemma*:
$[\![Transset(X);\ A{<=}X;\ n \in nat]\!] \Longrightarrow nat\text{-}rec(n,\ A,\ \lambda m\ r.\ \bigcup(r)) \subseteq X$
⟨*proof*⟩

lemma *eclose-least*:
$[\![Transset(X);\ A{<=}X]\!] \Longrightarrow eclose(A) \subseteq X$
⟨*proof*⟩

lemma *eclose-induct-down* [*consumes 1*]:
$[\![a \in eclose(b);$
$\bigwedge y.\ \ [\![y \in b]\!] \Longrightarrow P(y);$
$\bigwedge y\ z.\ [\![y \in eclose(b);\ P(y);\ z \in y]\!] \Longrightarrow P(z)$
$]\!] \Longrightarrow P(a)$
⟨*proof*⟩

lemma *Transset-eclose-eq-arg*: $Transset(X) \Longrightarrow eclose(X) = X$
⟨*proof*⟩

A transitive set either is empty or contains the empty set.

lemma *Transset-0-lemma* [*rule-format*]: $Transset(A) \Longrightarrow x{\in}A \longrightarrow 0{\in}A$
⟨*proof*⟩

lemma *Transset-0-disj*: $Transset(A) \Longrightarrow A{=}0 \mid 0{\in}A$

$\langle proof \rangle$

18.3 Epsilon Recursion

lemma *mem-eclose-trans*: $[\![A \in eclose(B);\ \ B \in eclose(C)]\!] \Longrightarrow A \in eclose(C)$
$\langle proof \rangle$

lemma *mem-eclose-sing-trans*:
$[\![A \in eclose(\{B\});\ \ B \in eclose(\{C\})]\!] \Longrightarrow A \in eclose(\{C\})$
$\langle proof \rangle$

lemma *under-Memrel*: $[\![Transset(i);\ \ j \in i]\!] \Longrightarrow Memrel(i)-``\{j\} = j$
$\langle proof \rangle$

lemma *lt-Memrel*: $j < i \Longrightarrow Memrel(i) -`` \{j\} = j$
$\langle proof \rangle$

lemmas *under-Memrel-eclose* = *Transset-eclose* [*THEN under-Memrel*]

lemmas *wfrec-ssubst* = *wf-Memrel* [*THEN wfrec*, *THEN ssubst*]

lemma *wfrec-eclose-eq*:
$[\![k \in eclose(\{j\});\ \ j \in eclose(\{i\})]\!] \Longrightarrow$
$wfrec(Memrel(eclose(\{i\})),\ k,\ H) = wfrec(Memrel(eclose(\{j\})),\ k,\ H)$
$\langle proof \rangle$

lemma *wfrec-eclose-eq2*:
$k \in i \Longrightarrow wfrec(Memrel(eclose(\{i\})),k,H) = wfrec(Memrel(eclose(\{k\})),k,H)$
$\langle proof \rangle$

lemma *transrec*: $transrec(a,H) = H(a,\ \lambda x{\in}a.\ transrec(x,H))$
$\langle proof \rangle$

lemma *def-transrec*:
$[\![\bigwedge x.\ f(x) \equiv transrec(x,H)]\!] \Longrightarrow f(a) = H(a,\ \lambda x{\in}a.\ f(x))$
$\langle proof \rangle$

lemma *transrec-type*:
$[\![\bigwedge x\ u.\ [\![x \in eclose(\{a\});\ \ u \in Pi(x,B)]\!] \Longrightarrow H(x,u) \in B(x)]\!]$
$\Longrightarrow transrec(a,H) \in B(a)$
$\langle proof \rangle$

lemma *eclose-sing-Ord*: $Ord(i) \Longrightarrow eclose(\{i\}) \subseteq succ(i)$
$\langle proof \rangle$

lemma *succ-subset-eclose-sing*: $succ(i) \subseteq eclose(\{i\})$

⟨*proof*⟩

lemma *eclose-sing-Ord-eq*: $Ord(i) \Longrightarrow eclose(\{i\}) = succ(i)$
⟨*proof*⟩

lemma *Ord-transrec-type*:
assumes *jini*: $j \in i$
and *ordi*: $Ord(i)$
and *minor*: $\bigwedge x\ u.\ [\![x \in i;\ \ u \in Pi(x,B)]\!] \Longrightarrow H(x,u) \in B(x)$
shows $transrec(j,H) \in B(j)$
⟨*proof*⟩

18.4 Rank

lemma *rank*: $rank(a) = (\bigcup y{\in}a.\ succ(rank(y)))$
⟨*proof*⟩

lemma *Ord-rank* [*simp*]: $Ord(rank(a))$
⟨*proof*⟩

lemma *rank-of-Ord*: $Ord(i) \Longrightarrow rank(i) = i$
⟨*proof*⟩

lemma *rank-lt*: $a \in b \Longrightarrow rank(a) < rank(b)$
⟨*proof*⟩

lemma *eclose-rank-lt*: $a \in eclose(b) \Longrightarrow rank(a) < rank(b)$
⟨*proof*⟩

lemma *rank-mono*: $a{<=}b \Longrightarrow rank(a) \leq rank(b)$
⟨*proof*⟩

lemma *rank-Pow*: $rank(Pow(a)) = succ(rank(a))$
⟨*proof*⟩

lemma *rank-0* [*simp*]: $rank(0) = 0$
⟨*proof*⟩

lemma *rank-succ* [*simp*]: $rank(succ(x)) = succ(rank(x))$
⟨*proof*⟩

lemma *rank-Union*: $rank(\bigcup(A)) = (\bigcup x{\in}A.\ rank(x))$
⟨*proof*⟩

lemma *rank-eclose*: $rank(eclose(a)) = rank(a)$
⟨*proof*⟩

lemma *rank-pair1*: $rank(a) < rank(\langle a,b\rangle)$
⟨*proof*⟩

lemma *rank-pair2*: $rank(b) < rank(\langle a,b\rangle)$
⟨*proof*⟩

lemma *the-equality-if*:
$P(a) \Longrightarrow (THE\ x.\ P(x)) = (if\ (\exists!x.\ P(x))\ then\ a\ else\ 0)$
⟨*proof*⟩

lemma *rank-apply*: $[\![i \in domain(f); function(f)]\!] \Longrightarrow rank(f`i) < rank(f)$
⟨*proof*⟩

18.5 Corollaries of Leastness

lemma *mem-eclose-subset*: $A \in B \Longrightarrow eclose(A)<=eclose(B)$
⟨*proof*⟩

lemma *eclose-mono*: $A<=B \Longrightarrow eclose(A) \subseteq eclose(B)$
⟨*proof*⟩

lemma *eclose-idem*: $eclose(eclose(A)) = eclose(A)$
⟨*proof*⟩

lemma *transrec2-0* [*simp*]: $transrec2(0,a,b) = a$
⟨*proof*⟩

lemma *transrec2-succ* [*simp*]: $transrec2(succ(i),a,b) = b(i, transrec2(i,a,b))$
⟨*proof*⟩

lemma *transrec2-Limit*:
$Limit(i) \Longrightarrow transrec2(i,a,b) = (\bigcup j<i.\ transrec2(j,a,b))$
⟨*proof*⟩

lemma *def-transrec2*:
$(\bigwedge x.\ f(x)\equiv transrec2(x,a,b))$
$\Longrightarrow f(0) = a \land$
$f(succ(i)) = b(i, f(i)) \land$
$(Limit(K) \longrightarrow f(K) = (\bigcup j<K.\ f(j)))$
⟨*proof*⟩

lemmas *recursor-lemma* = *recursor-def* [*THEN def-transrec*, *THEN trans*]

lemma *recursor-0*: *recursor*(*a*,*b*,*0*) = *a*
⟨*proof*⟩

lemma *recursor-succ*: *recursor*(*a*,*b*,*succ*(*m*)) = *b*(*m*, *recursor*(*a*,*b*,*m*))
⟨*proof*⟩

lemma *rec-0* [*simp*]: *rec*(*0*,*a*,*b*) = *a*
⟨*proof*⟩

lemma *rec-succ* [*simp*]: *rec*(*succ*(*m*),*a*,*b*) = *b*(*m*, *rec*(*m*,*a*,*b*))
⟨*proof*⟩

lemma *rec-type*:
⟦*n* ∈ *nat*;
a ∈ *C*(*0*);
⋀*m z*. ⟦*m* ∈ *nat*; *z* ∈ *C*(*m*)⟧ ⟹ *b*(*m*,*z*): *C*(*succ*(*m*))⟧
⟹ *rec*(*n*,*a*,*b*) ∈ *C*(*n*)
⟨*proof*⟩

end

19 Partial and Total Orderings: Basic Definitions and Properties

theory *Order* **imports** *WF Perm* **begin**

We adopt the following convention: *ord* is used for strict orders and *order* is used for their reflexive counterparts.

definition
part-ord :: [*i*,*i*]⇒*o* **where**
part-ord(*A*,*r*) ≡ *irrefl*(*A*,*r*) ∧ *trans*[*A*](*r*)

definition
linear :: [*i*,*i*]⇒*o* **where**
linear(*A*,*r*) ≡ (∀ *x*∈*A*. ∀ *y*∈*A*. ⟨*x*,*y*⟩:*r* | *x*=*y* | ⟨*y*,*x*⟩:*r*)

definition
tot-ord :: [*i*,*i*]⇒*o* **where**
tot-ord(*A*,*r*) ≡ *part-ord*(*A*,*r*) ∧ *linear*(*A*,*r*)

definition
preorder-on(*A*, *r*) ≡ *refl*(*A*, *r*) ∧ *trans*[*A*](*r*)

definition
partial-order-on$(A,\ r) \equiv$ *preorder-on*$(A,\ r) \wedge$ *antisym*(r)

abbreviation
Preorder$(r) \equiv$ *preorder-on*(*field*$(r),\ r)$

abbreviation
Partial-order$(r) \equiv$ *partial-order-on*(*field*$(r),\ r)$

definition
well-ord :: $[i,i]\Rightarrow o$ **where**
well-ord$(A,r) \equiv$ *tot-ord*$(A,r) \wedge$ *wf*$[A](r)$

definition
mono-map :: $[i,i,i,i]\Rightarrow i$ **where**
mono-map$(A,r,B,s) \equiv$
$\{f \in A{-}{>}B.\ \forall x{\in}A.\ \forall y{\in}A.\ \langle x,y\rangle{:}r \longrightarrow {<}f`x,f`y{>}{:}s\}$

definition
ord-iso :: $[i,i,i,i]\Rightarrow i$ (‹(‹notation=‹infix ord-iso››⟨-, -⟩ ≅/ ⟨-, -⟩)› *51*) **where**
$\langle A,r\rangle \cong \langle B,s\rangle \equiv$
$\{f \in bij(A,B).\ \forall x{\in}A.\ \forall y{\in}A.\ \langle x,y\rangle{:}r \longleftrightarrow {<}f`x,f`y{>}{:}s\}$

definition
pred :: $[i,i,i]\Rightarrow i$ **where**
pred$(A,x,r) \equiv \{y \in A.\ \langle y,x\rangle{:}r\}$

definition
ord-iso-map :: $[i,i,i,i]\Rightarrow i$ **where**
ord-iso-map$(A,r,B,s) \equiv$
$\bigcup x{\in}A.\ \bigcup y{\in}B.\ \bigcup f \in$ *ord-iso*(*pred*$(A,x,r),\ r,$ *pred*$(B,y,s),\ s).\ \{\langle x,y\rangle\}$

definition
first :: $[i,\ i,\ i] \Rightarrow o$ **where**
first$(u,\ X,\ R) \equiv u \in X \wedge (\forall v{\in}X.\ v{\neq}u \longrightarrow \langle u,v\rangle \in R)$

19.1 Immediate Consequences of the Definitions

lemma *part-ord-Imp-asym*:
part-ord$(A,r) \Longrightarrow$ *asym*$(r \cap A{*}A)$
⟨*proof*⟩

lemma *linearE*:
$[\![$*linear*$(A,r);\ \ x \in A;\ \ y \in A;$
$\langle x,y\rangle{:}r \Longrightarrow P;\ \ x{=}y \Longrightarrow P;\ \ \langle y,x\rangle{:}r \Longrightarrow P]\!]$
$\Longrightarrow P$
⟨*proof*⟩

lemma *well-ordI*:
  $[\![wf[A](r);\ linear(A,r)]\!] \Longrightarrow well\text{-}ord(A,r)$
⟨*proof*⟩

lemma *well-ord-is-wf*:
  $well\text{-}ord(A,r) \Longrightarrow wf[A](r)$
⟨*proof*⟩

lemma *well-ord-is-trans-on*:
  $well\text{-}ord(A,r) \Longrightarrow trans[A](r)$
⟨*proof*⟩

lemma *well-ord-is-linear*: $well\text{-}ord(A,r) \Longrightarrow linear(A,r)$
⟨*proof*⟩

lemma *pred-iff*: $y \in pred(A,x,r) \longleftrightarrow \langle y,x\rangle{:}r \wedge y \in A$
⟨*proof*⟩

lemmas *predI* = *conjI* [*THEN pred-iff* [*THEN iffD2*]]

lemma *predE*: $[\![y \in pred(A,x,r);\ \ [\![y \in A;\ \langle y,x\rangle{:}r]\!] \Longrightarrow P]\!] \Longrightarrow P$
⟨*proof*⟩

lemma *pred-subset-under*: $pred(A,x,r) \subseteq r -`` \{x\}$
⟨*proof*⟩

lemma *pred-subset*: $pred(A,x,r) \subseteq A$
⟨*proof*⟩

lemma *pred-pred-eq*:
  $pred(pred(A,x,r),\ y,\ r) = pred(A,x,r) \cap pred(A,y,r)$
⟨*proof*⟩

lemma *trans-pred-pred-eq*:
  $[\![trans[A](r);\ \ \langle y,x\rangle{:}r;\ \ x \in A;\ \ y \in A]\!]$
  $\Longrightarrow pred(pred(A,x,r),\ y,\ r) = pred(A,y,r)$
⟨*proof*⟩

19.2 Restricting an Ordering's Domain

lemma *part-ord-subset*:
  $[\![part\text{-}ord(A,r);\ \ B{<=}A]\!] \Longrightarrow part\text{-}ord(B,r)$
⟨*proof*⟩

lemma *linear-subset*:
$[\![linear(A,r);\ \ B<=A]\!] \Longrightarrow linear(B,r)$
⟨*proof*⟩

lemma *tot-ord-subset*:
$[\![tot\text{-}ord(A,r);\ \ B<=A]\!] \Longrightarrow tot\text{-}ord(B,r)$
⟨*proof*⟩

lemma *well-ord-subset*:
$[\![well\text{-}ord(A,r);\ \ B<=A]\!] \Longrightarrow well\text{-}ord(B,r)$
⟨*proof*⟩

lemma *irrefl-Int-iff*: $irrefl(A,r \cap A{*}A) \longleftrightarrow irrefl(A,r)$
⟨*proof*⟩

lemma *trans-on-Int-iff*: $trans[A](r \cap A{*}A) \longleftrightarrow trans[A](r)$
⟨*proof*⟩

lemma *part-ord-Int-iff*: $part\text{-}ord(A,r \cap A{*}A) \longleftrightarrow part\text{-}ord(A,r)$
⟨*proof*⟩

lemma *linear-Int-iff*: $linear(A,r \cap A{*}A) \longleftrightarrow linear(A,r)$
⟨*proof*⟩

lemma *tot-ord-Int-iff*: $tot\text{-}ord(A,r \cap A{*}A) \longleftrightarrow tot\text{-}ord(A,r)$
⟨*proof*⟩

lemma *wf-on-Int-iff*: $wf[A](r \cap A{*}A) \longleftrightarrow wf[A](r)$
⟨*proof*⟩

lemma *well-ord-Int-iff*: $well\text{-}ord(A,r \cap A{*}A) \longleftrightarrow well\text{-}ord(A,r)$
⟨*proof*⟩

19.3 Empty and Unit Domains

lemma *wf-on-any-0*: $wf[A](0)$
⟨*proof*⟩

19.3.1 Relations over the Empty Set

lemma *irrefl-0*: $irrefl(0,r)$
⟨*proof*⟩

lemma *trans-on-0*: $trans[0](r)$
⟨*proof*⟩

lemma *part-ord-0*: $part\text{-}ord(0,r)$

⟨*proof*⟩

lemma *linear-0*: $linear(0,r)$
⟨*proof*⟩

lemma *tot-ord-0*: $tot\text{-}ord(0,r)$
⟨*proof*⟩

lemma *wf-on-0*: $wf[0](r)$
⟨*proof*⟩

lemma *well-ord-0*: $well\text{-}ord(0,r)$
⟨*proof*⟩

19.3.2 The Empty Relation Well-Orders the Unit Set

by Grabczewski

lemma *tot-ord-unit*: $tot\text{-}ord(\{a\},0)$
⟨*proof*⟩

lemma *well-ord-unit*: $well\text{-}ord(\{a\},0)$
⟨*proof*⟩

19.4 Order-Isomorphisms

Suppes calls them "similarities"

lemma *mono-map-is-fun*: $f \in mono\text{-}map(A,r,B,s) \Longrightarrow f \in A{-}{>}B$
⟨*proof*⟩

lemma *mono-map-is-inj*:
$[\![linear(A,r);\ \ wf[B](s);\ \ f \in mono\text{-}map(A,r,B,s)]\!] \Longrightarrow f \in inj(A,B)$
⟨*proof*⟩

lemma *ord-isoI*:
$[\![f \in bij(A,\ B);$
$\bigwedge x\ y.\ [\![x \in A;\ y \in A]\!] \Longrightarrow \langle x,\ y\rangle \in r \longleftrightarrow <f`x,\ f`y> \in s]\!]$
$\Longrightarrow f \in ord\text{-}iso(A,r,B,s)$
⟨*proof*⟩

lemma *ord-iso-is-mono-map*:
$f \in ord\text{-}iso(A,r,B,s) \Longrightarrow f \in mono\text{-}map(A,r,B,s)$
⟨*proof*⟩

lemma *ord-iso-is-bij*:
$f \in ord\text{-}iso(A,r,B,s) \Longrightarrow f \in bij(A,B)$
⟨*proof*⟩

lemma *ord-iso-apply*:
$[\![f \in \textit{ord-iso}(A,r,B,s);\ \ \langle x,y\rangle: r;\ \ x \in A;\ \ y \in A]\!] \Longrightarrow <f`x,\ f`y> \in s$
⟨*proof*⟩

lemma *ord-iso-converse*:
$[\![f \in \textit{ord-iso}(A,r,B,s);\ \ \langle x,y\rangle: s;\ \ x \in B;\ \ y \in B]\!]$
$\Longrightarrow <\textit{converse}(f)\ `\ x,\ \textit{converse}(f)\ `\ y> \in r$
⟨*proof*⟩

lemma *ord-iso-refl*: $\textit{id}(A)$: $\textit{ord-iso}(A,r,A,r)$
⟨*proof*⟩

lemma *ord-iso-sym*: $f \in \textit{ord-iso}(A,r,B,s) \Longrightarrow \textit{converse}(f)$: $\textit{ord-iso}(B,s,A,r)$
⟨*proof*⟩

lemma *mono-map-trans*:
$[\![g \in \textit{mono-map}(A,r,B,s);\ \ f \in \textit{mono-map}(B,s,C,t)]\!]$
$\Longrightarrow (f\ O\ g)$: $\textit{mono-map}(A,r,C,t)$
⟨*proof*⟩

lemma *ord-iso-trans*:
$[\![g \in \textit{ord-iso}(A,r,B,s);\ \ f \in \textit{ord-iso}(B,s,C,t)]\!]$
$\Longrightarrow (f\ O\ g)$: $\textit{ord-iso}(A,r,C,t)$
⟨*proof*⟩

lemma *mono-ord-isoI*:
$[\![f \in \textit{mono-map}(A,r,B,s);\ \ g \in \textit{mono-map}(B,s,A,r);$
$f\ O\ g = \textit{id}(B);\ \ g\ O\ f = \textit{id}(A)]\!] \Longrightarrow f \in \textit{ord-iso}(A,r,B,s)$
⟨*proof*⟩

lemma *well-ord-mono-ord-isoI*:
$[\![\textit{well-ord}(A,r);\ \ \textit{well-ord}(B,s);$
$f \in \textit{mono-map}(A,r,B,s);\ \ \textit{converse}(f)$: $\textit{mono-map}(B,s,A,r)]\!]$
$\Longrightarrow f \in \textit{ord-iso}(A,r,B,s)$
⟨*proof*⟩

lemma *part-ord-ord-iso*:

⟦*part-ord*(*B*,*s*); *f* ∈ *ord-iso*(*A*,*r*,*B*,*s*)⟧ ⟹ *part-ord*(*A*,*r*)
⟨*proof*⟩

lemma *linear-ord-iso*:
⟦*linear*(*B*,*s*); *f* ∈ *ord-iso*(*A*,*r*,*B*,*s*)⟧ ⟹ *linear*(*A*,*r*)
⟨*proof*⟩

lemma *wf-on-ord-iso*:
⟦*wf*[*B*](*s*); *f* ∈ *ord-iso*(*A*,*r*,*B*,*s*)⟧ ⟹ *wf*[*A*](*r*)
⟨*proof*⟩

lemma *well-ord-ord-iso*:
⟦*well-ord*(*B*,*s*); *f* ∈ *ord-iso*(*A*,*r*,*B*,*s*)⟧ ⟹ *well-ord*(*A*,*r*)
⟨*proof*⟩

19.5 Main results of Kunen, Chapter 1 section 6

lemma *well-ord-iso-subset-lemma*:
⟦*well-ord*(*A*,*r*); *f* ∈ *ord-iso*(*A*,*r*, *A'*,*r*); *A'*<= *A*; *y* ∈ *A*⟧
⟹ ¬ <*f*‘*y*, *y*>: *r*
⟨*proof*⟩

lemma *well-ord-iso-predE*:
⟦*well-ord*(*A*,*r*); *f* ∈ *ord-iso*(*A*, *r*, *pred*(*A*,*x*,*r*), *r*); *x* ∈ *A*⟧ ⟹ *P*
⟨*proof*⟩

lemma *well-ord-iso-pred-eq*:
⟦*well-ord*(*A*,*r*); *f* ∈ *ord-iso*(*pred*(*A*,*a*,*r*), *r*, *pred*(*A*,*c*,*r*), *r*);
a ∈ *A*; *c* ∈ *A*⟧ ⟹ *a*=*c*
⟨*proof*⟩

lemma *ord-iso-image-pred*:
⟦*f* ∈ *ord-iso*(*A*,*r*,*B*,*s*); *a* ∈ *A*⟧ ⟹ *f* ‘‘ *pred*(*A*,*a*,*r*) = *pred*(*B*, *f*‘*a*, *s*)
⟨*proof*⟩

lemma *ord-iso-restrict-image*:
⟦*f* ∈ *ord-iso*(*A*,*r*,*B*,*s*); *C*<=*A*⟧
⟹ *restrict*(*f*,*C*) ∈ *ord-iso*(*C*, *r*, *f*‘‘*C*, *s*)
⟨*proof*⟩

lemma *ord-iso-restrict-pred*:
⟦*f* ∈ *ord-iso*(*A*,*r*,*B*,*s*); *a* ∈ *A*⟧
⟹ *restrict*(*f*, *pred*(*A*,*a*,*r*)) ∈ *ord-iso*(*pred*(*A*,*a*,*r*), *r*, *pred*(*B*, *f*‘*a*, *s*), *s*)
⟨*proof*⟩

lemma *well-ord-iso-preserving*:
⟦*well-ord*(*A*,*r*); *well-ord*(*B*,*s*); ⟨*a*,*c*⟩: *r*;
f ∈ *ord-iso*(*pred*(*A*,*a*,*r*), *r*, *pred*(*B*,*b*,*s*), *s*);
g ∈ *ord-iso*(*pred*(*A*,*c*,*r*), *r*, *pred*(*B*,*d*,*s*), *s*);
a ∈ *A*; *c* ∈ *A*; *b* ∈ *B*; *d* ∈ *B*⟧ ⟹ ⟨*b*,*d*⟩: *s*
⟨*proof*⟩

lemma *well-ord-iso-unique-lemma*:
⟦*well-ord*(*A*,*r*);
f ∈ *ord-iso*(*A*,*r*, *B*,*s*); *g* ∈ *ord-iso*(*A*,*r*, *B*,*s*); *y* ∈ *A*⟧
⟹ ¬ <*g*‘*y*, *f*‘*y*> ∈ *s*
⟨*proof*⟩

lemma *well-ord-iso-unique*: ⟦*well-ord*(*A*,*r*);
f ∈ *ord-iso*(*A*,*r*, *B*,*s*); *g* ∈ *ord-iso*(*A*,*r*, *B*,*s*)⟧ ⟹ *f* = *g*
⟨*proof*⟩

19.6 Towards Kunen's Theorem 6.3: Linearity of the Similarity Relation

lemma *ord-iso-map-subset*: *ord-iso-map*(*A*,*r*,*B*,*s*) ⊆ *A*∗*B*
⟨*proof*⟩

lemma *domain-ord-iso-map*: *domain*(*ord-iso-map*(*A*,*r*,*B*,*s*)) ⊆ *A*
⟨*proof*⟩

lemma *range-ord-iso-map*: *range*(*ord-iso-map*(*A*,*r*,*B*,*s*)) ⊆ *B*
⟨*proof*⟩

lemma *converse-ord-iso-map*:
converse(*ord-iso-map*(*A*,*r*,*B*,*s*)) = *ord-iso-map*(*B*,*s*,*A*,*r*)
⟨*proof*⟩

lemma *function-ord-iso-map*:
well-ord(*B*,*s*) ⟹ *function*(*ord-iso-map*(*A*,*r*,*B*,*s*))
⟨*proof*⟩

lemma *ord-iso-map-fun*: *well-ord*(*B*,*s*) ⟹ *ord-iso-map*(*A*,*r*,*B*,*s*)
∈ *domain*(*ord-iso-map*(*A*,*r*,*B*,*s*)) −> *range*(*ord-iso-map*(*A*,*r*,*B*,*s*))
⟨*proof*⟩

lemma *ord-iso-map-mono-map*:
⟦*well-ord*(*A*,*r*); *well-ord*(*B*,*s*)⟧
⟹ *ord-iso-map*(*A*,*r*,*B*,*s*)
∈ *mono-map*(*domain*(*ord-iso-map*(*A*,*r*,*B*,*s*)), *r*,

$range(ord\text{-}iso\text{-}map(A,r,B,s)),\ s)$
⟨*proof*⟩

lemma *ord-iso-map-ord-iso*:
$[\![well\text{-}ord(A,r);\ \ well\text{-}ord(B,s)]\!] \Longrightarrow ord\text{-}iso\text{-}map(A,r,B,s)$
$\in ord\text{-}iso(domain(ord\text{-}iso\text{-}map(A,r,B,s)),\ r,$
$range(ord\text{-}iso\text{-}map(A,r,B,s)),\ s)$
⟨*proof*⟩

lemma *domain-ord-iso-map-subset*:
$[\![well\text{-}ord(A,r);\ \ well\text{-}ord(B,s);$
$a \in A;\ \ a \notin domain(ord\text{-}iso\text{-}map(A,r,B,s))]\!]$
$\Longrightarrow\ \ domain(ord\text{-}iso\text{-}map(A,r,B,s)) \subseteq pred(A,\ a,\ r)$
⟨*proof*⟩

lemma *domain-ord-iso-map-cases*:
$[\![well\text{-}ord(A,r);\ \ well\text{-}ord(B,s)]\!]$
$\Longrightarrow domain(ord\text{-}iso\text{-}map(A,r,B,s)) = A\ |$
$(\exists\, x{\in}A.\ domain(ord\text{-}iso\text{-}map(A,r,B,s)) = pred(A,x,r))$
⟨*proof*⟩

lemma *range-ord-iso-map-cases*:
$[\![well\text{-}ord(A,r);\ \ well\text{-}ord(B,s)]\!]$
$\Longrightarrow range(ord\text{-}iso\text{-}map(A,r,B,s)) = B\ |$
$(\exists\, y{\in}B.\ range(ord\text{-}iso\text{-}map(A,r,B,s)) = pred(B,y,s))$
⟨*proof*⟩

Kunen's Theorem 6.3: Fundamental Theorem for Well-Ordered Sets

theorem *well-ord-trichotomy*:
$[\![well\text{-}ord(A,r);\ \ well\text{-}ord(B,s)]\!]$
$\Longrightarrow ord\text{-}iso\text{-}map(A,r,B,s) \in ord\text{-}iso(A,\ r,\ B,\ s)\ |$
$(\exists\, x{\in}A.\ ord\text{-}iso\text{-}map(A,r,B,s) \in ord\text{-}iso(pred(A,x,r),\ r,\ B,\ s))\ |$
$(\exists\, y{\in}B.\ ord\text{-}iso\text{-}map(A,r,B,s) \in ord\text{-}iso(A,\ r,\ pred(B,y,s),\ s))$
⟨*proof*⟩

19.7 Miscellaneous Results by Krzysztof Grabczewski

lemma *irrefl-converse*: $irrefl(A,r) \Longrightarrow irrefl(A,converse(r))$
⟨*proof*⟩

lemma *trans-on-converse*: $trans[A](r) \Longrightarrow trans[A](converse(r))$
⟨*proof*⟩

lemma *part-ord-converse*: $part\text{-}ord(A,r) \Longrightarrow part\text{-}ord(A,converse(r))$
⟨*proof*⟩

lemma *linear-converse*: $linear(A,r) \Longrightarrow linear(A,converse(r))$
⟨*proof*⟩

lemma *tot-ord-converse*: $tot\text{-}ord(A,r) \Longrightarrow tot\text{-}ord(A,converse(r))$
⟨*proof*⟩

lemma *first-is-elem*: $first(b,B,r) \Longrightarrow b \in B$
⟨*proof*⟩

lemma *well-ord-imp-ex1-first*:
$[\![well\text{-}ord(A,r);\ B<=A;\ B\neq 0]\!] \Longrightarrow (\exists!b.\ first(b,B,r))$
⟨*proof*⟩

lemma *the-first-in*:
$[\![well\text{-}ord(A,r);\ B<=A;\ B\neq 0]\!] \Longrightarrow (THE\ b.\ first(b,B,r)) \in B$
⟨*proof*⟩

19.8 Lemmas for the Reflexive Orders

lemma *subset-vimage-vimage-iff*:
$[\![Preorder(r);\ A \subseteq field(r);\ B \subseteq field(r)]\!] \Longrightarrow$
$r -`` A \subseteq r -`` B \longleftrightarrow (\forall a\in A.\ \exists b\in B.\ \langle a,\ b\rangle \in r)$
⟨*proof*⟩

lemma *subset-vimage1-vimage1-iff*:
$[\![Preorder(r);\ a \in field(r);\ b \in field(r)]\!] \Longrightarrow$
$r -`` \{a\} \subseteq r -`` \{b\} \longleftrightarrow \langle a,\ b\rangle \in r$
⟨*proof*⟩

lemma *Refl-antisym-eq-Image1-Image1-iff*:
$[\![refl(field(r),\ r);\ antisym(r);\ a \in field(r);\ b \in field(r)]\!] \Longrightarrow$
$r `` \{a\} = r `` \{b\} \longleftrightarrow a = b$
⟨*proof*⟩

lemma *Partial-order-eq-Image1-Image1-iff*:
$[\![Partial\text{-}order(r);\ a \in field(r);\ b \in field(r)]\!] \Longrightarrow$
$r `` \{a\} = r `` \{b\} \longleftrightarrow a = b$
⟨*proof*⟩

lemma *Refl-antisym-eq-vimage1-vimage1-iff*:
$[\![refl(field(r),\ r);\ antisym(r);\ a \in field(r);\ b \in field(r)]\!] \Longrightarrow$
$r -`` \{a\} = r -`` \{b\} \longleftrightarrow a = b$
⟨*proof*⟩

lemma *Partial-order-eq-vimage1-vimage1-iff*:

$[\![Partial\text{-}order(r);\ a \in field(r);\ b \in field(r)]\!] \Longrightarrow$
$r -`` \{a\} = r -`` \{b\} \longleftrightarrow a = b$
⟨*proof*⟩

end

20 Combining Orderings: Foundations of Ordinal Arithmetic

theory *OrderArith* **imports** *Order Sum Ordinal* **begin**

definition

$radd\ :: [i,i,i,i] \Rightarrow i$ **where**
$radd(A,r,B,s) \equiv$
$\{z: (A+B) * (A+B).$
$(\exists x\ y.\ z = \langle Inl(x),\ Inr(y)\rangle)\ \ |$
$(\exists x'\ x.\ z = \langle Inl(x'),\ Inl(x)\rangle \land \langle x',x\rangle{:}r)\ \ |$
$(\exists y'\ y.\ z = \langle Inr(y'),\ Inr(y)\rangle \land \langle y',y\rangle{:}s)\}$

definition

$rmult\ :: [i,i,i,i] \Rightarrow i$ **where**
$rmult(A,r,B,s) \equiv$
$\{z: (A*B) * (A*B).$
$\exists x'\ y'\ x\ y.\ z = \langle\langle x',y'\rangle,\ \langle x,y\rangle\rangle \land$
$(\langle x',x\rangle{:}\ r \mid (x'=x \land \langle y',y\rangle{:}\ s))\}$

definition

$rvimage :: [i,i,i] \Rightarrow i$ **where**
$rvimage(A,f,r) \equiv \{z \in A*A.\ \exists x\ y.\ z = \langle x,y\rangle \land \langle f`x,f`y\rangle{:}\ r\}$

definition
$measure :: [i,\ i \Rightarrow i] \Rightarrow i$ **where**
$measure(A,f) \equiv \{\langle x,y\rangle{:}\ A*A.\ f(x) < f(y)\}$

20.1 Addition of Relations – Disjoint Sum

20.1.1 Rewrite rules. Can be used to obtain introduction rules

lemma *radd-Inl-Inr-iff* [*iff*]:
$\langle Inl(a),\ Inr(b)\rangle \in radd(A,r,B,s) \longleftrightarrow a \in A \land b \in B$
⟨*proof*⟩

lemma *radd-Inl-iff* [*iff*]:
$\langle Inl(a'),\ Inl(a)\rangle \in radd(A,r,B,s) \longleftrightarrow a'{:}A \land a \in A \land \langle a',a\rangle{:}r$
⟨*proof*⟩

lemma *radd-Inr-iff* [*iff*]:
$\langle Inr(b'),\ Inr(b)\rangle \in radd(A,r,B,s) \longleftrightarrow\ b':B \wedge b \in B \wedge \langle b',b\rangle:s$
⟨*proof*⟩

lemma *radd-Inr-Inl-iff* [*simp*]:
$\langle Inr(b),\ Inl(a)\rangle \in radd(A,r,B,s) \longleftrightarrow False$
⟨*proof*⟩

declare *radd-Inr-Inl-iff* [*THEN iffD1*, *dest*!]

20.1.2 Elimination Rule

lemma *raddE*:
$[\![\langle p',p\rangle \in radd(A,r,B,s);$
$\bigwedge x\ y.\ [\![p'=Inl(x);\ x \in A;\ p=Inr(y);\ y \in B]\!] \Longrightarrow Q;$
$\bigwedge x'\ x.\ [\![p'=Inl(x');\ p=Inl(x);\ \langle x',x\rangle:\ r;\ x':A;\ x \in A]\!] \Longrightarrow Q;$
$\bigwedge y'\ y.\ [\![p'=Inr(y');\ p=Inr(y);\ \langle y',y\rangle:\ s;\ y':B;\ y \in B]\!] \Longrightarrow Q$
$]\!] \Longrightarrow Q$
⟨*proof*⟩

20.1.3 Type checking

lemma *radd-type*: $radd(A,r,B,s) \subseteq (A+B) * (A+B)$
⟨*proof*⟩

lemmas *field-radd* = *radd-type* [*THEN field-rel-subset*]

20.1.4 Linearity

lemma *linear-radd*:
$[\![linear(A,r);\ \ linear(B,s)]\!] \Longrightarrow linear(A+B,radd(A,r,B,s))$
⟨*proof*⟩

20.1.5 Well-foundedness

lemma *wf-on-radd*: $[\![wf[A](r);\ \ wf[B](s)]\!] \Longrightarrow wf[A+B](radd(A,r,B,s))$
⟨*proof*⟩

lemma *wf-radd*: $[\![wf(r);\ \ wf(s)]\!] \Longrightarrow wf(radd(field(r),r,field(s),s))$
⟨*proof*⟩

lemma *well-ord-radd*:
$[\![well\text{-}ord(A,r);\ \ well\text{-}ord(B,s)]\!] \Longrightarrow well\text{-}ord(A+B,\ radd(A,r,B,s))$
⟨*proof*⟩

20.1.6 An *ord-iso* congruence law

lemma *sum-bij*:
$[\![f \in bij(A,C);\ \ g \in bij(B,D)]\!]$
$\Longrightarrow (\lambda z{\in}A+B.\ case(\lambda x.\ Inl(f`x),\ \lambda y.\ Inr(g`y),\ z)) \in bij(A+B,\ C+D)$

⟨*proof*⟩

lemma *sum-ord-iso-cong*:
$[\![f \in \textit{ord-iso}(A,r,A',r');\ \ g \in \textit{ord-iso}(B,s,B',s')]\!] \Longrightarrow$
$(\lambda z{\in}A{+}B.\ \textit{case}(\lambda x.\ \textit{Inl}(f`x),\ \lambda y.\ \textit{Inr}(g`y),\ z))$
$\in \textit{ord-iso}(A{+}B,\ \textit{radd}(A,r,B,s),\ A'{+}B',\ \textit{radd}(A',r',B',s'))$
⟨*proof*⟩

lemma *sum-disjoint-bij*: $A \cap B = 0 \Longrightarrow$
$(\lambda z{\in}A{+}B.\ \textit{case}(\lambda x.\ x,\ \lambda y.\ y,\ z)) \in \textit{bij}(A{+}B,\ A \cup B)$
⟨*proof*⟩

20.1.7 Associativity

lemma *sum-assoc-bij*:
$(\lambda z{\in}(A{+}B){+}C.\ \textit{case}(\textit{case}(\textit{Inl},\ \lambda y.\ \textit{Inr}(\textit{Inl}(y))),\ \lambda y.\ \textit{Inr}(\textit{Inr}(y)),\ z))$
$\in \textit{bij}((A{+}B){+}C,\ A{+}(B{+}C))$
⟨*proof*⟩

lemma *sum-assoc-ord-iso*:
$(\lambda z{\in}(A{+}B){+}C.\ \textit{case}(\textit{case}(\textit{Inl},\ \lambda y.\ \textit{Inr}(\textit{Inl}(y))),\ \lambda y.\ \textit{Inr}(\textit{Inr}(y)),\ z))$
$\in \textit{ord-iso}((A{+}B){+}C,\ \textit{radd}(A{+}B,\ \textit{radd}(A,r,B,s),\ C,\ t),$
$A{+}(B{+}C),\ \textit{radd}(A,\ r,\ B{+}C,\ \textit{radd}(B,s,C,t)))$
⟨*proof*⟩

20.2 Multiplication of Relations – Lexicographic Product

20.2.1 Rewrite rule. Can be used to obtain introduction rules

lemma *rmult-iff* [*iff*]:
$\langle\langle a',b'\rangle,\ \langle a,b\rangle\rangle \in \textit{rmult}(A,r,B,s) \longleftrightarrow$
$(\langle a',a\rangle{:}\ r\ \wedge\ a'{:}A \wedge a \in A \wedge b'{:}\ B \wedge b \in B)\ |$
$(\langle b',b\rangle{:}\ s\ \wedge\ a'{=}a \wedge a \in A \wedge b'{:}\ B \wedge b \in B)$

⟨*proof*⟩

lemma *rmultE*:
$[\![\langle\langle a',b'\rangle,\ \langle a,b\rangle\rangle \in \textit{rmult}(A,r,B,s);$
$[\![\langle a',a\rangle{:}\ r;\ \ a'{:}A;\ \ a \in A;\ \ b'{:}B;\ \ b \in B]\!] \Longrightarrow Q;$
$[\![\langle b',b\rangle{:}\ s;\ \ a \in A;\ \ a'{=}a;\ \ b'{:}B;\ \ b \in B]\!] \Longrightarrow Q$
$]\!] \Longrightarrow Q$
⟨*proof*⟩

20.2.2 Type checking

lemma *rmult-type*: $\textit{rmult}(A,r,B,s) \subseteq (A{*}B) * (A{*}B)$
⟨*proof*⟩

lemmas *field-rmult* = *rmult-type* [*THEN field-rel-subset*]

20.2.3 Linearity

lemma *linear-rmult*:
$[\![linear(A,r);\ \ linear(B,s)]\!] \Longrightarrow linear(A{*}B,rmult(A,r,B,s))$
$\langle proof \rangle$

20.2.4 Well-foundedness

lemma *wf-on-rmult*: $[\![wf[A](r);\ \ wf[B](s)]\!] \Longrightarrow wf[A{*}B](rmult(A,r,B,s))$
$\langle proof \rangle$

lemma *wf-rmult*: $[\![wf(r);\ \ wf(s)]\!] \Longrightarrow wf(rmult(field(r),r,field(s),s))$
$\langle proof \rangle$

lemma *well-ord-rmult*:
$[\![well\text{-}ord(A,r);\ \ well\text{-}ord(B,s)]\!] \Longrightarrow well\text{-}ord(A{*}B,\ rmult(A,r,B,s))$
$\langle proof \rangle$

20.2.5 An *ord-iso* congruence law

lemma *prod-bij*:
$[\![f \in bij(A,C);\ \ g \in bij(B,D)]\!]$
$\Longrightarrow (lam\ \langle x,y\rangle{:}A{*}B.\ \langle f`x,\ g`y\rangle) \in bij(A{*}B,\ C{*}D)$
$\langle proof \rangle$

lemma *prod-ord-iso-cong*:
$[\![f \in ord\text{-}iso(A,r,A',r');\ \ g \in ord\text{-}iso(B,s,B',s')]\!]$
$\Longrightarrow (lam\ \langle x,y\rangle{:}A{*}B.\ \langle f`x,\ g`y\rangle)$
$\in ord\text{-}iso(A{*}B,\ rmult(A,r,B,s),\ A'{*}B',\ rmult(A',r',B',s'))$
$\langle proof \rangle$

lemma *singleton-prod-bij*: $(\lambda z{\in}A.\ \langle x,z\rangle) \in bij(A,\ \{x\}{*}A)$
$\langle proof \rangle$

lemma *singleton-prod-ord-iso*:
$well\text{-}ord(\{x\},xr) \Longrightarrow$
$(\lambda z{\in}A.\ \langle x,z\rangle) \in ord\text{-}iso(A,\ r,\ \{x\}{*}A,\ rmult(\{x\},\ xr,\ A,\ r))$
$\langle proof \rangle$

lemma *prod-sum-singleton-bij*:
$a{\notin}C \Longrightarrow$
$(\lambda x{\in}C{*}B + D.\ case(\lambda x.\ x,\ \lambda y.\langle a,y\rangle,\ x))$
$\in bij(C{*}B + D,\ C{*}B \cup \{a\}{*}D)$
$\langle proof \rangle$

lemma *prod-sum-singleton-ord-iso*:
$[\![a \in A;\ \ well\text{-}ord(A,r)]\!] \Longrightarrow$

$(\lambda x{\in}pred(A,a,r){*}B + pred(B,b,s).\ case(\lambda x.\ x,\ \lambda y.\langle a,y\rangle,\ x))$
$\in ord\text{-}iso(pred(A,a,r){*}B + pred(B,b,s),$
$radd(A{*}B,\ rmult(A,r,B,s),\ B,\ s),$
$pred(A,a,r){*}B \cup \{a\}{*}pred(B,b,s),\ rmult(A,r,B,s))$
⟨*proof*⟩

20.2.6 Distributive law

lemma *sum-prod-distrib-bij*:
$(lam\ \langle x,z\rangle{:}(A{+}B){*}C.\ case(\lambda y.\ Inl(\langle y,z\rangle),\ \lambda y.\ Inr(\langle y,z\rangle),\ x))$
$\in bij((A{+}B){*}C,\ (A{*}C){+}(B{*}C))$
⟨*proof*⟩

lemma *sum-prod-distrib-ord-iso*:
$(lam\ \langle x,z\rangle{:}(A{+}B){*}C.\ case(\lambda y.\ Inl(\langle y,z\rangle),\ \lambda y.\ Inr(\langle y,z\rangle),\ x))$
$\in ord\text{-}iso((A{+}B){*}C,\ rmult(A{+}B,\ radd(A,r,B,s),\ C,\ t),$
$(A{*}C){+}(B{*}C),\ radd(A{*}C,\ rmult(A,r,C,t),\ B{*}C,\ rmult(B,s,C,t)))$
⟨*proof*⟩

20.2.7 Associativity

lemma *prod-assoc-bij*:
$(lam\ \langle\langle x,y\rangle,\ z\rangle{:}(A{*}B){*}C.\ \langle x,\langle y,z\rangle\rangle) \in bij((A{*}B){*}C,\ A{*}(B{*}C))$
⟨*proof*⟩

lemma *prod-assoc-ord-iso*:
$(lam\ \langle\langle x,y\rangle,\ z\rangle{:}(A{*}B){*}C.\ \langle x,\langle y,z\rangle\rangle)$
$\in ord\text{-}iso((A{*}B){*}C,\ rmult(A{*}B,\ rmult(A,r,B,s),\ C,\ t),$
$A{*}(B{*}C),\ rmult(A,\ r,\ B{*}C,\ rmult(B,s,C,t)))$
⟨*proof*⟩

20.3 Inverse Image of a Relation

20.3.1 Rewrite rule

lemma *rvimage-iff*: $\langle a,b\rangle \in rvimage(A,f,r) \longleftrightarrow \langle f`a,f`b\rangle{:}\ r \land a \in A \land b \in A$
⟨*proof*⟩

20.3.2 Type checking

lemma *rvimage-type*: $rvimage(A,f,r) \subseteq A{*}A$
⟨*proof*⟩

lemmas *field-rvimage* = *rvimage-type* [*THEN field-rel-subset*]

lemma *rvimage-converse*: $rvimage(A,f,\ converse(r)) = converse(rvimage(A,f,r))$
⟨*proof*⟩

20.3.3 Partial Ordering Properties

lemma *irrefl-rvimage*:

$[\![f \in inj(A,B);\ \ irrefl(B,r)]\!] \Longrightarrow irrefl(A,\ rvimage(A,f,r))$
⟨*proof*⟩

lemma *trans-on-rvimage*:
$[\![f \in inj(A,B);\ \ trans[B](r)]\!] \Longrightarrow trans[A](rvimage(A,f,r))$
⟨*proof*⟩

lemma *part-ord-rvimage*:
$[\![f \in inj(A,B);\ \ part\text{-}ord(B,r)]\!] \Longrightarrow part\text{-}ord(A,\ rvimage(A,f,r))$
⟨*proof*⟩

20.3.4 Linearity

lemma *linear-rvimage*:
$[\![f \in inj(A,B);\ \ linear(B,r)]\!] \Longrightarrow linear(A,rvimage(A,f,r))$
⟨*proof*⟩

lemma *tot-ord-rvimage*:
$[\![f \in inj(A,B);\ \ tot\text{-}ord(B,r)]\!] \Longrightarrow tot\text{-}ord(A,\ rvimage(A,f,r))$
⟨*proof*⟩

20.3.5 Well-foundedness

lemma *wf-rvimage* [*intro*!]: $wf(r) \Longrightarrow wf(rvimage(A,f,r))$
⟨*proof*⟩

But note that the combination of *wf-imp-wf-on* and *wf-rvimage* gives $wf(r) \Longrightarrow wf[C](rvimage(A,\ f,\ r))$

lemma *wf-on-rvimage*: $[\![f \in A \rightarrow B;\ \ wf[B](r)]\!] \Longrightarrow wf[A](rvimage(A,f,r))$
⟨*proof*⟩

lemma *well-ord-rvimage*:
$[\![f \in inj(A,B);\ \ well\text{-}ord(B,r)]\!] \Longrightarrow well\text{-}ord(A,\ rvimage(A,f,r))$
⟨*proof*⟩

lemma *ord-iso-rvimage*:
$f \in bij(A,B) \Longrightarrow f \in ord\text{-}iso(A,\ rvimage(A,f,s),\ B,\ s)$
⟨*proof*⟩

lemma *ord-iso-rvimage-eq*:
$f \in ord\text{-}iso(A,r,\ B,s) \Longrightarrow rvimage(A,f,s) = r \cap A*A$
⟨*proof*⟩

20.4 Every well-founded relation is a subset of some inverse image of an ordinal

lemma *wf-rvimage-Ord*: $Ord(i) \Longrightarrow wf(rvimage(A,\ f,\ Memrel(i)))$
⟨*proof*⟩

definition
  wfrank :: $[i,i] \Rightarrow i$ **where**
    $wfrank(r,a) \equiv wfrec(r,\ a,\ \lambda x\ f.\ \bigcup y \in r-``\{x\}.\ succ(f`y))$

definition
  wftype :: $i \Rightarrow i$ **where**
    $wftype(r) \equiv \bigcup y \in range(r).\ succ(wfrank(r,y))$

lemma *wfrank*: $wf(r) \Longrightarrow wfrank(r,a) = (\bigcup y \in r-``\{a\}.\ succ(wfrank(r,y)))$
⟨*proof*⟩

lemma *Ord-wfrank*: $wf(r) \Longrightarrow Ord(wfrank(r,a))$
⟨*proof*⟩

lemma *wfrank-lt*: $[\![wf(r);\ \langle a,b\rangle \in r]\!] \Longrightarrow wfrank(r,a) < wfrank(r,b)$
⟨*proof*⟩

lemma *Ord-wftype*: $wf(r) \Longrightarrow Ord(wftype(r))$
⟨*proof*⟩

lemma *wftypeI*: $[\![wf(r);\ \ x \in field(r)]\!] \Longrightarrow wfrank(r,x) \in wftype(r)$
⟨*proof*⟩

lemma *wf-imp-subset-rvimage*:
    $[\![wf(r);\ r \subseteq A*A]\!] \Longrightarrow \exists\, i\ f.\ Ord(i) \wedge r \subseteq rvimage(A,\ f,\ Memrel(i))$
⟨*proof*⟩

theorem *wf-iff-subset-rvimage*:
  $relation(r) \Longrightarrow wf(r) \longleftrightarrow (\exists\, i\ f\ A.\ Ord(i) \wedge r \subseteq rvimage(A,\ f,\ Memrel(i)))$
⟨*proof*⟩

20.5 Other Results

lemma *wf-times*: $A \cap B = 0 \Longrightarrow wf(A*B)$
⟨*proof*⟩

Could also be used to prove *wf-radd*

lemma *wf-Un*:
    $[\![range(r) \cap domain(s) = 0;\ wf(r);\ \ wf(s)]\!] \Longrightarrow wf(r \cup s)$
⟨*proof*⟩

20.5.1 The Empty Relation

lemma *wf0*: $wf(0)$
⟨*proof*⟩

lemma *linear0*: $linear(0,0)$
⟨*proof*⟩

lemma *well-ord0*: $well\text{-}ord(0,0)$
⟨*proof*⟩

20.5.2 The "measure" relation is useful with wfrec

lemma *measure-eq-rvimage-Memrel*:
$measure(A,f) = rvimage(A,Lambda(A,f),Memrel(Collect(RepFun(A,f),Ord)))$
⟨*proof*⟩

lemma *wf-measure* [*iff*]: $wf(measure(A,f))$
⟨*proof*⟩

lemma *measure-iff* [*iff*]: $\langle x,y\rangle \in measure(A,f) \longleftrightarrow x \in A \wedge y \in A \wedge f(x)<f(y)$
⟨*proof*⟩

lemma *linear-measure*:
assumes *Ordf*: $\bigwedge x.\ x \in A \Longrightarrow Ord(f(x))$
and *inj*: $\bigwedge x\ y.\ [\![x \in A;\ y \in A;\ f(x) = f(y)]\!] \Longrightarrow x=y$
shows $linear(A,\ measure(A,f))$
⟨*proof*⟩

lemma *wf-on-measure*: $wf[B](measure(A,f))$
⟨*proof*⟩

lemma *well-ord-measure*:
assumes *Ordf*: $\bigwedge x.\ x \in A \Longrightarrow Ord(f(x))$
and *inj*: $\bigwedge x\ y.\ [\![x \in A;\ y \in A;\ f(x) = f(y)]\!] \Longrightarrow x=y$
shows $well\text{-}ord(A,\ measure(A,f))$
⟨*proof*⟩

lemma *measure-type*: $measure(A,f) \subseteq A*A$
⟨*proof*⟩

20.5.3 Well-foundedness of Unions

lemma *wf-on-Union*:
assumes *wfA*: $wf[A](r)$
and *wfB*: $\bigwedge a.\ a{\in}A \Longrightarrow wf[B(a)](s)$
and *ok*: $\bigwedge a\ u\ v.\ [\![\langle u,v\rangle \in s;\ v \in B(a);\ a \in A]\!]$
$\Longrightarrow (\exists\, a'{\in}A.\ \langle a',a\rangle \in r \wedge u \in B(a')) \mid u \in B(a)$
shows $wf[\bigcup a{\in}A.\ B(a)](s)$
⟨*proof*⟩

20.5.4 Bijections involving Powersets

lemma *Pow-sum-bij*:
$(\lambda Z \in Pow(A+B).\ \langle\{x \in A.\ Inl(x) \in Z\},\ \{y \in B.\ Inr(y) \in Z\}\rangle)$

$\in$ *bij*(*Pow*(*A*+*B*), *Pow*(*A*)∗*Pow*(*B*))
⟨*proof*⟩

As a special case, we have *bij*(*Pow*(*A* × *B*), *A* → *Pow*(*B*))

lemma *Pow-Sigma-bij*:
(λ*r* ∈ *Pow*(*Sigma*(*A*,*B*)). λ*x* ∈ *A*. *r*‘‘{*x*})
∈ *bij*(*Pow*(*Sigma*(*A*,*B*)), ∏ *x* ∈ *A*. *Pow*(*B*(*x*)))
⟨*proof*⟩

end

21 Order Types and Ordinal Arithmetic

theory *OrderType* **imports** *OrderArith OrdQuant Nat* **begin**

The order type of a well-ordering is the least ordinal isomorphic to it. Ordinal arithmetic is traditionally defined in terms of order types, as it is here. But a definition by transfinite recursion would be much simpler!

definition
ordermap :: [*i*,*i*]⇒*i* **where**
ordermap(*A*,*r*) ≡ λ*x*∈*A*. *wfrec*[*A*](*r*, *x*, λ*x f*. *f* ‘‘ *pred*(*A*,*x*,*r*))

definition
ordertype :: [*i*,*i*]⇒*i* **where**
ordertype(*A*,*r*) ≡ *ordermap*(*A*,*r*)‘‘*A*

definition

Ord-alt :: *i* ⇒ *o* **where**
Ord-alt(*X*) ≡ *well-ord*(*X*, *Memrel*(*X*)) ∧ (∀ *u*∈*X*. *u*=*pred*(*X*, *u*, *Memrel*(*X*)))

definition

ordify :: *i*⇒*i* **where**
ordify(*x*) ≡ *if Ord*(*x*) *then x else 0*

definition

omult :: [*i*,*i*]⇒*i* (**infixl** ‹∗∗› *70*) **where**
i ∗∗ *j* ≡ *ordertype*(*j*∗*i*, *rmult*(*j*,*Memrel*(*j*),*i*,*Memrel*(*i*)))

definition

raw-oadd :: [*i*,*i*]⇒*i* **where**
raw-oadd(*i*,*j*) ≡ *ordertype*(*i*+*j*, *radd*(*i*,*Memrel*(*i*),*j*,*Memrel*(*j*)))

definition
oadd :: [*i*,*i*]⇒*i* (**infixl** ‹++› *65*) **where**

$i \mathbin{++} j \equiv raw\text{-}oadd(ordify(i),ordify(j))$

definition

$odiff$ :: $[i,i]\Rightarrow i$ (**infixl** ‹−−› *65*) **where**
$i -- j \equiv ordertype(i-j,\ Memrel(i))$

21.1 Proofs needing the combination of Ordinal.thy and Order.thy

lemma *le-well-ord-Memrel*: $j \le i \Longrightarrow well\text{-}ord(j,\ Memrel(i))$
⟨*proof*⟩

lemmas *well-ord-Memrel* = *le-refl* [*THEN le-well-ord-Memrel*]

lemma *lt-pred-Memrel*:
$j<i \Longrightarrow pred(i,\ j,\ Memrel(i)) = j$
⟨*proof*⟩

lemma *pred-Memrel*:
$x \in A \Longrightarrow pred(A,\ x,\ Memrel(A)) = A \cap x$
⟨*proof*⟩

lemma *Ord-iso-implies-eq-lemma*:
$[\![j<i;\ f \in ord\text{-}iso(i,Memrel(i),j,Memrel(j))]\!] \Longrightarrow R$
⟨*proof*⟩

lemma *Ord-iso-implies-eq*:
$[\![Ord(i);\ Ord(j);\ f \in ord\text{-}iso(i,Memrel(i),j,Memrel(j))]\!]$
$\Longrightarrow i=j$
⟨*proof*⟩

21.2 Ordermap and ordertype

lemma *ordermap-type*:
$ordermap(A,r) \in A \mathbin{->} ordertype(A,r)$
⟨*proof*⟩

21.2.1 Unfolding of ordermap

lemma *ordermap-eq-image*:
$[\![wf[A](r);\ x \in A]\!]$
$\Longrightarrow ordermap(A,r)\ \text{‘}\ x = ordermap(A,r)\ \text{‘‘}\ pred(A,x,r)$
⟨*proof*⟩

lemma *ordermap-pred-unfold*:

$[\![wf[A](r);\ \ x \in A]\!]$
$\implies ordermap(A,r)\ `\ x = \{ordermap(A,r)`y\ .\ y \in pred(A,x,r)\}$
⟨proof⟩

lemmas *ordermap-unfold* = *ordermap-pred-unfold* [*simplified pred-def*]

21.2.2 Showing that ordermap, ordertype yield ordinals

lemma *Ord-ordermap*:
$[\![well\text{-}ord(A,r);\ \ x \in A]\!] \implies Ord(ordermap(A,r)\ `\ x)$
⟨proof⟩

lemma *Ord-ordertype*:
$well\text{-}ord(A,r) \implies Ord(ordertype(A,r))$
⟨proof⟩

21.2.3 ordermap preserves the orderings in both directions

lemma *ordermap-mono*:
$[\![\langle w,x\rangle: r;\ \ wf[A](r);\ \ w \in A;\ x \in A]\!]$
$\implies ordermap(A,r)`w \in ordermap(A,r)`x$
⟨proof⟩

lemma *converse-ordermap-mono*:
$[\![ordermap(A,r)`w \in ordermap(A,r)`x;\ \ well\text{-}ord(A,r);\ w \in A;\ x \in A]\!]$
$\implies \langle w,x\rangle: r$
⟨proof⟩

lemma *ordermap-surj*: $ordermap(A,\ r) \in surj(A,\ ordertype(A,\ r))$
⟨proof⟩

lemma *ordermap-bij*:
$well\text{-}ord(A,r) \implies ordermap(A,r) \in bij(A,\ ordertype(A,r))$
⟨proof⟩

21.2.4 Isomorphisms involving ordertype

lemma *ordertype-ord-iso*:
$well\text{-}ord(A,r)$
$\implies ordermap(A,r) \in ord\text{-}iso(A,r,\ ordertype(A,r),\ Memrel(ordertype(A,r)))$
⟨proof⟩

lemma *ordertype-eq*:
$[\![f \in ord\text{-}iso(A,r,B,s);\ \ well\text{-}ord(B,s)]\!]$
$\implies ordertype(A,r) = ordertype(B,s)$
⟨proof⟩

lemma *ordertype-eq-imp-ord-iso*:

$[\![ordertype(A,r) = ordertype(B,s);\ well\text{-}ord(A,r);\ \ well\text{-}ord(B,s)]\!]$
$\Longrightarrow \exists f.\ f \in ord\text{-}iso(A,r,B,s)$
⟨*proof*⟩

21.2.5 Basic equalities for ordertype

lemma *le-ordertype-Memrel*: $j \le i \Longrightarrow ordertype(j,Memrel(i)) = j$
⟨*proof*⟩

lemmas *ordertype-Memrel* = *le-refl* [*THEN le-ordertype-Memrel*]

lemma *ordertype-0* [*simp*]: $ordertype(0,r) = 0$
⟨*proof*⟩

lemmas *bij-ordertype-vimage* = *ord-iso-rvimage* [*THEN ordertype-eq*]

21.2.6 A fundamental unfolding law for ordertype.

lemma *ordermap-pred-eq-ordermap*:
$[\![well\text{-}ord(A,r);\ \ y \in A;\ \ z \in pred(A,y,r)]\!]$
$\Longrightarrow ordermap(pred(A,y,r),\ r)\ `\ z = ordermap(A,\ r)\ `\ z$
⟨*proof*⟩

lemma *ordertype-unfold*:
$ordertype(A,r) = \{ordermap(A,r)`y\ .\ y \in A\}$
⟨*proof*⟩

Theorems by Krzysztof Grabczewski; proofs simplified by lcp

lemma *ordertype-pred-subset*: $[\![well\text{-}ord(A,r);\ \ x \in A]\!] \Longrightarrow$
$ordertype(pred(A,x,r),r) \subseteq ordertype(A,r)$
⟨*proof*⟩

lemma *ordertype-pred-lt*:
$[\![well\text{-}ord(A,r);\ \ x \in A]\!]$
$\Longrightarrow ordertype(pred(A,x,r),r) < ordertype(A,r)$
⟨*proof*⟩

lemma *ordertype-pred-unfold*:
$well\text{-}ord(A,r)$
$\Longrightarrow ordertype(A,r) = \{ordertype(pred(A,x,r),r).\ x \in A\}$
⟨*proof*⟩

21.3 Alternative definition of ordinal

lemma *Ord-is-Ord-alt*: $Ord(i) \Longrightarrow Ord\text{-}alt(i)$
⟨*proof*⟩

lemma *Ord-alt-is-Ord*:
  $\mathit{Ord\text{-}alt}(i) \Longrightarrow \mathit{Ord}(i)$
⟨*proof*⟩

21.4 Ordinal Addition

21.4.1 Order Type calculations for radd

Addition with 0

lemma *bij-sum-0*: $(\lambda z \in A+0.\ \mathit{case}(\lambda x.\ x,\ \lambda y.\ y,\ z)) \in \mathit{bij}(A+0,\ A)$
⟨*proof*⟩

lemma *ordertype-sum-0-eq*:
  $\mathit{well\text{-}ord}(A,r) \Longrightarrow \mathit{ordertype}(A+0,\ \mathit{radd}(A,r,0,s)) = \mathit{ordertype}(A,r)$
⟨*proof*⟩

lemma *bij-0-sum*: $(\lambda z \in 0+A.\ \mathit{case}(\lambda x.\ x,\ \lambda y.\ y,\ z)) \in \mathit{bij}(0+A,\ A)$
⟨*proof*⟩

lemma *ordertype-0-sum-eq*:
  $\mathit{well\text{-}ord}(A,r) \Longrightarrow \mathit{ordertype}(0+A,\ \mathit{radd}(0,s,A,r)) = \mathit{ordertype}(A,r)$
⟨*proof*⟩

Initial segments of radd. Statements by Grabczewski

lemma *pred-Inl-bij*:
 $a \in A \Longrightarrow (\lambda x \in \mathit{pred}(A,a,r).\ \mathit{Inl}(x))$
  $\in \mathit{bij}(\mathit{pred}(A,a,r),\ \mathit{pred}(A+B,\ \mathit{Inl}(a),\ \mathit{radd}(A,r,B,s)))$
 ⟨*proof*⟩

lemma *ordertype-pred-Inl-eq*:
  $[\![a \in A;\ \ \mathit{well\text{-}ord}(A,r)]\!]$
  $\Longrightarrow \mathit{ordertype}(\mathit{pred}(A+B,\ \mathit{Inl}(a),\ \mathit{radd}(A,r,B,s)),\ \mathit{radd}(A,r,B,s)) =$
  $\mathit{ordertype}(\mathit{pred}(A,a,r),\ r)$
⟨*proof*⟩

lemma *pred-Inr-bij*:
 $b \in B \Longrightarrow$
  $\mathit{id}(A+\mathit{pred}(B,b,s))$
  $\in \mathit{bij}(A+\mathit{pred}(B,b,s),\ \mathit{pred}(A+B,\ \mathit{Inr}(b),\ \mathit{radd}(A,r,B,s)))$
 ⟨*proof*⟩

lemma *ordertype-pred-Inr-eq*:
  $[\![b \in B;\ \ \mathit{well\text{-}ord}(A,r);\ \ \mathit{well\text{-}ord}(B,s)]\!]$
  $\Longrightarrow \mathit{ordertype}(\mathit{pred}(A+B,\ \mathit{Inr}(b),\ \mathit{radd}(A,r,B,s)),\ \mathit{radd}(A,r,B,s)) =$
  $\mathit{ordertype}(A+\mathit{pred}(B,b,s),\ \mathit{radd}(A,r,\mathit{pred}(B,b,s),s))$
⟨*proof*⟩

21.4.2 ordify: trivial coercion to an ordinal

lemma *Ord-ordify* [*iff*, *TC*]: $Ord(ordify(x))$
⟨*proof*⟩

lemma *ordify-idem* [*simp*]: $ordify(ordify(x)) = ordify(x)$
⟨*proof*⟩

21.4.3 Basic laws for ordinal addition

lemma *Ord-raw-oadd*: $[\![Ord(i);\ Ord(j)]\!] \Longrightarrow Ord(raw\text{-}oadd(i,j))$
⟨*proof*⟩

lemma *Ord-oadd* [*iff*,*TC*]: $Ord(i{+}{+}j)$
⟨*proof*⟩

Ordinal addition with zero

lemma *raw-oadd-0*: $Ord(i) \Longrightarrow raw\text{-}oadd(i,0) = i$
⟨*proof*⟩

lemma *oadd-0* [*simp*]: $Ord(i) \Longrightarrow i{+}{+}0 = i$
⟨*proof*⟩

lemma *raw-oadd-0-left*: $Ord(i) \Longrightarrow raw\text{-}oadd(0,i) = i$
⟨*proof*⟩

lemma *oadd-0-left* [*simp*]: $Ord(i) \Longrightarrow 0{+}{+}i = i$
⟨*proof*⟩

lemma *oadd-eq-if-raw-oadd*:
$i{+}{+}j = (if\ Ord(i)\ then\ (if\ Ord(j)\ then\ raw\text{-}oadd(i,j)\ else\ i)$
$else\ (if\ Ord(j)\ then\ j\ else\ 0))$
⟨*proof*⟩

lemma *raw-oadd-eq-oadd*: $[\![Ord(i);\ Ord(j)]\!] \Longrightarrow raw\text{-}oadd(i,j) = i{+}{+}j$
⟨*proof*⟩

lemma *lt-oadd1*: $k{<}i \Longrightarrow k < i{+}{+}j$
⟨*proof*⟩

lemma *oadd-le-self*: $Ord(i) \Longrightarrow i \le i{+}{+}j$
⟨*proof*⟩

Various other results

lemma *id-ord-iso-Memrel*: $A<=B \Longrightarrow id(A) \in ord\text{-}iso(A,\ Memrel(A),\ A,\ Memrel(B))$
⟨*proof*⟩

lemma *subset-ord-iso-Memrel*:
$[\![f \in ord\text{-}iso(A,Memrel(B),C,r);\ A<=B]\!] \Longrightarrow f \in ord\text{-}iso(A,Memrel(A),C,r)$
⟨*proof*⟩

lemma *restrict-ord-iso*:
$[\![f \in ord\text{-}iso(i,\ Memrel(i),\ Order.pred(A,a,r),\ r);\ \ a \in A;\ j < i;$
$trans[A](r)]\!]$
$\Longrightarrow restrict(f,j) \in ord\text{-}iso(j,\ Memrel(j),\ Order.pred(A,f`j,r),\ r)$
⟨*proof*⟩

lemma *restrict-ord-iso2*:
$[\![f \in ord\text{-}iso(Order.pred(A,a,r),\ r,\ i,\ Memrel(i));\ \ a \in A;$
$j < i;\ trans[A](r)]\!]$
$\Longrightarrow converse(restrict(converse(f),\ j))$
$\in ord\text{-}iso(Order.pred(A,\ converse(f)`j,\ r),\ r,\ j,\ Memrel(j))$
⟨*proof*⟩

lemma *ordertype-sum-Memrel*:
$[\![well\text{-}ord(A,r);\ \ k<j]\!]$
$\Longrightarrow ordertype(A+k,\ radd(A,\ r,\ k,\ Memrel(j))) =$
$ordertype(A+k,\ radd(A,\ r,\ k,\ Memrel(k)))$
⟨*proof*⟩

lemma *oadd-lt-mono2*: $k<j \Longrightarrow i{+}{+}k < i{+}{+}j$
⟨*proof*⟩

lemma *oadd-lt-cancel2*: $[\![i{+}{+}j < i{+}{+}k;\ \ Ord(j)]\!] \Longrightarrow j<k$
⟨*proof*⟩

lemma *oadd-lt-iff2*: $Ord(j) \Longrightarrow i{+}{+}j < i{+}{+}k \longleftrightarrow j<k$
⟨*proof*⟩

lemma *oadd-inject*: $[\![i{+}{+}j = i{+}{+}k;\ \ Ord(j);\ Ord(k)]\!] \Longrightarrow j=k$
⟨*proof*⟩

lemma *lt-oadd-disj*: $k < i{+}{+}j \Longrightarrow k<i \mid (\exists l\in j.\ k = i{+}{+}l\ )$
⟨*proof*⟩

21.4.4 Ordinal addition with successor – via associativity!

lemma *oadd-assoc*: $(i{+}{+}j){+}{+}k = i{+}{+}(j{+}{+}k)$
⟨*proof*⟩

lemma *oadd-unfold*: $[\![Ord(i);\ \ Ord(j)]\!] \Longrightarrow i{+}{+}j = i \cup (\bigcup k\in j.\ \{i{+}{+}k\})$
⟨*proof*⟩

lemma *oadd-1*: $Ord(i) \Longrightarrow i{+}{+}1 = succ(i)$
⟨*proof*⟩

lemma *oadd-succ* [*simp*]: $Ord(j) \Longrightarrow i{+}{+}succ(j) = succ(i{+}{+}j)$
⟨*proof*⟩

Ordinal addition with limit ordinals

lemma *oadd-UN*:
$[\![\bigwedge x.\ x \in A \Longrightarrow Ord(j(x));\ \ a \in A]\!]$
$\Longrightarrow i ++ (\bigcup x{\in}A.\ j(x)) = (\bigcup x{\in}A.\ i{+}{+}j(x))$
⟨*proof*⟩

lemma *oadd-Limit*: $Limit(j) \Longrightarrow i{+}{+}j = (\bigcup k{\in}j.\ i{+}{+}k)$
⟨*proof*⟩

lemma *oadd-eq-0-iff*: $[\![Ord(i);\ Ord(j)]\!] \Longrightarrow (i ++ j) = 0 \longleftrightarrow i{=}0 \land j{=}0$
⟨*proof*⟩

lemma *oadd-eq-lt-iff*: $[\![Ord(i);\ Ord(j)]\!] \Longrightarrow 0 < (i ++ j) \longleftrightarrow 0{<}i \mid 0{<}j$
⟨*proof*⟩

lemma *oadd-LimitI*: $[\![Ord(i);\ Limit(j)]\!] \Longrightarrow Limit(i ++ j)$
⟨*proof*⟩

Order/monotonicity properties of ordinal addition

lemma *oadd-le-self2*: $Ord(i) \Longrightarrow i \le j{+}{+}i$
⟨*proof*⟩

lemma *oadd-le-mono1*: $k \le j \Longrightarrow k{+}{+}i \le j{+}{+}i$
⟨*proof*⟩

lemma *oadd-lt-mono*: $[\![i' \le i;\ \ j'{<}j]\!] \Longrightarrow i'{+}{+}j' < i{+}{+}j$
⟨*proof*⟩

lemma *oadd-le-mono*: $[\![i' \le i;\ \ j' \le j]\!] \Longrightarrow i'{+}{+}j' \le i{+}{+}j$
⟨*proof*⟩

lemma *oadd-le-iff2*: $[\![Ord(j);\ Ord(k)]\!] \Longrightarrow i{+}{+}j \le i{+}{+}k \longleftrightarrow j \le k$
⟨*proof*⟩

lemma *oadd-lt-self*: $[\![Ord(i);\ \ 0{<}j]\!] \Longrightarrow i < i{+}{+}j$
⟨*proof*⟩

Every ordinal is exceeded by some limit ordinal.

lemma *Ord-imp-greater-Limit*: $Ord(i) \Longrightarrow \exists k.\ i{<}k \land Limit(k)$
⟨*proof*⟩

lemma *Ord2-imp-greater-Limit*: $[\![Ord(i);\ Ord(j)]\!] \Longrightarrow \exists k.\ i{<}k \land j{<}k \land Limit(k)$

⟨*proof*⟩

21.5 Ordinal Subtraction

The difference is $ordertype(j - i, Memrel(j))$. It's probably simpler to define the difference recursively!

lemma *bij-sum-Diff*:
$A<=B \Longrightarrow (\lambda y{\in}B.\ if(y \in A,\ Inl(y),\ Inr(y))) \in bij(B,\ A{+}(B{-}A))$
⟨*proof*⟩

lemma *ordertype-sum-Diff*:
$i \le j \Longrightarrow$
$ordertype(i{+}(j{-}i),\ radd(i,Memrel(j),j{-}i,Memrel(j))) =$
$ordertype(j,\ Memrel(j))$
⟨*proof*⟩

lemma *Ord-odiff* [*simp,TC*]:
$[\![Ord(i);\ \ Ord(j)]\!] \Longrightarrow Ord(i{-}{-}j)$
⟨*proof*⟩

lemma *raw-oadd-ordertype-Diff*:
$i \le j$
$\Longrightarrow raw\text{-}oadd(i,j{-}{-}i) = ordertype(i{+}(j{-}i),\ radd(i,Memrel(j),j{-}i,Memrel(j)))$
⟨*proof*⟩

lemma *oadd-odiff-inverse*: $i \le j \Longrightarrow i \mathbin{++} (j{-}{-}i) = j$
⟨*proof*⟩

lemma *odiff-oadd-inverse*: $[\![Ord(i);\ Ord(j)]\!] \Longrightarrow (i{+}{+}j)\ {-}{-}\ i = j$
⟨*proof*⟩

lemma *odiff-lt-mono2*: $[\![i<j;\ \ k \le i]\!] \Longrightarrow i{-}{-}k < j{-}{-}k$
⟨*proof*⟩

21.6 Ordinal Multiplication

lemma *Ord-omult* [*simp,TC*]:
$[\![Ord(i);\ \ Ord(j)]\!] \Longrightarrow Ord(i{*}{*}j)$
⟨*proof*⟩

21.6.1 A useful unfolding law

lemma *pred-Pair-eq*:
$[\![a \in A;\ \ b \in B]\!] \Longrightarrow pred(A{*}B,\ \langle a,b\rangle,\ rmult(A,r,B,s)) =$
$pred(A,a,r){*}B \cup (\{a\} * pred(B,b,s))$
⟨*proof*⟩

lemma *ordertype-pred-Pair-eq*:
$[\![a \in A;\ \ b \in B;\ \ well\text{-}ord(A,r);\ \ well\text{-}ord(B,s)]\!] \Longrightarrow$
$ordertype(pred(A*B,\ \langle a,b\rangle,\ rmult(A,r,B,s)),\ rmult(A,r,B,s)) =$
$ordertype(pred(A,a,r)*B\ +\ pred(B,b,s),$
$radd(A*B,\ rmult(A,r,B,s),\ B,\ s))$
⟨*proof*⟩

lemma *ordertype-pred-Pair-lemma*:
$[\![i'<i;\ \ j'<j]\!]$
$\Longrightarrow ordertype(pred(i*j,\ <i',j'>,\ rmult(i,Memrel(i),j,Memrel(j))),$
$rmult(i,Memrel(i),j,Memrel(j))) =$
$raw\text{-}oadd\ (j{*}{*}i',\ j')$
⟨*proof*⟩

lemma *lt-omult*:
$[\![Ord(i);\ \ Ord(j);\ \ k<j{*}{*}i]\!]$
$\Longrightarrow \exists j'\ i'.\ k = j{*}{*}i'\ {+}{+}\ j' \wedge j'<j \wedge i'<i$
⟨*proof*⟩

lemma *omult-oadd-lt*:
$[\![j'<j;\ \ i'<i]\!] \Longrightarrow j{*}{*}i'\ {+}{+}\ j'\ \ <\ \ j{*}{*}i$
⟨*proof*⟩

lemma *omult-unfold*:
$[\![Ord(i);\ \ Ord(j)]\!] \Longrightarrow j{*}{*}i = (\bigcup j'{\in}j.\ \bigcup i'{\in}i.\ \{j{*}{*}i'\ {+}{+}\ j'\})$
⟨*proof*⟩

21.6.2 Basic laws for ordinal multiplication

Ordinal multiplication by zero

lemma *omult-0* [*simp*]: $i{*}{*}0 = 0$
⟨*proof*⟩

lemma *omult-0-left* [*simp*]: $0{*}{*}i = 0$
⟨*proof*⟩

Ordinal multiplication by 1

lemma *omult-1* [*simp*]: $Ord(i) \Longrightarrow i{*}{*}1 = i$
⟨*proof*⟩

lemma *omult-1-left* [*simp*]: $Ord(i) \Longrightarrow 1{*}{*}i = i$
⟨*proof*⟩

Distributive law for ordinal multiplication and addition

lemma *oadd-omult-distrib*:
$[\![Ord(i);\ \ Ord(j);\ \ Ord(k)]\!] \Longrightarrow i{*}{*}(j{+}{+}k) = (i{*}{*}j){+}{+}(i{*}{*}k)$
⟨*proof*⟩

lemma *omult-succ*: $[\![Ord(i);\ \ Ord(j)]\!] \Longrightarrow i{*}{*}succ(j) = (i{*}{*}j){+}{+}i$
⟨*proof*⟩

Associative law

lemma *omult-assoc*:
$[\![Ord(i);\ \ Ord(j);\ \ Ord(k)]\!] \Longrightarrow (i{*}{*}j){*}{*}k = i{*}{*}(j{*}{*}k)$
⟨*proof*⟩

Ordinal multiplication with limit ordinals

lemma *omult-UN*:
$[\![Ord(i);\ \ \bigwedge x.\ x \in A \Longrightarrow Ord(j(x))]\!]$
$\Longrightarrow i\ {*}{*}\ (\bigcup x{\in}A.\ j(x)) = (\bigcup x{\in}A.\ i{*}{*}j(x))$
⟨*proof*⟩

lemma *omult-Limit*: $[\![Ord(i);\ \ Limit(j)]\!] \Longrightarrow i{*}{*}j = (\bigcup k{\in}j.\ i{*}{*}k)$
⟨*proof*⟩

21.6.3 Ordering/monotonicity properties of ordinal multiplication

lemma *lt-omult1*: $[\![k<i;\ \ 0<j]\!] \Longrightarrow k < i{*}{*}j$
⟨*proof*⟩

lemma *omult-le-self*: $[\![Ord(i);\ \ 0<j]\!] \Longrightarrow i \le i{*}{*}j$
⟨*proof*⟩

lemma *omult-le-mono1*:
assumes *kj*: $k \le j$ **and** *i*: $Ord(i)$ **shows** $k{*}{*}i \le j{*}{*}i$
⟨*proof*⟩

lemma *omult-lt-mono2*: $[\![k<j;\ \ 0<i]\!] \Longrightarrow i{*}{*}k < i{*}{*}j$
⟨*proof*⟩

lemma *omult-le-mono2*: $[\![k \le j;\ \ Ord(i)]\!] \Longrightarrow i{*}{*}k \le i{*}{*}j$
⟨*proof*⟩

lemma *omult-le-mono*: $[\![i' \le i;\ \ j' \le j]\!] \Longrightarrow i'{*}{*}j' \le i{*}{*}j$
⟨*proof*⟩

lemma *omult-lt-mono*: $[\![i' \le i;\ \ j'<j;\ \ 0<i]\!] \Longrightarrow i'{*}{*}j' < i{*}{*}j$
⟨*proof*⟩

lemma *omult-le-self2*:
assumes *i*: $Ord(i)$ **and** *j*: $0<j$ **shows** $i \le j{*}{*}i$
⟨*proof*⟩

Further properties of ordinal multiplication

lemma *omult-inject*: $[\![i{*}{*}j = i{*}{*}k;\ \ 0<i;\ \ Ord(j);\ \ Ord(k)]\!] \Longrightarrow j{=}k$
⟨*proof*⟩

21.7 The Relation *Lt*

lemma *wf-Lt*: *wf*(*Lt*)
⟨*proof*⟩

lemma *irrefl-Lt*: *irrefl*(*A*,*Lt*)
⟨*proof*⟩

lemma *trans-Lt*: *trans*[*A*](*Lt*)
⟨*proof*⟩

lemma *part-ord-Lt*: *part-ord*(*A*,*Lt*)
⟨*proof*⟩

lemma *linear-Lt*: *linear*(*nat*,*Lt*)
⟨*proof*⟩

lemma *tot-ord-Lt*: *tot-ord*(*nat*,*Lt*)
⟨*proof*⟩

lemma *well-ord-Lt*: *well-ord*(*nat*,*Lt*)
⟨*proof*⟩

end

22 Finite Powerset Operator and Finite Function Space

theory *Finite* **imports** *Inductive Epsilon Nat* **begin**

rep-datatype
  elimination *natE*
  induction *nat-induct*
  case-eqns *nat-case-0 nat-case-succ*
  recursor-eqns *recursor-0 recursor-succ*

consts
  Fin :: *i*⇒*i*
  FiniteFun :: [*i*,*i*]⇒*i* (‹(‹*notation*=‹*infix* −||>››- −||>/ -)› [*61*, *60*] *60*)

inductive
  domains *Fin*(*A*) ⊆ *Pow*(*A*)
  intros
    emptyI: *0* ∈ *Fin*(*A*)
    consI: ⟦*a* ∈ *A*; *b* ∈ *Fin*(*A*)⟧ ⟹ *cons*(*a*,*b*) ∈ *Fin*(*A*)
  type-intros *empty-subsetI cons-subsetI PowI*
  type-elims *PowD* [*elim-format*]

inductive
domains $FiniteFun(A,B) \subseteq Fin(A*B)$
intros
emptyI: $0 \in A -||> B$
consI: $[\![a \in A;\ \ b \in B;\ \ h \in A -||> B;\ \ a \notin domain(h)]\!]$
$\Longrightarrow cons(\langle a,b\rangle,h) \in A -||> B$
type-intros *Fin.intros*

22.1 Finite Powerset Operator

lemma *Fin-mono*: $A<=B \Longrightarrow Fin(A) \subseteq Fin(B)$
$\langle proof \rangle$

lemmas *FinD* = *Fin.dom-subset* [*THEN subsetD*, *THEN PowD*]

lemma *Fin-induct* [*case-names 0 cons*, *induct set*: *Fin*]:
$[\![b \in Fin(A);$
$P(0);$
$\bigwedge x\ y.\ [\![x \in A;\ \ y \in Fin(A);\ \ x\notin y;\ \ P(y)]\!] \Longrightarrow P(cons(x,y))$
$]\!] \Longrightarrow P(b)$
$\langle proof \rangle$

declare *Fin.intros* [*simp*]

lemma *Fin-0*: $Fin(0) = \{0\}$
$\langle proof \rangle$

lemma *Fin-UnI* [*simp*]: $[\![b \in Fin(A);\ \ c \in Fin(A)]\!] \Longrightarrow b \cup c \in Fin(A)$
$\langle proof \rangle$

lemma *Fin-UnionI*: $C \in Fin(Fin(A)) \Longrightarrow \bigcup(C) \in Fin(A)$
$\langle proof \rangle$

lemma *Fin-subset-lemma* [*rule-format*]: $b \in Fin(A) \Longrightarrow \forall z.\ z<=b \longrightarrow z \in Fin(A)$
$\langle proof \rangle$

lemma *Fin-subset*: $[\![c<=b;\ \ b \in Fin(A)]\!] \Longrightarrow c \in Fin(A)$
$\langle proof \rangle$

lemma *Fin-IntI1* [*intro*,*simp*]: $b \in Fin(A) \Longrightarrow b \cap c \in Fin(A)$
⟨*proof*⟩

lemma *Fin-IntI2* [*intro*,*simp*]: $c \in Fin(A) \Longrightarrow b \cap c \in Fin(A)$
⟨*proof*⟩

lemma *Fin-0-induct-lemma* [*rule-format*]:
$[\![c \in Fin(A);\ \ b \in Fin(A);\ P(b);$
$\bigwedge x\ y.\ [\![x \in A;\ \ y \in Fin(A);\ \ x \in y;\ \ P(y)]\!] \Longrightarrow P(y-\{x\})$
$]\!] \Longrightarrow c<=b \longrightarrow P(b-c)$
⟨*proof*⟩

lemma *Fin-0-induct*:
$[\![b \in Fin(A);$
$P(b);$
$\bigwedge x\ y.\ [\![x \in A;\ \ y \in Fin(A);\ \ x \in y;\ \ P(y)]\!] \Longrightarrow P(y-\{x\})$
$]\!] \Longrightarrow P(0)$
⟨*proof*⟩

lemma *nat-fun-subset-Fin*: $n \in nat \Longrightarrow n->A \subseteq Fin(nat*A)$
⟨*proof*⟩

22.2 Finite Function Space

lemma *FiniteFun-mono*:
$[\![A<=C;\ \ B<=D]\!] \Longrightarrow A -||> B\ \subseteq\ C -||> D$
⟨*proof*⟩

lemma *FiniteFun-mono1*: $A<=B \Longrightarrow A -||> A\ \subseteq\ B -||> B$
⟨*proof*⟩

lemma *FiniteFun-is-fun*: $h \in A -||>B \Longrightarrow h \in domain(h) -> B$
⟨*proof*⟩

lemma *FiniteFun-domain-Fin*: $h \in A -||>B \Longrightarrow domain(h) \in Fin(A)$
⟨*proof*⟩

lemmas *FiniteFun-apply-type* = *FiniteFun-is-fun* [*THEN apply-type*]

lemma *FiniteFun-subset-lemma* [*rule-format*]:
$b \in A-||>B \Longrightarrow \forall z.\ z<=b \longrightarrow z \in A-||>B$
⟨*proof*⟩

lemma *FiniteFun-subset*: $[\![c<=b;\ \ b \in A-||>B]\!] \Longrightarrow c \in A-||>B$
⟨*proof*⟩

lemma *fun-FiniteFunI* [*rule-format*]: $A \in Fin(X) \Longrightarrow \forall f.\ f \in A{-}{>}B \longrightarrow f \in A{-}||{>}B$
⟨*proof*⟩

lemma *lam-FiniteFun*: $A \in Fin(X) \Longrightarrow (\lambda x{\in}A.\ b(x)) \in A\ {-}||{>}\ \{b(x).\ x \in A\}$
⟨*proof*⟩

lemma *FiniteFun-Collect-iff*:
$f \in FiniteFun(A,\ \{y \in B.\ P(y)\})$
$\longleftrightarrow f \in FiniteFun(A,B) \wedge (\forall x{\in}domain(f).\ P(f`x))$
⟨*proof*⟩

22.3 The Contents of a Singleton Set

definition
contents :: $i{\Rightarrow}i$ **where**
$contents(X) \equiv THE\ x.\ X = \{x\}$

lemma *contents-eq* [*simp*]: $contents\ (\{x\}) = x$
⟨*proof*⟩

end

23 Cardinal Numbers Without the Axiom of Choice

theory *Cardinal* **imports** *OrderType Finite Nat Sum* **begin**

definition

Least :: $(i{\Rightarrow}o) \Rightarrow i$ (**binder** ‹μ › *10*) **where**
$Least(P) \equiv THE\ i.\ Ord(i) \wedge P(i) \wedge (\forall j.\ j{<}i \longrightarrow \neg P(j))$

definition
eqpoll :: $[i,i] \Rightarrow o$ (**infixl** ‹≈› *50*) **where**
$A \approx B \equiv \exists f.\ f \in bij(A,B)$

definition
lepoll :: $[i,i] \Rightarrow o$ (**infixl** ‹≲› *50*) **where**
$A \lesssim B \equiv \exists f.\ f \in inj(A,B)$

definition
lesspoll :: $[i,i] \Rightarrow o$ (**infixl** ‹≺› *50*) **where**
$A \prec B \equiv A \lesssim B \wedge \neg(A \approx B)$

definition
cardinal :: $i{\Rightarrow}i$ (‹(‹open-block notation=‹mixfix cardinal››|-|)›)
where $|A| \equiv (\mu\ i.\ i \approx A)$

definition
Finite :: $i \Rightarrow o$ **where**
$Finite(A) \equiv \exists n \in nat.\ A \approx n$

definition
Card :: $i \Rightarrow o$ **where**
$Card(i) \equiv (i = |i|)$

23.1 The Schroeder-Bernstein Theorem

See Davey and Priestly, page 106

lemma *decomp-bnd-mono*: $bnd\text{-}mono(X, \lambda W.\ X - g``(Y - f``W))$
⟨*proof*⟩

lemma *Banach-last-equation*:
$g \in Y -> X$
$\Longrightarrow g``(Y - f``\ lfp(X, \lambda W.\ X - g``(Y - f``W))) =$
$X - lfp(X, \lambda W.\ X - g``(Y - f``W))$
⟨*proof*⟩

lemma *decomposition*:
$[\![f \in X -> Y;\ g \in Y -> X]\!] \Longrightarrow$
$\exists XA\ XB\ YA\ YB.\ (XA \cap XB = 0) \wedge (XA \cup XB = X) \wedge$
$(YA \cap YB = 0) \wedge (YA \cup YB = Y) \wedge$
$f``XA = YA \wedge g``YB = XB$
⟨*proof*⟩

lemma *schroeder-bernstein*:
$[\![f \in inj(X,Y);\ g \in inj(Y,X)]\!] \Longrightarrow \exists h.\ h \in bij(X,Y)$
⟨*proof*⟩

lemma *bij-imp-eqpoll*: $f \in bij(A,B) \Longrightarrow A \approx B$
⟨*proof*⟩

lemmas *eqpoll-refl* = *id-bij* [*THEN bij-imp-eqpoll*, *simp*]

lemma *eqpoll-sym*: $X \approx Y \Longrightarrow Y \approx X$
⟨*proof*⟩

lemma *eqpoll-trans* [*trans*]:
$[\![X \approx Y;\ Y \approx Z]\!] \Longrightarrow X \approx Z$
⟨*proof*⟩

lemma *subset-imp-lepoll*: $X<=Y \Longrightarrow X \lesssim Y$
⟨*proof*⟩

lemmas *lepoll-refl* = *subset-refl* [*THEN subset-imp-lepoll*, *simp*]

lemmas *le-imp-lepoll* = *le-imp-subset* [*THEN subset-imp-lepoll*]

lemma *eqpoll-imp-lepoll*: $X \approx Y \Longrightarrow X \lesssim Y$
⟨*proof*⟩

lemma *lepoll-trans* [*trans*]: $[\![X \lesssim Y;\ \ Y \lesssim Z]\!] \Longrightarrow X \lesssim Z$
⟨*proof*⟩

lemma *eq-lepoll-trans* [*trans*]: $[\![X \approx Y;\ \ Y \lesssim Z]\!] \Longrightarrow X \lesssim Z$
⟨*proof*⟩

lemma *lepoll-eq-trans* [*trans*]: $[\![X \lesssim Y;\ \ Y \approx Z]\!] \Longrightarrow X \lesssim Z$
⟨*proof*⟩

lemma *eqpollI*: $[\![X \lesssim Y;\ \ Y \lesssim X]\!] \Longrightarrow X \approx Y$
⟨*proof*⟩

lemma *eqpollE*:
$[\![X \approx Y; [\![X \lesssim Y; Y \lesssim X]\!] \Longrightarrow P]\!] \Longrightarrow P$
⟨*proof*⟩

lemma *eqpoll-iff*: $X \approx Y \longleftrightarrow X \lesssim Y \wedge Y \lesssim X$
⟨*proof*⟩

lemma *lepoll-0-is-0*: $A \lesssim 0 \Longrightarrow A = 0$
⟨*proof*⟩

lemmas *empty-lepollI* = *empty-subsetI* [*THEN subset-imp-lepoll*]

lemma *lepoll-0-iff*: $A \lesssim 0 \longleftrightarrow A=0$
⟨*proof*⟩

lemma *Un-lepoll-Un*:
$[\![A \lesssim B; C \lesssim D; B \cap D = 0]\!] \Longrightarrow A \cup C \lesssim B \cup D$
⟨*proof*⟩

lemmas *eqpoll-0-is-0* = *eqpoll-imp-lepoll* [*THEN lepoll-0-is-0*]

lemma *eqpoll-0-iff*: $A \approx 0 \longleftrightarrow A=0$
⟨*proof*⟩

lemma *eqpoll-disjoint-Un*:
  $[\![A \approx B;\ \ C \approx D;\ \ A \cap C = 0;\ \ B \cap D = 0]\!]$
  $\Longrightarrow A \cup C \approx B \cup D$
  ⟨*proof*⟩

23.2 lesspoll: contributions by Krzysztof Grabczewski

lemma *lesspoll-not-refl*: $\neg\ (i \prec i)$
⟨*proof*⟩

lemma *lesspoll-irrefl* [*elim*!]: $i \prec i \Longrightarrow P$
⟨*proof*⟩

lemma *lesspoll-imp-lepoll*: $A \prec B \Longrightarrow A \lesssim B$
⟨*proof*⟩

lemma *lepoll-well-ord*: $[\![A \lesssim B;\ well\text{-}ord(B,r)]\!] \Longrightarrow \exists\, s.\ well\text{-}ord(A,s)$
  ⟨*proof*⟩

lemma *lepoll-iff-leqpoll*: $A \lesssim B \longleftrightarrow A \prec B \mid A \approx B$
  ⟨*proof*⟩

lemma *inj-not-surj-succ*:
  assumes *fi*: $f \in inj(A,\ succ(m))$ **and** *fns*: $f \notin surj(A,\ succ(m))$
  shows $\exists f.\ f \in inj(A,m)$
⟨*proof*⟩

lemma *lesspoll-trans* [*trans*]:
  $[\![X \prec Y;\ Y \prec Z]\!] \Longrightarrow X \prec Z$
  ⟨*proof*⟩

lemma *lesspoll-trans1* [*trans*]:
  $[\![X \lesssim Y;\ Y \prec Z]\!] \Longrightarrow X \prec Z$
  ⟨*proof*⟩

lemma *lesspoll-trans2* [*trans*]:
  $[\![X \prec Y;\ Y \lesssim Z]\!] \Longrightarrow X \prec Z$
  ⟨*proof*⟩

lemma *eq-lesspoll-trans* [*trans*]:
  $[\![X \approx Y;\ Y \prec Z]\!] \Longrightarrow X \prec Z$
  ⟨*proof*⟩

lemma *lesspoll-eq-trans* [*trans*]:
  $[\![X \prec Y;\ Y \approx Z]\!] \Longrightarrow X \prec Z$
  ⟨*proof*⟩

lemma *Least-equality*:
$[\![P(i);\ \ Ord(i);\ \ \bigwedge x.\ x<i \Longrightarrow \neg P(x)]\!] \Longrightarrow (\mu\ x.\ P(x)) = i$
⟨*proof*⟩

lemma *LeastI*:
assumes P: $P(i)$ **and** i: $Ord(i)$ **shows** $P(\mu\ x.\ P(x))$
⟨*proof*⟩

The proof is almost identical to the one above!

lemma *Least-le*:
assumes P: $P(i)$ **and** i: $Ord(i)$ **shows** $(\mu\ x.\ P(x)) \leq i$
⟨*proof*⟩

lemma *less-LeastE*: $[\![P(i);\ \ i < (\mu\ x.\ P(x))]\!] \Longrightarrow Q$
⟨*proof*⟩

lemma *LeastI2*:
$[\![P(i);\ \ Ord(i);\ \ \bigwedge j.\ P(j) \Longrightarrow Q(j)]\!] \Longrightarrow Q(\mu\ j.\ P(j))$
⟨*proof*⟩

lemma *Least-0*:
$[\![\neg\ (\exists\, i.\ Ord(i) \wedge P(i))]\!] \Longrightarrow (\mu\ x.\ P(x)) = 0$
⟨*proof*⟩

lemma *Ord-Least* [*intro*,*simp*,*TC*]: $Ord(\mu\ x.\ P(x))$
⟨*proof*⟩

23.3 Basic Properties of Cardinals

lemma *Least-cong*: $(\bigwedge y.\ P(y) \longleftrightarrow Q(y)) \Longrightarrow (\mu\ x.\ P(x)) = (\mu\ x.\ Q(x))$
⟨*proof*⟩

lemma *cardinal-cong*: $X \approx Y \Longrightarrow |X| = |Y|$
⟨*proof*⟩

lemma *well-ord-cardinal-eqpoll*:
assumes r: $well\text{-}ord(A,r)$ **shows** $|A| \approx A$
⟨*proof*⟩

lemmas *Ord-cardinal-eqpoll* = *well-ord-Memrel* [*THEN well-ord-cardinal-eqpoll*]

lemma *Ord-cardinal-idem*: $Ord(A) \Longrightarrow ||A|| = |A|$
⟨*proof*⟩

lemma *well-ord-cardinal-eqE*:
assumes *woX*: *well-ord*(X,r) **and** *woY*: *well-ord*(Y,s) **and** *eq*: $|X| = |Y|$
shows $X \approx Y$
⟨*proof*⟩

lemma *well-ord-cardinal-eqpoll-iff*:
$[\![$*well-ord*(X,r); *well-ord*$(Y,s)]\!] \Longrightarrow |X| = |Y| \longleftrightarrow X \approx Y$
⟨*proof*⟩

lemma *Ord-cardinal-le*: $Ord(i) \Longrightarrow |i| \leq i$
⟨*proof*⟩

lemma *Card-cardinal-eq*: $Card(K) \Longrightarrow |K| = K$
⟨*proof*⟩

lemma *CardI*: $[\![Ord(i); \bigwedge j.\ j<i \Longrightarrow \neg(j \approx i)]\!] \Longrightarrow Card(i)$
⟨*proof*⟩

lemma *Card-is-Ord*: $Card(i) \Longrightarrow Ord(i)$
⟨*proof*⟩

lemma *Card-cardinal-le*: $Card(K) \Longrightarrow K \leq |K|$
⟨*proof*⟩

lemma *Ord-cardinal* [*simp,intro!*]: $Ord(|A|)$
⟨*proof*⟩

The cardinals are the initial ordinals.

lemma *Card-iff-initial*: $Card(K) \longleftrightarrow Ord(K) \wedge (\forall j.\ j<K \longrightarrow \neg\ j \approx K)$
⟨*proof*⟩

lemma *lt-Card-imp-lesspoll*: $[\![Card(a); i<a]\!] \Longrightarrow i \prec a$
⟨*proof*⟩

lemma *Card-0*: $Card(0)$
⟨*proof*⟩

lemma *Card-Un*: $[\![Card(K); Card(L)]\!] \Longrightarrow Card(K \cup L)$
⟨*proof*⟩

lemma *Card-cardinal* [*iff*]: $Card(|A|)$
⟨*proof*⟩

lemma *cardinal-eq-lemma*:
assumes i:$|i| \le j$ **and** j: $j \le i$ **shows** $|j| = |i|$
⟨*proof*⟩

lemma *cardinal-mono*:
assumes ij: $i \le j$ **shows** $|i| \le |j|$
⟨*proof*⟩

Since we have $|succ(nat)| \le |nat|$, the converse of *cardinal-mono* fails!

lemma *cardinal-lt-imp-lt*: $[\![|i| < |j|;\ Ord(i);\ Ord(j)]\!] \Longrightarrow i < j$
⟨*proof*⟩

lemma *Card-lt-imp-lt*: $[\![|i| < K;\ Ord(i);\ Card(K)]\!] \Longrightarrow i < K$
⟨*proof*⟩

lemma *Card-lt-iff*: $[\![Ord(i);\ Card(K)]\!] \Longrightarrow (|i| < K) \longleftrightarrow (i < K)$
⟨*proof*⟩

lemma *Card-le-iff*: $[\![Ord(i);\ Card(K)]\!] \Longrightarrow (K \le |i|) \longleftrightarrow (K \le i)$
⟨*proof*⟩

lemma *well-ord-lepoll-imp-cardinal-le*:
assumes wB: $well\text{-}ord(B,r)$ **and** AB: $A \lesssim B$
shows $|A| \le |B|$
⟨*proof*⟩

lemma *lepoll-cardinal-le*: $[\![A \lesssim i;\ Ord(i)]\!] \Longrightarrow |A| \le i$
⟨*proof*⟩

lemma *lepoll-Ord-imp-eqpoll*: $[\![A \lesssim i;\ Ord(i)]\!] \Longrightarrow |A| \approx A$
⟨*proof*⟩

lemma *lesspoll-imp-eqpoll*: $[\![A \prec i;\ Ord(i)]\!] \Longrightarrow |A| \approx A$
⟨*proof*⟩

lemma *cardinal-subset-Ord*: $[\![A<=i;\ Ord(i)]\!] \Longrightarrow |A| \subseteq i$
⟨*proof*⟩

23.4 The finite cardinals

lemma *cons-lepoll-consD*:
$[\![cons(u,A) \lesssim cons(v,B);\ u \notin A;\ v \notin B]\!] \Longrightarrow A \lesssim B$

⟨*proof*⟩

lemma *cons-eqpoll-consD*: $[\![cons(u,A) \approx cons(v,B);\ \ u \notin A;\ \ v \notin B]\!] \Longrightarrow A \approx B$
⟨*proof*⟩

lemma *succ-lepoll-succD*: $succ(m) \lesssim succ(n) \Longrightarrow m \lesssim n$
⟨*proof*⟩

lemma *nat-lepoll-imp-le*:
$m \in nat \Longrightarrow n \in nat \Longrightarrow m \lesssim n \Longrightarrow m \leq n$
⟨*proof*⟩

lemma *nat-eqpoll-iff*: $[\![m \in nat;\ n \in nat]\!] \Longrightarrow m \approx n \longleftrightarrow m = n$
⟨*proof*⟩

lemma *nat-into-Card*:
assumes n: $n \in nat$ **shows** $Card(n)$
⟨*proof*⟩

lemmas *cardinal-0* = *nat-0I* [*THEN nat-into-Card*, *THEN Card-cardinal-eq*, *iff*]
lemmas *cardinal-1* = *nat-1I* [*THEN nat-into-Card*, *THEN Card-cardinal-eq*, *iff*]

lemma *succ-lepoll-natE*: $[\![succ(n) \lesssim n;\ \ n \in nat]\!] \Longrightarrow P$
⟨*proof*⟩

lemma *nat-lepoll-imp-ex-eqpoll-n*:
$[\![n \in nat;\ \ nat \lesssim X]\!] \Longrightarrow \exists\, Y.\ Y \subseteq X \wedge n \approx Y$
⟨*proof*⟩

lemma *lepoll-succ*: $i \lesssim succ(i)$
⟨*proof*⟩

lemma *lepoll-imp-lesspoll-succ*:
assumes A: $A \lesssim m$ **and** m: $m \in nat$
shows $A \prec succ(m)$
⟨*proof*⟩

lemma *lesspoll-succ-imp-lepoll*:
$[\![A \prec succ(m);\ m \in nat]\!] \Longrightarrow A \lesssim m$
⟨*proof*⟩

lemma *lesspoll-succ-iff*: $m \in nat \Longrightarrow A \prec succ(m) \longleftrightarrow A \lesssim m$
⟨*proof*⟩

lemma *lepoll-succ-disj*: $[\![A \lesssim succ(m);\ \ m \in nat]\!] \Longrightarrow A \lesssim m \mid A \approx succ(m)$
⟨*proof*⟩

lemma *lesspoll-cardinal-lt*: $[\![A \prec i;\ Ord(i)]\!] \Longrightarrow |A| < i$
⟨*proof*⟩

23.5 The first infinite cardinal: Omega, or nat

lemma *lt-not-lepoll*:
assumes n: $n<i$ $n \in nat$ **shows** $\neg\ i \lesssim n$
⟨*proof*⟩

A slightly weaker version of *nat-eqpoll-iff*

lemma *Ord-nat-eqpoll-iff*:
assumes i: $Ord(i)$ **and** n: $n \in nat$ **shows** $i \approx n \longleftrightarrow i=n$
⟨*proof*⟩

lemma *Card-nat*: $Card(nat)$
⟨*proof*⟩

lemma *nat-le-cardinal*: $nat \le i \Longrightarrow nat \le |i|$
⟨*proof*⟩

lemma *n-lesspoll-nat*: $n \in nat \Longrightarrow n \prec nat$
⟨*proof*⟩

23.6 Towards Cardinal Arithmetic

lemma *cons-lepoll-cong*:
$[\![A \lesssim B;\ \ b \notin B]\!] \Longrightarrow cons(a,A) \lesssim cons(b,B)$
⟨*proof*⟩

lemma *cons-eqpoll-cong*:
$[\![A \approx B;\ \ a \notin A;\ \ b \notin B]\!] \Longrightarrow cons(a,A) \approx cons(b,B)$
⟨*proof*⟩

lemma *cons-lepoll-cons-iff*:
$[\![a \notin A;\ \ b \notin B]\!] \Longrightarrow cons(a,A) \lesssim cons(b,B) \ \longleftrightarrow\ A \lesssim B$
⟨*proof*⟩

lemma *cons-eqpoll-cons-iff*:
$[\![a \notin A;\ \ b \notin B]\!] \Longrightarrow cons(a,A) \approx cons(b,B) \ \longleftrightarrow\ A \approx B$
⟨*proof*⟩

lemma *singleton-eqpoll-1*: $\{a\} \approx 1$

⟨*proof*⟩

lemma *cardinal-singleton*: $|\{a\}| = 1$
⟨*proof*⟩

lemma *not-0-is-lepoll-1*: $A \neq 0 \Longrightarrow 1 \lesssim A$
⟨*proof*⟩

lemma *succ-eqpoll-cong*: $A \approx B \Longrightarrow succ(A) \approx succ(B)$
⟨*proof*⟩

lemma *sum-eqpoll-cong*: $[\![A \approx C;\ B \approx D]\!] \Longrightarrow A+B \approx C+D$
⟨*proof*⟩

lemma *prod-eqpoll-cong*:
$[\![A \approx C;\ B \approx D]\!] \Longrightarrow A*B \approx C*D$
⟨*proof*⟩

lemma *inj-disjoint-eqpoll*:
$[\![f \in inj(A,B);\ A \cap B = 0]\!] \Longrightarrow A \cup (B - range(f)) \approx B$
⟨*proof*⟩

23.7 Lemmas by Krzysztof Grabczewski

If A has at most $n + 1$ elements and $a \in A$ then $A - \{a\}$ has at most n.

lemma *Diff-sing-lepoll*:
$[\![a \in A;\ A \lesssim succ(n)]\!] \Longrightarrow A - \{a\} \lesssim n$
⟨*proof*⟩

If A has at least $n + 1$ elements then $A - \{a\}$ has at least n.

lemma *lepoll-Diff-sing*:
assumes A: $succ(n) \lesssim A$ **shows** $n \lesssim A - \{a\}$
⟨*proof*⟩

lemma *Diff-sing-eqpoll*: $[\![a \in A;\ A \approx succ(n)]\!] \Longrightarrow A - \{a\} \approx n$
⟨*proof*⟩

lemma *lepoll-1-is-sing*: $[\![A \lesssim 1;\ a \in A]\!] \Longrightarrow A = \{a\}$
⟨*proof*⟩

lemma *Un-lepoll-sum*: $A \cup B \lesssim A+B$
⟨*proof*⟩

lemma *well-ord-Un*:
$[\![well\text{-}ord(X,R);\ well\text{-}ord(Y,S)]\!] \Longrightarrow \exists\, T.\ well\text{-}ord(X \cup Y,\ T)$
⟨*proof*⟩

lemma *disj-Un-eqpoll-sum*: $A \cap B = 0 \Longrightarrow A \cup B \approx A + B$
⟨*proof*⟩

23.8 Finite and infinite sets

lemma *eqpoll-imp-Finite-iff*: $A \approx B \Longrightarrow Finite(A) \longleftrightarrow Finite(B)$
⟨*proof*⟩

lemma *Finite-0* [*simp*]: $Finite(0)$
⟨*proof*⟩

lemma *Finite-cons*: $Finite(x) \Longrightarrow Finite(cons(y,x))$
⟨*proof*⟩

lemma *Finite-succ*: $Finite(x) \Longrightarrow Finite(succ(x))$
⟨*proof*⟩

lemma *lepoll-nat-imp-Finite*:
assumes *A*: $A \lesssim n$ **and** *n*: $n \in nat$ **shows** $Finite(A)$
⟨*proof*⟩

lemma *lesspoll-nat-is-Finite*:
$A \prec nat \Longrightarrow Finite(A)$
⟨*proof*⟩

lemma *lepoll-Finite*:
assumes *Y*: $Y \lesssim X$ **and** *X*: $Finite(X)$ **shows** $Finite(Y)$
⟨*proof*⟩

lemmas *subset-Finite* = *subset-imp-lepoll* [*THEN lepoll-Finite*]

lemma *Finite-cons-iff* [*iff*]: $Finite(cons(y,x)) \longleftrightarrow Finite(x)$
⟨*proof*⟩

lemma *Finite-succ-iff* [*iff*]: $Finite(succ(x)) \longleftrightarrow Finite(x)$
⟨*proof*⟩

lemma *Finite-Int*: $Finite(A) \mid Finite(B) \Longrightarrow Finite(A \cap B)$
⟨*proof*⟩

lemmas *Finite-Diff* = *Diff-subset* [*THEN subset-Finite*]

lemma *nat-le-infinite-Ord*:
$[\![Ord(i);\ \neg\ Finite(i)]\!] \Longrightarrow nat \le i$
⟨*proof*⟩

lemma *Finite-imp-well-ord*:

$Finite(A) \Longrightarrow \exists r.\ well\text{-}ord(A,r)$
⟨proof⟩

lemma *succ-lepoll-imp-not-empty*: $succ(x) \lesssim y \Longrightarrow y \neq 0$
⟨proof⟩

lemma *eqpoll-succ-imp-not-empty*: $x \approx succ(n) \Longrightarrow x \neq 0$
⟨proof⟩

lemma *Finite-Fin-lemma* [*rule-format*]:
$n \in nat \Longrightarrow \forall A.\ (A \approx n \wedge A \subseteq X) \longrightarrow A \in Fin(X)$
⟨proof⟩

lemma *Finite-Fin*: $[\![Finite(A);\ A \subseteq X]\!] \Longrightarrow A \in Fin(X)$
⟨proof⟩

lemma *Fin-lemma* [*rule-format*]: $n \in nat \Longrightarrow \forall A.\ A \approx n \longrightarrow A \in Fin(A)$
⟨proof⟩

lemma *Finite-into-Fin*: $Finite(A) \Longrightarrow A \in Fin(A)$
⟨proof⟩

lemma *Fin-into-Finite*: $A \in Fin(U) \Longrightarrow Finite(A)$
⟨proof⟩

lemma *Finite-Fin-iff*: $Finite(A) \longleftrightarrow A \in Fin(A)$
⟨proof⟩

lemma *Finite-Un*: $[\![Finite(A);\ Finite(B)]\!] \Longrightarrow Finite(A \cup B)$
⟨proof⟩

lemma *Finite-Un-iff* [*simp*]: $Finite(A \cup B) \longleftrightarrow (Finite(A) \wedge Finite(B))$
⟨proof⟩

The converse must hold too.

lemma *Finite-Union*: $[\![\forall y \in X.\ Finite(y);\ \ Finite(X)]\!] \Longrightarrow Finite(\bigcup(X))$
⟨proof⟩

lemma *Finite-induct* [*case-names 0 cons*, *induct set*: *Finite*]:
$[\![Finite(A);\ P(0);$
$\bigwedge x\ B.\ \ [\![Finite(B);\ x \notin B;\ P(B)]\!] \Longrightarrow P(cons(x,\ B))]\!]$
$\Longrightarrow P(A)$
⟨proof⟩

lemma *Diff-sing-Finite*: $Finite(A - \{a\}) \Longrightarrow Finite(A)$
⟨proof⟩

lemma *Diff-Finite* [*rule-format*]: $Finite(B) \Longrightarrow Finite(A-B) \longrightarrow Finite(A)$
⟨*proof*⟩

lemma *Finite-RepFun*: $Finite(A) \Longrightarrow Finite(RepFun(A,f))$
⟨*proof*⟩

lemma *Finite-RepFun-iff-lemma* [*rule-format*]:
$[\![Finite(x); \bigwedge x\ y.\ f(x)=f(y) \Longrightarrow x=y]\!]$
$\Longrightarrow \forall A.\ x = RepFun(A,f) \longrightarrow Finite(A)$
⟨*proof*⟩

I don't know why, but if the premise is expressed using meta-connectives then the simplifier cannot prove it automatically in conditional rewriting.

lemma *Finite-RepFun-iff*:
$(\forall x\ y.\ f(x)=f(y) \longrightarrow x=y) \Longrightarrow Finite(RepFun(A,f)) \longleftrightarrow Finite(A)$
⟨*proof*⟩

lemma *Finite-Pow*: $Finite(A) \Longrightarrow Finite(Pow(A))$
⟨*proof*⟩

lemma *Finite-Pow-imp-Finite*: $Finite(Pow(A)) \Longrightarrow Finite(A)$
⟨*proof*⟩

lemma *Finite-Pow-iff* [*iff*]: $Finite(Pow(A)) \longleftrightarrow Finite(A)$
⟨*proof*⟩

lemma *Finite-cardinal-iff*:
assumes *i*: $Ord(i)$ **shows** $Finite(|i|) \longleftrightarrow Finite(i)$
⟨*proof*⟩

lemma *nat-wf-on-converse-Memrel*: $n \in nat \Longrightarrow wf[n](converse(Memrel(n)))$
⟨*proof*⟩

lemma *nat-well-ord-converse-Memrel*: $n \in nat \Longrightarrow well\text{-}ord(n,converse(Memrel(n)))$
⟨*proof*⟩

lemma *well-ord-converse*:
$[\![well\text{-}ord(A,r);$
$well\text{-}ord(ordertype(A,r),\ converse(Memrel(ordertype(A,\ r))))]\!]$
$\Longrightarrow well\text{-}ord(A,converse(r))$
⟨*proof*⟩

lemma *ordertype-eq-n*:
assumes *r*: $well\text{-}ord(A,r)$ **and** *A*: $A \approx n$ **and** *n*: $n \in nat$
shows $ordertype(A,r) = n$

⟨*proof*⟩

lemma *Finite-well-ord-converse*:
$[\![Finite(A);\ \ well\text{-}ord(A,r)]\!] \Longrightarrow well\text{-}ord(A,converse(r))$
⟨*proof*⟩

lemma *nat-into-Finite*: $n \in nat \Longrightarrow Finite(n)$
⟨*proof*⟩

lemma *nat-not-Finite*: $\neg\ Finite(nat)$
⟨*proof*⟩

end

24 The Cumulative Hierarchy and a Small Universe for Recursive Types

theory *Univ* **imports** *Epsilon Cardinal* **begin**

definition
Vfrom :: $[i,i]\Rightarrow i$ **where**
$Vfrom(A,i) \equiv transrec(i, \lambda x\, f.\ A \cup (\bigcup y{\in}x.\ Pow(f`y)))$

abbreviation
Vset :: $i\Rightarrow i$ **where**
$Vset(x) \equiv Vfrom(0,x)$

definition
Vrec :: $[i, [i,i]\Rightarrow i] \Rightarrow i$ **where**
$Vrec(a,H) \equiv transrec(rank(a), \lambda x\ g.\ \lambda z{\in}Vset(succ(x)).$
$H(z, \lambda w{\in}Vset(x).\ g`rank(w)`w))\ `\ a$

definition
Vrecursor :: $[[i,i]\Rightarrow i,\ i] \Rightarrow i$ **where**
$Vrecursor(H,a) \equiv transrec(rank(a), \lambda x\ g.\ \lambda z{\in}Vset(succ(x)).$
$H(\lambda w{\in}Vset(x).\ g`rank(w)`w,\ z))\ `\ a$

definition
univ :: $i\Rightarrow i$ **where**
$univ(A) \equiv Vfrom(A,nat)$

24.1 Immediate Consequences of the Definition of $Vfrom(A, i)$

NOT SUITABLE FOR REWRITING – RECURSIVE!

lemma *Vfrom*: $Vfrom(A,i) = A \cup (\bigcup j{\in}i.\ Pow(Vfrom(A,j)))$
⟨*proof*⟩

24.1.1 Monotonicity

lemma *Vfrom-mono* [*rule-format*]:
$A<=B \Longrightarrow \forall j.\ i<=j \longrightarrow Vfrom(A,i) \subseteq Vfrom(B,j)$
⟨*proof*⟩

lemma *VfromI*: $[\![a \in Vfrom(A,j);\ j<i]\!] \Longrightarrow a \in Vfrom(A,i)$
⟨*proof*⟩

24.1.2 A fundamental equality: Vfrom does not require ordinals!

lemma *Vfrom-rank-subset1*: $Vfrom(A,x) \subseteq Vfrom(A,rank(x))$
⟨*proof*⟩

lemma *Vfrom-rank-subset2*: $Vfrom(A,rank(x)) \subseteq Vfrom(A,x)$
⟨*proof*⟩

lemma *Vfrom-rank-eq*: $Vfrom(A,rank(x)) = Vfrom(A,x)$
⟨*proof*⟩

24.2 Basic Closure Properties

lemma *zero-in-Vfrom*: $y{:}x \Longrightarrow 0 \in Vfrom(A,x)$
⟨*proof*⟩

lemma *i-subset-Vfrom*: $i \subseteq Vfrom(A,i)$
⟨*proof*⟩

lemma *A-subset-Vfrom*: $A \subseteq Vfrom(A,i)$
⟨*proof*⟩

lemmas *A-into-Vfrom* = *A-subset-Vfrom* [*THEN subsetD*]

lemma *subset-mem-Vfrom*: $a \subseteq Vfrom(A,i) \Longrightarrow a \in Vfrom(A,succ(i))$
⟨*proof*⟩

24.2.1 Finite sets and ordered pairs

lemma *singleton-in-Vfrom*: $a \in Vfrom(A,i) \Longrightarrow \{a\} \in Vfrom(A,succ(i))$
⟨*proof*⟩

lemma *doubleton-in-Vfrom*:
$[\![a \in Vfrom(A,i);\ b \in Vfrom(A,i)]\!] \Longrightarrow \{a,b\} \in Vfrom(A,succ(i))$
⟨*proof*⟩

lemma *Pair-in-Vfrom*:
$[\![a \in Vfrom(A,i);\ b \in Vfrom(A,i)]\!] \Longrightarrow \langle a,b\rangle \in Vfrom(A,succ(succ(i)))$
⟨*proof*⟩

lemma *succ-in-Vfrom*: $a \subseteq Vfrom(A,i) \Longrightarrow succ(a) \in Vfrom(A,succ(succ(i)))$

⟨*proof*⟩

24.3 0, Successor and Limit Equations for *Vfrom*

lemma *Vfrom-0*: $Vfrom(A,0) = A$
⟨*proof*⟩

lemma *Vfrom-succ-lemma*: $Ord(i) \Longrightarrow Vfrom(A,succ(i)) = A \cup Pow(Vfrom(A,i))$
⟨*proof*⟩

lemma *Vfrom-succ*: $Vfrom(A,succ(i)) = A \cup Pow(Vfrom(A,i))$
⟨*proof*⟩

lemma *Vfrom-Union*: $y{:}X \Longrightarrow Vfrom(A,\bigcup(X)) = (\bigcup y{\in}X.\ Vfrom(A,y))$
⟨*proof*⟩

24.4 *Vfrom* applied to Limit Ordinals

lemma *Limit-Vfrom-eq*:
$Limit(i) \Longrightarrow Vfrom(A,i) = (\bigcup y{\in}i.\ Vfrom(A,y))$
⟨*proof*⟩

lemma *Limit-VfromE*:
$[\![a \in Vfrom(A,i);\ \ \neg R \Longrightarrow Limit(i);$
$\bigwedge x.\ [\![x<i;\ \ a \in Vfrom(A,x)]\!] \Longrightarrow R$
$]\!] \Longrightarrow R$
⟨*proof*⟩

lemma *singleton-in-VLimit*:
$[\![a \in Vfrom(A,i);\ \ Limit(i)]\!] \Longrightarrow \{a\} \in Vfrom(A,i)$
⟨*proof*⟩

lemmas *Vfrom-UnI1* =
Un-upper1 [*THEN subset-refl* [*THEN Vfrom-mono*, *THEN subsetD*]]
lemmas *Vfrom-UnI2* =
Un-upper2 [*THEN subset-refl* [*THEN Vfrom-mono*, *THEN subsetD*]]

Hard work is finding a single j:i such that a,b<=Vfrom(A,j)

lemma *doubleton-in-VLimit*:
$[\![a \in Vfrom(A,i);\ \ b \in Vfrom(A,i);\ \ Limit(i)]\!] \Longrightarrow \{a,b\} \in Vfrom(A,i)$
⟨*proof*⟩

lemma *Pair-in-VLimit*:
$[\![a \in Vfrom(A,i);\ \ b \in Vfrom(A,i);\ \ Limit(i)]\!] \Longrightarrow \langle a,b\rangle \in Vfrom(A,i)$

Infer that a, b occur at ordinals x,xa < i.

⟨*proof*⟩

lemma *product-VLimit*: $Limit(i) \Longrightarrow Vfrom(A,i) * Vfrom(A,i) \subseteq Vfrom(A,i)$
⟨*proof*⟩

lemmas *Sigma-subset-VLimit* =
subset-trans [*OF Sigma-mono product-VLimit*]

lemmas *nat-subset-VLimit* =
subset-trans [*OF nat-le-Limit* [*THEN le-imp-subset*] *i-subset-Vfrom*]

lemma *nat-into-VLimit*: $[\![n: nat;\ \ Limit(i)]\!] \Longrightarrow n \in Vfrom(A,i)$
⟨*proof*⟩

24.4.1 Closure under Disjoint Union

lemmas *zero-in-VLimit* = *Limit-has-0* [*THEN ltD*, *THEN zero-in-Vfrom*]

lemma *one-in-VLimit*: $Limit(i) \Longrightarrow 1 \in Vfrom(A,i)$
⟨*proof*⟩

lemma *Inl-in-VLimit*:
$[\![a \in Vfrom(A,i);\ Limit(i)]\!] \Longrightarrow Inl(a) \in Vfrom(A,i)$
⟨*proof*⟩

lemma *Inr-in-VLimit*:
$[\![b \in Vfrom(A,i);\ Limit(i)]\!] \Longrightarrow Inr(b) \in Vfrom(A,i)$
⟨*proof*⟩

lemma *sum-VLimit*: $Limit(i) \Longrightarrow Vfrom(C,i)+Vfrom(C,i) \subseteq Vfrom(C,i)$
⟨*proof*⟩

lemmas *sum-subset-VLimit* = *subset-trans* [*OF sum-mono sum-VLimit*]

24.5 Properties assuming *Transset*(*A*)

lemma *Transset-Vfrom*: $Transset(A) \Longrightarrow Transset(Vfrom(A,i))$
⟨*proof*⟩

lemma *Transset-Vfrom-succ*:
$Transset(A) \Longrightarrow Vfrom(A,\ succ(i)) = Pow(Vfrom(A,i))$
⟨*proof*⟩

lemma *Transset-Pair-subset*: $[\![\langle a,b\rangle \subseteq C;\ Transset(C)]\!] \Longrightarrow a: C \wedge b: C$
⟨*proof*⟩

lemma *Transset-Pair-subset-VLimit*:
$[\![\langle a,b\rangle \subseteq Vfrom(A,i);\ \ Transset(A);\ \ Limit(i)]\!]$
$\Longrightarrow \langle a,b\rangle \in Vfrom(A,i)$
⟨*proof*⟩

lemma *Union-in-Vfrom*:

$[\![X \in Vfrom(A,j);\ \ Transset(A)]\!] \Longrightarrow \bigcup(X) \in Vfrom(A,\ succ(j))$
⟨*proof*⟩

lemma *Union-in-VLimit*:
$[\![X \in Vfrom(A,i);\ \ Limit(i);\ \ Transset(A)]\!] \Longrightarrow \bigcup(X) \in Vfrom(A,i)$
⟨*proof*⟩

General theorem for membership in Vfrom(A,i) when i is a limit ordinal

lemma *in-VLimit*:
$[\![a \in Vfrom(A,i);\ \ b \in Vfrom(A,i);\ \ Limit(i);$
$\bigwedge x\ y\ j.\ [\![j<i;\ 1{:}j;\ x \in Vfrom(A,j);\ y \in Vfrom(A,j)]\!]$
$\Longrightarrow \exists\, k.\ h(x,y) \in Vfrom(A,k) \wedge k<i]\!]$
$\Longrightarrow h(a,b) \in Vfrom(A,i)$

Infer that a, b occur at ordinals x,xa < i.

⟨*proof*⟩

24.5.1 Products

lemma *prod-in-Vfrom*:
$[\![a \in Vfrom(A,j);\ \ b \in Vfrom(A,j);\ \ Transset(A)]\!]$
$\Longrightarrow a{*}b \in Vfrom(A,\ succ(succ(succ(j))))$
⟨*proof*⟩

lemma *prod-in-VLimit*:
$[\![a \in Vfrom(A,i);\ \ b \in Vfrom(A,i);\ \ Limit(i);\ \ Transset(A)]\!]$
$\Longrightarrow a{*}b \in Vfrom(A,i)$
⟨*proof*⟩

24.5.2 Disjoint Sums, or Quine Ordered Pairs

lemma *sum-in-Vfrom*:
$[\![a \in Vfrom(A,j);\ \ b \in Vfrom(A,j);\ \ Transset(A);\ \ 1{:}j]\!]$
$\Longrightarrow a{+}b \in Vfrom(A,\ succ(succ(succ(j))))$
⟨*proof*⟩

lemma *sum-in-VLimit*:
$[\![a \in Vfrom(A,i);\ \ b \in Vfrom(A,i);\ \ Limit(i);\ \ Transset(A)]\!]$
$\Longrightarrow a{+}b \in Vfrom(A,i)$
⟨*proof*⟩

24.5.3 Function Space!

lemma *fun-in-Vfrom*:
$[\![a \in Vfrom(A,j);\ \ b \in Vfrom(A,j);\ \ Transset(A)]\!] \Longrightarrow$
$a{-}{>}b \in Vfrom(A,\ succ(succ(succ(succ(j)))))$
⟨*proof*⟩

lemma *fun-in-VLimit*:
$[\![a \in Vfrom(A,i);\ \ b \in Vfrom(A,i);\ \ Limit(i);\ \ Transset(A)]\!]$

$\Longrightarrow a{-}{>}b \in Vfrom(A,i)$
⟨proof⟩

lemma *Pow-in-Vfrom*:
$[\![a \in Vfrom(A,j);\ \ Transset(A)]\!] \Longrightarrow Pow(a) \in Vfrom(A,\ succ(succ(j)))$
⟨proof⟩

lemma *Pow-in-VLimit*:
$[\![a \in Vfrom(A,i);\ \ Limit(i);\ \ Transset(A)]\!] \Longrightarrow Pow(a) \in Vfrom(A,i)$
⟨proof⟩

24.6 The Set *Vset*(*i*)

lemma *Vset*: $Vset(i) = (\bigcup j{\in}i.\ Pow(Vset(j)))$
⟨proof⟩

lemmas *Vset-succ* = *Transset-0* [*THEN Transset-Vfrom-succ*]
lemmas *Transset-Vset* = *Transset-0* [*THEN Transset-Vfrom*]

24.6.1 Characterisation of the elements of *Vset*(*i*)

lemma *VsetD* [*rule-format*]: $Ord(i) \Longrightarrow \forall\, b.\ b \in Vset(i) \longrightarrow rank(b) < i$
⟨proof⟩

lemma *VsetI-lemma* [*rule-format*]:
$Ord(i) \Longrightarrow \forall\, b.\ rank(b) \in i \longrightarrow b \in Vset(i)$
⟨proof⟩

lemma *VsetI*: $rank(x){<}i \Longrightarrow x \in Vset(i)$
⟨proof⟩

Merely a lemma for the next result

lemma *Vset-Ord-rank-iff*: $Ord(i) \Longrightarrow b \in Vset(i) \longleftrightarrow rank(b) < i$
⟨proof⟩

lemma *Vset-rank-iff* [*simp*]: $b \in Vset(a) \longleftrightarrow rank(b) < rank(a)$
⟨proof⟩

This is rank(rank(a)) = rank(a)

declare *Ord-rank* [*THEN rank-of-Ord*, *simp*]

lemma *rank-Vset*: $Ord(i) \Longrightarrow rank(Vset(i)) = i$
⟨proof⟩

lemma *Finite-Vset*: $i \in nat \Longrightarrow Finite(Vset(i))$
⟨proof⟩

24.6.2 Reasoning about Sets in Terms of Their Elements' Ranks

lemma *arg-subset-Vset-rank*: $a \subseteq Vset(rank(a))$

⟨*proof*⟩

lemma *Int-Vset-subset*:
$[\![\bigwedge i.\ Ord(i) \Longrightarrow a \cap Vset(i) \subseteq b]\!] \Longrightarrow a \subseteq b$
⟨*proof*⟩

24.6.3 Set Up an Environment for Simplification

lemma *rank-Inl*: $rank(a) < rank(Inl(a))$
⟨*proof*⟩

lemma *rank-Inr*: $rank(a) < rank(Inr(a))$
⟨*proof*⟩

lemmas *rank-rls = rank-Inl rank-Inr rank-pair1 rank-pair2*

24.6.4 Recursion over Vset Levels!

NOT SUITABLE FOR REWRITING: recursive!

lemma *Vrec*: $Vrec(a,H) = H(a,\ \lambda x\in Vset(rank(a)).\ Vrec(x,H))$
⟨*proof*⟩

This form avoids giant explosions in proofs. NOTE the form of the premise!

lemma *def-Vrec*:
$[\![\bigwedge x.\ h(x)\equiv Vrec(x,H)]\!] \Longrightarrow$
$h(a) = H(a,\ \lambda x\in Vset(rank(a)).\ h(x))$
⟨*proof*⟩

NOT SUITABLE FOR REWRITING: recursive!

lemma *Vrecursor*:
$Vrecursor(H,a) = H(\lambda x\in Vset(rank(a)).\ Vrecursor(H,x),\ \ a)$
⟨*proof*⟩

This form avoids giant explosions in proofs. NOTE the form of the premise!

lemma *def-Vrecursor*:
$h \equiv Vrecursor(H) \Longrightarrow h(a) = H(\lambda x\in Vset(rank(a)).\ h(x),\ \ a)$
⟨*proof*⟩

24.7 The Datatype Universe: *univ*(*A*)

lemma *univ-mono*: $A<=B \Longrightarrow univ(A) \subseteq univ(B)$
⟨*proof*⟩

lemma *Transset-univ*: $Transset(A) \Longrightarrow Transset(univ(A))$
⟨*proof*⟩

24.7.1 The Set *univ*(*A*) as a Limit

lemma *univ-eq-UN*: $univ(A) = (\bigcup i \in nat.\ Vfrom(A,i))$
⟨*proof*⟩

lemma *subset-univ-eq-Int*: $c \subseteq univ(A) \Longrightarrow c = (\bigcup i \in nat.\ c \cap Vfrom(A,i))$
⟨*proof*⟩

lemma *univ-Int-Vfrom-subset*:
$[\![a \subseteq univ(X);$
$\bigwedge i.\ i{:}nat \Longrightarrow a \cap Vfrom(X,i) \subseteq b]\!]$
$\Longrightarrow a \subseteq b$
⟨*proof*⟩

lemma *univ-Int-Vfrom-eq*:
$[\![a \subseteq univ(X);\quad b \subseteq univ(X);$
$\bigwedge i.\ i{:}nat \Longrightarrow a \cap Vfrom(X,i) = b \cap Vfrom(X,i)$
$]\!] \Longrightarrow a = b$
⟨*proof*⟩

24.8 Closure Properties for *univ*(*A*)

lemma *zero-in-univ*: $0 \in univ(A)$
⟨*proof*⟩

lemma *zero-subset-univ*: $\{0\} \subseteq univ(A)$
⟨*proof*⟩

lemma *A-subset-univ*: $A \subseteq univ(A)$
⟨*proof*⟩

lemmas *A-into-univ* = *A-subset-univ* [*THEN subsetD*]

24.8.1 Closure under Unordered and Ordered Pairs

lemma *singleton-in-univ*: $a{:}\ univ(A) \Longrightarrow \{a\} \in univ(A)$
⟨*proof*⟩

lemma *doubleton-in-univ*:
$[\![a{:}\ univ(A);\quad b{:}\ univ(A)]\!] \Longrightarrow \{a,b\} \in univ(A)$
⟨*proof*⟩

lemma *Pair-in-univ*:
$[\![a{:}\ univ(A);\quad b{:}\ univ(A)]\!] \Longrightarrow \langle a,b\rangle \in univ(A)$
⟨*proof*⟩

lemma *Union-in-univ*:
$[\![X{:}\ univ(A);\quad Transset(A)]\!] \Longrightarrow \bigcup(X) \in univ(A)$
⟨*proof*⟩

lemma *product-univ*: $univ(A)*univ(A) \subseteq univ(A)$
⟨*proof*⟩

24.8.2 The Natural Numbers

lemma *nat-subset-univ*: $nat \subseteq univ(A)$
⟨*proof*⟩

lemma *nat-into-univ*: $n \in nat \Longrightarrow n \in univ(A)$
⟨*proof*⟩

24.8.3 Instances for 1 and 2

lemma *one-in-univ*: $1 \in univ(A)$
⟨*proof*⟩

unused!

lemma *two-in-univ*: $2 \in univ(A)$
⟨*proof*⟩

lemma *bool-subset-univ*: $bool \subseteq univ(A)$
⟨*proof*⟩

lemmas *bool-into-univ* = *bool-subset-univ* [*THEN subsetD*]

24.8.4 Closure under Disjoint Union

lemma *Inl-in-univ*: $a: univ(A) \Longrightarrow Inl(a) \in univ(A)$
⟨*proof*⟩

lemma *Inr-in-univ*: $b: univ(A) \Longrightarrow Inr(b) \in univ(A)$
⟨*proof*⟩

lemma *sum-univ*: $univ(C)+univ(C) \subseteq univ(C)$
⟨*proof*⟩

lemmas *sum-subset-univ* = *subset-trans* [*OF sum-mono sum-univ*]

lemma *Sigma-subset-univ*:
$[\![A \subseteq univ(D); \bigwedge x.\ x \in A \Longrightarrow B(x) \subseteq univ(D)]\!] \Longrightarrow Sigma(A,B) \subseteq univ(D)$
⟨*proof*⟩

24.9 Finite Branching Closure Properties

24.9.1 Closure under Finite Powerset

lemma *Fin-Vfrom-lemma*:
$[\![b: Fin(Vfrom(A,i));\ \ Limit(i)]\!] \Longrightarrow \exists j.\ b \subseteq Vfrom(A,j) \wedge j<i$
⟨*proof*⟩

lemma *Fin-VLimit*: *Limit*(i) $\Longrightarrow$ *Fin*(*Vfrom*(A,i)) $\subseteq$ *Vfrom*(A,i)
⟨*proof*⟩

lemmas *Fin-subset-VLimit* = *subset-trans* [*OF Fin-mono Fin-VLimit*]

lemma *Fin-univ*: *Fin*(*univ*(A)) $\subseteq$ *univ*(A)
⟨*proof*⟩

24.9.2 Closure under Finite Powers: Functions from a Natural Number

lemma *nat-fun-VLimit*:
⟦n: *nat*; *Limit*(i)⟧ $\Longrightarrow$ n -> *Vfrom*(A,i) $\subseteq$ *Vfrom*(A,i)
⟨*proof*⟩

lemmas *nat-fun-subset-VLimit* = *subset-trans* [*OF Pi-mono nat-fun-VLimit*]

lemma *nat-fun-univ*: n: *nat* $\Longrightarrow$ n -> *univ*(A) $\subseteq$ *univ*(A)
⟨*proof*⟩

24.9.3 Closure under Finite Function Space

General but seldom-used version; normally the domain is fixed

lemma *FiniteFun-VLimit1*:
Limit(i) $\Longrightarrow$ *Vfrom*(A,i) -||> *Vfrom*(A,i) $\subseteq$ *Vfrom*(A,i)
⟨*proof*⟩

lemma *FiniteFun-univ1*: *univ*(A) -||> *univ*(A) $\subseteq$ *univ*(A)
⟨*proof*⟩

Version for a fixed domain

lemma *FiniteFun-VLimit*:
⟦W $\subseteq$ *Vfrom*(A,i); *Limit*(i)⟧ $\Longrightarrow$ W -||> *Vfrom*(A,i) $\subseteq$ *Vfrom*(A,i)
⟨*proof*⟩

lemma *FiniteFun-univ*:
W $\subseteq$ *univ*(A) $\Longrightarrow$ W -||> *univ*(A) $\subseteq$ *univ*(A)
⟨*proof*⟩

lemma *FiniteFun-in-univ*:
⟦f: W -||> *univ*(A); W $\subseteq$ *univ*(A)⟧ $\Longrightarrow$ $f \in$ *univ*(A)
⟨*proof*⟩

Remove $\subseteq$ from the rule above

lemmas *FiniteFun-in-univ'* = *FiniteFun-in-univ* [*OF - subsetI*]

24.10 * For QUniv. Properties of Vfrom analogous to the "take-lemma" *

Intersecting a*b with Vfrom...

This version says a, b exist one level down, in the smaller set Vfrom(X,i)

lemma *doubleton-in-Vfrom-D*:
$[\![\{a,b\} \in Vfrom(X,succ(i)); \ Transset(X)]\!]$
$\Longrightarrow a \in Vfrom(X,i) \ \wedge \ b \in Vfrom(X,i)$
⟨*proof*⟩

This weaker version says a, b exist at the same level

lemmas *Vfrom-doubleton-D* = *Transset-Vfrom* [*THEN Transset-doubleton-D*]

lemma *Pair-in-Vfrom-D*:
$[\![\langle a,b\rangle \in Vfrom(X,succ(i)); \ Transset(X)]\!]$
$\Longrightarrow a \in Vfrom(X,i) \ \wedge \ b \in Vfrom(X,i)$
⟨*proof*⟩

lemma *product-Int-Vfrom-subset*:
$Transset(X) \Longrightarrow$
$(a*b) \cap Vfrom(X, \ succ(i)) \subseteq (a \cap Vfrom(X,i)) * (b \cap Vfrom(X,i))$
⟨*proof*⟩

⟨*ML*⟩

end

25 A Small Universe for Lazy Recursive Types

theory *QUniv* **imports** *Univ QPair* **begin**

rep-datatype
elimination *sumE*
induction *TrueI*
case-eqns *case-Inl case-Inr*

rep-datatype
elimination *qsumE*
induction *TrueI*
case-eqns *qcase-QInl qcase-QInr*

definition

quniv :: $i \Rightarrow i$ **where**
quniv$(A) \equiv$ *Pow*(*univ*(*eclose*(A)))

25.1 Properties involving Transset and Sum

lemma *Transset-includes-summands*:
$[\![$*Transset*(C); $A{+}B \subseteq C]\!] \Longrightarrow A \subseteq C \wedge B \subseteq C$
⟨*proof*⟩

lemma *Transset-sum-Int-subset*:
Transset$(C) \Longrightarrow (A{+}B) \cap C \subseteq (A \cap C) + (B \cap C)$
⟨*proof*⟩

25.2 Introduction and Elimination Rules

lemma *qunivI*: $X \subseteq$ *univ*(*eclose*(A)) $\Longrightarrow X \in$ *quniv*(A)
⟨*proof*⟩

lemma *qunivD*: $X \in$ *quniv*$(A) \Longrightarrow X \subseteq$ *univ*(*eclose*(A))
⟨*proof*⟩

lemma *quniv-mono*: $A{<}{=}B \Longrightarrow$ *quniv*$(A) \subseteq$ *quniv*(B)
⟨*proof*⟩

25.3 Closure Properties

lemma *univ-eclose-subset-quniv*: *univ*(*eclose*(A)) $\subseteq$ *quniv*(A)
⟨*proof*⟩

lemma *univ-subset-quniv*: *univ*$(A) \subseteq$ *quniv*(A)
⟨*proof*⟩

lemmas *univ-into-quniv* = *univ-subset-quniv* [*THEN subsetD*]

lemma *Pow-univ-subset-quniv*: *Pow*(*univ*(A)) $\subseteq$ *quniv*(A)
⟨*proof*⟩

lemmas *univ-subset-into-quniv* =
PowI [*THEN Pow-univ-subset-quniv* [*THEN subsetD*]]

lemmas *zero-in-quniv* = *zero-in-univ* [*THEN univ-into-quniv*]
lemmas *one-in-quniv* = *one-in-univ* [*THEN univ-into-quniv*]
lemmas *two-in-quniv* = *two-in-univ* [*THEN univ-into-quniv*]

lemmas *A-subset-quniv* = *subset-trans* [*OF A-subset-univ univ-subset-quniv*]

lemmas *A-into-quniv* = *A-subset-quniv* [*THEN subsetD*]

lemma *QPair-subset-univ*:
⟦*a* ⊆ *univ*(*A*); *b* ⊆ *univ*(*A*)⟧ ⟹ <*a*;*b*> ⊆ *univ*(*A*)
⟨*proof*⟩

25.4 Quine Disjoint Sum

lemma *QInl-subset-univ*: *a* ⊆ *univ*(*A*) ⟹ *QInl*(*a*) ⊆ *univ*(*A*)
⟨*proof*⟩

lemmas *naturals-subset-nat* =
Ord-nat [*THEN Ord-is-Transset*, *unfolded Transset-def*, *THEN bspec*]

lemmas *naturals-subset-univ* =
subset-trans [*OF naturals-subset-nat nat-subset-univ*]

lemma *QInr-subset-univ*: *a* ⊆ *univ*(*A*) ⟹ *QInr*(*a*) ⊆ *univ*(*A*)
⟨*proof*⟩

25.5 Closure for Quine-Inspired Products and Sums

lemma *QPair-in-quniv*:
⟦*a*: *quniv*(*A*); *b*: *quniv*(*A*)⟧ ⟹ <*a*;*b*> ∈ *quniv*(*A*)
⟨*proof*⟩

lemma *QSigma-quniv*: *quniv*(*A*) <*> *quniv*(*A*) ⊆ *quniv*(*A*)
⟨*proof*⟩

lemmas *QSigma-subset-quniv* = *subset-trans* [*OF QSigma-mono QSigma-quniv*]

lemma *quniv-QPair-D*:
<*a*;*b*> ∈ *quniv*(*A*) ⟹ *a*: *quniv*(*A*) ∧ *b*: *quniv*(*A*)
⟨*proof*⟩

lemmas *quniv-QPair-E* = *quniv-QPair-D* [*THEN conjE*]

lemma *quniv-QPair-iff*: <*a*;*b*> ∈ *quniv*(*A*) ⟷ *a*: *quniv*(*A*) ∧ *b*: *quniv*(*A*)
⟨*proof*⟩

25.6 Quine Disjoint Sum

lemma *QInl-in-quniv*: *a*: *quniv*(*A*) ⟹ *QInl*(*a*) ∈ *quniv*(*A*)
⟨*proof*⟩

lemma *QInr-in-quniv*: *b*: *quniv*(*A*) ⟹ *QInr*(*b*) ∈ *quniv*(*A*)
⟨*proof*⟩

lemma *qsum-quniv*: *quniv*(*C*) <+> *quniv*(*C*) ⊆ *quniv*(*C*)

⟨*proof*⟩

lemmas *qsum-subset-quniv* = *subset-trans* [*OF qsum-mono qsum-quniv*]

25.7 The Natural Numbers

lemmas *nat-subset-quniv* = *subset-trans* [*OF nat-subset-univ univ-subset-quniv*]

lemmas *nat-into-quniv* = *nat-subset-quniv* [*THEN subsetD*]

lemmas *bool-subset-quniv* = *subset-trans* [*OF bool-subset-univ univ-subset-quniv*]

lemmas *bool-into-quniv* = *bool-subset-quniv* [*THEN subsetD*]

lemma *QPair-Int-Vfrom-succ-subset*:
Transset(*X*) ⟹
<*a*;*b*> ∩ *Vfrom*(*X*, *succ*(*i*)) ⊆ <*a* ∩ *Vfrom*(*X*,*i*); *b* ∩ *Vfrom*(*X*,*i*)>
⟨*proof*⟩

25.8 "Take-Lemma" Rules

lemma *QPair-Int-Vfrom-subset*:
Transset(*X*) ⟹
<*a*;*b*> ∩ *Vfrom*(*X*,*i*) ⊆ <*a* ∩ *Vfrom*(*X*,*i*); *b* ∩ *Vfrom*(*X*,*i*)>
⟨*proof*⟩

lemmas *QPair-Int-Vset-subset-trans* =
subset-trans [*OF Transset-0* [*THEN QPair-Int-Vfrom-subset*] *QPair-mono*]

lemma *QPair-Int-Vset-subset-UN*:
Ord(*i*) ⟹ <*a*;*b*> ∩ *Vset*(*i*) ⊆ (⋃*j*∈*i*. <*a* ∩ *Vset*(*j*); *b* ∩ *Vset*(*j*)>)
⟨*proof*⟩

end

26 Datatype and CoDatatype Definitions

theory *Datatype*
imports *Inductive Univ QUniv*
keywords *datatype codatatype* :: *thy-decl*
begin

⟨*ML*⟩

end

27 Arithmetic Operators and Their Definitions

theory *Arith* **imports** *Univ* **begin**

Proofs about elementary arithmetic: addition, multiplication, etc.

definition
  pred :: $i \Rightarrow i$ **where**
    $pred(y) \equiv nat\text{-}case(0, \lambda x. x, y)$

definition
  natify :: $i \Rightarrow i$ **where**
    $natify \equiv Vrecursor(\lambda f\ a.\ if\ a = succ(pred(a))\ then\ succ(f`pred(a))$
$else\ 0)$

consts
  raw-add :: $[i,i] \Rightarrow i$
  raw-diff :: $[i,i] \Rightarrow i$
  raw-mult :: $[i,i] \Rightarrow i$

primrec
  $raw\text{-}add\ (0, n) = n$
  $raw\text{-}add\ (succ(m), n) = succ(raw\text{-}add(m, n))$

primrec
  raw-diff-0: $raw\text{-}diff(m, 0) = m$
  raw-diff-succ: $raw\text{-}diff(m, succ(n)) =$
$nat\text{-}case(0, \lambda x. x, raw\text{-}diff(m, n))$

primrec
  $raw\text{-}mult(0, n) = 0$
  $raw\text{-}mult(succ(m), n) = raw\text{-}add\ (n, raw\text{-}mult(m, n))$

definition
  add :: $[i,i] \Rightarrow i$ (**infixl** ‹#+› *65*) **where**
    $m \#+ n \equiv raw\text{-}add\ (natify(m), natify(n))$

definition
  diff :: $[i,i] \Rightarrow i$ (**infixl** ‹#−› *65*) **where**
    $m \#- n \equiv raw\text{-}diff\ (natify(m), natify(n))$

definition
  mult :: $[i,i] \Rightarrow i$ (**infixl** ‹#∗› *70*) **where**
    $m \#* n \equiv raw\text{-}mult\ (natify(m), natify(n))$

definition
  raw-div :: $[i,i] \Rightarrow i$ **where**
    $raw\text{-}div\ (m, n) \equiv$

$transrec(m, \lambda j\, f.\ if\ j{<}n \mid n{=}0\ then\ 0\ else\ succ(f`(j\#{-}n)))$

definition
raw-mod :: $[i,i]{\Rightarrow}i$ **where**
raw-mod $(m, n) \equiv$
$transrec(m, \lambda j\, f.\ if\ j{<}n \mid n{=}0\ then\ j\ else\ f`(j\#{-}n))$

definition
div :: $[i,i]{\Rightarrow}i$ (**infixl** ‹*div*› *70*) **where**
$m\ div\ n \equiv raw\text{-}div\ (natify(m), natify(n))$

definition
mod :: $[i,i]{\Rightarrow}i$ (**infixl** ‹*mod*› *70*) **where**
$m\ mod\ n \equiv raw\text{-}mod\ (natify(m), natify(n))$

declare *rec-type* [*simp*]
nat-0-le [*simp*]

lemma *zero-lt-lemma*: $[\![0{<}k;\ k \in nat]\!] \Longrightarrow \exists j{\in}nat.\ k = succ(j)$
⟨*proof*⟩

lemmas *zero-lt-natE* = *zero-lt-lemma* [*THEN bexE*]

27.1 *natify*, the Coercion to *nat*

lemma *pred-succ-eq* [*simp*]: $pred(succ(y)) = y$
⟨*proof*⟩

lemma *natify-succ*: $natify(succ(x)) = succ(natify(x))$
⟨*proof*⟩

lemma *natify-0* [*simp*]: $natify(0) = 0$
⟨*proof*⟩

lemma *natify-non-succ*: $\forall z.\ x \neq succ(z) \Longrightarrow natify(x) = 0$
⟨*proof*⟩

lemma *natify-in-nat* [*iff*,*TC*]: $natify(x) \in nat$
⟨*proof*⟩

lemma *natify-ident* [*simp*]: $n \in nat \Longrightarrow natify(n) = n$
⟨*proof*⟩

lemma *natify-eqE*: $[\![natify(x) = y;\ \ x \in nat]\!] \Longrightarrow x{=}y$
⟨*proof*⟩

lemma *natify-idem* [*simp*]: $natify(natify(x)) = natify(x)$
⟨*proof*⟩

lemma *add-natify1* [*simp*]: $natify(m)$ #+ $n = m$ #+ n
⟨*proof*⟩

lemma *add-natify2* [*simp*]: m #+ $natify(n) = m$ #+ n
⟨*proof*⟩

lemma *mult-natify1* [*simp*]: $natify(m)$ #∗ $n = m$ #∗ n
⟨*proof*⟩

lemma *mult-natify2* [*simp*]: m #∗ $natify(n) = m$ #∗ n
⟨*proof*⟩

lemma *diff-natify1* [*simp*]: $natify(m)$ #− $n = m$ #− n
⟨*proof*⟩

lemma *diff-natify2* [*simp*]: m #− $natify(n) = m$ #− n
⟨*proof*⟩

lemma *mod-natify1* [*simp*]: $natify(m)\ mod\ n = m\ mod\ n$
⟨*proof*⟩

lemma *mod-natify2* [*simp*]: $m\ mod\ natify(n) = m\ mod\ n$
⟨*proof*⟩

lemma *div-natify1* [*simp*]: $natify(m)\ div\ n = m\ div\ n$
⟨*proof*⟩

lemma *div-natify2* [*simp*]: $m\ div\ natify(n) = m\ div\ n$
⟨*proof*⟩

27.2 Typing rules

lemma *raw-add-type*: $[\![m \in nat;\ \ n \in nat]\!] \Longrightarrow raw\text{-}add\ (m,\ n) \in nat$

⟨*proof*⟩

lemma *add-type* [*iff*,*TC*]: $m \#+ n \in nat$
⟨*proof*⟩

lemma *raw-mult-type*: $[\![m{\in}nat;\ \ n{\in}nat]\!] \Longrightarrow raw\text{-}mult\ (m,\ n) \in nat$
⟨*proof*⟩

lemma *mult-type* [*iff*,*TC*]: $m \#* n \in nat$
⟨*proof*⟩

lemma *raw-diff-type*: $[\![m{\in}nat;\ \ n{\in}nat]\!] \Longrightarrow raw\text{-}diff\ (m,\ n) \in nat$
⟨*proof*⟩

lemma *diff-type* [*iff*,*TC*]: $m \#- n \in nat$
⟨*proof*⟩

lemma *diff-0-eq-0* [*simp*]: $0 \#- n = 0$
⟨*proof*⟩

lemma *diff-succ-succ* [*simp*]: $succ(m) \#- succ(n) = m \#- n$
⟨*proof*⟩

declare *raw-diff-succ* [*simp del*]

lemma *diff-0* [*simp*]: $m \#- 0 = natify(m)$
⟨*proof*⟩

lemma *diff-le-self*: $m{\in}nat \Longrightarrow (m \#- n) \le m$
⟨*proof*⟩

27.3 Addition

lemma *add-0-natify* [*simp*]: $0 \#+ m = natify(m)$
⟨*proof*⟩

lemma *add-succ* [*simp*]: $succ(m) \#+ n = succ(m \#+ n)$
⟨*proof*⟩

lemma *add-0*: $m \in nat \Longrightarrow 0 \#+ m = m$
⟨*proof*⟩

lemma *add-assoc*: $(m \#+ n) \#+ k = m \#+ (n \#+ k)$
⟨*proof*⟩

lemma *add-0-right-natify* [*simp*]: $m \#+ 0 = natify(m)$
⟨*proof*⟩

lemma *add-succ-right* [*simp*]: $m \#+ succ(n) = succ(m \#+ n)$
⟨*proof*⟩

lemma *add-0-right*: $m \in nat \Longrightarrow m \#+ 0 = m$
⟨*proof*⟩

lemma *add-commute*: $m \#+ n = n \#+ m$
⟨*proof*⟩

lemma *add-left-commute*: $m\#+(n\#+k)=n\#+(m\#+k)$
⟨*proof*⟩

lemmas *add-ac* = *add-assoc add-commute add-left-commute*

lemma *raw-add-left-cancel*:
$[\![raw\text{-}add(k, m) = raw\text{-}add(k, n);\ \ k{\in}nat]\!] \Longrightarrow m{=}n$
⟨*proof*⟩

lemma *add-left-cancel-natify*: $k \#+ m = k \#+ n \Longrightarrow natify(m) = natify(n)$
⟨*proof*⟩

lemma *add-left-cancel*:
$[\![i = j;\ \ i \#+ m = j \#+ n;\ \ m{\in}nat;\ \ n{\in}nat]\!] \Longrightarrow m = n$
⟨*proof*⟩

lemma *add-le-elim1-natify*: $k\#+m \le k\#+n \Longrightarrow natify(m) \le natify(n)$
⟨*proof*⟩

lemma *add-le-elim1*: $[\![k\#+m \le k\#+n;\ m \in nat;\ n \in nat]\!] \Longrightarrow m \le n$
⟨*proof*⟩

lemma *add-lt-elim1-natify*: $k\#+m < k\#+n \Longrightarrow natify(m) < natify(n)$
⟨*proof*⟩

lemma *add-lt-elim1*: $[\![k\#+m < k\#+n;\ m \in nat;\ n \in nat]\!] \Longrightarrow m < n$

⟨*proof*⟩

lemma *zero-less-add*: $[\![n \in nat;\ m \in nat]\!] \Longrightarrow 0 < m \#{+}\ n \longleftrightarrow (0{<}m \mid 0{<}n)$
⟨*proof*⟩

27.4 Monotonicity of Addition

lemma *add-lt-mono1*: $[\![i{<}j;\ j{\in}nat]\!] \Longrightarrow i\#{+}k < j\#{+}k$
⟨*proof*⟩

strict, in second argument

lemma *add-lt-mono2*: $[\![i{<}j;\ j{\in}nat]\!] \Longrightarrow k\#{+}i < k\#{+}j$
⟨*proof*⟩

A [clumsy] way of lifting < monotonicity to ≤ monotonicity

lemma *Ord-lt-mono-imp-le-mono*:
 assumes *lt-mono*: $\bigwedge i\ j.\ [\![i{<}j;\ j{:}k]\!] \Longrightarrow f(i) < f(j)$
  and *ford*: $\bigwedge i.\ i{:}k \Longrightarrow Ord(f(i))$
  and *leij*: $i \le j$
  and *jink*: $j{:}k$
 shows $f(i) \le f(j)$
⟨*proof*⟩

≤ monotonicity, 1st argument

lemma *add-le-mono1*: $[\![i \le j;\ j{\in}nat]\!] \Longrightarrow i\#{+}k \le j\#{+}k$
⟨*proof*⟩

≤ monotonicity, both arguments

lemma *add-le-mono*: $[\![i \le j;\ k \le l;\ j{\in}nat;\ l{\in}nat]\!] \Longrightarrow i\#{+}k \le j\#{+}l$
⟨*proof*⟩

Combinations of less-than and less-than-or-equals

lemma *add-lt-le-mono*: $[\![i{<}j;\ k{\le}l;\ j{\in}nat;\ l{\in}nat]\!] \Longrightarrow i\#{+}k < j\#{+}l$
⟨*proof*⟩

lemma *add-le-lt-mono*: $[\![i{\le}j;\ k{<}l;\ j{\in}nat;\ l{\in}nat]\!] \Longrightarrow i\#{+}k < j\#{+}l$
⟨*proof*⟩

Less-than: in other words, strict in both arguments

lemma *add-lt-mono*: $[\![i{<}j;\ k{<}l;\ j{\in}nat;\ l{\in}nat]\!] \Longrightarrow i\#{+}k < j\#{+}l$
⟨*proof*⟩

lemma *diff-add-inverse*: $(n\#{+}m)\ \#{-}\ n = natify(m)$
⟨*proof*⟩

lemma *diff-add-inverse2*: $(m\#{+}n)\ \#{-}\ n = natify(m)$

⟨*proof*⟩

lemma *diff-cancel*: $(k\#{+}m)\ \#{-}\ (k\#{+}n) = m\ \#{-}\ n$
⟨*proof*⟩

lemma *diff-cancel2*: $(m\#{+}k)\ \#{-}\ (n\#{+}k) = m\ \#{-}\ n$
⟨*proof*⟩

lemma *diff-add-0*: $n\ \#{-}\ (n\#{+}m) = 0$
⟨*proof*⟩

lemma *pred-0* [*simp*]: $pred(0) = 0$
⟨*proof*⟩

lemma *eq-succ-imp-eq-m1*: $[\![i = succ(j);\ i{\in}nat]\!] \Longrightarrow j = i\ \#{-}\ 1 \wedge j \in nat$
⟨*proof*⟩

lemma *pred-Un-distrib*:
$[\![i{\in}nat;\ j{\in}nat]\!] \Longrightarrow pred(i \cup j) = pred(i) \cup pred(j)$
⟨*proof*⟩

lemma *pred-type* [*TC*,*simp*]:
$i \in nat \Longrightarrow pred(i) \in nat$
⟨*proof*⟩

lemma *nat-diff-pred*: $[\![i{\in}nat;\ j{\in}nat]\!] \Longrightarrow i\ \#{-}\ succ(j) = pred(i\ \#{-}\ j)$
⟨*proof*⟩

lemma *diff-succ-eq-pred*: $i\ \#{-}\ succ(j) = pred(i\ \#{-}\ j)$
⟨*proof*⟩

lemma *nat-diff-Un-distrib*:
$[\![i{\in}nat;\ j{\in}nat;\ k{\in}nat]\!] \Longrightarrow (i \cup j)\ \#{-}\ k = (i\#{-}k) \cup (j\#{-}k)$
⟨*proof*⟩

lemma *diff-Un-distrib*:
$[\![i{\in}nat;\ j{\in}nat]\!] \Longrightarrow (i \cup j)\ \#{-}\ k = (i\#{-}k) \cup (j\#{-}k)$
⟨*proof*⟩

We actually prove $i\ \#{-}\ j\ \#{-}\ k = i\ \#{-}\ (j\ \#{+}\ k)$

lemma *diff-diff-left* [*simplified*]:
$natify(i)\#{-}natify(j)\#{-}k = natify(i)\ \#{-}\ (natify(j)\#{+}k)$
⟨*proof*⟩

lemma *eq-add-iff*: $(u\ \#{+}\ m = u\ \#{+}\ n) \longleftrightarrow (0\ \#{+}\ m = natify(n))$
⟨*proof*⟩

lemma *less-add-iff*: $(u \#+ m < u \#+ n) \longleftrightarrow (0 \#+ m < natify(n))$
⟨*proof*⟩

lemma *diff-add-eq*: $((u \#+ m) \#- (u \#+ n)) = ((0 \#+ m) \#- n)$
⟨*proof*⟩

lemma *eq-cong2*: $u = u' \Longrightarrow (t \equiv u) \equiv (t \equiv u')$
⟨*proof*⟩

lemma *iff-cong2*: $u \longleftrightarrow u' \Longrightarrow (t \equiv u) \equiv (t \equiv u')$
⟨*proof*⟩

27.5 Multiplication

lemma *mult-0* [*simp*]: $0 \#* m = 0$
⟨*proof*⟩

lemma *mult-succ* [*simp*]: $succ(m) \#* n = n \#+ (m \#* n)$
⟨*proof*⟩

lemma *mult-0-right* [*simp*]: $m \#* 0 = 0$
⟨*proof*⟩

lemma *mult-succ-right* [*simp*]: $m \#* succ(n) = m \#+ (m \#* n)$
⟨*proof*⟩

lemma *mult-1-natify* [*simp*]: $1 \#* n = natify(n)$
⟨*proof*⟩

lemma *mult-1-right-natify* [*simp*]: $n \#* 1 = natify(n)$
⟨*proof*⟩

lemma *mult-1*: $n \in nat \Longrightarrow 1 \#* n = n$
⟨*proof*⟩

lemma *mult-1-right*: $n \in nat \Longrightarrow n \#* 1 = n$
⟨*proof*⟩

lemma *mult-commute*: $m \#* n = n \#* m$
⟨*proof*⟩

lemma *add-mult-distrib*: $(m \#+ n) \#* k = (m \#* k) \#+ (n \#* k)$
⟨*proof*⟩

lemma *add-mult-distrib-left*: $k \#* (m \#+ n) = (k \#* m) \#+ (k \#* n)$
⟨*proof*⟩

lemma *mult-assoc*: $(m \#* n) \#* k = m \#* (n \#* k)$
⟨*proof*⟩

lemma *mult-left-commute*: $m \#* (n \#* k) = n \#* (m \#* k)$
⟨*proof*⟩

lemmas *mult-ac* = *mult-assoc mult-commute mult-left-commute*

lemma *lt-succ-eq-0-disj*:
$[\![m \in nat; n \in nat]\!]$
$\Longrightarrow (m < succ(n)) \longleftrightarrow (m = 0 \mid (\exists j \in nat.\ m = succ(j) \wedge j < n))$
⟨*proof*⟩

lemma *less-diff-conv* [*rule-format*]:
$[\![j \in nat; k \in nat]\!] \Longrightarrow \forall i \in nat.\ (i < j \#- k) \longleftrightarrow (i \#+ k < j)$
⟨*proof*⟩

lemmas *nat-typechecks* = *rec-type nat-0I nat-1I nat-succI Ord-nat*

end

28 Arithmetic with simplification

theory *ArithSimp*
imports *Arith*
begin

28.1 Arithmetic simplification

⟨*ML*⟩

28.1.1 Examples

lemma $x \#+ y = x \#+ z$ ⟨*proof*⟩
lemma $y \#+ x = x \#+ z$ ⟨*proof*⟩
lemma $x \#+ y \#+ z = x \#+ z$ ⟨*proof*⟩
lemma $y \#+ (z \#+ x) = z \#+ x$ ⟨*proof*⟩
lemma $x \#+ y \#+ z = (z \#+ y) \#+ (x \#+ w)$ ⟨*proof*⟩
lemma $x\#*y \#+ z = (z \#+ y) \#+ (y\#*x \#+ w)$ ⟨*proof*⟩

lemma $x \#+ succ(y) = x \#+ z$ ⟨*proof*⟩

lemma $x \#+ succ(y) = succ(z \#+ x)$ ⟨*proof*⟩
lemma $succ(x) \#+ succ(y) \#+ z = succ(z \#+ y) \#+ succ(x \#+ w)$ ⟨*proof*⟩

lemma $(x \#+ y) \#- (x \#+ z) = w$ ⟨*proof*⟩
lemma $(y \#+ x) \#- (x \#+ z) = dd$ ⟨*proof*⟩
lemma $(x \#+ y \#+ z) \#- (x \#+ z) = dd$ ⟨*proof*⟩
lemma $(y \#+ (z \#+ x)) \#- (z \#+ x) = dd$ ⟨*proof*⟩
lemma $(x \#+ y \#+ z) \#- ((z \#+ y) \#+ (x \#+ w)) = dd$ ⟨*proof*⟩
lemma $(x\#*y \#+ z) \#- ((z \#+ y) \#+ (y\#*x \#+ w)) = dd$ ⟨*proof*⟩

lemma $(x \#+ succ(y)) \#- (x \#+ z) = dd$ ⟨*proof*⟩

lemma $x \#* y2 \#+ y \#* x2 = y \#* x2 \#+ x \#* y2$ ⟨*proof*⟩

lemma $(x \#+ succ(y)) \#- (succ(z \#+ x)) = dd$ ⟨*proof*⟩
lemma $(succ(x) \#+ succ(y) \#+ z) \#- (succ(z \#+ y) \#+ succ(x \#+ w)) = dd$
⟨*proof*⟩

lemma $x : nat ==> x \#+ y = x$ ⟨*proof*⟩
lemma $x : nat --> x \#+ y = x$ ⟨*proof*⟩
lemma $x : nat ==> x \#+ y < x$ ⟨*proof*⟩
lemma $x : nat ==> x < y\#+x$ ⟨*proof*⟩
lemma $x : nat ==> x \le succ(x)$ ⟨*proof*⟩

lemma $x \#+ y = x$ ⟨*proof*⟩

lemma $x \#+ y < x \#+ z$ ⟨*proof*⟩
lemma $y \#+ x < x \#+ z$ ⟨*proof*⟩
lemma $x \#+ y \#+ z < x \#+ z$ ⟨*proof*⟩
lemma $y \#+ z \#+ x < x \#+ z$ ⟨*proof*⟩
lemma $y \#+ (z \#+ x) < z \#+ x$ ⟨*proof*⟩
lemma $x \#+ y \#+ z < (z \#+ y) \#+ (x \#+ w)$ ⟨*proof*⟩
lemma $x\#*y \#+ z < (z \#+ y) \#+ (y\#*x \#+ w)$ ⟨*proof*⟩

lemma $x \#+ succ(y) < x \#+ z$ ⟨*proof*⟩
lemma $x \#+ succ(y) < succ(z \#+ x)$ ⟨*proof*⟩
lemma $succ(x) \#+ succ(y) \#+ z < succ(z \#+ y) \#+ succ(x \#+ w)$ ⟨*proof*⟩

lemma $x \#+ succ(y) \le succ(z \#+ x)$ ⟨*proof*⟩

28.2 Difference

lemma *diff-self-eq-0* [*simp*]: $m \#- m = 0$
⟨*proof*⟩

lemma *add-diff-inverse*: $[\![n \leq m;\ \ m{:}nat]\!] \Longrightarrow n\ \#{+}\ (m\#{-}n) = m$
⟨*proof*⟩

lemma *add-diff-inverse2*: $[\![n \leq m;\ \ m{:}nat]\!] \Longrightarrow (m\#{-}n)\ \#{+}\ n = m$
⟨*proof*⟩

lemma *diff-succ*: $[\![n \leq m;\ \ m{:}nat]\!] \Longrightarrow succ(m)\ \#{-}\ n = succ(m\#{-}n)$
⟨*proof*⟩

lemma *zero-less-diff* [*simp*]:
$[\![m{:}\ nat;\ n{:}\ nat]\!] \Longrightarrow 0 < (n\ \#{-}\ m) \quad \longleftrightarrow \quad m{<}n$
⟨*proof*⟩

lemma *diff-mult-distrib*: $(m\ \#{-}\ n)\ \#{*}\ k = (m\ \#{*}\ k)\ \#{-}\ (n\ \#{*}\ k)$
⟨*proof*⟩

lemma *diff-mult-distrib2*: $k\ \#{*}\ (m\ \#{-}\ n) = (k\ \#{*}\ m)\ \#{-}\ (k\ \#{*}\ n)$
⟨*proof*⟩

28.3 Remainder

lemma *div-termination*: $[\![0{<}n;\ \ n \leq m;\ \ m{:}nat]\!] \Longrightarrow m\ \#{-}\ n < m$
⟨*proof*⟩

lemmas *div-rls* =
nat-typechecks Ord-transrec-type apply-funtype
div-termination [*THEN ltD*]
nat-into-Ord not-lt-iff-le [*THEN iffD1*]

lemma *raw-mod-type*: $[\![m{:}nat;\ \ n{:}nat]\!] \Longrightarrow raw\text{-}mod\ (m,\ n) \in nat$
⟨*proof*⟩

lemma *mod-type* [*TC*,*iff*]: $m\ mod\ n \in nat$
⟨*proof*⟩

lemma *DIVISION-BY-ZERO-DIV*: $a\ div\ 0 = 0$
⟨*proof*⟩

lemma *DIVISION-BY-ZERO-MOD*: $a\ mod\ 0 = natify(a)$

⟨*proof*⟩

lemma *raw-mod-less*: $m<n \Longrightarrow$ *raw-mod* $(m,n) = m$
⟨*proof*⟩

lemma *mod-less* [*simp*]: $[\![m<n;\ n \in nat]\!] \Longrightarrow m\ mod\ n = m$
⟨*proof*⟩

lemma *raw-mod-geq*:
$[\![0<n;\ n \leq m;\ \ m{:}nat]\!] \Longrightarrow$ *raw-mod* $(m,\ n) =$ *raw-mod* $(m\#-n,\ n)$
⟨*proof*⟩

lemma *mod-geq*: $[\![n \leq m;\ \ m{:}nat]\!] \Longrightarrow m\ mod\ n = (m\#-n)\ mod\ n$
⟨*proof*⟩

28.4 Division

lemma *raw-div-type*: $[\![m{:}nat;\ \ n{:}nat]\!] \Longrightarrow$ *raw-div* $(m,\ n) \in nat$
⟨*proof*⟩

lemma *div-type* [*TC*,*iff*]: $m\ div\ n \in nat$
⟨*proof*⟩

lemma *raw-div-less*: $m<n \Longrightarrow$ *raw-div* $(m,n) = 0$
⟨*proof*⟩

lemma *div-less* [*simp*]: $[\![m<n;\ n \in nat]\!] \Longrightarrow m\ div\ n = 0$
⟨*proof*⟩

lemma *raw-div-geq*: $[\![0<n;\ \ n \leq m;\ \ m{:}nat]\!] \Longrightarrow$ *raw-div*$(m,n) =$ *succ*(*raw-div*$(m\#-n,$
$n)$)
⟨*proof*⟩

lemma *div-geq* [*simp*]:
$[\![0<n;\ \ n \leq m;\ \ m{:}nat]\!] \Longrightarrow m\ div\ n = succ\ ((m\#-n)\ div\ n)$
⟨*proof*⟩

declare *div-less* [*simp*] *div-geq* [*simp*]

lemma *mod-div-lemma*: $[\![m{:}\ nat;\ \ n{:}\ nat]\!] \Longrightarrow (m\ div\ n)\#*n\ \#+\ m\ mod\ n = m$
⟨*proof*⟩

lemma *mod-div-equality-natify*: $(m\ div\ n)\#*n\ \#+\ m\ mod\ n = natify(m)$
⟨*proof*⟩

lemma *mod-div-equality*: $m{:}\ nat \Longrightarrow (m\ div\ n)\#*n\ \#+\ m\ mod\ n = m$

⟨*proof*⟩

28.5 Further Facts about Remainder

(mainly for mutilated chess board)

lemma *mod-succ-lemma*:
$[\![0<n; \ m{:}nat; \ n{:}nat]\!]$
$\Longrightarrow succ(m) \ mod \ n = (if \ succ(m \ mod \ n) = n \ then \ 0 \ else \ succ(m \ mod \ n))$
⟨*proof*⟩

lemma *mod-succ*:
$n{:}nat \Longrightarrow succ(m) \ mod \ n = (if \ succ(m \ mod \ n) = n \ then \ 0 \ else \ succ(m \ mod \ n))$
⟨*proof*⟩

lemma *mod-less-divisor*: $[\![0<n; \ n{:}nat]\!] \Longrightarrow m \ mod \ n < n$
⟨*proof*⟩

lemma *mod-1-eq* [*simp*]: $m \ mod \ 1 = 0$
⟨*proof*⟩

lemma *mod2-cases*: $b<2 \Longrightarrow k \ mod \ 2 = b \mid k \ mod \ 2 = (if \ b=1 \ then \ 0 \ else \ 1)$
⟨*proof*⟩

lemma *mod2-succ-succ* [*simp*]: $succ(succ(m)) \ mod \ 2 = m \ mod \ 2$
⟨*proof*⟩

lemma *mod2-add-more* [*simp*]: $(m\#{+}m\#{+}n) \ mod \ 2 = n \ mod \ 2$
⟨*proof*⟩

lemma *mod2-add-self* [*simp*]: $(m\#{+}m) \ mod \ 2 = 0$
⟨*proof*⟩

28.6 Additional theorems about ≤

lemma *add-le-self*: $m{:}nat \Longrightarrow m \le (m \ \#{+} \ n)$
⟨*proof*⟩

lemma *add-le-self2*: $m{:}nat \Longrightarrow m \le (n \ \#{+} \ m)$
⟨*proof*⟩

lemma *mult-le-mono1*: $[\![i \le j; \ j{:}nat]\!] \Longrightarrow (i\#{*}k) \le (j\#{*}k)$
⟨*proof*⟩

lemma *mult-le-mono*: $[\![i \le j; \ k \le l; \ j{:}nat; \ l{:}nat]\!] \Longrightarrow i\#{*}k \le j\#{*}l$
⟨*proof*⟩

lemma *mult-lt-mono2*: $[\![i<j;\ 0<k;\ j{:}nat;\ k{:}nat]\!] \Longrightarrow k\#{*}i < k\#{*}j$
⟨*proof*⟩

lemma *mult-lt-mono1*: $[\![i<j;\ 0<k;\ j{:}nat;\ k{:}nat]\!] \Longrightarrow i\#{*}k < j\#{*}k$
⟨*proof*⟩

lemma *add-eq-0-iff* [*iff*]: $m\#{+}n = 0 \longleftrightarrow natify(m)=0 \wedge natify(n)=0$
⟨*proof*⟩

lemma *zero-lt-mult-iff* [*iff*]: $0 < m\#{*}n \longleftrightarrow 0 < natify(m) \wedge 0 < natify(n)$
⟨*proof*⟩

lemma *mult-eq-1-iff* [*iff*]: $m\#{*}n = 1 \longleftrightarrow natify(m)=1 \wedge natify(n)=1$
⟨*proof*⟩

lemma *mult-is-zero*: $[\![m{:}\ nat;\ n{:}\ nat]\!] \Longrightarrow (m \#{*}\ n = 0) \longleftrightarrow (m = 0 \mid n = 0)$
⟨*proof*⟩

lemma *mult-is-zero-natify* [*iff*]:
$(m \#{*}\ n = 0) \longleftrightarrow (natify(m) = 0 \mid natify(n) = 0)$
⟨*proof*⟩

28.7 Cancellation Laws for Common Factors in Comparisons

lemma *mult-less-cancel-lemma*:
$[\![k{:}\ nat;\ m{:}\ nat;\ n{:}\ nat]\!] \Longrightarrow (m\#{*}k < n\#{*}k) \longleftrightarrow (0<k \wedge m<n)$
⟨*proof*⟩

lemma *mult-less-cancel2* [*simp*]:
$(m\#{*}k < n\#{*}k) \longleftrightarrow (0 < natify(k) \wedge natify(m) < natify(n))$
⟨*proof*⟩

lemma *mult-less-cancel1* [*simp*]:
$(k\#{*}m < k\#{*}n) \longleftrightarrow (0 < natify(k) \wedge natify(m) < natify(n))$
⟨*proof*⟩

lemma *mult-le-cancel2* [*simp*]: $(m\#{*}k \le n\#{*}k) \longleftrightarrow (0 < natify(k) \longrightarrow natify(m) \le natify(n))$
⟨*proof*⟩

lemma *mult-le-cancel1* [*simp*]: $(k\#{*}m \le k\#{*}n) \longleftrightarrow (0 < natify(k) \longrightarrow natify(m) \le natify(n))$
⟨*proof*⟩

lemma *mult-le-cancel-le1*: $k \in nat \Longrightarrow k \#{*}\ m \le k \longleftrightarrow (0 < k \longrightarrow natify(m) \le 1)$
⟨*proof*⟩

lemma *Ord-eq-iff-le*: $[\![Ord(m);\ Ord(n)]\!] \Longrightarrow m{=}n \longleftrightarrow (m \le n \wedge n \le m)$
⟨*proof*⟩

lemma *mult-cancel2-lemma*:
$[\![k\text{: } nat;\ m\text{: } nat;\ n\text{: } nat]\!] \Longrightarrow (m\#{*}k = n\#{*}k) \longleftrightarrow (m{=}n \mid k{=}0)$
⟨*proof*⟩

lemma *mult-cancel2* [*simp*]:
$(m\#{*}k = n\#{*}k) \longleftrightarrow (natify(m) = natify(n) \mid natify(k) = 0)$
⟨*proof*⟩

lemma *mult-cancel1* [*simp*]:
$(k\#{*}m = k\#{*}n) \longleftrightarrow (natify(m) = natify(n) \mid natify(k) = 0)$
⟨*proof*⟩

lemma *div-cancel-raw*:
$[\![0{<}n;\ 0{<}k;\ k{:}nat;\ m{:}nat;\ n{:}nat]\!] \Longrightarrow (k\#{*}m)\ div\ (k\#{*}n) = m\ div\ n$
⟨*proof*⟩

lemma *div-cancel*:
$[\![0 < natify(n);\ \ 0 < natify(k)]\!] \Longrightarrow (k\#{*}m)\ div\ (k\#{*}n) = m\ div\ n$
⟨*proof*⟩

28.8 More Lemmas about Remainder

lemma *mult-mod-distrib-raw*:
$[\![k{:}nat;\ m{:}nat;\ n{:}nat]\!] \Longrightarrow (k\#{*}m)\ mod\ (k\#{*}n) = k\ \#{*}\ (m\ mod\ n)$
⟨*proof*⟩

lemma *mod-mult-distrib2*: $k\ \#{*}\ (m\ mod\ n) = (k\#{*}m)\ mod\ (k\#{*}n)$
⟨*proof*⟩

lemma *mult-mod-distrib*: $(m\ mod\ n)\ \#{*}\ k = (m\#{*}k)\ mod\ (n\#{*}k)$
⟨*proof*⟩

lemma *mod-add-self2-raw*: $n \in nat \Longrightarrow (m\ \#{+}\ n)\ mod\ n = m\ mod\ n$
⟨*proof*⟩

lemma *mod-add-self2* [*simp*]: $(m\ \#{+}\ n)\ mod\ n = m\ mod\ n$
⟨*proof*⟩

lemma *mod-add-self1* [*simp*]: $(n\#{+}m)\ mod\ n = m\ mod\ n$
⟨*proof*⟩

lemma *mod-mult-self1-raw*: $k \in nat \Longrightarrow (m\ \#{+}\ k\#{*}n)\ mod\ n = m\ mod\ n$

⟨*proof*⟩

lemma *mod-mult-self1* [*simp*]: $(m \#+ k\#*n)\ mod\ n = m\ mod\ n$
⟨*proof*⟩

lemma *mod-mult-self2* [*simp*]: $(m \#+ n\#*k)\ mod\ n = m\ mod\ n$
⟨*proof*⟩

lemma *mult-eq-self-implies-10*: $m = m\#*n \Longrightarrow natify(n)=1 \mid m=0$
⟨*proof*⟩

lemma *less-imp-succ-add* [*rule-format*]:
$[\![m<n;\ n:\ nat]\!] \Longrightarrow \exists\, k\in nat.\ n = succ(m\#+k)$
⟨*proof*⟩

lemma *less-iff-succ-add*:
$[\![m:\ nat;\ n:\ nat]\!] \Longrightarrow (m<n) \longleftrightarrow (\exists\, k\in nat.\ n = succ(m\#+k))$
⟨*proof*⟩

lemma *add-lt-elim2*:
$[\![a \#+ d = b \#+ c;\ a < b;\ b \in nat;\ c \in nat;\ d \in nat]\!] \Longrightarrow c < d$
⟨*proof*⟩

lemma *add-le-elim2*:
$[\![a \#+ d = b \#+ c;\ a \le b;\ b \in nat;\ c \in nat;\ d \in nat]\!] \Longrightarrow c \le d$
⟨*proof*⟩

28.8.1 More Lemmas About Difference

lemma *diff-is-0-lemma*:
$[\![m:\ nat;\ n:\ nat]\!] \Longrightarrow m \#- n = 0 \longleftrightarrow m \le n$
⟨*proof*⟩

lemma *diff-is-0-iff*: $m \#- n = 0 \longleftrightarrow natify(m) \le natify(n)$
⟨*proof*⟩

lemma *nat-lt-imp-diff-eq-0*:
$[\![a{:}nat;\ b{:}nat;\ a<b]\!] \Longrightarrow a \#- b = 0$
⟨*proof*⟩

lemma *raw-nat-diff-split*:
$[\![a{:}nat;\ b{:}nat]\!] \Longrightarrow$
$(P(a \#- b)) \longleftrightarrow ((a < b \longrightarrow P(0)) \wedge (\forall\, d\in nat.\ a = b \#+ d \longrightarrow P(d)))$
⟨*proof*⟩

lemma *nat-diff-split*:
$(P(a \#- b)) \longleftrightarrow$
$(natify(a) < natify(b) \longrightarrow P(0)) \wedge (\forall\, d\in nat.\ natify(a) = b \#+ d \longrightarrow P(d))$

⟨*proof*⟩

Difference and less-than

lemma *diff-lt-imp-lt*: $[\![(k\#-i) < (k\#-j);\ i\in nat;\ j\in nat;\ k\in nat]\!] \Longrightarrow j<i$
⟨*proof*⟩

lemma *lt-imp-diff-lt*: $[\![j<i;\ i\le k;\ k\in nat]\!] \Longrightarrow (k\#-i) < (k\#-j)$
⟨*proof*⟩

lemma *diff-lt-iff-lt*: $[\![i\le k;\ j\in nat;\ k\in nat]\!] \Longrightarrow (k\#-i) < (k\#-j) \longleftrightarrow j<i$
⟨*proof*⟩

end

29 Lists in Zermelo-Fraenkel Set Theory

theory *List* **imports** *Datatype ArithSimp* **begin**

consts
  list :: *i*⇒*i*

datatype
  list(*A*) = *Nil* | *Cons* (*a* ∈ *A*, *l* ∈ *list*(*A*))

notation *Nil* (‹[]›)

syntax
  -List :: *is* ⇒ *i* (‹(‹*indent=1 notation=*‹*mixfix list enumeration*››[-])›)
translations
  [*x*, *xs*] == *CONST Cons*(*x*, [*xs*])
  [*x*] == *CONST Cons*(*x*, [])

consts
  length :: *i*⇒*i*
  hd :: *i*⇒*i*
  tl :: *i*⇒*i*

primrec
  length([]) = *0*
  length(*Cons*(*a*,*l*)) = *succ*(*length*(*l*))

primrec
  hd([]) = *0*
  hd(*Cons*(*a*,*l*)) = *a*

primrec
  tl([]) = []
  tl(*Cons*(*a*,*l*)) = *l*

consts
map :: $[i \Rightarrow i,\ i] \Rightarrow i$
set-of-list :: $i \Rightarrow i$
app :: $[i,i] \Rightarrow i$ (**infixr** ‹@› *60*)

primrec
$map(f,[]) = []$
$map(f,Cons(a,l)) = Cons(f(a),\ map(f,l))$

primrec
$set\text{-}of\text{-}list([]) = 0$
$set\text{-}of\text{-}list(Cons(a,l)) = cons(a,\ set\text{-}of\text{-}list(l))$

primrec
app-Nil: $[]\ @\ ys = ys$
app-Cons: $(Cons(a,l))\ @\ ys = Cons(a,\ l\ @\ ys)$

consts
rev :: $i \Rightarrow i$
flat :: $i \Rightarrow i$
list-add :: $i \Rightarrow i$

primrec
$rev([]) = []$
$rev(Cons(a,l)) = rev(l)\ @\ [a]$

primrec
$flat([]) = []$
$flat(Cons(l,ls)) = l\ @\ flat(ls)$

primrec
$list\text{-}add([]) = 0$
$list\text{-}add(Cons(a,l)) = a\ \#+\ list\text{-}add(l)$

consts
drop :: $[i,i] \Rightarrow i$

primrec
drop-0: $drop(0,l) = l$
drop-succ: $drop(succ(i),\ l) = tl\ (drop(i,l))$

definition

take :: $[i,i]\Rightarrow i$ **where**
take$(n,\ as) \equiv$ *list-rec*$(\lambda n{\in}nat.\ [],$
$\lambda a\ l\ r.\ \lambda n{\in}nat.\ nat\text{-}case([],\ \lambda m.\ Cons(a,\ r`m),\ n),\ as)`n$

definition
nth :: $[i,\ i]\Rightarrow i$ **where**
— returns the (n+1)th element of a list, or 0 if the list is too short.
nth$(n,\ as) \equiv$ *list-rec*$(\lambda n{\in}nat.\ 0,$
$\lambda a\ l\ r.\ \lambda n{\in}nat.\ nat\text{-}case(a,\ \lambda m.\ r`m,\ n),\ as)\ `\ n$

definition
list-update :: $[i,\ i,\ i]\Rightarrow i$ **where**
list-update$(xs,\ i,\ v) \equiv$ *list-rec*$(\lambda n{\in}nat.\ Nil,$
$\lambda u\ us\ vs.\ \lambda n{\in}nat.\ nat\text{-}case(Cons(v,\ us),\ \lambda m.\ Cons(u,\ vs`m),\ n),\ xs)`i$

consts
filter :: $[i\Rightarrow o,\ i] \Rightarrow i$
upt :: $[i,\ i] \Rightarrow i$

primrec
filter$(P,\ Nil) = Nil$
filter$(P,\ Cons(x,\ xs)) =$
$(if\ P(x)\ then\ Cons(x,\ filter(P,\ xs))\ else\ filter(P,\ xs))$

primrec
upt$(i,\ 0) = Nil$
upt$(i,\ succ(j)) = (if\ i \le j\ then\ upt(i,\ j)@[j]\ else\ Nil)$

definition
min :: $[i,i] \Rightarrow i$ **where**
min$(x,\ y) \equiv (if\ x \le y\ then\ x\ else\ y)$

definition
max :: $[i,\ i] \Rightarrow i$ **where**
max$(x,\ y) \equiv (if\ x \le y\ then\ y\ else\ x)$

declare *list.intros* $[simp, TC]$

inductive-cases *ConsE*: $Cons(a,l) \in list(A)$

lemma *Cons-type-iff* [*simp*]: $Cons(a,l) \in list(A) \longleftrightarrow a \in A \land l \in list(A)$
⟨*proof*⟩

lemma *Cons-iff*: $Cons(a,l)=Cons(a',l') \longleftrightarrow a=a' \land l=l'$

⟨*proof*⟩

lemma *Nil-Cons-iff*: $\neg$ *Nil*=*Cons*(*a*,*l*)
⟨*proof*⟩

lemma *list-unfold*: *list*(*A*) = {*0*} + (*A* $*$ *list*(*A*))
⟨*proof*⟩

lemma *list-mono*: *A*<=*B* $\Longrightarrow$ *list*(*A*) $\subseteq$ *list*(*B*)
⟨*proof*⟩

lemma *list-univ*: *list*(*univ*(*A*)) $\subseteq$ *univ*(*A*)
⟨*proof*⟩

lemmas *list-subset-univ* = *subset-trans* [*OF list-mono list-univ*]

lemma *list-into-univ*: ⟦*l* $\in$ *list*(*A*); *A* $\subseteq$ *univ*(*B*)⟧ $\Longrightarrow$ *l* $\in$ *univ*(*B*)
⟨*proof*⟩

lemma *list-case-type*:
⟦*l* $\in$ *list*(*A*);
c $\in$ *C*(*Nil*);
$\bigwedge$*x y*. ⟦*x* $\in$ *A*; *y* $\in$ *list*(*A*)⟧ $\Longrightarrow$ *h*(*x*,*y*): *C*(*Cons*(*x*,*y*))
⟧ $\Longrightarrow$ *list-case*(*c*,*h*,*l*) $\in$ *C*(*l*)
⟨*proof*⟩

lemma *list-0-triv*: *list*(*0*) = {*Nil*}
⟨*proof*⟩

lemma *tl-type*: *l* $\in$ *list*(*A*) $\Longrightarrow$ *tl*(*l*) $\in$ *list*(*A*)
⟨*proof*⟩

lemma *drop-Nil* [*simp*]: *i* $\in$ *nat* $\Longrightarrow$ *drop*(*i*, *Nil*) = *Nil*
⟨*proof*⟩

lemma *drop-succ-Cons* [*simp*]: *i* $\in$ *nat* $\Longrightarrow$ *drop*(*succ*(*i*), *Cons*(*a*,*l*)) = *drop*(*i*,*l*)
⟨*proof*⟩

lemma *drop-type* [*simp*,*TC*]: ⟦*i* $\in$ *nat*; *l* $\in$ *list*(*A*)⟧ $\Longrightarrow$ *drop*(*i*,*l*) $\in$ *list*(*A*)

⟨*proof*⟩

declare *drop-succ* [*simp del*]

lemma *list-rec-type* [*TC*]:
$[\![l \in list(A);$
$c \in C(Nil);$
$\bigwedge x\ y\ r.\ [\![x \in A;\ \ y \in list(A);\ \ r \in C(y)]\!] \Longrightarrow h(x,y,r){:}\ C(Cons(x,y))$
$]\!] \Longrightarrow list\text{-}rec(c,h,l) \in C(l)$
⟨*proof*⟩

lemma *map-type* [*TC*]:
$[\![l \in list(A);\ \ \bigwedge x.\ x \in A \Longrightarrow h(x){:}\ B]\!] \Longrightarrow map(h,l) \in list(B)$
⟨*proof*⟩

lemma *map-type2* [*TC*]: $l \in list(A) \Longrightarrow map(h,l) \in list(\{h(u).\ u \in A\})$
⟨*proof*⟩

lemma *length-type* [*TC*]: $l \in list(A) \Longrightarrow length(l) \in nat$
⟨*proof*⟩

lemma *lt-length-in-nat*:
$[\![x < length(xs);\ xs \in list(A)]\!] \Longrightarrow x \in nat$
⟨*proof*⟩

lemma *app-type* [*TC*]: $[\![xs{:}\ list(A);\ \ ys{:}\ list(A)]\!] \Longrightarrow xs@ys \in list(A)$
⟨*proof*⟩

lemma *rev-type* [*TC*]: $xs{:}\ list(A) \Longrightarrow rev(xs) \in list(A)$
⟨*proof*⟩

lemma *flat-type* [*TC*]: $ls{:}\ list(list(A)) \Longrightarrow flat(ls) \in list(A)$
⟨*proof*⟩

lemma *set-of-list-type* [*TC*]: $l \in list(A) \Longrightarrow set\text{-}of\text{-}list(l) \in Pow(A)$
⟨*proof*⟩

lemma *set-of-list-append*:
$xs: list(A) \Longrightarrow set\text{-}of\text{-}list\ (xs@ys) = set\text{-}of\text{-}list(xs) \cup set\text{-}of\text{-}list(ys)$
⟨*proof*⟩

lemma *list-add-type* [*TC*]: $xs: list(nat) \Longrightarrow list\text{-}add(xs) \in nat$
⟨*proof*⟩

lemma *map-ident* [*simp*]: $l \in list(A) \Longrightarrow map(\lambda u.\ u,\ l) = l$
⟨*proof*⟩

lemma *map-compose*: $l \in list(A) \Longrightarrow map(h,\ map(j,l)) = map(\lambda u.\ h(j(u)),\ l)$
⟨*proof*⟩

lemma *map-app-distrib*: $xs: list(A) \Longrightarrow map(h,\ xs@ys) = map(h,xs)\ @\ map(h,ys)$
⟨*proof*⟩

lemma *map-flat*: $ls: list(list(A)) \Longrightarrow map(h,\ flat(ls)) = flat(map(map(h),ls))$
⟨*proof*⟩

lemma *list-rec-map*:
$l \in list(A) \Longrightarrow$
$list\text{-}rec(c,\ d,\ map(h,l)) =$
$list\text{-}rec(c,\ \lambda x\ xs\ r.\ d(h(x),\ map(h,xs),\ r),\ l)$
⟨*proof*⟩

lemmas *list-CollectD* = *Collect-subset* [*THEN list-mono*, *THEN subsetD*]

lemma *map-list-Collect*: $l \in list(\{x \in A.\ h(x)=j(x)\}) \Longrightarrow map(h,l) = map(j,l)$
⟨*proof*⟩

lemma *length-map* [*simp*]: $xs: list(A) \Longrightarrow length(map(h,xs)) = length(xs)$
⟨*proof*⟩

lemma *length-app* [*simp*]:
$[\![xs\colon list(A);\ ys\colon list(A)]\!]$
$\Longrightarrow length(xs@ys) = length(xs) \#+ length(ys)$
⟨*proof*⟩

lemma *length-rev* [*simp*]: $xs\colon list(A) \Longrightarrow length(rev(xs)) = length(xs)$
⟨*proof*⟩

lemma *length-flat*:
$ls\colon list(list(A)) \Longrightarrow length(flat(ls)) = list\text{-}add(map(length,ls))$
⟨*proof*⟩

lemma *drop-length-Cons* [*rule-format*]:
$xs\colon list(A) \Longrightarrow$
$\forall x.\ \exists z\ zs.\ drop(length(xs),\ Cons(x,xs)) = Cons(z,zs)$
⟨*proof*⟩

lemma *drop-length* [*rule-format*]:
$l \in list(A) \Longrightarrow \forall i \in length(l).\ (\exists z\ zs.\ drop(i,l) = Cons(z,zs))$
⟨*proof*⟩

lemma *app-right-Nil* [*simp*]: $xs\colon list(A) \Longrightarrow xs@Nil=xs$
⟨*proof*⟩

lemma *app-assoc*: $xs\colon list(A) \Longrightarrow (xs@ys)@zs = xs@(ys@zs)$
⟨*proof*⟩

lemma *flat-app-distrib*: $ls\colon list(list(A)) \Longrightarrow flat(ls@ms) = flat(ls)@flat(ms)$
⟨*proof*⟩

lemma *rev-map-distrib*: $l \in list(A) \Longrightarrow rev(map(h,l)) = map(h,rev(l))$
⟨*proof*⟩

lemma *rev-app-distrib*:
$[\![xs\colon list(A);\ ys\colon list(A)]\!] \Longrightarrow rev(xs@ys) = rev(ys)@rev(xs)$
⟨*proof*⟩

lemma *rev-rev-ident* [*simp*]: $l \in list(A) \Longrightarrow rev(rev(l))=l$
⟨*proof*⟩

lemma *rev-flat*: *ls*: $list(list(A)) \Longrightarrow rev(flat(ls)) = flat(map(rev,rev(ls)))$
⟨*proof*⟩

lemma *list-add-app*:
$[\![xs: list(nat); \; ys: list(nat)]\!]$
$\Longrightarrow list\text{-}add(xs@ys) = list\text{-}add(ys) \; \#+ \; list\text{-}add(xs)$
⟨*proof*⟩

lemma *list-add-rev*: $l \in list(nat) \Longrightarrow list\text{-}add(rev(l)) = list\text{-}add(l)$
⟨*proof*⟩

lemma *list-add-flat*:
ls: $list(list(nat)) \Longrightarrow list\text{-}add(flat(ls)) = list\text{-}add(map(list\text{-}add,ls))$
⟨*proof*⟩

lemma *list-append-induct* [*case-names Nil snoc*, *consumes 1*]:
$[\![l \in list(A);$
$P(Nil);$
$\bigwedge x \; y. \; [\![x \in A; \; y \in list(A); \; P(y)]\!] \Longrightarrow P(y \; @ \; [x])$
$]\!] \Longrightarrow P(l)$
⟨*proof*⟩

lemma *list-complete-induct-lemma* [*rule-format*]:
assumes *ih*:
$\bigwedge l. \; [\![l \in list(A);$
$\forall \, l' \in list(A). \; length(l') < length(l) \longrightarrow P(l')]\!]$
$\Longrightarrow P(l)$
shows $n \in nat \Longrightarrow \forall \, l \in list(A). \; length(l) < n \longrightarrow P(l)$
⟨*proof*⟩

theorem *list-complete-induct*:
$[\![l \in list(A);$
$\bigwedge l. \; [\![l \in list(A);$
$\forall \, l' \in list(A). \; length(l') < length(l) \longrightarrow P(l')]\!]$
$\Longrightarrow P(l)$
$]\!] \Longrightarrow P(l)$
⟨*proof*⟩

lemma *min-sym*: $[\![i \in nat; \; j \in nat]\!] \Longrightarrow min(i,j)=min(j,i)$

$\langle proof \rangle$

lemma *min-type* [*simp,TC*]: $[\![i \in nat; j \in nat]\!] \Longrightarrow min(i,j){:}nat$
$\langle proof \rangle$

lemma *min-0* [*simp*]: $i \in nat \Longrightarrow min(0,i) = 0$
$\langle proof \rangle$

lemma *min-02* [*simp*]: $i \in nat \Longrightarrow min(i,\ 0) = 0$
$\langle proof \rangle$

lemma *lt-min-iff*: $[\![i \in nat; j \in nat; k \in nat]\!] \Longrightarrow i{<}min(j,k) \longleftrightarrow i{<}j \wedge i{<}k$
$\langle proof \rangle$

lemma *min-succ-succ* [*simp*]:
$[\![i \in nat; j \in nat]\!] \Longrightarrow\ min(succ(i),\ succ(j)) = succ(min(i,\ j))$
$\langle proof \rangle$

lemma *filter-append* [*simp*]:
$xs{:}list(A) \Longrightarrow filter(P,\ xs@ys) = filter(P,\ xs)\ @\ filter(P,\ ys)$
$\langle proof \rangle$

lemma *filter-type* [*simp,TC*]: $xs{:}list(A) \Longrightarrow filter(P,\ xs){:}list(A)$
$\langle proof \rangle$

lemma *length-filter*: $xs{:}list(A) \Longrightarrow length(filter(P,\ xs)) \le length(xs)$
$\langle proof \rangle$

lemma *filter-is-subset*: $xs{:}list(A) \Longrightarrow set\text{-}of\text{-}list(filter(P,xs)) \subseteq set\text{-}of\text{-}list(xs)$
$\langle proof \rangle$

lemma *filter-False* [*simp*]: $xs{:}list(A) \Longrightarrow filter(\lambda p.\ False,\ xs) = Nil$
$\langle proof \rangle$

lemma *filter-True* [*simp*]: $xs{:}list(A) \Longrightarrow filter(\lambda p.\ True,\ xs) = xs$
$\langle proof \rangle$

lemma *length-is-0-iff* [*simp*]: $xs{:}list(A) \Longrightarrow length(xs){=}0 \longleftrightarrow xs{=}Nil$
$\langle proof \rangle$

lemma *length-is-0-iff2* [*simp*]: $xs{:}list(A) \Longrightarrow 0 = length(xs) \longleftrightarrow xs{=}Nil$
$\langle proof \rangle$

lemma *length-tl* [*simp*]: $xs{:}list(A) \Longrightarrow length(tl(xs)) = length(xs) \#- 1$
⟨*proof*⟩

lemma *length-greater-0-iff*: $xs{:}list(A) \Longrightarrow 0<length(xs) \longleftrightarrow xs \neq Nil$
⟨*proof*⟩

lemma *length-succ-iff*: $xs{:}list(A) \Longrightarrow length(xs)=succ(n) \longleftrightarrow (\exists\, y\ ys.\ xs=Cons(y, ys) \wedge length(ys)=n)$
⟨*proof*⟩

lemma *append-is-Nil-iff* [*simp*]:
$xs{:}list(A) \Longrightarrow (xs@ys = Nil) \longleftrightarrow (xs=Nil \wedge ys = Nil)$
⟨*proof*⟩

lemma *append-is-Nil-iff2* [*simp*]:
$xs{:}list(A) \Longrightarrow (Nil = xs@ys) \longleftrightarrow (xs=Nil \wedge ys = Nil)$
⟨*proof*⟩

lemma *append-left-is-self-iff* [*simp*]:
$xs{:}list(A) \Longrightarrow (xs@ys = xs) \longleftrightarrow (ys = Nil)$
⟨*proof*⟩

lemma *append-left-is-self-iff2* [*simp*]:
$xs{:}list(A) \Longrightarrow (xs = xs@ys) \longleftrightarrow (ys = Nil)$
⟨*proof*⟩

lemma *append-left-is-Nil-iff* [*rule-format*]:
$[\![xs{:}list(A);\ ys{:}list(A);\ zs{:}list(A)]\!] \Longrightarrow$
$length(ys)=length(zs) \longrightarrow (xs@ys=zs \longleftrightarrow (xs=Nil \wedge ys=zs))$
⟨*proof*⟩

lemma *append-left-is-Nil-iff2* [*rule-format*]:
$[\![xs{:}list(A);\ ys{:}list(A);\ zs{:}list(A)]\!] \Longrightarrow$
$length(ys)=length(zs) \longrightarrow (zs=ys@xs \longleftrightarrow (xs=Nil \wedge ys=zs))$
⟨*proof*⟩

lemma *append-eq-append-iff* [*rule-format*]:
$xs{:}list(A) \Longrightarrow \forall\, ys \in list(A).$
$length(xs)=length(ys) \longrightarrow (xs@us = ys@vs) \longleftrightarrow (xs=ys \wedge us=vs)$
⟨*proof*⟩
declare *append-eq-append-iff* [*simp*]

lemma *append-eq-append* [*rule-format*]:
$xs{:}list(A) \Longrightarrow$
$\forall\, ys \in list(A).\ \forall\, us \in list(A).\ \forall\, vs \in list(A).$

$length(us) = length(vs) \longrightarrow (xs@us = ys@vs) \longrightarrow (xs=ys \wedge us=vs)$
⟨*proof*⟩

lemma *append-eq-append-iff2* [*simp*]:
$[\![xs{:}list(A);\ ys{:}list(A);\ us{:}list(A);\ vs{:}list(A);\ length(us)=length(vs)]\!]$
$\Longrightarrow\ xs@us = ys@vs \longleftrightarrow (xs=ys \wedge us=vs)$
⟨*proof*⟩

lemma *append-self-iff* [*simp*]:
$[\![xs{:}list(A);\ ys{:}list(A);\ zs{:}list(A)]\!] \Longrightarrow xs@ys=xs@zs \longleftrightarrow ys=zs$
⟨*proof*⟩

lemma *append-self-iff2* [*simp*]:
$[\![xs{:}list(A);\ ys{:}list(A);\ zs{:}list(A)]\!] \Longrightarrow ys@xs=zs@xs \longleftrightarrow ys=zs$
⟨*proof*⟩

lemma *append1-eq-iff* [*rule-format*]:
$xs{:}list(A) \Longrightarrow \forall\, ys \in list(A).\ xs@[x] = ys@[y] \longleftrightarrow (xs = ys \wedge x=y)$
⟨*proof*⟩
declare *append1-eq-iff* [*simp*]

lemma *append-right-is-self-iff* [*simp*]:
$[\![xs{:}list(A);\ ys{:}list(A)]\!] \Longrightarrow (xs@ys = ys) \longleftrightarrow (xs=Nil)$
⟨*proof*⟩

lemma *append-right-is-self-iff2* [*simp*]:
$[\![xs{:}list(A);\ ys{:}list(A)]\!] \Longrightarrow (ys = xs@ys) \longleftrightarrow (xs=Nil)$
⟨*proof*⟩

lemma *hd-append* [*rule-format*]:
$xs{:}list(A) \Longrightarrow xs \neq Nil \longrightarrow hd(xs\ @\ ys) = hd(xs)$
⟨*proof*⟩
declare *hd-append* [*simp*]

lemma *tl-append* [*rule-format*]:
$xs{:}list(A) \Longrightarrow xs{\neq}Nil \longrightarrow tl(xs\ @\ ys) = tl(xs)@ys$
⟨*proof*⟩
declare *tl-append* [*simp*]

lemma *rev-is-Nil-iff* [*simp*]: $xs{:}list(A) \Longrightarrow (rev(xs) = Nil \longleftrightarrow xs = Nil)$
⟨*proof*⟩

lemma *Nil-is-rev-iff* [*simp*]: $xs{:}list(A) \Longrightarrow (Nil = rev(xs) \longleftrightarrow xs = Nil)$
⟨*proof*⟩

lemma *rev-is-rev-iff* [*rule-format*]:
$xs{:}list(A) \Longrightarrow \forall\, ys \in list(A).\ rev(xs)=rev(ys) \longleftrightarrow xs=ys$

⟨*proof*⟩
declare *rev-is-rev-iff* [*simp*]

lemma *rev-list-elim* [*rule-format*]:
$xs{:}list(A) \Longrightarrow$
$(xs{=}Nil \longrightarrow P) \longrightarrow (\forall ys \in list(A).\ \forall y \in A.\ xs = ys@[y] \longrightarrow P) \longrightarrow P$
⟨*proof*⟩

lemma *length-drop* [*rule-format*]:
$n \in nat \Longrightarrow \forall xs \in list(A).\ length(drop(n, xs)) = length(xs)\ \#-\ n$
⟨*proof*⟩
declare *length-drop* [*simp*]

lemma *drop-all* [*rule-format*]:
$n \in nat \Longrightarrow \forall xs \in list(A).\ length(xs) \le n \longrightarrow drop(n, xs){=}Nil$
⟨*proof*⟩
declare *drop-all* [*simp*]

lemma *drop-append* [*rule-format*]:
$n \in nat \Longrightarrow$
$\forall xs \in list(A).\ drop(n, xs@ys) = drop(n,xs)\ @\ drop(n\ \#-\ length(xs), ys)$
⟨*proof*⟩

lemma *drop-drop*:
$m \in nat \Longrightarrow \forall xs \in list(A).\ \forall n \in nat.\ drop(n, drop(m, xs)){=}drop(n\ \#+\ m, xs)$
⟨*proof*⟩

lemma *take-0* [*simp*]: $xs{:}list(A) \Longrightarrow take(0, xs) = Nil$
⟨*proof*⟩

lemma *take-succ-Cons* [*simp*]:
$n \in nat \Longrightarrow take(succ(n), Cons(a, xs)) = Cons(a, take(n, xs))$
⟨*proof*⟩

lemma *take-Nil* [*simp*]: $n \in nat \Longrightarrow take(n, Nil) = Nil$
⟨*proof*⟩

lemma *take-all* [*rule-format*]:
$n \in nat \Longrightarrow \forall xs \in list(A).\ length(xs) \le n \longrightarrow take(n, xs) = xs$
⟨*proof*⟩
declare *take-all* [*simp*]

lemma *take-type* [*rule-format*]:

$xs{:}list(A) \Longrightarrow \forall n \in nat.\ take(n,\ xs){:}list(A)$
⟨*proof*⟩
declare *take-type* [*simp*, *TC*]

lemma *take-append* [*rule-format*]:
$xs{:}list(A) \Longrightarrow$
$\forall ys \in list(A).\ \forall n \in nat.\ take(n,\ xs\ @\ ys) =$
$take(n,\ xs)\ @\ take(n\ \#-\ length(xs),\ ys)$
⟨*proof*⟩
declare *take-append* [*simp*]

lemma *take-take* [*rule-format*]:
$m \in nat \Longrightarrow$
$\forall xs \in list(A).\ \forall n \in nat.\ take(n,\ take(m,xs)) = take(min(n,\ m),\ xs)$
⟨*proof*⟩

lemma *nth-0* [*simp*]: $nth(0,\ Cons(a,\ l)) = a$
⟨*proof*⟩

lemma *nth-Cons* [*simp*]: $n \in nat \Longrightarrow nth(succ(n),\ Cons(a,l)) = nth(n,l)$
⟨*proof*⟩

lemma *nth-empty* [*simp*]: $nth(n,\ Nil) = 0$
⟨*proof*⟩

lemma *nth-type* [*rule-format*]:
$xs{:}list(A) \Longrightarrow \forall n.\ n < length(xs) \longrightarrow nth(n,xs) \in A$
⟨*proof*⟩
declare *nth-type* [*simp*, *TC*]

lemma *nth-eq-0* [*rule-format*]:
$xs{:}list(A) \Longrightarrow \forall n \in nat.\ length(xs) \le n \longrightarrow nth(n,xs) = 0$
⟨*proof*⟩

lemma *nth-append* [*rule-format*]:
$xs{:}list(A) \Longrightarrow$
$\forall n \in nat.\ nth(n,\ xs\ @\ ys) = (if\ n < length(xs)\ then\ nth(n,xs)$
$else\ nth(n\ \#-\ length(xs),\ ys))$
⟨*proof*⟩

lemma *set-of-list-conv-nth*:
$xs{:}list(A)$
$\Longrightarrow set\text{-}of\text{-}list(xs) = \{x \in A.\ \exists i{\in}nat.\ i{<}length(xs) \wedge x = nth(i,xs)\}$
⟨*proof*⟩

lemma *nth-take-lemma* [*rule-format*]:
$k \in nat \Longrightarrow$
$\forall xs \in list(A).\ (\forall ys \in list(A).\ k \le length(xs) \longrightarrow k \le length(ys) \longrightarrow$
$(\forall i \in nat.\ i<k \longrightarrow nth(i,xs) = nth(i,ys))\longrightarrow take(k,xs) = take(k,ys))$
⟨*proof*⟩

lemma *nth-equalityI* [*rule-format*]:
$[\![xs{:}list(A);\ ys{:}list(A);\ length(xs) = length(ys);$
$\forall i \in nat.\ i < length(xs) \longrightarrow nth(i,xs) = nth(i,ys)]\!]$
$\Longrightarrow xs = ys$
⟨*proof*⟩

lemma *take-equalityI* [*rule-format*]:
$[\![xs{:}list(A);\ ys{:}list(A);\ (\forall i \in nat.\ take(i,\ xs) = take(i,ys))]\!]$
$\Longrightarrow xs = ys$
⟨*proof*⟩

lemma *nth-drop* [*rule-format*]:
$n \in nat \Longrightarrow \forall i \in nat.\ \forall xs \in list(A).\ nth(i,\ drop(n,\ xs)) = nth(n \#+ i,\ xs)$
⟨*proof*⟩

lemma *take-succ* [*rule-format*]:
$xs{\in}list(A)$
$\Longrightarrow \forall i.\ i < length(xs) \longrightarrow take(succ(i),\ xs) = take(i,xs)\ @\ [nth(i,\ xs)]$
⟨*proof*⟩

lemma *take-add* [*rule-format*]:
$[\![xs{\in}list(A);\ j{\in}nat]\!]$
$\Longrightarrow \forall i{\in}nat.\ take(i \#+ j,\ xs) = take(i,xs)\ @\ take(j,\ drop(i,xs))$
⟨*proof*⟩

lemma *length-take*:
$l{\in}list(A) \Longrightarrow \forall n{\in}nat.\ length(take(n,l)) = min(n,\ length(l))$
⟨*proof*⟩

29.1 The function zip

Crafty definition to eliminate a type argument

consts
zip-aux :: $[i,i]{\Rightarrow}i$

primrec
$zip\text{-}aux(B,[]) =$
$(\lambda ys \in list(B).\ list\text{-}case([],\ \lambda y\ l.\ [],\ ys))$

$zip\text{-}aux(B,Cons(x,l)) =$
$(\lambda ys \in list(B).$

list-case(*Nil*, λy *zs*. *Cons*($\langle x,y\rangle$, *zip-aux*(*B*,*l*)‘*zs*), *ys*))

definition
zip :: [*i*, *i*]⇒*i* **where**
zip(*xs*, *ys*) ≡ *zip-aux*(*set-of-list*(*ys*),*xs*)‘*ys*

lemma *list-on-set-of-list*: *xs* ∈ *list*(*A*) ⟹ *xs* ∈ *list*(*set-of-list*(*xs*))
⟨*proof*⟩

lemma *zip-Nil* [*simp*]: *ys*:*list*(*A*) ⟹ *zip*(*Nil*, *ys*)=*Nil*
⟨*proof*⟩

lemma *zip-Nil2* [*simp*]: *xs*:*list*(*A*) ⟹ *zip*(*xs*, *Nil*)=*Nil*
⟨*proof*⟩

lemma *zip-aux-unique* [*rule-format*]:
⟦*B*<=*C*; *xs* ∈ *list*(*A*)⟧
⟹ ∀ *ys* ∈ *list*(*B*). *zip-aux*(*C*,*xs*) ‘ *ys* = *zip-aux*(*B*,*xs*) ‘ *ys*
⟨*proof*⟩

lemma *zip-Cons-Cons* [*simp*]:
⟦*xs*:*list*(*A*); *ys*:*list*(*B*); *x* ∈ *A*; *y* ∈ *B*⟧ ⟹
zip(*Cons*(*x*,*xs*), *Cons*(*y*, *ys*)) = *Cons*(⟨*x*,*y*⟩, *zip*(*xs*, *ys*))
⟨*proof*⟩

lemma *zip-type* [*rule-format*]:
xs:*list*(*A*) ⟹ ∀ *ys* ∈ *list*(*B*). *zip*(*xs*, *ys*):*list*(*A*∗*B*)
⟨*proof*⟩
declare *zip-type* [*simp*,*TC*]

lemma *length-zip* [*rule-format*]:
xs:*list*(*A*) ⟹ ∀ *ys* ∈ *list*(*B*). *length*(*zip*(*xs*,*ys*)) =
min(*length*(*xs*), *length*(*ys*))
⟨*proof*⟩
declare *length-zip* [*simp*]

lemma *zip-append1* [*rule-format*]:
⟦*ys*:*list*(*A*); *zs*:*list*(*B*)⟧ ⟹
∀ *xs* ∈ *list*(*A*). *zip*(*xs* @ *ys*, *zs*) =
zip(*xs*, *take*(*length*(*xs*), *zs*)) @ *zip*(*ys*, *drop*(*length*(*xs*),*zs*))
⟨*proof*⟩

lemma *zip-append2* [*rule-format*]:
⟦*xs*:*list*(*A*); *zs*:*list*(*B*)⟧ ⟹ ∀ *ys* ∈ *list*(*B*). *zip*(*xs*, *ys*@*zs*) =
zip(*take*(*length*(*ys*), *xs*), *ys*) @ *zip*(*drop*(*length*(*ys*), *xs*), *zs*)

⟨*proof*⟩

lemma *zip-append* [*simp*]:
⟦*length*(*xs*) = *length*(*us*); *length*(*ys*) = *length*(*vs*);
xs:*list*(*A*); *us*:*list*(*B*); *ys*:*list*(*A*); *vs*:*list*(*B*)⟧
⟹ *zip*(*xs*@*ys*,*us*@*vs*) = *zip*(*xs*, *us*) @ *zip*(*ys*, *vs*)
⟨*proof*⟩

lemma *zip-rev* [*rule-format*]:
ys:*list*(*B*) ⟹ ∀ *xs* ∈ *list*(*A*).
length(*xs*) = *length*(*ys*) ⟶ *zip*(*rev*(*xs*), *rev*(*ys*)) = *rev*(*zip*(*xs*, *ys*))
⟨*proof*⟩
declare *zip-rev* [*simp*]

lemma *nth-zip* [*rule-format*]:
ys:*list*(*B*) ⟹ ∀ *i* ∈ *nat*. ∀ *xs* ∈ *list*(*A*).
i < *length*(*xs*) ⟶ *i* < *length*(*ys*) ⟶
nth(*i*,*zip*(*xs*, *ys*)) = <*nth*(*i*,*xs*),*nth*(*i*, *ys*)>
⟨*proof*⟩
declare *nth-zip* [*simp*]

lemma *set-of-list-zip* [*rule-format*]:
⟦*xs*:*list*(*A*); *ys*:*list*(*B*); *i* ∈ *nat*⟧
⟹ *set-of-list*(*zip*(*xs*, *ys*)) =
{⟨*x*, *y*⟩:*A*∗*B*. ∃ *i*∈*nat*. *i* < *min*(*length*(*xs*), *length*(*ys*))
∧ *x* = *nth*(*i*, *xs*) ∧ *y* = *nth*(*i*, *ys*)}
⟨*proof*⟩

lemma *list-update-Nil* [*simp*]: *i* ∈ *nat* ⟹*list-update*(*Nil*, *i*, *v*) = *Nil*
⟨*proof*⟩

lemma *list-update-Cons-0* [*simp*]: *list-update*(*Cons*(*x*, *xs*), *0*, *v*)= *Cons*(*v*, *xs*)
⟨*proof*⟩

lemma *list-update-Cons-succ* [*simp*]:
n ∈ *nat* ⟹
list-update(*Cons*(*x*, *xs*), *succ*(*n*), *v*)= *Cons*(*x*, *list-update*(*xs*, *n*, *v*))
⟨*proof*⟩

lemma *list-update-type* [*rule-format*]:
⟦*xs*:*list*(*A*); *v* ∈ *A*⟧ ⟹ ∀ *n* ∈ *nat*. *list-update*(*xs*, *n*, *v*):*list*(*A*)
⟨*proof*⟩
declare *list-update-type* [*simp*,*TC*]

lemma *length-list-update* [*rule-format*]:
xs:*list*(*A*) ⟹ ∀ *i* ∈ *nat*. *length*(*list-update*(*xs*, *i*, *v*))=*length*(*xs*)

⟨*proof*⟩
declare *length-list-update* [*simp*]

lemma *nth-list-update* [*rule-format*]:
$[\![xs{:}list(A)]\!] \Longrightarrow \forall i \in nat.\ \forall j \in nat.\ i < length(xs) \longrightarrow$
$nth(j,\ list\text{-}update(xs,\ i,\ x)) = (if\ i{=}j\ then\ x\ else\ nth(j,\ xs))$
⟨*proof*⟩

lemma *nth-list-update-eq* [*simp*]:
$[\![i < length(xs);\ xs{:}list(A)]\!] \Longrightarrow nth(i,\ list\text{-}update(xs,\ i,x)) = x$
⟨*proof*⟩

lemma *nth-list-update-neq* [*rule-format*]:
$xs{:}list(A) \Longrightarrow$
$\forall i \in nat.\ \forall j \in nat.\ i \neq j \longrightarrow nth(j,\ list\text{-}update(xs,i,x)) = nth(j,xs)$
⟨*proof*⟩
declare *nth-list-update-neq* [*simp*]

lemma *list-update-overwrite* [*rule-format*]:
$xs{:}list(A) \Longrightarrow \forall i \in nat.\ i < length(xs)$
$\longrightarrow list\text{-}update(list\text{-}update(xs,\ i,\ x),\ i,\ y) = list\text{-}update(xs,\ i,y)$
⟨*proof*⟩
declare *list-update-overwrite* [*simp*]

lemma *list-update-same-conv* [*rule-format*]:
$xs{:}list(A) \Longrightarrow$
$\forall i \in nat.\ i < length(xs) \longrightarrow$
$(list\text{-}update(xs,\ i,\ x) = xs) \longleftrightarrow (nth(i,\ xs) = x)$
⟨*proof*⟩

lemma *update-zip* [*rule-format*]:
$ys{:}list(B) \Longrightarrow$
$\forall i \in nat.\ \forall xy \in A{*}B.\ \forall xs \in list(A).$
$length(xs) = length(ys) \longrightarrow$
$list\text{-}update(zip(xs,\ ys),\ i,\ xy) = zip(list\text{-}update(xs,\ i,\ fst(xy)),$
$list\text{-}update(ys,\ i,\ snd(xy)))$
⟨*proof*⟩

lemma *set-update-subset-cons* [*rule-format*]:
$xs{:}list(A) \Longrightarrow$
$\forall i \in nat.\ set\text{-}of\text{-}list(list\text{-}update(xs,\ i,\ x)) \subseteq cons(x,\ set\text{-}of\text{-}list(xs))$
⟨*proof*⟩

lemma *set-of-list-update-subsetI*:
$[\![set\text{-}of\text{-}list(xs) \subseteq A;\ xs{:}list(A);\ x \in A;\ i \in nat]\!]$
$\Longrightarrow set\text{-}of\text{-}list(list\text{-}update(xs,\ i,x)) \subseteq A$
⟨*proof*⟩

lemma *upt-rec*:
$j \in nat \Longrightarrow upt(i,j) = (if\ i<j\ then\ Cons(i,\ upt(succ(i),\ j))\ else\ Nil)$
⟨*proof*⟩

lemma *upt-conv-Nil* [*simp*]: $[\![j \le i;\ j \in nat]\!] \Longrightarrow upt(i,j) = Nil$
⟨*proof*⟩

lemma *upt-succ-append*:
$[\![i \le j;\ j \in nat]\!] \Longrightarrow upt(i,succ(j)) = upt(i,\ j)@[j]$
⟨*proof*⟩

lemma *upt-conv-Cons*:
$[\![i<j;\ j \in nat]\!] \Longrightarrow upt(i,j) = Cons(i,upt(succ(i),j))$
⟨*proof*⟩

lemma *upt-type* [*simp*,*TC*]: $j \in nat \Longrightarrow upt(i,j):list(nat)$
⟨*proof*⟩

lemma *upt-add-eq-append*:
$[\![i \le j;\ j \in nat;\ k \in nat]\!] \Longrightarrow upt(i,\ j \#+k) = upt(i,j)@upt(j,j\#+k)$
⟨*proof*⟩

lemma *length-upt* [*simp*]: $[\![i \in nat;\ j \in nat]\!] \Longrightarrow length(upt(i,j)) = j \#- i$
⟨*proof*⟩

lemma *nth-upt* [*simp*]:
$[\![i \in nat;\ j \in nat;\ k \in nat;\ i \#+ k < j]\!] \Longrightarrow nth(k,\ upt(i,j)) = i \#+ k$
⟨*proof*⟩

lemma *take-upt* [*rule-format*]:
$[\![m \in nat;\ n \in nat]\!] \Longrightarrow$
$\forall i \in nat.\ i \#+ m \le n \longrightarrow take(m,\ upt(i,n)) = upt(i,i\#+m)$
⟨*proof*⟩
declare *take-upt* [*simp*]

lemma *map-succ-upt*:
$[\![m \in nat;\ n \in nat]\!] \Longrightarrow map(succ,\ upt(m,n))= upt(succ(m),\ succ(n))$
⟨*proof*⟩

lemma *nth-map* [*rule-format*]:
$xs{:}list(A) \Longrightarrow$
$\forall n \in nat.\ n < length(xs) \longrightarrow nth(n,\ map(f,\ xs)) = f(nth(n,\ xs))$
⟨*proof*⟩
declare *nth-map* [*simp*]

lemma *nth-map-upt* [*rule-format*]:
$[\![m \in nat; n \in nat]\!] \Longrightarrow$
$\forall i \in nat.\ i < n \#- m \longrightarrow nth(i, map(f, upt(m,n))) = f(m \#+ i)$
⟨*proof*⟩

definition
sublist :: $[i, i] \Rightarrow i$ **where**
$sublist(xs, A) \equiv$
$map(fst, (filter(\lambda p.\ snd(p): A, zip(xs, upt(0,length(xs))))))$

lemma *sublist-0* [*simp*]: $xs{:}list(A) \Longrightarrow sublist(xs, 0) = Nil$
⟨*proof*⟩

lemma *sublist-Nil* [*simp*]: $sublist(Nil, A) = Nil$
⟨*proof*⟩

lemma *sublist-shift-lemma*:
$[\![xs{:}list(B); i \in nat]\!] \Longrightarrow$
$map(fst, filter(\lambda p.\ snd(p){:}A, zip(xs, upt(i,i \#+ length(xs))))) =$
$map(fst, filter(\lambda p.\ snd(p){:}nat \wedge snd(p) \#+ i \in A, zip(xs,upt(0,length(xs)))))$
⟨*proof*⟩

lemma *sublist-type* [*simp*,*TC*]:
$xs{:}list(B) \Longrightarrow sublist(xs, A){:}list(B)$
⟨*proof*⟩

lemma *upt-add-eq-append2*:
$[\![i \in nat; j \in nat]\!] \Longrightarrow upt(0, i \#+ j) = upt(0, i) @ upt(i, i \#+ j)$
⟨*proof*⟩

lemma *sublist-append*:
$[\![xs{:}list(B); ys{:}list(B)]\!] \Longrightarrow$
$sublist(xs@ys, A) = sublist(xs, A) @ sublist(ys, \{j \in nat.\ j \#+ length(xs): A\})$
⟨*proof*⟩

lemma *sublist-Cons*:
$[\![xs{:}list(B); x \in B]\!] \Longrightarrow$
$sublist(Cons(x, xs), A) =$
$(if\ 0 \in A\ then\ [x]\ else\ []) @ sublist(xs, \{j \in nat.\ succ(j) \in A\})$
⟨*proof*⟩

lemma *sublist-singleton* [*simp*]:
$sublist([x], A) = (if\ 0 \in A\ then\ [x]\ else\ [])$
⟨*proof*⟩

lemma *sublist-upt-eq-take* [*rule-format*]:

xs:*list*(*A*) ⟹ ∀ *n*∈*nat*. *sublist*(*xs*,*n*) = *take*(*n*,*xs*)
⟨*proof*⟩
declare *sublist-upt-eq-take* [*simp*]

lemma *sublist-Int-eq*:
xs ∈ *list*(*B*) ⟹ *sublist*(*xs*, *A* ∩ *nat*) = *sublist*(*xs*, *A*)
⟨*proof*⟩

Repetition of a List Element

consts *repeat* :: [*i*,*i*]⇒*i*
primrec
repeat(*a*,*0*) = []

repeat(*a*,*succ*(*n*)) = *Cons*(*a*,*repeat*(*a*,*n*))

lemma *length-repeat*: *n* ∈ *nat* ⟹ *length*(*repeat*(*a*,*n*)) = *n*
⟨*proof*⟩

lemma *repeat-succ-app*: *n* ∈ *nat* ⟹ *repeat*(*a*,*succ*(*n*)) = *repeat*(*a*,*n*) @ [*a*]
⟨*proof*⟩

lemma *repeat-type* [*TC*]: ⟦*a* ∈ *A*; *n* ∈ *nat*⟧ ⟹ *repeat*(*a*,*n*) ∈ *list*(*A*)
⟨*proof*⟩

end

30 Equivalence Relations

theory *EquivClass* **imports** *Trancl Perm* **begin**

definition
quotient :: [*i*,*i*]⇒*i* (**infixl** ‹'/'/› *90*) **where**
A//*r* ≡ {*r*``{*x*} . *x* ∈ *A*}

definition
congruent :: [*i*,*i*⇒*i*]⇒*o* **where**
congruent(*r*,*b*) ≡ ∀ *y z*. ⟨*y*,*z*⟩:*r* ⟶ *b*(*y*)=*b*(*z*)

definition
congruent2 :: [*i*,*i*,[*i*,*i*]⇒*i*]⇒*o* **where**
congruent2(*r1*,*r2*,*b*) ≡ ∀ *y1 z1 y2 z2*.
⟨*y1*,*z1*⟩:*r1* ⟶ ⟨*y2*,*z2*⟩:*r2* ⟶ *b*(*y1*,*y2*) = *b*(*z1*,*z2*)

abbreviation
RESPECTS ::[*i*⇒*i*, *i*] ⇒ *o* (**infixr** ‹*respects*› *80*) **where**
f respects r ≡ *congruent*(*r*,*f*)

abbreviation
RESPECTS2 ::[*i*⇒*i*⇒*i*, *i*] ⇒ *o* (**infixr** ‹*respects2* › *80*) **where**

f respects2 r ≡ *congruent2*(*r*,*r*,*f*)
— Abbreviation for the common case where the relations are identical

30.1 Suppes, Theorem 70: r is an equiv relation iff $converse(r)$ $O\ r = r$

lemma *sym-trans-comp-subset*:
$[\![sym(r);\ trans(r)]\!] \Longrightarrow converse(r)\ O\ r \subseteq r$
⟨*proof*⟩

lemma *refl-comp-subset*:
$[\![refl(A,r);\ r \subseteq A*A]\!] \Longrightarrow r \subseteq converse(r)\ O\ r$
⟨*proof*⟩

lemma *equiv-comp-eq*:
$equiv(A,r) \Longrightarrow converse(r)\ O\ r = r$
⟨*proof*⟩

lemma *comp-equivI*:
$[\![converse(r)\ O\ r = r;\ \ domain(r) = A]\!] \Longrightarrow equiv(A,r)$
⟨*proof*⟩

lemma *equiv-class-subset*:
$[\![sym(r);\ \ trans(r);\ \ \langle a,b\rangle: r]\!] \Longrightarrow r``\{a\} \subseteq r``\{b\}$
⟨*proof*⟩

lemma *equiv-class-eq*:
$[\![equiv(A,r);\ \ \langle a,b\rangle: r]\!] \Longrightarrow r``\{a\} = r``\{b\}$
⟨*proof*⟩

lemma *equiv-class-self*:
$[\![equiv(A,r);\ \ a \in A]\!] \Longrightarrow a \in r``\{a\}$
⟨*proof*⟩

lemma *subset-equiv-class*:
$[\![equiv(A,r);\ \ r``\{b\} \subseteq r``\{a\};\ \ b \in A]\!] \Longrightarrow \langle a,b\rangle: r$
⟨*proof*⟩

lemma *eq-equiv-class*: $[\![r``\{a\} = r``\{b\};\ \ equiv(A,r);\ \ b \in A]\!] \Longrightarrow \langle a,b\rangle: r$
⟨*proof*⟩

lemma *equiv-class-nondisjoint*:
$[\![equiv(A,r);\ \ x: (r``\{a\} \cap r``\{b\})]\!] \Longrightarrow \langle a,b\rangle: r$

$\langle proof \rangle$

lemma *equiv-type*: $equiv(A,r) \Longrightarrow r \subseteq A*A$
$\langle proof \rangle$

lemma *equiv-class-eq-iff*:
$equiv(A,r) \Longrightarrow \langle x,y\rangle: r \longleftrightarrow r``\{x\} = r``\{y\} \land x \in A \land y \in A$
$\langle proof \rangle$

lemma *eq-equiv-class-iff*:
$[\![equiv(A,r);\ \ x \in A;\ \ y \in A]\!] \Longrightarrow r``\{x\} = r``\{y\} \longleftrightarrow \langle x,y\rangle: r$
$\langle proof \rangle$

lemma *quotientI* [*TC*]: $x \in A \Longrightarrow r``\{x\}: A//r$
$\langle proof \rangle$

lemma *quotientE*:
$[\![X \in A//r;\ \ \bigwedge x.\ [\![X = r``\{x\};\ \ x \in A]\!] \Longrightarrow P]\!] \Longrightarrow P$
$\langle proof \rangle$

lemma *Union-quotient*:
$equiv(A,r) \Longrightarrow \bigcup(A//r) = A$
$\langle proof \rangle$

lemma *quotient-disj*:
$[\![equiv(A,r);\ \ X \in A//r;\ \ Y \in A//r]\!] \Longrightarrow X{=}Y \mid (X \cap Y \subseteq 0)$
$\langle proof \rangle$

30.2 Defining Unary Operations upon Equivalence Classes

lemma *UN-equiv-class*:
$[\![equiv(A,r);\ \ b\ respects\ r;\ \ a \in A]\!] \Longrightarrow (\bigcup x{\in}r``\{a\}.\ b(x)) = b(a)$
$\langle proof \rangle$

lemma *UN-equiv-class-type*:
$[\![equiv(A,r);\ \ b\ respects\ r;\ \ X \in A//r;\ \ \bigwedge x.\ \ x \in A \Longrightarrow b(x) \in B]\!]$
$\Longrightarrow (\bigcup x{\in}X.\ b(x)) \in B$
$\langle proof \rangle$

lemma *UN-equiv-class-inject*:
$[\![equiv(A,r);\ \ \ b\ respects\ r;$
$(\bigcup x{\in}X.\ b(x)){=}(\bigcup y{\in}Y.\ b(y));\ \ X \in A//r;\ \ Y \in A//r;$
$\bigwedge x\ y.\ [\![x \in A;\ y \in A;\ b(x){=}b(y)]\!] \Longrightarrow \langle x,y\rangle{:}r]\!]$

$\Longrightarrow X{=}Y$
⟨proof⟩

30.3 Defining Binary Operations upon Equivalence Classes

lemma *congruent2-implies-congruent*:
$\llbracket equiv(A,r1);\ \ congruent2(r1,r2,b);\ \ a \in A\rrbracket \Longrightarrow congruent(r2,b(a))$
⟨proof⟩

lemma *congruent2-implies-congruent-UN*:
$\llbracket equiv(A1,r1);\ \ equiv(A2,r2);\ \ congruent2(r1,r2,b);\ \ a \in A2\rrbracket \Longrightarrow$
$congruent(r1,\ \lambda x1.\ \bigcup x2 \in r2``\{a\}.\ b(x1,x2))$
⟨proof⟩

lemma *UN-equiv-class2*:
$\llbracket equiv(A1,r1);\ \ equiv(A2,r2);\ \ congruent2(r1,r2,b);\ \ a1\colon A1;\ \ a2\colon A2\rrbracket$
$\Longrightarrow (\bigcup x1 \in r1``\{a1\}.\ \bigcup x2 \in r2``\{a2\}.\ b(x1,x2)) = b(a1,a2)$
⟨proof⟩

lemma *UN-equiv-class-type2*:
$\llbracket equiv(A,r);\ \ b\ respects2\ r;$
$X1\colon A//r;\ \ X2\colon A//r;$
$\bigwedge x1\ x2.\ \ \llbracket x1\colon A;\ x2\colon A\rrbracket \Longrightarrow b(x1,x2) \in B$
$\rrbracket \Longrightarrow (\bigcup x1{\in}X1.\ \bigcup x2{\in}X2.\ b(x1,x2)) \in B$
⟨proof⟩

lemma *congruent2I*:
$\llbracket equiv(A1,r1);\ \ equiv(A2,r2);$
$\bigwedge y\ z\ w.\ \llbracket w \in A2;\ \ \langle y,z\rangle \in r1\rrbracket \Longrightarrow b(y,w) = b(z,w);$
$\bigwedge y\ z\ w.\ \llbracket w \in A1;\ \ \langle y,z\rangle \in r2\rrbracket \Longrightarrow b(w,y) = b(w,z)$
$\rrbracket \Longrightarrow congruent2(r1,r2,b)$
⟨proof⟩

lemma *congruent2-commuteI*:
assumes *equivA*: $equiv(A,r)$
and *commute*: $\bigwedge y\ z.\ \llbracket y \in A;\ \ z \in A\rrbracket \Longrightarrow b(y,z) = b(z,y)$
and *congt*: $\bigwedge y\ z\ w.\ \llbracket w \in A;\ \ \langle y,z\rangle\colon r\rrbracket \Longrightarrow b(w,y) = b(w,z)$
shows *b respects2 r*
⟨proof⟩

lemma *congruent-commuteI*:
$\llbracket equiv(A,r);\ \ Z \in A//r;$
$\bigwedge w.\ \llbracket w \in A\rrbracket \Longrightarrow congruent(r,\ \lambda z.\ b(w,z));$
$\bigwedge x\ y.\ \llbracket x \in A;\ \ y \in A\rrbracket \Longrightarrow b(y,x) = b(x,y)$
$\rrbracket \Longrightarrow congruent(r,\ \lambda w.\ \bigcup z{\in}Z.\ b(w,z))$

⟨*proof*⟩

end

31 The Integers as Equivalence Classes Over Pairs of Natural Numbers

theory *Int* **imports** *EquivClass ArithSimp* **begin**

definition
intrel :: *i* **where**
intrel ≡ {*p* ∈ (*nat*∗*nat*)∗(*nat*∗*nat*).
∃ *x1 y1 x2 y2*. *p*=<⟨*x1*,*y1*⟩,⟨*x2*,*y2*⟩> ∧ *x1*#+*y2* = *x2*#+*y1*}

definition
int :: *i* **where**
int ≡ (*nat*∗*nat*)//*intrel*

definition
int-of :: *i*⇒*i* — coercion from nat to int (‹(‹*open-block notation*=‹*prefix* $#››$# -)› [*80*] *80*)
where $# *m* ≡ *intrel* ‘‘ {<*natify*(*m*), *0*>}

definition
intify :: *i*⇒*i* — coercion from ANYTHING to int **where**
intify(*m*) ≡ *if m* ∈ *int then m else* $#*0*

definition
raw-zminus :: *i*⇒*i* **where**
raw-zminus(*z*) ≡ ⋃⟨*x*,*y*⟩∈*z*. *intrel*‘‘{⟨*y*,*x*⟩}

definition
zminus :: *i*⇒*i* (‹(‹*open-block notation*=‹*prefix* $−››$− -)› [*80*] *80*)
where $− *z* ≡ *raw-zminus* (*intify*(*z*))

definition
znegative :: *i*⇒*o* **where**
znegative(*z*) ≡ ∃ *x y*. *x*<*y* ∧ *y*∈*nat* ∧ ⟨*x*,*y*⟩∈*z*

definition
iszero :: *i*⇒*o* **where**
iszero(*z*) ≡ *z* = $# *0*

definition
raw-nat-of :: *i*⇒*i* **where**
raw-nat-of(*z*) ≡ *natify* (⋃⟨*x*,*y*⟩∈*z*. *x*#−*y*)

definition

nat-of :: *i*⇒*i* **where**
nat-of(*z*) ≡ *raw-nat-of* (*intify*(*z*))

definition
zmagnitude :: *i*⇒*i* **where**
— could be replaced by an absolute value function from int to int?
zmagnitude(*z*) ≡
THE m. m∈*nat* ∧ ((¬ *znegative*(*z*) ∧ *z* = $# *m*) |
(*znegative*(*z*) ∧ $− *z* = $# *m*))

definition
raw-zmult :: [*i*,*i*]⇒*i* **where**

raw-zmult(*z1*,*z2*) ≡
⋃*p1*∈*z1*. ⋃*p2*∈*z2*. *split*(λ*x1 y1*. *split*(λ*x2 y2*.
intrel``{<*x1*#∗*x2* #+ *y1*#∗*y2*, *x1*#∗*y2* #+ *y1*#∗*x2*>}, *p2*), *p1*)

definition
zmult :: [*i*,*i*]⇒*i* (**infixl** ‹$∗› *70*) **where**
z1 $∗ *z2* ≡ *raw-zmult* (*intify*(*z1*),*intify*(*z2*))

definition
raw-zadd :: [*i*,*i*]⇒*i* **where**
raw-zadd (*z1*, *z2*) ≡
⋃*z1*∈*z1*. ⋃*z2*∈*z2*. *let* ⟨*x1*,*y1*⟩=*z1*; ⟨*x2*,*y2*⟩=*z2*
in intrel``{<*x1*#+*x2*, *y1*#+*y2*>}

definition
zadd :: [*i*,*i*]⇒*i* (**infixl** ‹$+› *65*) **where**
z1 $+ *z2* ≡ *raw-zadd* (*intify*(*z1*),*intify*(*z2*))

definition
zdiff :: [*i*,*i*]⇒*i* (**infixl** ‹$−› *65*) **where**
z1 $− *z2* ≡ *z1* $+ *zminus*(*z2*)

definition
zless :: [*i*,*i*]⇒*o* (**infixl** ‹$<› *50*) **where**
z1 $< *z2* ≡ *znegative*(*z1* $− *z2*)

definition
zle :: [*i*,*i*]⇒*o* (**infixl** ‹$≤› *50*) **where**
z1 $≤ *z2* ≡ *z1* $< *z2* | *intify*(*z1*)=*intify*(*z2*)

declare *quotientE* [*elim*!]

31.1 Proving that *intrel* is an equivalence relation

lemma *intrel-iff* [*simp*]:

<$\langle$*x1*,*y1*$\rangle$,$\langle$*x2*,*y2*$\rangle$>: *intrel* $\longleftrightarrow$
x1$\in$*nat* $\wedge$ *y1*$\in$*nat* $\wedge$ *x2*$\in$*nat* $\wedge$ *y2*$\in$*nat* $\wedge$ *x1* #+*y2* = *x2* #+*y1*
$\langle$*proof*$\rangle$

lemma *intrelI* [*intro*!]:
[[*x1* #+*y2* = *x2* #+*y1*; *x1*$\in$*nat*; *y1*$\in$*nat*; *x2*$\in$*nat*; *y2*$\in$*nat*]]
$\Longrightarrow$ <$\langle$*x1*,*y1*$\rangle$,$\langle$*x2*,*y2*$\rangle$>: *intrel*
$\langle$*proof*$\rangle$

lemma *intrelE* [*elim*!]:
[[*p* $\in$ *intrel*;
$\bigwedge$*x1 y1 x2 y2*. [[*p* = <$\langle$*x1*,*y1*$\rangle$,$\langle$*x2*,*y2*$\rangle$>; *x1* #+*y2* = *x2* #+*y1*;
x1$\in$*nat*; *y1*$\in$*nat*; *x2*$\in$*nat*; *y2*$\in$*nat*]] $\Longrightarrow$ *Q*]]
$\Longrightarrow$ *Q*
$\langle$*proof*$\rangle$

lemma *int-trans-lemma*:
[[*x1* #+ *y2* = *x2* #+ *y1*; *x2* #+ *y3* = *x3* #+ *y2*]] $\Longrightarrow$ *x1* #+ *y3* = *x3* #+ *y1*
$\langle$*proof*$\rangle$

lemma *equiv-intrel*: *equiv*(*nat*∗*nat*, *intrel*)
$\langle$*proof*$\rangle$

lemma *image-intrel-int*: [[*m*$\in$*nat*; *n*$\in$*nat*]] $\Longrightarrow$ *intrel* `` {$\langle$*m*,*n*$\rangle$} $\in$ *int*
$\langle$*proof*$\rangle$

declare *equiv-intrel* [*THEN eq-equiv-class-iff*, *simp*]
declare *conj-cong* [*cong*]

lemmas *eq-intrelD* = *eq-equiv-class* [*OF - equiv-intrel*]

lemma *int-of-type* [*simp*,*TC*]: $#*m* $\in$ *int*
$\langle$*proof*$\rangle$

lemma *int-of-eq* [*iff*]: ($# *m* = $# *n*) $\longleftrightarrow$ *natify*(*m*)=*natify*(*n*)
$\langle$*proof*$\rangle$

lemma *int-of-inject*: [[$#*m* = $#*n*; *m*$\in$*nat*; *n*$\in$*nat*]] $\Longrightarrow$ *m*=*n*
$\langle$*proof*$\rangle$

lemma *intify-in-int* [*iff*,*TC*]: *intify*(*x*) $\in$ *int*
$\langle$*proof*$\rangle$

lemma *intify-ident* [*simp*]: $n \in int \Longrightarrow intify(n) = n$
⟨*proof*⟩

31.2 Collapsing rules: to remove *intify* from arithmetic expressions

lemma *intify-idem* [*simp*]: $intify(intify(x)) = intify(x)$
⟨*proof*⟩

lemma *int-of-natify* [*simp*]: $\$\#\ (natify(m)) = \$\#\ m$
⟨*proof*⟩

lemma *zminus-intify* [*simp*]: $\$-\ (intify(m)) = \$-\ m$
⟨*proof*⟩

lemma *zadd-intify1* [*simp*]: $intify(x)\ \$+\ y = x\ \$+\ y$
⟨*proof*⟩

lemma *zadd-intify2* [*simp*]: $x\ \$+\ intify(y) = x\ \$+\ y$
⟨*proof*⟩

lemma *zdiff-intify1* [*simp*]:$intify(x)\ \$-\ y = x\ \$-\ y$
⟨*proof*⟩

lemma *zdiff-intify2* [*simp*]:$x\ \$-\ intify(y) = x\ \$-\ y$
⟨*proof*⟩

lemma *zmult-intify1* [*simp*]:$intify(x)\ \$*\ y = x\ \$*\ y$
⟨*proof*⟩

lemma *zmult-intify2* [*simp*]:$x\ \$*\ intify(y) = x\ \$*\ y$
⟨*proof*⟩

lemma *zless-intify1* [*simp*]:$intify(x)\ \$<\ y \longleftrightarrow x\ \$<\ y$
⟨*proof*⟩

lemma *zless-intify2* [*simp*]:$x\ \$<\ intify(y) \longleftrightarrow x\ \$<\ y$
⟨*proof*⟩

lemma *zle-intify1* [*simp*]:$intify(x)\ \$\le\ y \longleftrightarrow x\ \$\le\ y$
⟨*proof*⟩

lemma *zle-intify2* [*simp*]:$x \;\$\!\le\; intify(y) \longleftrightarrow x \;\$\!\le\; y$
⟨*proof*⟩

31.3 *zminus*: unary negation on *int*

lemma *zminus-congruent*: $(\lambda\langle x,y\rangle.\ intrel``\{\langle y,x\rangle\})$ *respects intrel*
⟨*proof*⟩

lemma *raw-zminus-type*: $z \in int \Longrightarrow raw\text{-}zminus(z) \in int$
⟨*proof*⟩

lemma *zminus-type* [*TC*,*iff*]: $\$-z \in int$
⟨*proof*⟩

lemma *raw-zminus-inject*:
$[\![raw\text{-}zminus(z) = raw\text{-}zminus(w);\ \ z \in int;\ \ w \in int]\!] \Longrightarrow z=w$
⟨*proof*⟩

lemma *zminus-inject-intify* [*dest*!]: $\$-z = \$-w \Longrightarrow intify(z) = intify(w)$
⟨*proof*⟩

lemma *zminus-inject*: $[\![\$-z = \$-w;\ \ z \in int;\ \ w \in int]\!] \Longrightarrow z=w$
⟨*proof*⟩

lemma *raw-zminus*:
$[\![x{\in}nat;\ \ y{\in}nat]\!] \Longrightarrow raw\text{-}zminus(intrel``\{\langle x,y\rangle\}) = intrel\ ``\ \{\langle y,x\rangle\}$
⟨*proof*⟩

lemma *zminus*:
$[\![x{\in}nat;\ \ y{\in}nat]\!]$
$\Longrightarrow \$-\ (intrel``\{\langle x,y\rangle\}) = intrel\ ``\ \{\langle y,x\rangle\}$
⟨*proof*⟩

lemma *raw-zminus-zminus*: $z \in int \Longrightarrow raw\text{-}zminus\ (raw\text{-}zminus(z)) = z$
⟨*proof*⟩

lemma *zminus-zminus-intify* [*simp*]: $\$-\ (\$-\ z) = intify(z)$
⟨*proof*⟩

lemma *zminus-int0* [*simp*]: $\$-\ (\$\#0) = \$\#0$
⟨*proof*⟩

lemma *zminus-zminus*: $z \in int \Longrightarrow \$-\ (\$-\ z) = z$
⟨*proof*⟩

31.4 *znegative*: the test for negative integers

lemma *znegative*: $[\![x{\in}nat;\ y{\in}nat]\!] \Longrightarrow znegative(intrel``\{\langle x,y\rangle\}) \longleftrightarrow x<y$
⟨*proof*⟩

lemma *not-znegative-int-of* [*iff*]: $\neg$ *znegative*($# *n*)
⟨*proof*⟩

lemma *znegative-zminus-int-of* [*simp*]: *znegative*($− $# *succ*(*n*))
⟨*proof*⟩

lemma *not-znegative-imp-zero*: $\neg$ *znegative*($− $# *n*) $\Longrightarrow$ *natify*(*n*)=*0*
⟨*proof*⟩

31.5 *nat-of*: Coercion of an Integer to a Natural Number

lemma *nat-of-intify* [*simp*]: *nat-of*(*intify*(*z*)) = *nat-of*(*z*)
⟨*proof*⟩

lemma *nat-of-congruent*: (λ*x*. ($\lambda\langle x,y\rangle$. *x* #− *y*)(*x*)) *respects intrel*
⟨*proof*⟩

lemma *raw-nat-of*:
  ⟦*x*∈*nat*; *y*∈*nat*⟧ $\Longrightarrow$ *raw-nat-of*(*intrel*``{⟨*x*,*y*⟩}) = *x*#−*y*
⟨*proof*⟩

lemma *raw-nat-of-int-of*: *raw-nat-of*($# *n*) = *natify*(*n*)
⟨*proof*⟩

lemma *nat-of-int-of* [*simp*]: *nat-of*($# *n*) = *natify*(*n*)
⟨*proof*⟩

lemma *raw-nat-of-type*: *raw-nat-of*(*z*) ∈ *nat*
⟨*proof*⟩

lemma *nat-of-type* [*iff*,*TC*]: *nat-of*(*z*) ∈ *nat*
⟨*proof*⟩

31.6 zmagnitude: magnitide of an integer, as a natural number

lemma *zmagnitude-int-of* [*simp*]: *zmagnitude*($# *n*) = *natify*(*n*)
⟨*proof*⟩

lemma *natify-int-of-eq*: *natify*(*x*)=*n* $\Longrightarrow$ $#*x* = $# *n*
⟨*proof*⟩

lemma *zmagnitude-zminus-int-of* [*simp*]: *zmagnitude*($− $# *n*) = *natify*(*n*)
⟨*proof*⟩

lemma *zmagnitude-type* [*iff*,*TC*]: *zmagnitude*(*z*)∈*nat*
⟨*proof*⟩

lemma *not-zneg-int-of*:
  $[\![z \in int; \neg\ znegative(z)]\!] \Longrightarrow \exists\, n{\in}nat.\ z = \$\#\ n$
⟨*proof*⟩

lemma *not-zneg-mag* [*simp*]:
  $[\![z \in int; \neg\ znegative(z)]\!] \Longrightarrow \$\#\ (zmagnitude(z)) = z$
⟨*proof*⟩

lemma *zneg-int-of*:
  $[\![znegative(z);\ z \in int]\!] \Longrightarrow \exists\, n{\in}nat.\ z = \$-\ (\$\#\ succ(n))$
⟨*proof*⟩

lemma *zneg-mag* [*simp*]:
  $[\![znegative(z);\ z \in int]\!] \Longrightarrow \$\#\ (zmagnitude(z)) = \$-\ z$
⟨*proof*⟩

lemma *int-cases*: $z \in int \Longrightarrow \exists\, n{\in}nat.\ z = \$\#\ n \mid z = \$-\ (\$\#\ succ(n))$
⟨*proof*⟩

lemma *not-zneg-raw-nat-of*:
  $[\![\neg\ znegative(z);\ z \in int]\!] \Longrightarrow \$\#\ (raw\text{-}nat\text{-}of(z)) = z$
⟨*proof*⟩

lemma *not-zneg-nat-of-intify*:
  $\neg\ znegative(intify(z)) \Longrightarrow \$\#\ (nat\text{-}of(z)) = intify(z)$
⟨*proof*⟩

lemma *not-zneg-nat-of*: $[\![\neg\ znegative(z);\ z \in int]\!] \Longrightarrow \$\#\ (nat\text{-}of(z)) = z$
⟨*proof*⟩

lemma *zneg-nat-of* [*simp*]: $znegative(intify(z)) \Longrightarrow nat\text{-}of(z) = 0$
⟨*proof*⟩

31.7 ($+): addition on int

Congruence Property for Addition

lemma *zadd-congruent2*:
  $(\lambda z1\ z2.\ let\ \langle x1,y1\rangle{=}z1;\ \langle x2,y2\rangle{=}z2$
      $in\ intrel``\{<x1\#{+}x2,\ y1\#{+}y2>\})$
  respects2 intrel
⟨*proof*⟩

lemma *raw-zadd-type*: $[\![z \in int;\ \ w \in int]\!] \Longrightarrow raw\text{-}zadd(z,w) \in int$
⟨*proof*⟩

lemma *zadd-type* [*iff*,*TC*]: $z\ \$+\ w \in int$
⟨*proof*⟩

lemma *raw-zadd*:
⟦*x1*∈*nat*; *y1*∈*nat*; *x2*∈*nat*; *y2*∈*nat*⟧
⟹ *raw-zadd* (*intrel*‘‘{⟨*x1*,*y1*⟩}, *intrel*‘‘{⟨*x2*,*y2*⟩}) =
intrel ‘‘ {<*x1*#+*x2*, *y1*#+*y2*>}
⟨*proof*⟩

lemma *zadd*:
⟦*x1*∈*nat*; *y1*∈*nat*; *x2*∈*nat*; *y2*∈*nat*⟧
⟹ (*intrel*‘‘{⟨*x1*,*y1*⟩}) $+ (*intrel*‘‘{⟨*x2*,*y2*⟩}) =
intrel ‘‘ {<*x1*#+*x2*, *y1*#+*y2*>}
⟨*proof*⟩

lemma *raw-zadd-int0*: *z* ∈ *int* ⟹ *raw-zadd* ($#*0*,*z*) = *z*
⟨*proof*⟩

lemma *zadd-int0-intify* [*simp*]: $#*0* $+ *z* = *intify*(*z*)
⟨*proof*⟩

lemma *zadd-int0*: *z* ∈ *int* ⟹ $#*0* $+ *z* = *z*
⟨*proof*⟩

lemma *raw-zminus-zadd-distrib*:
⟦*z* ∈ *int*; *w* ∈ *int*⟧ ⟹ $− *raw-zadd*(*z*,*w*) = *raw-zadd*($− *z*, $− *w*)
⟨*proof*⟩

lemma *zminus-zadd-distrib* [*simp*]: $− (*z* $+ *w*) = $− *z* $+ $− *w*
⟨*proof*⟩

lemma *raw-zadd-commute*:
⟦*z* ∈ *int*; *w* ∈ *int*⟧ ⟹ *raw-zadd*(*z*,*w*) = *raw-zadd*(*w*,*z*)
⟨*proof*⟩

lemma *zadd-commute*: *z* $+ *w* = *w* $+ *z*
⟨*proof*⟩

lemma *raw-zadd-assoc*:
⟦*z1*: *int*; *z2*: *int*; *z3*: *int*⟧
⟹ *raw-zadd* (*raw-zadd*(*z1*,*z2*),*z3*) = *raw-zadd*(*z1*,*raw-zadd*(*z2*,*z3*))
⟨*proof*⟩

lemma *zadd-assoc*: (*z1* $+ *z2*) $+ *z3* = *z1* $+ (*z2* $+ *z3*)
⟨*proof*⟩

lemma *zadd-left-commute*: *z1*$+(*z2*$+*z3*) = *z2*$+(*z1*$+*z3*)
⟨*proof*⟩

lemmas *zadd-ac* = *zadd-assoc zadd-commute zadd-left-commute*

lemma *int-of-add*: $\$\#\ (m\ \#+\ n) = (\$\#m)\ \$+\ (\$\#n)$
⟨*proof*⟩

lemma *int-succ-int-1*: $\$\#\ succ(m) = \$\#\ 1\ \$+\ (\$\#\ m)$
⟨*proof*⟩

lemma *int-of-diff*:
$[\![m{\in}nat;\ \ n \le m]\!] \Longrightarrow \$\#\ (m\ \#-\ n) = (\$\#m)\ \$-\ (\$\#n)$
⟨*proof*⟩

lemma *raw-zadd-zminus-inverse*: $z \in int \Longrightarrow raw\text{-}zadd\ (z, \$-\ z) = \$\#0$
⟨*proof*⟩

lemma *zadd-zminus-inverse* [*simp*]: $z\ \$+\ (\$-\ z) = \$\#0$
⟨*proof*⟩

lemma *zadd-zminus-inverse2* [*simp*]: $(\$-\ z)\ \$+\ z = \$\#0$
⟨*proof*⟩

lemma *zadd-int0-right-intify* [*simp*]: $z\ \$+\ \$\#0 = intify(z)$
⟨*proof*⟩

lemma *zadd-int0-right*: $z \in int \Longrightarrow z\ \$+\ \$\#0 = z$
⟨*proof*⟩

31.8 (\$*): Integer Multiplication

Congruence property for multiplication

lemma *zmult-congruent2*:
$(\lambda p1\ p2.\ split(\lambda x1\ y1.\ split(\lambda x2\ y2.$
$intrel``\{<x1\#*x2\ \#+\ y1\#*y2,\ x1\#*y2\ \#+\ y1\#*x2>\},\ p2),\ p1))$
respects2 intrel
⟨*proof*⟩

lemma *raw-zmult-type*: $[\![z \in int;\ \ w \in int]\!] \Longrightarrow raw\text{-}zmult(z,w) \in int$
⟨*proof*⟩

lemma *zmult-type* [*iff*,*TC*]: $z\ \$*\ w \in int$
⟨*proof*⟩

lemma *raw-zmult*:
$[\![x1{\in}nat;\ y1{\in}nat;\ \ x2{\in}nat;\ y2{\in}nat]\!]$
$\Longrightarrow raw\text{-}zmult(intrel``\{\langle x1,y1\rangle\},\ intrel``\{\langle x2,y2\rangle\}) =$
$intrel\ ``\ \{<x1\#*x2\ \#+\ y1\#*y2,\ x1\#*y2\ \#+\ y1\#*x2>\}$
⟨*proof*⟩

lemma *zmult*:

⟦x1∈nat; y1∈nat; x2∈nat; y2∈nat⟧
 ⟹ (intrel``{⟨x1,y1⟩}) $* (intrel``{⟨x2,y2⟩}) =
   intrel `` {<x1#*x2 #+ y1#*y2, x1#*y2 #+ y1#*x2>}
⟨proof⟩

lemma *raw-zmult-int0*: z ∈ int ⟹ raw-zmult ($#0,z) = $#0
⟨proof⟩

lemma *zmult-int0* [simp]: $#0 $* z = $#0
⟨proof⟩

lemma *raw-zmult-int1*: z ∈ int ⟹ raw-zmult ($#1,z) = z
⟨proof⟩

lemma *zmult-int1-intify* [simp]: $#1 $* z = intify(z)
⟨proof⟩

lemma *zmult-int1*: z ∈ int ⟹ $#1 $* z = z
⟨proof⟩

lemma *raw-zmult-commute*:
 ⟦z ∈ int; w ∈ int⟧ ⟹ raw-zmult(z,w) = raw-zmult(w,z)
⟨proof⟩

lemma *zmult-commute*: z $* w = w $* z
⟨proof⟩

lemma *raw-zmult-zminus*:
 ⟦z ∈ int; w ∈ int⟧ ⟹ raw-zmult($− z, w) = $− raw-zmult(z, w)
⟨proof⟩

lemma *zmult-zminus* [simp]: ($− z) $* w = $− (z $* w)
⟨proof⟩

lemma *zmult-zminus-right* [simp]: w $* ($− z) = $− (w $* z)
⟨proof⟩

lemma *raw-zmult-assoc*:
 ⟦z1: int; z2: int; z3: int⟧
 ⟹ raw-zmult (raw-zmult(z1,z2),z3) = raw-zmult(z1,raw-zmult(z2,z3))
⟨proof⟩

lemma *zmult-assoc*: (z1 $* z2) $* z3 = z1 $* (z2 $* z3)
⟨proof⟩

lemma *zmult-left-commute*: z1$*(z2$*z3) = z2$*(z1$*z3)
⟨proof⟩

lemmas *zmult-ac* = *zmult-assoc zmult-commute zmult-left-commute*

lemma *raw-zadd-zmult-distrib*:
  $[\![z1: int;\ \ z2: int;\ \ w \in int]\!]$
  $\Longrightarrow$ *raw-zmult*(*raw-zadd*(*z1*,*z2*), *w*) =
    raw-zadd (*raw-zmult*(*z1*,*w*), *raw-zmult*(*z2*,*w*))
⟨*proof*⟩

lemma *zadd-zmult-distrib*: (*z1* \$+ *z2*) \$∗ *w* = (*z1* \$∗ *w*) \$+ (*z2* \$∗ *w*)
⟨*proof*⟩

lemma *zadd-zmult-distrib2*: *w* \$∗ (*z1* \$+ *z2*) = (*w* \$∗ *z1*) \$+ (*w* \$∗ *z2*)
⟨*proof*⟩

lemmas *int-typechecks* =
 int-of-type zminus-type zmagnitude-type zadd-type zmult-type

lemma *zdiff-type* [*iff*,*TC*]: *z* \$− *w* ∈ *int*
⟨*proof*⟩

lemma *zminus-zdiff-eq* [*simp*]: \$− (*z* \$− *y*) = *y* \$− *z*
⟨*proof*⟩

lemma *zdiff-zmult-distrib*: (*z1* \$− *z2*) \$∗ *w* = (*z1* \$∗ *w*) \$− (*z2* \$∗ *w*)
⟨*proof*⟩

lemma *zdiff-zmult-distrib2*: *w* \$∗ (*z1* \$− *z2*) = (*w* \$∗ *z1*) \$− (*w* \$∗ *z2*)
⟨*proof*⟩

lemma *zadd-zdiff-eq*: *x* \$+ (*y* \$− *z*) = (*x* \$+ *y*) \$− *z*
⟨*proof*⟩

lemma *zdiff-zadd-eq*: (*x* \$− *y*) \$+ *z* = (*x* \$+ *z*) \$− *y*
⟨*proof*⟩

31.9 The "Less Than" Relation

lemma *zless-linear-lemma*:
  $[\![z \in int;\ w \in int]\!] \Longrightarrow$ *z*\$<*w* | *z*=*w* | *w*\$<*z*
⟨*proof*⟩

lemma *zless-linear*: *z*\$<*w* | *intify*(*z*)=*intify*(*w*) | *w*\$<*z*
⟨*proof*⟩

lemma *zless-not-refl* [*iff*]: ¬ (*z*\$<*z*)

⟨proof⟩

lemma *neq-iff-zless*: $[\![x \in int;\ y \in int]\!] \Longrightarrow (x \neq y) \longleftrightarrow (x \ \$< \ y \mid y \ \$< \ x)$
⟨proof⟩

lemma *zless-imp-intify-neq*: $w \ \$< \ z \Longrightarrow intify(w) \neq intify(z)$
⟨proof⟩

lemma *zless-imp-succ-zadd-lemma*:
$[\![w \ \$< \ z;\ w \in int;\ z \in int]\!] \Longrightarrow (\exists\, n{\in}nat.\ z = w \ \$+ \ \$\#(succ(n)))$
⟨proof⟩

lemma *zless-imp-succ-zadd*:
$w \ \$< \ z \Longrightarrow (\exists\, n{\in}nat.\ w \ \$+ \ \$\#(succ(n)) = intify(z))$
⟨proof⟩

lemma *zless-succ-zadd-lemma*:
$w \in int \Longrightarrow w \ \$< \ w \ \$+ \ \$\# \ succ(n)$
⟨proof⟩

lemma *zless-succ-zadd*: $w \ \$< \ w \ \$+ \ \$\# \ succ(n)$
⟨proof⟩

lemma *zless-iff-succ-zadd*:
$w \ \$< \ z \longleftrightarrow (\exists\, n{\in}nat.\ w \ \$+ \ \$\#(succ(n)) = intify(z))$
⟨proof⟩

lemma *zless-int-of* [*simp*]: $[\![m{\in}nat;\ n{\in}nat]\!] \Longrightarrow (\$\#m \ \$< \ \$\#n) \longleftrightarrow (m{<}n)$
⟨proof⟩

lemma *zless-trans-lemma*:
$[\![x \ \$< \ y;\ y \ \$< \ z;\ x \in int;\ y \in int;\ z \in int]\!] \Longrightarrow x \ \$< \ z$
⟨proof⟩

lemma *zless-trans* [*trans*]: $[\![x \ \$< \ y;\ y \ \$< \ z]\!] \Longrightarrow x \ \$< \ z$
⟨proof⟩

lemma *zless-not-sym*: $z \ \$< \ w \Longrightarrow \neg\ (w \ \$< \ z)$
⟨proof⟩

lemmas *zless-asym* = *zless-not-sym* [*THEN swap*]

lemma *zless-imp-zle*: $z \ \$< \ w \Longrightarrow z \ \$\leq \ w$
⟨proof⟩

lemma *zle-linear*: $z \ \$\leq \ w \mid w \ \$\leq \ z$
⟨proof⟩

31.10 Less Than or Equals

lemma *zle-refl*: $z \$\le z$
⟨*proof*⟩

lemma *zle-eq-refl*: $x=y \Longrightarrow x \$\le y$
⟨*proof*⟩

lemma *zle-anti-sym-intify*: $[\![x \$\le y; y \$\le x]\!] \Longrightarrow intify(x) = intify(y)$
⟨*proof*⟩

lemma *zle-anti-sym*: $[\![x \$\le y; y \$\le x; x \in int; y \in int]\!] \Longrightarrow x=y$
⟨*proof*⟩

lemma *zle-trans-lemma*:
$[\![x \in int; y \in int; z \in int; x \$\le y; y \$\le z]\!] \Longrightarrow x \$\le z$
⟨*proof*⟩

lemma *zle-trans* [*trans*]: $[\![x \$\le y; y \$\le z]\!] \Longrightarrow x \$\le z$
⟨*proof*⟩

lemma *zle-zless-trans* [*trans*]: $[\![i \$\le j; j \$< k]\!] \Longrightarrow i \$< k$
⟨*proof*⟩

lemma *zless-zle-trans* [*trans*]: $[\![i \$< j; j \$\le k]\!] \Longrightarrow i \$< k$
⟨*proof*⟩

lemma *not-zless-iff-zle*: $\neg (z \$< w) \longleftrightarrow (w \$\le z)$
⟨*proof*⟩

lemma *not-zle-iff-zless*: $\neg (z \$\le w) \longleftrightarrow (w \$< z)$
⟨*proof*⟩

31.11 More subtraction laws (for *zcompare-rls*)

lemma *zdiff-zdiff-eq*: $(x \$- y) \$- z = x \$- (y \$+ z)$
⟨*proof*⟩

lemma *zdiff-zdiff-eq2*: $x \$- (y \$- z) = (x \$+ z) \$- y$
⟨*proof*⟩

lemma *zdiff-zless-iff*: $(x\$-y \$< z) \longleftrightarrow (x \$< z \$+ y)$
⟨*proof*⟩

lemma *zless-zdiff-iff*: $(x \$< z\$-y) \longleftrightarrow (x \$+ y \$< z)$
⟨*proof*⟩

lemma *zdiff-eq-iff*: $[\![x \in int; z \in int]\!] \Longrightarrow (x\$-y = z) \longleftrightarrow (x = z \$+ y)$
⟨*proof*⟩

lemma *eq-zdiff-iff*: $[\![x \in int; z \in int]\!] \Longrightarrow (x = z\$-y) \longleftrightarrow (x \$+ y = z)$
⟨*proof*⟩

lemma *zdiff-zle-iff-lemma*:
$[\![x \in int; z \in int]\!] \Longrightarrow (x\$-y \$\leq z) \longleftrightarrow (x \$\leq z \$+ y)$
⟨*proof*⟩

lemma *zdiff-zle-iff*: $(x\$-y \$\leq z) \longleftrightarrow (x \$\leq z \$+ y)$
⟨*proof*⟩

lemma *zle-zdiff-iff-lemma*:
$[\![x \in int; z \in int]\!] \Longrightarrow (x \$\leq z\$-y) \longleftrightarrow (x \$+ y \$\leq z)$
⟨*proof*⟩

lemma *zle-zdiff-iff*: $(x \$\leq z\$-y) \longleftrightarrow (x \$+ y \$\leq z)$
⟨*proof*⟩

This list of rewrites simplifies (in)equalities by bringing subtractions to the top and then moving negative terms to the other side. Use with *zadd-ac*

lemmas *zcompare-rls* =
zdiff-def [*symmetric*]
zadd-zdiff-eq zdiff-zadd-eq zdiff-zdiff-eq zdiff-zdiff-eq2
zdiff-zless-iff zless-zdiff-iff zdiff-zle-iff zle-zdiff-iff
zdiff-eq-iff eq-zdiff-iff

31.12 Monotonicity and Cancellation Results for Instantiation of the CancelNumerals Simprocs

lemma *zadd-left-cancel*:
$[\![w \in int; w': int]\!] \Longrightarrow (z \$+ w' = z \$+ w) \longleftrightarrow (w' = w)$
⟨*proof*⟩

lemma *zadd-left-cancel-intify* [*simp*]:
$(z \$+ w' = z \$+ w) \longleftrightarrow intify(w') = intify(w)$
⟨*proof*⟩

lemma *zadd-right-cancel*:
$[\![w \in int; w': int]\!] \Longrightarrow (w' \$+ z = w \$+ z) \longleftrightarrow (w' = w)$
⟨*proof*⟩

lemma *zadd-right-cancel-intify* [*simp*]:
$(w' \$+ z = w \$+ z) \longleftrightarrow intify(w') = intify(w)$
⟨*proof*⟩

lemma *zadd-right-cancel-zless* [*simp*]: $(w' \$+ z \$< w \$+ z) \longleftrightarrow (w' \$< w)$
⟨*proof*⟩

lemma *zadd-left-cancel-zless* [*simp*]: $(z \$+ w' \$< z \$+ w) \longleftrightarrow (w' \$< w)$
⟨*proof*⟩

lemma *zadd-right-cancel-zle* [*simp*]: $(w' \$+ z \$\leq w \$+ z) \longleftrightarrow w' \$\leq w$
⟨*proof*⟩

lemma *zadd-left-cancel-zle* [*simp*]: $(z \$+ w' \$\leq z \$+ w) \longleftrightarrow w' \$\leq w$
⟨*proof*⟩

lemmas *zadd-zless-mono1* = *zadd-right-cancel-zless* [*THEN iffD2*]

lemmas *zadd-zless-mono2* = *zadd-left-cancel-zless* [*THEN iffD2*]

lemmas *zadd-zle-mono1* = *zadd-right-cancel-zle* [*THEN iffD2*]

lemmas *zadd-zle-mono2* = *zadd-left-cancel-zle* [*THEN iffD2*]

lemma *zadd-zle-mono*: $[\![w' \$\leq w; z' \$\leq z]\!] \Longrightarrow w' \$+ z' \$\leq w \$+ z$
⟨*proof*⟩

lemma *zadd-zless-mono*: $[\![w' \$< w; z' \$\leq z]\!] \Longrightarrow w' \$+ z' \$< w \$+ z$
⟨*proof*⟩

31.13 Comparison laws

lemma *zminus-zless-zminus* [*simp*]: $(\$- x \$< \$- y) \longleftrightarrow (y \$< x)$
⟨*proof*⟩

lemma *zminus-zle-zminus* [*simp*]: $(\$- x \$\leq \$- y) \longleftrightarrow (y \$\leq x)$
⟨*proof*⟩

31.13.1 More inequality lemmas

lemma *equation-zminus*: $[\![x \in int; \ y \in int]\!] \Longrightarrow (x = \$- y) \longleftrightarrow (y = \$- x)$
⟨*proof*⟩

lemma *zminus-equation*: $[\![x \in int; \ y \in int]\!] \Longrightarrow (\$- x = y) \longleftrightarrow (\$- y = x)$
⟨*proof*⟩

lemma *equation-zminus-intify*: $(intify(x) = \$- y) \longleftrightarrow (intify(y) = \$- x)$
⟨*proof*⟩

lemma *zminus-equation-intify*: $(\$- x = intify(y)) \longleftrightarrow (\$- y = intify(x))$
⟨*proof*⟩

31.13.2 The next several equations are permutative: watch out!

lemma *zless-zminus*: (x $< $− y) ⟷ (y $< $− x)
⟨*proof*⟩

lemma *zminus-zless*: ($− x $< y) ⟷ ($− y $< x)
⟨*proof*⟩

lemma *zle-zminus*: (x $≤ $− y) ⟷ (y $≤ $− x)
⟨*proof*⟩

lemma *zminus-zle*: ($− x $≤ y) ⟷ ($− y $≤ x)
⟨*proof*⟩

end

32 Arithmetic on Binary Integers

theory *Bin*
imports *Int Datatype*
begin

```
consts  bin :: i
datatype
  bin = Pls
      | Min
      | Bit (w ∈ bin, b ∈ bool)     (infixl ‹BIT› 90)

consts
  integ-of  :: i⇒i
  NCons     :: [i,i]⇒i
  bin-succ  :: i⇒i
  bin-pred  :: i⇒i
  bin-minus :: i⇒i
  bin-adder :: i⇒i
  bin-mult  :: [i,i]⇒i

primrec
  integ-of-Pls:  integ-of (Pls)     = $# 0
  integ-of-Min:  integ-of (Min)     = $−($#1)
  integ-of-BIT:  integ-of (w BIT b) = $#b $+ integ-of(w) $+ integ-of(w)

primrec
  NCons-Pls: NCons (Pls,b)     = cond(b,Pls BIT b,Pls)
  NCons-Min: NCons (Min,b)     = cond(b,Min,Min BIT b)
  NCons-BIT: NCons (w BIT c,b) = w BIT c BIT b
```

primrec
bin-succ-Pls: *bin-succ* (*Pls*) = *Pls BIT 1*
bin-succ-Min: *bin-succ* (*Min*) = *Pls*
bin-succ-BIT: *bin-succ* (*w BIT b*) = *cond*(*b*, *bin-succ*(*w*) *BIT 0*, *NCons*(*w*,*1*))

primrec
bin-pred-Pls: *bin-pred* (*Pls*) = *Min*
bin-pred-Min: *bin-pred* (*Min*) = *Min BIT 0*
bin-pred-BIT: *bin-pred* (*w BIT b*) = *cond*(*b*, *NCons*(*w*,*0*), *bin-pred*(*w*) *BIT 1*)

primrec
bin-minus-Pls:
bin-minus (*Pls*) = *Pls*
bin-minus-Min:
bin-minus (*Min*) = *Pls BIT 1*
bin-minus-BIT:
bin-minus (*w BIT b*) = *cond*(*b*, *bin-pred*(*NCons*(*bin-minus*(*w*),*0*)),
bin-minus(*w*) *BIT 0*)

primrec
bin-adder-Pls:
bin-adder (*Pls*) = ($\lambda w \in bin$. *w*)
bin-adder-Min:
bin-adder (*Min*) = ($\lambda w \in bin$. *bin-pred*(*w*))
bin-adder-BIT:
bin-adder (*v BIT x*) =
($\lambda w \in bin$.
bin-case (*v BIT x*, *bin-pred*(*v BIT x*),
$\lambda w\ y$. *NCons*(*bin-adder* (*v*) ‘ *cond*(*x and y*, *bin-succ*(*w*), *w*),
x xor y),
w))

definition
bin-add :: [*i*,*i*]$\Rightarrow$*i* **where**
bin-add(*v*,*w*) $\equiv$ *bin-adder*(*v*)‘*w*

primrec
bin-mult-Pls:
bin-mult (*Pls*,*w*) = *Pls*
bin-mult-Min:
bin-mult (*Min*,*w*) = *bin-minus*(*w*)
bin-mult-BIT:
bin-mult (*v BIT b*,*w*) = *cond*(*b*, *bin-add*(*NCons*(*bin-mult*(*v*,*w*),*0*),*w*),
NCons(*bin-mult*(*v*,*w*),*0*))

syntax

-Int0 :: *i* (‹#()*0*›)
-Int1 :: *i* (‹#()*1*›)
-Int2 :: *i* (‹#()*2*›)
-Neg-Int1 :: *i* (‹#−()*1*›)
-Neg-Int2 :: *i* (‹#−()*2*›)
translations
#0 ⇌ *CONST integ-of*(*CONST Pls*)
#1 ⇌ *CONST integ-of*(*CONST Pls BIT 1*)
#2 ⇌ *CONST integ-of*(*CONST Pls BIT 1 BIT 0*)
#−1 ⇌ *CONST integ-of*(*CONST Min*)
#−2 ⇌ *CONST integ-of*(*CONST Min BIT 0*)

syntax
-Int :: *num-token* ⇒ *i* (‹(‹*open-block notation*=‹*literal number*››#-)› *1000*)
-Neg-Int :: *num-token* ⇒ *i* (‹(‹*open-block notation*=‹*literal number*››#−-)› *1000*)

syntax-consts
-Int0 -Int1 -Int2 -Int -Neg-Int1 -Neg-Int2 -Neg-Int ⇌ *integ-of*

⟨*ML*⟩

declare *bin.intros* [*simp*,*TC*]

lemma *NCons-Pls-0*: *NCons*(*Pls*,*0*) = *Pls*
⟨*proof*⟩

lemma *NCons-Pls-1*: *NCons*(*Pls*,*1*) = *Pls BIT 1*
⟨*proof*⟩

lemma *NCons-Min-0*: *NCons*(*Min*,*0*) = *Min BIT 0*
⟨*proof*⟩

lemma *NCons-Min-1*: *NCons*(*Min*,*1*) = *Min*
⟨*proof*⟩

lemma *NCons-BIT*: *NCons*(*w BIT x*,*b*) = *w BIT x BIT b*
⟨*proof*⟩

lemmas *NCons-simps* [*simp*] =
NCons-Pls-0 NCons-Pls-1 NCons-Min-0 NCons-Min-1 NCons-BIT

lemma *integ-of-type* [*TC*]: *w* ∈ *bin* ⟹ *integ-of*(*w*) ∈ *int*
⟨*proof*⟩

lemma *NCons-type* [*TC*]: ⟦*w* ∈ *bin*; *b* ∈ *bool*⟧ ⟹ *NCons*(*w*,*b*) ∈ *bin*
⟨*proof*⟩

lemma *bin-succ-type* [*TC*]: *w* ∈ *bin* ⟹ *bin-succ*(*w*) ∈ *bin*
⟨*proof*⟩

lemma *bin-pred-type* [*TC*]: *w* ∈ *bin* ⟹ *bin-pred*(*w*) ∈ *bin*
⟨*proof*⟩

lemma *bin-minus-type* [*TC*]: *w* ∈ *bin* ⟹ *bin-minus*(*w*) ∈ *bin*
⟨*proof*⟩

lemma *bin-add-type* [*rule-format*]:
  v ∈ *bin* ⟹ ∀ *w*∈*bin*. *bin-add*(*v*,*w*) ∈ *bin*
 ⟨*proof*⟩

declare *bin-add-type* [*TC*]

lemma *bin-mult-type* [*TC*]: ⟦*v* ∈ *bin*; *w* ∈ *bin*⟧ ⟹ *bin-mult*(*v*,*w*) ∈ *bin*
⟨*proof*⟩

32.0.1 The Carry and Borrow Functions, *bin-succ* and *bin-pred*

lemma *integ-of-NCons* [*simp*]:
  ⟦*w* ∈ *bin*; *b* ∈ *bool*⟧ ⟹ *integ-of*(*NCons*(*w*,*b*)) = *integ-of*(*w BIT b*)
⟨*proof*⟩

lemma *integ-of-succ* [*simp*]:
  w ∈ *bin* ⟹ *integ-of*(*bin-succ*(*w*)) = $#*1* $+ *integ-of*(*w*)
⟨*proof*⟩

lemma *integ-of-pred* [*simp*]:
  w ∈ *bin* ⟹ *integ-of*(*bin-pred*(*w*)) = $− ($#*1*) $+ *integ-of*(*w*)
⟨*proof*⟩

32.0.2 *bin-minus*: Unary Negation of Binary Integers

lemma *integ-of-minus*: *w* ∈ *bin* ⟹ *integ-of*(*bin-minus*(*w*)) = $− *integ-of*(*w*)
⟨*proof*⟩

32.0.3 *bin-add*: Binary Addition

lemma *bin-add-Pls* [*simp*]: *w* ∈ *bin* ⟹ *bin-add*(*Pls*,*w*) = *w*
⟨*proof*⟩

lemma *bin-add-Pls-right*: *w* ∈ *bin* ⟹ *bin-add*(*w*,*Pls*) = *w*
 ⟨*proof*⟩

lemma *bin-add-Min* [*simp*]: *w* ∈ *bin* ⟹ *bin-add*(*Min*,*w*) = *bin-pred*(*w*)

⟨*proof*⟩

lemma *bin-add-Min-right*: $w \in bin \Longrightarrow bin\text{-}add(w,Min) = bin\text{-}pred(w)$
⟨*proof*⟩

lemma *bin-add-BIT-Pls* [*simp*]: $bin\text{-}add(v\ BIT\ x,Pls) = v\ BIT\ x$
⟨*proof*⟩

lemma *bin-add-BIT-Min* [*simp*]: $bin\text{-}add(v\ BIT\ x,Min) = bin\text{-}pred(v\ BIT\ x)$
⟨*proof*⟩

lemma *bin-add-BIT-BIT* [*simp*]:
$[\![w \in bin;\ y \in bool]\!]$
$\Longrightarrow bin\text{-}add(v\ BIT\ x,\ w\ BIT\ y) =$
$NCons(bin\text{-}add(v,\ cond(x\ and\ y,\ bin\text{-}succ(w),\ w)),\ x\ xor\ y)$
⟨*proof*⟩

lemma *integ-of-add* [*rule-format*]:
$v \in bin \Longrightarrow$
$\forall w{\in}bin.\ integ\text{-}of(bin\text{-}add(v,w)) = integ\text{-}of(v)\ \$+\ integ\text{-}of(w)$
⟨*proof*⟩

lemma *diff-integ-of-eq*:
$[\![v \in bin;\ w \in bin]\!]$
$\Longrightarrow integ\text{-}of(v)\ \$-\ integ\text{-}of(w) = integ\text{-}of(bin\text{-}add\ (v,\ bin\text{-}minus(w)))$
⟨*proof*⟩

32.0.4 *bin-mult*: Binary Multiplication

lemma *integ-of-mult*:
$[\![v \in bin;\ w \in bin]\!]$
$\Longrightarrow integ\text{-}of(bin\text{-}mult(v,w)) = integ\text{-}of(v)\ \$*\ integ\text{-}of(w)$
⟨*proof*⟩

32.1 Computations

lemma *bin-succ-1*: $bin\text{-}succ(w\ BIT\ 1) = bin\text{-}succ(w)\ BIT\ 0$
⟨*proof*⟩

lemma *bin-succ-0*: $bin\text{-}succ(w\ BIT\ 0) = NCons(w,1)$
⟨*proof*⟩

lemma *bin-pred-1*: $bin\text{-}pred(w\ BIT\ 1) = NCons(w,0)$
⟨*proof*⟩

lemma *bin-pred-0*: $bin\text{-}pred(w\ BIT\ 0) = bin\text{-}pred(w)\ BIT\ 1$
⟨*proof*⟩

lemma *bin-minus-1*: *bin-minus*(*w BIT 1*) = *bin-pred*(*NCons*(*bin-minus*(*w*), *0*))
⟨*proof*⟩

lemma *bin-minus-0*: *bin-minus*(*w BIT 0*) = *bin-minus*(*w*) *BIT 0*
⟨*proof*⟩

lemma *bin-add-BIT-11*: *w* ∈ *bin* ⟹ *bin-add*(*v BIT 1*, *w BIT 1*) =
NCons(*bin-add*(*v*, *bin-succ*(*w*)), *0*)
⟨*proof*⟩

lemma *bin-add-BIT-10*: *w* ∈ *bin* ⟹ *bin-add*(*v BIT 1*, *w BIT 0*) =
NCons(*bin-add*(*v*,*w*), *1*)
⟨*proof*⟩

lemma *bin-add-BIT-0*: ⟦*w* ∈ *bin*; *y* ∈ *bool*⟧
⟹ *bin-add*(*v BIT 0*, *w BIT y*) = *NCons*(*bin-add*(*v*,*w*), *y*)
⟨*proof*⟩

lemma *bin-mult-1*: *bin-mult*(*v BIT 1*, *w*) = *bin-add*(*NCons*(*bin-mult*(*v*,*w*),*0*), *w*)
⟨*proof*⟩

lemma *bin-mult-0*: *bin-mult*(*v BIT 0*, *w*) = *NCons*(*bin-mult*(*v*,*w*),*0*)
⟨*proof*⟩

lemma *int-of-0*: $# *#0* = *#0*
⟨*proof*⟩

lemma *int-of-succ*: $# *succ*(*n*) = *#1* $+ $#*n*
⟨*proof*⟩

lemma *zminus-0* [*simp*]: $− *#0* = *#0*
⟨*proof*⟩

lemma *zadd-0-intify* [*simp*]: *#0* $+ *z* = *intify*(*z*)
⟨*proof*⟩

lemma *zadd-0-right-intify* [*simp*]: *z* $+ *#0* = *intify*(*z*)
⟨*proof*⟩

lemma *zmult-1-intify* [*simp*]: *#1* $∗ *z* = *intify*(*z*)
⟨*proof*⟩

lemma *zmult-1-right-intify* [*simp*]: $z \ \$* \ \#1 = intify(z)$
⟨*proof*⟩

lemma *zmult-0* [*simp*]: $\#0 \ \$* \ z = \#0$
⟨*proof*⟩

lemma *zmult-0-right* [*simp*]: $z \ \$* \ \#0 = \#0$
⟨*proof*⟩

lemma *zmult-minus1* [*simp*]: $\#-1 \ \$* \ z = \$-z$
⟨*proof*⟩

lemma *zmult-minus1-right* [*simp*]: $z \ \$* \ \#-1 = \$-z$
⟨*proof*⟩

32.2 Simplification Rules for Comparison of Binary Numbers

Thanks to Norbert Voelker

lemma *eq-integ-of-eq*:
$[\![v \in bin; \ w \in bin]\!]$
$\Longrightarrow ((integ\text{-}of(v)) = integ\text{-}of(w)) \longleftrightarrow$
$iszero \ (integ\text{-}of \ (bin\text{-}add \ (v, \ bin\text{-}minus(w))))$
⟨*proof*⟩

lemma *iszero-integ-of-Pls*: $iszero \ (integ\text{-}of(Pls))$
⟨*proof*⟩

lemma *nonzero-integ-of-Min*: $\neg \ iszero \ (integ\text{-}of(Min))$
⟨*proof*⟩

lemma *iszero-integ-of-BIT*:
$[\![w \in bin; \ x \in bool]\!]$
$\Longrightarrow iszero \ (integ\text{-}of \ (w \ BIT \ x)) \longleftrightarrow (x{=}0 \land iszero \ (integ\text{-}of(w)))$
⟨*proof*⟩

lemma *iszero-integ-of-0*:
$w \in bin \Longrightarrow iszero \ (integ\text{-}of \ (w \ BIT \ 0)) \longleftrightarrow iszero \ (integ\text{-}of(w))$
⟨*proof*⟩

lemma *iszero-integ-of-1*: $w \in bin \Longrightarrow \neg \ iszero \ (integ\text{-}of \ (w \ BIT \ 1))$
⟨*proof*⟩

lemma *less-integ-of-eq-neg*:
  $[\![v \in bin; \; w \in bin]\!]$
  $\Longrightarrow$ *integ-of*(*v*) \$< *integ-of*(*w*)
    $\longleftrightarrow$ *znegative* (*integ-of* (*bin-add* (*v*, *bin-minus*(*w*))))
 ⟨*proof*⟩

lemma *not-neg-integ-of-Pls*: ¬ *znegative* (*integ-of*(*Pls*))
⟨*proof*⟩

lemma *neg-integ-of-Min*: *znegative* (*integ-of*(*Min*))
⟨*proof*⟩

lemma *neg-integ-of-BIT*:
  $[\![w \in bin; x \in bool]\!]$
  $\Longrightarrow$ *znegative* (*integ-of* (*w BIT x*)) $\longleftrightarrow$ *znegative* (*integ-of*(*w*))
⟨*proof*⟩

lemma *le-integ-of-eq-not-less*:
  (*integ-of*(*x*) \$≤ (*integ-of*(*w*))) $\longleftrightarrow$ ¬ (*integ-of*(*w*) \$< (*integ-of*(*x*)))
⟨*proof*⟩

declare *bin-succ-BIT* [*simp del*]
    bin-pred-BIT [*simp del*]
    bin-minus-BIT [*simp del*]
    NCons-Pls [*simp del*]
    NCons-Min [*simp del*]
    bin-adder-BIT [*simp del*]
    bin-mult-BIT [*simp del*]

declare *integ-of-Pls* [*simp del*] *integ-of-Min* [*simp del*] *integ-of-BIT* [*simp del*]

lemmas *bin-arith-extra-simps* =
  integ-of-add [*symmetric*]
  integ-of-minus [*symmetric*]
  integ-of-mult [*symmetric*]
  bin-succ-1 bin-succ-0
  bin-pred-1 bin-pred-0
  bin-minus-1 bin-minus-0
  bin-add-Pls-right bin-add-Min-right
  bin-add-BIT-0 bin-add-BIT-10 bin-add-BIT-11
  diff-integ-of-eq
  bin-mult-1 bin-mult-0 NCons-simps

lemmas *bin-arith-simps* =
bin-pred-Pls bin-pred-Min
bin-succ-Pls bin-succ-Min
bin-add-Pls bin-add-Min
bin-minus-Pls bin-minus-Min
bin-mult-Pls bin-mult-Min
bin-arith-extra-simps

lemmas *bin-rel-simps* =
eq-integ-of-eq iszero-integ-of-Pls nonzero-integ-of-Min
iszero-integ-of-0 iszero-integ-of-1
less-integ-of-eq-neg
not-neg-integ-of-Pls neg-integ-of-Min neg-integ-of-BIT
le-integ-of-eq-not-less

declare *bin-arith-simps* [*simp*]
declare *bin-rel-simps* [*simp*]

lemma *add-integ-of-left* [*simp*]:
$[\![v \in bin;\ \ w \in bin]\!]$
$\Longrightarrow integ\text{-}of(v)\ \$+\ (integ\text{-}of(w)\ \$+\ z) = (integ\text{-}of(bin\text{-}add(v,w))\ \$+\ z)$
⟨*proof*⟩

lemma *mult-integ-of-left* [*simp*]:
$[\![v \in bin;\ \ w \in bin]\!]$
$\Longrightarrow integ\text{-}of(v)\ \$*\ (integ\text{-}of(w)\ \$*\ z) = (integ\text{-}of(bin\text{-}mult(v,w))\ \$*\ z)$
⟨*proof*⟩

lemma *add-integ-of-diff1* [*simp*]:
$[\![v \in bin;\ \ w \in bin]\!]$
$\Longrightarrow integ\text{-}of(v)\ \$+\ (integ\text{-}of(w)\ \$-\ c) = integ\text{-}of(bin\text{-}add(v,w))\ \$-\ (c)$
⟨*proof*⟩

lemma *add-integ-of-diff2* [*simp*]:
$[\![v \in bin;\ \ w \in bin]\!]$
$\Longrightarrow integ\text{-}of(v)\ \$+\ (c\ \$-\ integ\text{-}of(w)) =$
$integ\text{-}of\ (bin\text{-}add\ (v,\ bin\text{-}minus(w)))\ \$+\ (c)$
⟨*proof*⟩

declare *int-of-0* [*simp*] *int-of-succ* [*simp*]

lemma *zdiff0* [*simp*]: $\#0\ \$-\ x = \$-x$
⟨*proof*⟩

lemma *zdiff0-right* [*simp*]: $x\ \$-\ \#0 = intify(x)$
⟨*proof*⟩

lemma *zdiff-self* [*simp*]: $x\ \$-\ x = \#0$
⟨*proof*⟩

lemma *znegative-iff-zless-0*: $k \in int \Longrightarrow znegative(k) \longleftrightarrow k\ \$<\ \#0$
⟨*proof*⟩

lemma *zero-zless-imp-znegative-zminus*: $[\![\#0\ \$<\ k;\ k \in int]\!] \Longrightarrow znegative(\$-k)$
⟨*proof*⟩

lemma *zero-zle-int-of* [*simp*]: $\#0\ \$\leq\ \$\#\ n$
⟨*proof*⟩

lemma *nat-of-0* [*simp*]: $nat\text{-}of(\#0) = 0$
⟨*proof*⟩

lemma *nat-le-int0-lemma*: $[\![z\ \$\leq\ \$\#0;\ z \in int]\!] \Longrightarrow nat\text{-}of(z) = 0$
⟨*proof*⟩

lemma *nat-le-int0*: $z\ \$\leq\ \$\#0 \Longrightarrow nat\text{-}of(z) = 0$
⟨*proof*⟩

lemma *int-of-eq-0-imp-natify-eq-0*: $\$\#\ n = \#0 \Longrightarrow natify(n) = 0$
⟨*proof*⟩

lemma *nat-of-zminus-int-of*: $nat\text{-}of(\$-\ \$\#\ n) = 0$
⟨*proof*⟩

lemma *int-of-nat-of*: $\#0\ \$\leq\ z \Longrightarrow \$\#\ nat\text{-}of(z) = intify(z)$
⟨*proof*⟩

declare *int-of-nat-of* [*simp*] *nat-of-zminus-int-of* [*simp*]

lemma *int-of-nat-of-if*: $\$\#\ nat\text{-}of(z) = (if\ \#0\ \$\leq\ z\ then\ intify(z)\ else\ \#0)$
⟨*proof*⟩

lemma *zless-nat-iff-int-zless*: $[\![m \in nat;\ z \in int]\!] \Longrightarrow (m < nat\text{-}of(z)) \longleftrightarrow (\$\#m$
$\$<\ z)$
⟨*proof*⟩

lemma *zless-nat-conj-lemma*: $\$\#0\ \$<\ z \Longrightarrow (nat\text{-}of(w) < nat\text{-}of(z)) \longleftrightarrow (w\ \$<\ z)$
⟨*proof*⟩

lemma *zless-nat-conj*: $(nat\text{-}of(w) < nat\text{-}of(z)) \longleftrightarrow (\$\#0\ \$<\ z \wedge w\ \$<\ z)$
⟨*proof*⟩

lemma *integ-of-minus-reorient* [*simp*]:
$(integ\text{-}of(w) = \$-\ x) \longleftrightarrow (\$-\ x = integ\text{-}of(w))$
⟨*proof*⟩

lemma *integ-of-add-reorient* [*simp*]:
$(integ\text{-}of(w) = x\ \$+\ y) \longleftrightarrow (x\ \$+\ y = integ\text{-}of(w))$
⟨*proof*⟩

lemma *integ-of-diff-reorient* [*simp*]:
$(integ\text{-}of(w) = x\ \$-\ y) \longleftrightarrow (x\ \$-\ y = integ\text{-}of(w))$
⟨*proof*⟩

lemma *integ-of-mult-reorient* [*simp*]:
$(integ\text{-}of(w) = x\ \$*\ y) \longleftrightarrow (x\ \$*\ y = integ\text{-}of(w))$
⟨*proof*⟩

lemmas [*simp*] =
zminus-equation [**where** $y = integ\text{-}of(w)$]
equation-zminus [**where** $x = integ\text{-}of(w)$]
for w

lemmas [*iff*] =
zminus-zless [**where** $y = integ\text{-}of(w)$]
zless-zminus [**where** $x = integ\text{-}of(w)$]
for w

lemmas [*iff*] =
zminus-zle [**where** $y = integ\text{-}of(w)$]
zle-zminus [**where** $x = integ\text{-}of(w)$]
for w

lemmas [*simp*] =
Let-def [**where** $s = integ\text{-}of(w)$] **for** w

lemma *zless-iff-zdiff-zless-0*: $(x \$< y) \longleftrightarrow (x\$-y \$< \#0)$
⟨*proof*⟩

lemma *eq-iff-zdiff-eq-0*: $[\![x \in int; y \in int]\!] \Longrightarrow (x = y) \longleftrightarrow (x\$-y = \#0)$
⟨*proof*⟩

lemma *zle-iff-zdiff-zle-0*: $(x \$\leq y) \longleftrightarrow (x\$-y \$\leq \#0)$
⟨*proof*⟩

lemma *left-zadd-zmult-distrib*: $i\$*u \$+ (j\$*u \$+ k) = (i\$+j)\$*u \$+ k$
⟨*proof*⟩

lemma *eq-add-iff1*: $(i\$*u \$+ m = j\$*u \$+ n) \longleftrightarrow ((i\$-j)\$*u \$+ m = intify(n))$
⟨*proof*⟩

lemma *eq-add-iff2*: $(i\$*u \$+ m = j\$*u \$+ n) \longleftrightarrow (intify(m) = (j\$-i)\$*u \$+ n)$
⟨*proof*⟩

context fixes $n :: i$
begin

lemmas *rel-iff-rel-0-rls* =
zless-iff-zdiff-zless-0 [**where** $y = u \$+ v$]
eq-iff-zdiff-eq-0 [**where** $y = u \$+ v$]
zle-iff-zdiff-zle-0 [**where** $y = u \$+ v$]
zless-iff-zdiff-zless-0 [**where** $y = n$]
eq-iff-zdiff-eq-0 [**where** $y = n$]
zle-iff-zdiff-zle-0 [**where** $y = n$]
for $u\ v$

lemma *less-add-iff1*: $(i\$*u \$+ m \$< j\$*u \$+ n) \longleftrightarrow ((i\$-j)\$*u \$+ m \$< n)$
⟨*proof*⟩

lemma *less-add-iff2*: $(i\$*u \$+ m \$< j\$*u \$+ n) \longleftrightarrow (m \$< (j\$-i)\$*u \$+ n)$
⟨*proof*⟩

end

lemma *le-add-iff1*: $(i\$*u \$+ m \$\leq j\$*u \$+ n) \longleftrightarrow ((i\$-j)\$*u \$+ m \$\leq n)$
⟨*proof*⟩

lemma *le-add-iff2*: $(i\$*u\ \$+\ m\ \$\le\ j\$*u\ \$+\ n) \longleftrightarrow (m\ \$\le\ (j\$-i)\$*u\ \$+\ n)$
⟨*proof*⟩

⟨*ML*⟩

32.2.1 Examples

combine-numerals-prod (products of separate literals)

lemma $\#5\ \$*\ x\ \$*\ \#3\ =\ y$ ⟨*proof*⟩

schematic-goal $y2\ \$+\ ?x42\ =\ y\ \$+\ y2$ ⟨*proof*⟩

lemma $oo : int \Longrightarrow l\ \$+\ (l\ \$+\ \#2)\ \$+\ oo\ =\ oo$ ⟨*proof*⟩

lemma $\#9\$*x\ \$+\ y\ =\ x\$*\#23\ \$+\ z$ ⟨*proof*⟩
lemma $y\ \$+\ x\ =\ x\ \$+\ z$ ⟨*proof*⟩

lemma $x : int \Longrightarrow x\ \$+\ y\ \$+\ z\ =\ x\ \$+\ z$ ⟨*proof*⟩
lemma $x : int \Longrightarrow y\ \$+\ (z\ \$+\ x)\ =\ z\ \$+\ x$ ⟨*proof*⟩
lemma $z : int \Longrightarrow x\ \$+\ y\ \$+\ z\ =\ (z\ \$+\ y)\ \$+\ (x\ \$+\ w)$ ⟨*proof*⟩
lemma $z : int \Longrightarrow x\$*y\ \$+\ z\ =\ (z\ \$+\ y)\ \$+\ (y\$*x\ \$+\ w)$ ⟨*proof*⟩

lemma $\#-3\ \$*\ x\ \$+\ y\ \$\le\ x\ \$*\ \#2\ \$+\ z$ ⟨*proof*⟩
lemma $y\ \$+\ x\ \$\le\ x\ \$+\ z$ ⟨*proof*⟩
lemma $x\ \$+\ y\ \$+\ z\ \$\le\ x\ \$+\ z$ ⟨*proof*⟩

lemma $y\ \$+\ (z\ \$+\ x)\ \$<\ z\ \$+\ x$ ⟨*proof*⟩
lemma $x\ \$+\ y\ \$+\ z\ \$<\ (z\ \$+\ y)\ \$+\ (x\ \$+\ w)$ ⟨*proof*⟩
lemma $x\$*y\ \$+\ z\ \$<\ (z\ \$+\ y)\ \$+\ (y\$*x\ \$+\ w)$ ⟨*proof*⟩

lemma $l\ \$+\ \#2\ \$+\ \#2\ \$+\ \#2\ \$+\ (l\ \$+\ \#2)\ \$+\ (oo\ \$+\ \#2)\ =\ uu$ ⟨*proof*⟩
lemma $u : int \Longrightarrow \#2\ \$*\ u\ =\ u$ ⟨*proof*⟩
lemma $(i\ \$+\ j\ \$+\ \#12\ \$+\ k)\ \$-\ \#15\ =\ y$ ⟨*proof*⟩
lemma $(i\ \$+\ j\ \$+\ \#12\ \$+\ k)\ \$-\ \#5\ =\ y$ ⟨*proof*⟩

lemma $y\ \$-\ b\ \$<\ b$ ⟨*proof*⟩
lemma $y\ \$-\ (\#3\ \$*\ b\ \$+\ c)\ \$<\ b\ \$-\ \#2\ \$*\ c$ ⟨*proof*⟩

lemma $(\#2\ \$*\ x\ \$-\ (u\ \$*\ v)\ \$+\ y)\ \$-\ v\ \$*\ \#3\ \$*\ u\ =\ w$ ⟨*proof*⟩
lemma $(\#2\ \$*\ x\ \$*\ u\ \$*\ v\ \$+\ (u\ \$*\ v)\ \$*\ \#4\ \$+\ y)\ \$-\ v\ \$*\ u\ \$*\ \#4\ =\ w$
⟨*proof*⟩
lemma $(\#2\ \$*\ x\ \$*\ u\ \$*\ v\ \$+\ (u\ \$*\ v)\ \$*\ \#4\ \$+\ y)\ \$-\ v\ \$*\ u\ =\ w$ ⟨*proof*⟩
lemma $u\ \$*\ v\ \$-\ (x\ \$*\ u\ \$*\ v\ \$+\ (u\ \$*\ v)\ \$*\ \#4\ \$+\ y)\ =\ w$ ⟨*proof*⟩

lemma $(i\ \$+\ j\ \$+\ \#12\ \$+\ k)\ =\ u\ \$+\ \#15\ \$+\ y$ ⟨*proof*⟩
lemma $(i\ \$+\ j\ \$*\ \#2\ \$+\ \#12\ \$+\ k)\ =\ j\ \$+\ \#5\ \$+\ y$ ⟨*proof*⟩

lemma $\#2\ \$*\ y\ \$+\ \#3\ \$*\ z\ \$+\ \#6\ \$*\ w\ \$+\ \#2\ \$*\ y\ \$+\ \#3\ \$*\ z\ \$+\ \#2\ \$*\ u\ =\ \#2\ \$*\ y'\ \$+\ \#3\ \$*\ z'\ \$+\ \#6\ \$*\ w'\ \$+\ \#2\ \$*\ y'\ \$+\ \#3\ \$*\ z'\ \$+\ u\ \$+\ vv$

⟨*proof*⟩

lemma $a \$+ \$-(b\$+c) \$+ b = d$ ⟨*proof*⟩
lemma $a \$+ \$-(b\$+c) \$- b = d$ ⟨*proof*⟩

negative numerals

lemma $(i \$+ j \$+ \#-2 \$+ k) \$- (u \$+ \#5 \$+ y) = zz$ ⟨*proof*⟩
lemma $(i \$+ j \$+ \#-3 \$+ k) \$< u \$+ \#5 \$+ y$ ⟨*proof*⟩
lemma $(i \$+ j \$+ \#3 \$+ k) \$< u \$+ \#-6 \$+ y$ ⟨*proof*⟩
lemma $(i \$+ j \$+ \#-12 \$+ k) \$- \#15 = y$ ⟨*proof*⟩
lemma $(i \$+ j \$+ \#12 \$+ k) \$- \#-15 = y$ ⟨*proof*⟩
lemma $(i \$+ j \$+ \#-12 \$+ k) \$- \#-15 = y$ ⟨*proof*⟩

Multiplying separated numerals

lemma $\#6 \$* (\$\# x \$* \#2) = uu$ ⟨*proof*⟩
lemma $\#4 \$* (\$\# x \$* \$\# x) \$* (\#2 \$* \$\# x) = uu$ ⟨*proof*⟩

end

33 The Division Operators Div and Mod

theory *IntDiv*
imports *Bin OrderArith*
begin

definition
  quorem :: $[i,i] \Rightarrow o$ **where**
    $quorem \equiv \lambda\langle a,b\rangle\ \langle q,r\rangle.$
      $a = b\$*q \$+ r \wedge$
      $(\#0\$<b \wedge \#0\$\leq r \wedge r\$<b \mid \neg(\#0\$<b) \wedge b\$<r \wedge r \$\leq \#0)$

definition
  adjust :: $[i,i] \Rightarrow i$ **where**
    $adjust(b) \equiv \lambda\langle q,r\rangle.\ if\ \#0\ \$\leq r\$-b\ then\ <\#2\$*q \$+ \#1,r\$-b>$
      $else\ <\#2\$*q,r>$

definition
  posDivAlg :: $i \Rightarrow i$ **where**

    $posDivAlg(ab) \equiv$
      $wfrec(measure(int*int, \lambda\langle a,b\rangle.\ nat\text{-}of\ (a \$- b \$+ \#1)),$
        $ab,$
        $\lambda\langle a,b\rangle\ f.\ if\ (a\$<b \mid b\$\leq\#0)\ then\ <\#0,a>$
          $else\ adjust(b, f\ \text{`}\ <a,\#2\$*b>))$

definition
negDivAlg :: *i* ⇒ *i* **where**

negDivAlg(*ab*) ≡
wfrec(*measure*(*int*∗*int*, λ⟨*a*,*b*⟩. *nat-of* ($− *a* $− *b*)),
ab,
λ⟨*a*,*b*⟩ *f*. *if* (#*0* $≤ *a*$+*b* | *b*$≤#*0*) *then* <#−*1*,*a*$+*b*>
else adjust(*b*, *f* ‘ <*a*,#*2*$∗*b*>))

definition
negateSnd :: *i* ⇒ *i* **where**
negateSnd ≡ λ⟨*q*,*r*⟩. <*q*, $−*r*>

definition
divAlg :: *i* ⇒ *i* **where**
divAlg ≡
λ⟨*a*,*b*⟩. *if* #*0* $≤ *a then*
if #*0* $≤ *b then posDivAlg* (⟨*a*,*b*⟩)
else if a=#*0 then* <#*0*,#*0*>
else negateSnd (*negDivAlg* (<$−*a*,$−*b*>))
else
if #*0*$<*b then negDivAlg* (⟨*a*,*b*⟩)
else *negateSnd* (*posDivAlg* (<$−*a*,$−*b*>))

definition
zdiv :: [*i*,*i*]⇒*i* (**infixl** ‹*zdiv*› *70*) **where**
a zdiv b ≡ *fst* (*divAlg* (<*intify*(*a*), *intify*(*b*)>))

definition
zmod :: [*i*,*i*]⇒*i* (**infixl** ‹*zmod*› *70*) **where**
a zmod b ≡ *snd* (*divAlg* (<*intify*(*a*), *intify*(*b*)>))

lemma *zspos-add-zspos-imp-zspos*: ⟦#*0* $< *x*; #*0* $< *y*⟧ ⟹ #*0* $< *x* $+ *y*
⟨*proof*⟩

lemma *zpos-add-zpos-imp-zpos*: ⟦#*0* $≤ *x*; #*0* $≤ *y*⟧ ⟹ #*0* $≤ *x* $+ *y*
⟨*proof*⟩

lemma *zneg-add-zneg-imp-zneg*: ⟦*x* $< #*0*; *y* $< #*0*⟧ ⟹ *x* $+ *y* $< #*0*
⟨*proof*⟩

lemma *zneg-or-0-add-zneg-or-0-imp-zneg-or-0*:
$[\![x \$\leq \#0; \ \ y \$\leq \#0]\!] \Longrightarrow x \$+ y \$\leq \#0$
⟨*proof*⟩

lemma *zero-lt-zmagnitude*: $[\![\#0 \$< k; k \in int]\!] \Longrightarrow 0 < zmagnitude(k)$
⟨*proof*⟩

lemma *zless-add-succ-iff*:
$(w \$< z \$+ \$\# \ succ(m)) \longleftrightarrow (w \$< z \$+ \$\#m \mid intify(w) = z \$+ \$\#m)$
⟨*proof*⟩

lemma *zadd-succ-lemma*:
$z \in int \Longrightarrow (w \$+ \$\# \ succ(m) \$\leq z) \longleftrightarrow (w \$+ \$\#m \$< z)$
⟨*proof*⟩

lemma *zadd-succ-zle-iff*: $(w \$+ \$\# \ succ(m) \$\leq z) \longleftrightarrow (w \$+ \$\#m \$< z)$
⟨*proof*⟩

lemma *zless-add1-iff-zle*: $(w \$< z \$+ \#1) \longleftrightarrow (w\$\leq z)$
⟨*proof*⟩

lemma *add1-zle-iff*: $(w \$+ \#1 \$\leq z) \longleftrightarrow (w \$< z)$
⟨*proof*⟩

lemma *add1-left-zle-iff*: $(\#1 \$+ w \$\leq z) \longleftrightarrow (w \$< z)$
⟨*proof*⟩

lemma *zmult-mono-lemma*: $k \in nat \Longrightarrow i \$\leq j \Longrightarrow i \$* \$\#k \$\leq j \$* \$\#k$
⟨*proof*⟩

lemma *zmult-zle-mono1*: $[\![i \$\leq j; \ \ \#0 \$\leq k]\!] \Longrightarrow i\$*k \$\leq j\$*k$
⟨*proof*⟩

lemma *zmult-zle-mono1-neg*: $[\![i \$\leq j; \ \ k \$\leq \#0]\!] \Longrightarrow j\$*k \$\leq i\$*k$
⟨*proof*⟩

lemma *zmult-zle-mono2*: $[\![i \$\leq j; \ \ \#0 \$\leq k]\!] \Longrightarrow k\$*i \$\leq k\$*j$
⟨*proof*⟩

lemma *zmult-zle-mono2-neg*: $[\![i \ \$\le j;\ \ k \ \$\le \#0]\!] \Longrightarrow k\$*j \ \$\le k\$*i$
⟨*proof*⟩

lemma *zmult-zle-mono*:
$[\![i \ \$\le j;\ \ k \ \$\le l;\ \ \#0 \ \$\le j;\ \ \#0 \ \$\le k]\!] \Longrightarrow i\$*k \ \$\le j\$*l$
⟨*proof*⟩

lemma *zmult-zless-mono2-lemma* [*rule-format*]:
$[\![i\$<j;\ k \in nat]\!] \Longrightarrow 0<k \longrightarrow \$\#k \ \$* \ i \ \$< \$\#k \ \$* \ j$
⟨*proof*⟩

lemma *zmult-zless-mono2*: $[\![i\$<j;\ \ \#0 \ \$< k]\!] \Longrightarrow k\$*i \ \$< k\$*j$
⟨*proof*⟩

lemma *zmult-zless-mono1*: $[\![i\$<j;\ \ \#0 \ \$< k]\!] \Longrightarrow i\$*k \ \$< j\$*k$
⟨*proof*⟩

lemma *zmult-zless-mono*:
$[\![i \ \$< j;\ \ k \ \$< l;\ \ \#0 \ \$< j;\ \ \#0 \ \$< k]\!] \Longrightarrow i\$*k \ \$< j\$*l$
⟨*proof*⟩

lemma *zmult-zless-mono1-neg*: $[\![i \ \$< j;\ \ k \ \$< \#0]\!] \Longrightarrow j\$*k \ \$< i\$*k$
⟨*proof*⟩

lemma *zmult-zless-mono2-neg*: $[\![i \ \$< j;\ \ k \ \$< \#0]\!] \Longrightarrow k\$*j \ \$< k\$*i$
⟨*proof*⟩

lemma *zmult-eq-lemma*:
$[\![m \in int;\ n \in int]\!] \Longrightarrow (m = \#0 \mid n = \#0) \longleftrightarrow (m\$*n = \#0)$
⟨*proof*⟩

lemma *zmult-eq-0-iff* [*iff*]: $(m\$*n = \#0) \longleftrightarrow (intify(m) = \#0 \mid intify(n) = \#0)$
⟨*proof*⟩

lemma *zmult-zless-lemma*:
$[\![k \in int;\ m \in int;\ n \in int]\!]$
$\Longrightarrow (m\$*k \ \$< n\$*k) \longleftrightarrow ((\#0 \ \$< k \wedge m\$<n) \mid (k \ \$< \#0 \wedge n\$<m))$
⟨*proof*⟩

lemma *zmult-zless-cancel2*:
$(m\$*k\ \$<\ n\$*k) \longleftrightarrow ((\#0\ \$<\ k \wedge m\$<n) \mid (k\ \$<\ \#0 \wedge n\$<m))$
⟨*proof*⟩

lemma *zmult-zless-cancel1*:
$(k\$*m\ \$<\ k\$*n) \longleftrightarrow ((\#0\ \$<\ k \wedge m\$<n) \mid (k\ \$<\ \#0 \wedge n\$<m))$
⟨*proof*⟩

lemma *zmult-zle-cancel2*:
$(m\$*k\ \$\leq\ n\$*k) \longleftrightarrow ((\#0\ \$<\ k \longrightarrow m\$\leq n) \wedge (k\ \$<\ \#0 \longrightarrow n\$\leq m))$
⟨*proof*⟩

lemma *zmult-zle-cancel1*:
$(k\$*m\ \$\leq\ k\$*n) \longleftrightarrow ((\#0\ \$<\ k \longrightarrow m\$\leq n) \wedge (k\ \$<\ \#0 \longrightarrow n\$\leq m))$
⟨*proof*⟩

lemma *int-eq-iff-zle*: $[\![m \in int;\ n \in int]\!] \Longrightarrow m=n \longleftrightarrow (m\ \$\leq\ n \wedge n\ \$\leq\ m)$
⟨*proof*⟩

lemma *zmult-cancel2-lemma*:
$[\![k \in int;\ m \in int;\ n \in int]\!] \Longrightarrow (m\$*k = n\$*k) \longleftrightarrow (k=\#0 \mid m=n)$
⟨*proof*⟩

lemma *zmult-cancel2* [*simp*]:
$(m\$*k = n\$*k) \longleftrightarrow (intify(k) = \#0 \mid intify(m) = intify(n))$
⟨*proof*⟩

lemma *zmult-cancel1* [*simp*]:
$(k\$*m = k\$*n) \longleftrightarrow (intify(k) = \#0 \mid intify(m) = intify(n))$
⟨*proof*⟩

33.1 Uniqueness and monotonicity of quotients and remainders

lemma *unique-quotient-lemma*:
$[\![b\$*q'\ \$+\ r'\ \$\leq\ b\$*q\ \$+\ r;\ \ \#0\ \$\leq\ r';\ \ \#0\ \$<\ b;\ \ r\ \$<\ b]\!]$
$\Longrightarrow q'\ \$\leq\ q$
⟨*proof*⟩

lemma *unique-quotient-lemma-neg*:
$[\![b\$*q'\ \$+\ r'\ \$\leq\ b\$*q\ \$+\ r;\ \ r\ \$\leq\ \#0;\ \ b\ \$<\ \#0;\ \ b\ \$<\ r']\!]$
$\Longrightarrow q\ \$\leq\ q'$
⟨*proof*⟩

lemma *unique-quotient*:
$[\![quorem\ (\langle a,b\rangle,\ \langle q,r\rangle);\ \ quorem\ (\langle a,b\rangle,\ <q',r'>);\ \ b \in int;\ b \neq \#0;$
$q \in int;\ q' \in int]\!] \Longrightarrow q = q'$

$\langle proof \rangle$

lemma *unique-remainder*:

$[\![quorem\ (\langle a,b\rangle,\ \langle q,r\rangle);\ \ quorem\ (\langle a,b\rangle,\ <q',r'>);\ \ b \in int;\ b \neq \#0;$
$\quad q \in int;\ q' \in int;$
$\quad r \in int;\ r' \in int]\!] \Longrightarrow r = r'$

$\langle proof \rangle$

33.2 Correctness of posDivAlg, the Division Algorithm for $a \geq 0$ and $b > 0$

lemma *adjust-eq* [*simp*]:

$adjust(b,\ \langle q,r\rangle) = (let\ diff = r\$-b\ in$
$\quad if\ \#0\ \$\leq\ diff\ then\ <\#2\$*q\ \$+\ \#1,diff>$
$\qquad else\ <\#2\$*q,r>)$

$\langle proof \rangle$

lemma *posDivAlg-termination*:

$[\![\#0\ \$<\ b;\ \neg\ a\ \$<\ b]\!]$
$\Longrightarrow nat\text{-}of(a\ \$-\ \#2\ \$*\ b\ \$+\ \#1) < nat\text{-}of(a\ \$-\ b\ \$+\ \#1)$

$\langle proof \rangle$

lemmas *posDivAlg-unfold* = *def-wfrec* [*OF posDivAlg-def wf-measure*]

lemma *posDivAlg-eqn*:

$[\![\#0\ \$<\ b;\ a \in int;\ b \in int]\!] \Longrightarrow$
$posDivAlg(\langle a,b\rangle) =$
$(if\ a\$<b\ then\ <\#0,a>\ else\ adjust(b,\ posDivAlg\ (<a,\ \#2\$*b>)))$

$\langle proof \rangle$

lemma *posDivAlg-induct-lemma* [*rule-format*]:
assumes *prem*:

$\bigwedge a\ b.\ [\![a \in int;\ b \in int;$
$\quad \neg\ (a\ \$<\ b\ |\ b\ \$\leq\ \#0) \longrightarrow P(<a,\ \#2\ \$*\ b>)]\!] \Longrightarrow P(\langle a,b\rangle)$

shows $\langle u,v\rangle \in int*int \Longrightarrow P(\langle u,v\rangle)$

$\langle proof \rangle$

lemma *posDivAlg-induct* [*consumes 2*]:
assumes *u-int*: $u \in int$
and *v-int*: $v \in int$
and *ih*: $\bigwedge a\ b.\ [\![a \in int;\ b \in int;$
$\quad \neg\ (a\ \$<\ b\ |\ b\ \$\leq\ \#0) \longrightarrow P(a,\ \#2\ \$*\ b)]\!] \Longrightarrow P(a,b)$
shows $P(u,v)$

$\langle proof \rangle$

lemma *intify-eq-0-iff-zle*: $intify(m) = \#0 \longleftrightarrow (m\ \$\leq\ \#0 \wedge \#0\ \$\leq\ m)$

⟨*proof*⟩

33.3 Some convenient biconditionals for products of signs

lemma *zmult-pos*: $[\![\#0 \ \$< \ i; \ \#0 \ \$< \ j]\!] \Longrightarrow \#0 \ \$< \ i \ \$* \ j$
⟨*proof*⟩

lemma *zmult-neg*: $[\![i \ \$< \ \#0; \ j \ \$< \ \#0]\!] \Longrightarrow \#0 \ \$< \ i \ \$* \ j$
⟨*proof*⟩

lemma *zmult-pos-neg*: $[\![\#0 \ \$< \ i; \ j \ \$< \ \#0]\!] \Longrightarrow i \ \$* \ j \ \$< \ \#0$
⟨*proof*⟩

lemma *int-0-less-lemma*:
$[\![x \in int; \ y \in int]\!]$
$\Longrightarrow (\#0 \ \$< \ x \ \$* \ y) \longleftrightarrow (\#0 \ \$< \ x \wedge \#0 \ \$< \ y \mid x \ \$< \ \#0 \wedge y \ \$< \ \#0)$
⟨*proof*⟩

lemma *int-0-less-mult-iff*:
$(\#0 \ \$< \ x \ \$* \ y) \longleftrightarrow (\#0 \ \$< \ x \wedge \#0 \ \$< \ y \mid x \ \$< \ \#0 \wedge y \ \$< \ \#0)$
⟨*proof*⟩

lemma *int-0-le-lemma*:
$[\![x \in int; \ y \in int]\!]$
$\Longrightarrow (\#0 \ \$\leq \ x \ \$* \ y) \longleftrightarrow (\#0 \ \$\leq \ x \wedge \#0 \ \$\leq \ y \mid x \ \$\leq \ \#0 \wedge y \ \$\leq \ \#0)$
⟨*proof*⟩

lemma *int-0-le-mult-iff*:
$(\#0 \ \$\leq \ x \ \$* \ y) \longleftrightarrow ((\#0 \ \$\leq \ x \wedge \#0 \ \$\leq \ y) \mid (x \ \$\leq \ \#0 \wedge y \ \$\leq \ \#0))$
⟨*proof*⟩

lemma *zmult-less-0-iff*:
$(x \ \$* \ y \ \$< \ \#0) \longleftrightarrow (\#0 \ \$< \ x \wedge y \ \$< \ \#0 \mid x \ \$< \ \#0 \wedge \#0 \ \$< \ y)$
⟨*proof*⟩

lemma *zmult-le-0-iff*:
$(x \ \$* \ y \ \$\leq \ \#0) \longleftrightarrow (\#0 \ \$\leq \ x \wedge y \ \$\leq \ \#0 \mid x \ \$\leq \ \#0 \wedge \#0 \ \$\leq \ y)$
⟨*proof*⟩

lemma *posDivAlg-type* [*rule-format*]:
$[\![a \in int; \ b \in int]\!] \Longrightarrow posDivAlg(\langle a,b \rangle) \in int * int$
⟨*proof*⟩

lemma *posDivAlg-correct* [*rule-format*]:
  ⟦*a* ∈ *int*; *b* ∈ *int*⟧
  ⟹ *#0* $≤ *a* ⟶ *#0* $< *b* ⟶ *quorem* (⟨*a*,*b*⟩, *posDivAlg*(⟨*a*,*b*⟩))
⟨*proof*⟩

33.4 Correctness of negDivAlg, the division algorithm for a<0 and b>0

lemma *negDivAlg-termination*:
  ⟦*#0* $< *b*; *a* $+ *b* $< *#0*⟧
  ⟹ *nat-of*($− *a* $− *#2* $∗ *b*) < *nat-of*($− *a* $− *b*)
⟨*proof*⟩

lemmas *negDivAlg-unfold* = *def-wfrec* [*OF negDivAlg-def wf-measure*]

lemma *negDivAlg-eqn*:
  ⟦*#0* $< *b*; *a* ∈ *int*; *b* ∈ *int*⟧ ⟹
  negDivAlg(⟨*a*,*b*⟩) =
  (*if* *#0* $≤ *a*$+*b* *then* <#−*1*,*a*$+*b*>
   else *adjust*(*b*, *negDivAlg* (<*a*, #*2*$∗*b*>)))
⟨*proof*⟩

lemma *negDivAlg-induct-lemma* [*rule-format*]:
 assumes *prem*:
  ⋀*a b*. ⟦*a* ∈ *int*; *b* ∈ *int*;
   ¬ (*#0* $≤ *a* $+ *b* | *b* $≤ *#0*) ⟶ *P*(<*a*, *#2* $∗ *b*>)⟧
  ⟹ *P*(⟨*a*,*b*⟩)
 shows ⟨*u*,*v*⟩ ∈ *int*∗*int* ⟹ *P*(⟨*u*,*v*⟩)
⟨*proof*⟩

lemma *negDivAlg-induct* [*consumes 2*]:
 assumes *u-int*: *u* ∈ *int*
  and *v-int*: *v* ∈ *int*
  and *ih*: ⋀*a b*. ⟦*a* ∈ *int*; *b* ∈ *int*;
   ¬ (*#0* $≤ *a* $+ *b* | *b* $≤ *#0*) ⟶ *P*(*a*, *#2* $∗ *b*)⟧
  ⟹ *P*(*a*,*b*)
 shows *P*(*u*,*v*)
⟨*proof*⟩

lemma *negDivAlg-type*:
  ⟦*a* ∈ *int*; *b* ∈ *int*⟧ ⟹ *negDivAlg*(⟨*a*,*b*⟩) ∈ *int* ∗ *int*
⟨*proof*⟩

lemma *negDivAlg-correct* [*rule-format*]:
  ⟦*a* ∈ *int*; *b* ∈ *int*⟧

$\Longrightarrow a \ \$< \#0 \longrightarrow \#0 \ \$< b \longrightarrow quorem\ (\langle a,b\rangle,\ negDivAlg(\langle a,b\rangle))$
⟨proof⟩

33.5 Existence shown by proving the division algorithm to be correct

lemma *quorem-0*: $[\![b \neq \#0;\ \ b \in int]\!] \Longrightarrow quorem\ (<\#0,b>,\ <\#0,\#0>)$
⟨proof⟩

lemma *posDivAlg-zero-divisor*: $posDivAlg(<a,\#0>) = <\#0,a>$
⟨proof⟩

lemma *posDivAlg-0* [*simp*]: $posDivAlg\ (<\#0,b>) = <\#0,\#0>$
⟨proof⟩

lemma *linear-arith-lemma*: $\neg\ (\#0\ \$\leq \#{-1}\ \$+\ b) \Longrightarrow (b\ \$\leq \#0)$
⟨proof⟩

lemma *negDivAlg-minus1* [*simp*]: $negDivAlg\ (<\#{-1},b>) = <\#{-1},\ b\$-\#1>$
⟨proof⟩

lemma *negateSnd-eq* [*simp*]: $negateSnd\ (\langle q,r\rangle) = <q,\ \$-r>$
⟨proof⟩

lemma *negateSnd-type*: $qr \in int * int \Longrightarrow negateSnd\ (qr) \in int * int$
⟨proof⟩

lemma *quorem-neg*:
$[\![quorem\ (<\$-a,\$-b>,\ qr);\ \ a \in int;\ \ b \in int;\ \ qr \in int * int]\!]$
$\Longrightarrow quorem\ (\langle a,b\rangle,\ negateSnd(qr))$
⟨proof⟩

lemma *divAlg-correct*:
$[\![b \neq \#0;\ \ a \in int;\ \ b \in int]\!] \Longrightarrow quorem\ (\langle a,b\rangle,\ divAlg(\langle a,b\rangle))$
⟨proof⟩

lemma *divAlg-type*: $[\![a \in int;\ \ b \in int]\!] \Longrightarrow divAlg(\langle a,b\rangle) \in int * int$
⟨proof⟩

lemma *zdiv-intify1* [*simp*]: $intify(x)\ zdiv\ y = x\ zdiv\ y$
⟨proof⟩

lemma *zdiv-intify2* [*simp*]: $x\ zdiv\ intify(y) = x\ zdiv\ y$
⟨proof⟩

lemma *zdiv-type* [*iff*,*TC*]: z *zdiv* $w \in int$
⟨*proof*⟩

lemma *zmod-intify1* [*simp*]: *intify*(x) *zmod* $y = x$ *zmod* y
⟨*proof*⟩

lemma *zmod-intify2* [*simp*]: x *zmod* *intify*(y) $= x$ *zmod* y
⟨*proof*⟩

lemma *zmod-type* [*iff*,*TC*]: z *zmod* $w \in int$
⟨*proof*⟩

lemma *DIVISION-BY-ZERO-ZDIV*: a *zdiv* #*0* = #*0*
⟨*proof*⟩

lemma *DIVISION-BY-ZERO-ZMOD*: a *zmod* #*0* = *intify*(a)
⟨*proof*⟩

lemma *raw-zmod-zdiv-equality*:
$[\![a \in int;\ b \in int]\!] \Longrightarrow a = b$ \$∗ (a *zdiv* b) \$+ (a *zmod* b)
⟨*proof*⟩

lemma *zmod-zdiv-equality*: *intify*(a) = b \$∗ (a *zdiv* b) \$+ (a *zmod* b)
⟨*proof*⟩

lemma *pos-mod*: #*0* \$< $b \Longrightarrow$ #*0* \$≤ a *zmod* $b \wedge a$ *zmod* b \$< b
⟨*proof*⟩

lemmas *pos-mod-sign* = *pos-mod* [*THEN conjunct1*]
and *pos-mod-bound* = *pos-mod* [*THEN conjunct2*]

lemma *neg-mod*: b \$< #*0* $\Longrightarrow a$ *zmod* b \$≤ #*0* $\wedge$ b \$< a *zmod* b
⟨*proof*⟩

lemmas *neg-mod-sign* = *neg-mod* [*THEN conjunct1*]
and *neg-mod-bound* = *neg-mod* [*THEN conjunct2*]

lemma *quorem-div-mod*:
$[\![b \neq$ #*0*; $a \in int$; $b \in int]\!]$

$\Longrightarrow$ *quorem* ($\langle a,b\rangle$, <*a zdiv b*, *a zmod b*>)
⟨*proof*⟩

lemma *quorem-div*:
$[\![quorem(\langle a,b\rangle,\langle q,r\rangle);\ \ b \neq \#0;\ \ a \in int;\ \ b \in int;\ \ q \in int]\!]$
$\Longrightarrow a\ zdiv\ b = q$
⟨*proof*⟩

lemma *quorem-mod*:
$[\![quorem(\langle a,b\rangle,\langle q,r\rangle);\ b \neq \#0;\ a \in int;\ b \in int;\ q \in int;\ r \in int]\!]$
$\Longrightarrow a\ zmod\ b = r$
⟨*proof*⟩

lemma *zdiv-pos-pos-trivial-raw*:
$[\![a \in int;\ \ b \in int;\ \ \#0\ \$\leq\ a;\ \ a\ \$<\ b]\!] \Longrightarrow a\ zdiv\ b = \#0$
⟨*proof*⟩

lemma *zdiv-pos-pos-trivial*: $[\![\#0\ \$\leq\ a;\ \ a\ \$<\ b]\!] \Longrightarrow a\ zdiv\ b = \#0$
⟨*proof*⟩

lemma *zdiv-neg-neg-trivial-raw*:
$[\![a \in int;\ \ b \in int;\ \ a\ \$\leq\ \#0;\ \ b\ \$<\ a]\!] \Longrightarrow a\ zdiv\ b = \#0$
⟨*proof*⟩

lemma *zdiv-neg-neg-trivial*: $[\![a\ \$\leq\ \#0;\ \ b\ \$<\ a]\!] \Longrightarrow a\ zdiv\ b = \#0$
⟨*proof*⟩

lemma *zadd-le-0-lemma*: $[\![a\$+b\ \$\leq\ \#0;\ \ \#0\ \$<\ a;\ \ \#0\ \$<\ b]\!] \Longrightarrow False$
⟨*proof*⟩

lemma *zdiv-pos-neg-trivial-raw*:
$[\![a \in int;\ \ b \in int;\ \ \#0\ \$<\ a;\ \ a\$+b\ \$\leq\ \#0]\!] \Longrightarrow a\ zdiv\ b = \#{-1}$
⟨*proof*⟩

lemma *zdiv-pos-neg-trivial*: $[\![\#0\ \$<\ a;\ \ a\$+b\ \$\leq\ \#0]\!] \Longrightarrow a\ zdiv\ b = \#{-1}$
⟨*proof*⟩

lemma *zmod-pos-pos-trivial-raw*:
$[\![a \in int;\ \ b \in int;\ \ \#0\ \$\leq\ a;\ \ a\ \$<\ b]\!] \Longrightarrow a\ zmod\ b = a$
⟨*proof*⟩

lemma *zmod-pos-pos-trivial*: $[\![\#0\ \$\leq\ a;\ \ a\ \$<\ b]\!] \Longrightarrow a\ zmod\ b = intify(a)$
⟨*proof*⟩

lemma *zmod-neg-neg-trivial-raw*:

$[\![a \in int;\ \ b \in int;\ \ a \ \$\leq \#0;\ \ b \ \$< a]\!] \Longrightarrow a\ zmod\ b = a$
⟨proof⟩

lemma *zmod-neg-neg-trivial*: $[\![a \ \$\leq \#0;\ \ b \ \$< a]\!] \Longrightarrow a\ zmod\ b = intify(a)$
⟨proof⟩

lemma *zmod-pos-neg-trivial-raw*:
$[\![a \in int;\ \ b \in int;\ \ \#0 \ \$< a;\ \ a\$+b \ \$\leq \#0]\!] \Longrightarrow a\ zmod\ b = a\$+b$
⟨proof⟩

lemma *zmod-pos-neg-trivial*: $[\![\#0 \ \$< a;\ \ a\$+b \ \$\leq \#0]\!] \Longrightarrow a\ zmod\ b = a\$+b$
⟨proof⟩

lemma *zdiv-zminus-zminus-raw*:
$[\![a \in int;\ \ b \in int]\!] \Longrightarrow (\$-a)\ zdiv\ (\$-b) = a\ zdiv\ b$
⟨proof⟩

lemma *zdiv-zminus-zminus* [*simp*]: $(\$-a)\ zdiv\ (\$-b) = a\ zdiv\ b$
⟨proof⟩

lemma *zmod-zminus-zminus-raw*:
$[\![a \in int;\ \ b \in int]\!] \Longrightarrow (\$-a)\ zmod\ (\$-b) = \$-\ (a\ zmod\ b)$
⟨proof⟩

lemma *zmod-zminus-zminus* [*simp*]: $(\$-a)\ zmod\ (\$-b) = \$-\ (a\ zmod\ b)$
⟨proof⟩

33.6 division of a number by itself

lemma *self-quotient-aux1*: $[\![\#0 \ \$< a;\ a = r \ \$+\ a\$*q;\ r \ \$< a]\!] \Longrightarrow \#1 \ \$\leq q$
⟨proof⟩

lemma *self-quotient-aux2*: $[\![\#0 \ \$< a;\ a = r \ \$+\ a\$*q;\ \#0 \ \$\leq r]\!] \Longrightarrow q \ \$\leq \#1$
⟨proof⟩

lemma *self-quotient*:
$[\![quorem(\langle a,a\rangle,\langle q,r\rangle);\ \ a \in int;\ \ q \in int;\ \ a \neq \#0]\!] \Longrightarrow q = \#1$
⟨proof⟩

lemma *self-remainder*:
$[\![quorem(\langle a,a\rangle,\langle q,r\rangle);\ a \in int;\ q \in int;\ r \in int;\ a \neq \#0]\!] \Longrightarrow r = \#0$
⟨proof⟩

lemma *zdiv-self-raw*: $[\![a \neq \#0;\ a \in int]\!] \Longrightarrow a\ zdiv\ a = \#1$
$\langle proof \rangle$

lemma *zdiv-self* [*simp*]: $intify(a) \neq \#0 \Longrightarrow a\ zdiv\ a = \#1$
$\langle proof \rangle$

lemma *zmod-self-raw*: $a \in int \Longrightarrow a\ zmod\ a = \#0$
$\langle proof \rangle$

lemma *zmod-self* [*simp*]: $a\ zmod\ a = \#0$
$\langle proof \rangle$

33.7 Computation of division and remainder

lemma *zdiv-zero* [*simp*]: $\#0\ zdiv\ b = \#0$
$\langle proof \rangle$

lemma *zdiv-eq-minus1*: $\#0\ \$<\ b \Longrightarrow \#-1\ zdiv\ b = \#-1$
$\langle proof \rangle$

lemma *zmod-zero* [*simp*]: $\#0\ zmod\ b = \#0$
$\langle proof \rangle$

lemma *zdiv-minus1*: $\#0\ \$<\ b \Longrightarrow \#-1\ zdiv\ b = \#-1$
$\langle proof \rangle$

lemma *zmod-minus1*: $\#0\ \$<\ b \Longrightarrow \#-1\ zmod\ b = b\ \$-\ \#1$
$\langle proof \rangle$

lemma *zdiv-pos-pos*: $[\![\#0\ \$<\ a;\ \ \#0\ \$\leq\ b]\!]$
$\Longrightarrow a\ zdiv\ b = fst\ (posDivAlg(<intify(a),\ intify(b)>))$
$\langle proof \rangle$

lemma *zmod-pos-pos*:
$[\![\#0\ \$<\ a;\ \ \#0\ \$\leq\ b]\!]$
$\Longrightarrow a\ zmod\ b = snd\ (posDivAlg(<intify(a),\ intify(b)>))$
$\langle proof \rangle$

lemma *zdiv-neg-pos*:
$[\![a\ \$<\ \#0;\ \ \#0\ \$<\ b]\!]$
$\Longrightarrow a\ zdiv\ b = fst\ (negDivAlg(<intify(a),\ intify(b)>))$
$\langle proof \rangle$

lemma *zmod-neg-pos*:

⟦*a* $< #*0*; #*0* $< *b*⟧
⟹ *a zmod b* = *snd* (*negDivAlg*(<*intify*(*a*), *intify*(*b*)>))
⟨*proof*⟩

lemma *zdiv-pos-neg*:
⟦#*0* $< *a*; *b* $< #*0*⟧
⟹ *a zdiv b* = *fst* (*negateSnd*(*negDivAlg* (<$−*a*, $−*b*>)))
⟨*proof*⟩

lemma *zmod-pos-neg*:
⟦#*0* $< *a*; *b* $< #*0*⟧
⟹ *a zmod b* = *snd* (*negateSnd*(*negDivAlg* (<$−*a*, $−*b*>)))
⟨*proof*⟩

lemma *zdiv-neg-neg*:
⟦*a* $< #*0*; *b* $≤ #*0*⟧
⟹ *a zdiv b* = *fst* (*negateSnd*(*posDivAlg*(<$−*a*, $−*b*>)))
⟨*proof*⟩

lemma *zmod-neg-neg*:
⟦*a* $< #*0*; *b* $≤ #*0*⟧
⟹ *a zmod b* = *snd* (*negateSnd*(*posDivAlg*(<$−*a*, $−*b*>)))
⟨*proof*⟩

declare *zdiv-pos-pos* [*of integ-of* (*v*) *integ-of* (*w*), *simp*] **for** *v w*
declare *zdiv-neg-pos* [*of integ-of* (*v*) *integ-of* (*w*), *simp*] **for** *v w*
declare *zdiv-pos-neg* [*of integ-of* (*v*) *integ-of* (*w*), *simp*] **for** *v w*
declare *zdiv-neg-neg* [*of integ-of* (*v*) *integ-of* (*w*), *simp*] **for** *v w*
declare *zmod-pos-pos* [*of integ-of* (*v*) *integ-of* (*w*), *simp*] **for** *v w*
declare *zmod-neg-pos* [*of integ-of* (*v*) *integ-of* (*w*), *simp*] **for** *v w*
declare *zmod-pos-neg* [*of integ-of* (*v*) *integ-of* (*w*), *simp*] **for** *v w*
declare *zmod-neg-neg* [*of integ-of* (*v*) *integ-of* (*w*), *simp*] **for** *v w*
declare *posDivAlg-eqn* [*of concl*: *integ-of* (*v*) *integ-of* (*w*), *simp*] **for** *v w*
declare *negDivAlg-eqn* [*of concl*: *integ-of* (*v*) *integ-of* (*w*), *simp*] **for** *v w*

lemma *zmod-1* [*simp*]: *a zmod* #*1* = #*0*
⟨*proof*⟩

lemma *zdiv-1* [*simp*]: *a zdiv* #*1* = *intify*(*a*)
⟨*proof*⟩

lemma *zmod-minus1-right* [*simp*]: *a zmod* #−*1* = #*0*

⟨*proof*⟩

lemma *zdiv-minus1-right-raw*: $a \in int \Longrightarrow a\ zdiv\ \#{-1} = \$-a$
⟨*proof*⟩

lemma *zdiv-minus1-right*: $a\ zdiv\ \#{-1} = \$-a$
⟨*proof*⟩
declare *zdiv-minus1-right* [*simp*]

33.8 Monotonicity in the first argument (divisor)

lemma *zdiv-mono1*: $[\![a\ \$\le\ a';\ \ \#0\ \$<\ b]\!] \Longrightarrow a\ zdiv\ b\ \$\le\ a'\ zdiv\ b$
⟨*proof*⟩

lemma *zdiv-mono1-neg*: $[\![a\ \$\le\ a';\ \ b\ \$<\ \#0]\!] \Longrightarrow a'\ zdiv\ b\ \$\le\ a\ zdiv\ b$
⟨*proof*⟩

33.9 Monotonicity in the second argument (dividend)

lemma *q-pos-lemma*:
$[\![\#0\ \$\le\ b'\$*q'\ \$+\ r';\ r'\ \$<\ b';\ \ \#0\ \$<\ b']\!] \Longrightarrow \#0\ \$\le\ q'$
⟨*proof*⟩

lemma *zdiv-mono2-lemma*:
$[\![b\$*q\ \$+\ r = b'\$*q'\ \$+\ r';\ \ \#0\ \$\le\ b'\$*q'\ \$+\ r';$
$r'\ \$<\ b';\ \ \#0\ \$\le\ r;\ \ \#0\ \$<\ b';\ \ b'\ \$\le\ b]\!]$
$\Longrightarrow q\ \$\le\ q'$
⟨*proof*⟩

lemma *zdiv-mono2-raw*:
$[\![\#0\ \$\le\ a;\ \ \#0\ \$<\ b';\ \ b'\ \$\le\ b;\ \ a \in int]\!]$
$\Longrightarrow a\ zdiv\ b\ \$\le\ a\ zdiv\ b'$
⟨*proof*⟩

lemma *zdiv-mono2*:
$[\![\#0\ \$\le\ a;\ \ \#0\ \$<\ b';\ \ b'\ \$\le\ b]\!]$
$\Longrightarrow a\ zdiv\ b\ \$\le\ a\ zdiv\ b'$
⟨*proof*⟩

lemma *q-neg-lemma*:
$[\![b'\$*q'\ \$+\ r'\ \$<\ \#0;\ \ \#0\ \$\le\ r';\ \ \#0\ \$<\ b']\!] \Longrightarrow q'\ \$<\ \#0$
⟨*proof*⟩

lemma *zdiv-mono2-neg-lemma*:
$[\![b\$*q\ \$+\ r = b'\$*q'\ \$+\ r';\ \ b'\$*q'\ \$+\ r'\ \$<\ \#0;$
$r\ \$<\ b;\ \ \#0\ \$\le\ r';\ \ \#0\ \$<\ b';\ \ b'\ \$\le\ b]\!]$
$\Longrightarrow q'\ \$\le\ q$

⟨*proof*⟩

lemma *zdiv-mono2-neg-raw*:
  $[\![a \ \$< \#0; \ \#0 \ \$< b'; \ b' \ \$\le b; \ a \in int]\!]$
  $\Longrightarrow a \ zdiv \ b' \ \$\le a \ zdiv \ b$
⟨*proof*⟩

lemma *zdiv-mono2-neg*: $[\![a \ \$< \#0; \ \#0 \ \$< b'; \ b' \ \$\le b]\!]$
  $\Longrightarrow a \ zdiv \ b' \ \$\le a \ zdiv \ b$
⟨*proof*⟩

33.10 More algebraic laws for zdiv and zmod

lemma *zmult1-lemma*:
  $[\![quorem(\langle b,c\rangle, \langle q,r\rangle); \ c \in int; \ c \ne \#0]\!]$
  $\Longrightarrow quorem \ (<a\$*b, \ c>, <a\$*q \ \$+ \ (a\$*r) \ zdiv \ c, \ (a\$*r) \ zmod \ c>)$
⟨*proof*⟩

lemma *zdiv-zmult1-eq-raw*:
  $[\![b \in int; \ c \in int]\!]$
  $\Longrightarrow (a\$*b) \ zdiv \ c = a\$*(b \ zdiv \ c) \ \$+ \ a\$*(b \ zmod \ c) \ zdiv \ c$
⟨*proof*⟩

lemma *zdiv-zmult1-eq*: $(a\$*b) \ zdiv \ c = a\$*(b \ zdiv \ c) \ \$+ \ a\$*(b \ zmod \ c) \ zdiv \ c$
⟨*proof*⟩

lemma *zmod-zmult1-eq-raw*:
  $[\![b \in int; \ c \in int]\!] \Longrightarrow (a\$*b) \ zmod \ c = a\$*(b \ zmod \ c) \ zmod \ c$
⟨*proof*⟩

lemma *zmod-zmult1-eq*: $(a\$*b) \ zmod \ c = a\$*(b \ zmod \ c) \ zmod \ c$
⟨*proof*⟩

lemma *zmod-zmult1-eq'*: $(a\$*b) \ zmod \ c = ((a \ zmod \ c) \ \$* \ b) \ zmod \ c$
⟨*proof*⟩

lemma *zmod-zmult-distrib*: $(a\$*b) \ zmod \ c = ((a \ zmod \ c) \ \$* \ (b \ zmod \ c)) \ zmod \ c$
⟨*proof*⟩

lemma *zdiv-zmult-self1* [*simp*]: $intify(b) \ne \#0 \Longrightarrow (a\$*b) \ zdiv \ b = intify(a)$
⟨*proof*⟩

lemma *zdiv-zmult-self2* [*simp*]: $intify(b) \ne \#0 \Longrightarrow (b\$*a) \ zdiv \ b = intify(a)$
⟨*proof*⟩

lemma *zmod-zmult-self1* [*simp*]: $(a\$*b) \ zmod \ b = \#0$
⟨*proof*⟩

lemma *zmod-zmult-self2* [*simp*]: $(b\$*a) \ zmod \ b = \#0$

⟨proof⟩

lemma *zadd1-lemma*:
⟦quorem(⟨a,c⟩, ⟨aq,ar⟩); quorem(⟨b,c⟩, ⟨bq,br⟩);
c ∈ int; c ≠ #0⟧
⟹ quorem (<a$+b, c>, <aq $+ bq $+ (ar$+br) zdiv c, (ar$+br) zmod c>)
⟨proof⟩

lemma *zdiv-zadd1-eq-raw*:
⟦a ∈ int; b ∈ int; c ∈ int⟧ ⟹
(a$+b) zdiv c = a zdiv c $+ b zdiv c $+ ((a zmod c $+ b zmod c) zdiv c)
⟨proof⟩

lemma *zdiv-zadd1-eq*:
(a$+b) zdiv c = a zdiv c $+ b zdiv c $+ ((a zmod c $+ b zmod c) zdiv c)
⟨proof⟩

lemma *zmod-zadd1-eq-raw*:
⟦a ∈ int; b ∈ int; c ∈ int⟧
⟹ (a$+b) zmod c = (a zmod c $+ b zmod c) zmod c
⟨proof⟩

lemma *zmod-zadd1-eq*: (a$+b) zmod c = (a zmod c $+ b zmod c) zmod c
⟨proof⟩

lemma *zmod-div-trivial-raw*:
⟦a ∈ int; b ∈ int⟧ ⟹ (a zmod b) zdiv b = #0
⟨proof⟩

lemma *zmod-div-trivial* [simp]: (a zmod b) zdiv b = #0
⟨proof⟩

lemma *zmod-mod-trivial-raw*:
⟦a ∈ int; b ∈ int⟧ ⟹ (a zmod b) zmod b = a zmod b
⟨proof⟩

lemma *zmod-mod-trivial* [simp]: (a zmod b) zmod b = a zmod b
⟨proof⟩

lemma *zmod-zadd-left-eq*: (a$+b) zmod c = ((a zmod c) $+ b) zmod c
⟨proof⟩

lemma *zmod-zadd-right-eq*: (a$+b) zmod c = (a $+ (b zmod c)) zmod c
⟨proof⟩

lemma *zdiv-zadd-self1* [*simp*]:
  intify(*a*) ≠ #*0* ⟹ (*a*$+*b*) *zdiv a* = *b zdiv a* $+ #*1*
⟨*proof*⟩

lemma *zdiv-zadd-self2* [*simp*]:
  intify(*a*) ≠ #*0* ⟹ (*b*$+*a*) *zdiv a* = *b zdiv a* $+ #*1*
⟨*proof*⟩

lemma *zmod-zadd-self1* [*simp*]: (*a*$+*b*) *zmod a* = *b zmod a*
⟨*proof*⟩

lemma *zmod-zadd-self2* [*simp*]: (*b*$+*a*) *zmod a* = *b zmod a*
⟨*proof*⟩

33.11 proving a zdiv (b*c) = (a zdiv b) zdiv c

lemma *zdiv-zmult2-aux1*:
  ⟦#0 $< c; b $< r; r $≤ #0⟧ ⟹ b$*c $< b$*(q zmod c) $+ r
⟨*proof*⟩

lemma *zdiv-zmult2-aux2*:
  ⟦#0 $< c; b $< r; r $≤ #0⟧ ⟹ b $* (q zmod c) $+ r $≤ #0
⟨*proof*⟩

lemma *zdiv-zmult2-aux3*:
  ⟦#0 $< c; #0 $≤ r; r $< b⟧ ⟹ #0 $≤ b $* (q zmod c) $+ r
⟨*proof*⟩

lemma *zdiv-zmult2-aux4*:
  ⟦#0 $< c; #0 $≤ r; r $< b⟧ ⟹ b $* (q zmod c) $+ r $< b $* c
⟨*proof*⟩

lemma *zdiv-zmult2-lemma*:
  ⟦quorem (⟨a,b⟩, ⟨q,r⟩); a ∈ int; b ∈ int; b ≠ #0; #0 $< c⟧
  ⟹ quorem (<a,b$*c>, <q zdiv c, b$*(q zmod c) $+ r>)
⟨*proof*⟩

lemma *zdiv-zmult2-eq-raw*:
  ⟦#0 $< c; a ∈ int; b ∈ int⟧ ⟹ a zdiv (b$*c) = (a zdiv b) zdiv c
⟨*proof*⟩

lemma *zdiv-zmult2-eq*: #0 $< c ⟹ a zdiv (b$*c) = (a zdiv b) zdiv c
⟨*proof*⟩

lemma *zmod-zmult2-eq-raw*:
  ⟦#0 $< c; a ∈ int; b ∈ int⟧
  ⟹ a zmod (b$*c) = b$*(a zdiv b zmod c) $+ a zmod b
⟨*proof*⟩

lemma *zmod-zmult2-eq*:
$\#0 \ \$< \ c \Longrightarrow a \ zmod \ (b\$*c) = b\$*(a \ zdiv \ b \ zmod \ c) \ \$+ \ a \ zmod \ b$
⟨*proof*⟩

33.12 Cancellation of common factors in "zdiv"

lemma *zdiv-zmult-zmult1-aux1*:
$[\![\#0 \ \$< \ b; \ \ intify(c) \neq \#0]\!] \Longrightarrow (c\$*a) \ zdiv \ (c\$*b) = a \ zdiv \ b$
⟨*proof*⟩

lemma *zdiv-zmult-zmult1-aux2*:
$[\![b \ \$< \ \#0; \ \ intify(c) \neq \#0]\!] \Longrightarrow (c\$*a) \ zdiv \ (c\$*b) = a \ zdiv \ b$
⟨*proof*⟩

lemma *zdiv-zmult-zmult1-raw*:
$[\![intify(c) \neq \#0; \ b \in int]\!] \Longrightarrow (c\$*a) \ zdiv \ (c\$*b) = a \ zdiv \ b$
⟨*proof*⟩

lemma *zdiv-zmult-zmult1*: $intify(c) \neq \#0 \Longrightarrow (c\$*a) \ zdiv \ (c\$*b) = a \ zdiv \ b$
⟨*proof*⟩

lemma *zdiv-zmult-zmult2*: $intify(c) \neq \#0 \Longrightarrow (a\$*c) \ zdiv \ (b\$*c) = a \ zdiv \ b$
⟨*proof*⟩

33.13 Distribution of factors over "zmod"

lemma *zmod-zmult-zmult1-aux1*:
$[\![\#0 \ \$< \ b; \ \ intify(c) \neq \#0]\!]$
$\Longrightarrow (c\$*a) \ zmod \ (c\$*b) = c \ \$* \ (a \ zmod \ b)$
⟨*proof*⟩

lemma *zmod-zmult-zmult1-aux2*:
$[\![b \ \$< \ \#0; \ \ intify(c) \neq \#0]\!]$
$\Longrightarrow (c\$*a) \ zmod \ (c\$*b) = c \ \$* \ (a \ zmod \ b)$
⟨*proof*⟩

lemma *zmod-zmult-zmult1-raw*:
$[\![b \in int; \ c \in int]\!] \Longrightarrow (c\$*a) \ zmod \ (c\$*b) = c \ \$* \ (a \ zmod \ b)$
⟨*proof*⟩

lemma *zmod-zmult-zmult1*: $(c\$*a) \ zmod \ (c\$*b) = c \ \$* \ (a \ zmod \ b)$
⟨*proof*⟩

lemma *zmod-zmult-zmult2*: $(a\$*c) \ zmod \ (b\$*c) = (a \ zmod \ b) \ \$* \ c$
⟨*proof*⟩

lemma *zdiv-neg-pos-less0*: $[\![a \$< \#0;\ \ \#0 \$< b]\!] \Longrightarrow a\ zdiv\ b \$< \#0$
⟨*proof*⟩

lemma *zdiv-nonneg-neg-le0*: $[\![\#0 \$\leq a;\ \ b \$< \#0]\!] \Longrightarrow a\ zdiv\ b \$\leq \#0$
⟨*proof*⟩

lemma *pos-imp-zdiv-nonneg-iff*: $\#0 \$< b \Longrightarrow (\#0 \$\leq a\ zdiv\ b) \longleftrightarrow (\#0 \$\leq a)$
⟨*proof*⟩

lemma *neg-imp-zdiv-nonneg-iff*: $b \$< \#0 \Longrightarrow (\#0 \$\leq a\ zdiv\ b) \longleftrightarrow (a \$\leq \#0)$
⟨*proof*⟩

lemma *pos-imp-zdiv-neg-iff*: $\#0 \$< b \Longrightarrow (a\ zdiv\ b \$< \#0) \longleftrightarrow (a \$< \#0)$
⟨*proof*⟩

lemma *neg-imp-zdiv-neg-iff*: $b \$< \#0 \Longrightarrow (a\ zdiv\ b \$< \#0) \longleftrightarrow (\#0 \$< a)$
⟨*proof*⟩

end

34 Cardinal Arithmetic Without the Axiom of Choice

theory *CardinalArith* **imports** *Cardinal OrderArith ArithSimp Finite* **begin**

definition
InfCard :: $i \Rightarrow o$ **where**
InfCard(i) ≡ *Card*(i) ∧ *nat* ≤ i

definition
cmult :: $[i,i] \Rightarrow i$ (**infixl** ‹⊗› *70*) **where**
$i \otimes j \equiv |i*j|$

definition
cadd :: $[i,i] \Rightarrow i$ (**infixl** ‹⊕› *65*) **where**
$i \oplus j \equiv |i+j|$

definition
csquare-rel :: $i \Rightarrow i$ **where**
csquare-rel(K) ≡
rvimage($K*K$,
lam ⟨x,y⟩:$K*K$. <$x \cup y$, x, y>,
rmult(K,*Memrel*(K), $K*K$, *rmult*(K,*Memrel*(K), K,*Memrel*(K))))

definition
jump-cardinal :: $i \Rightarrow i$ **where**

— This definition is more complex than Kunen's but it more easily proved to be a cardinal
jump-cardinal$(K) \equiv$
$\bigcup X{\in}Pow(K).\ \{z.\ r \in Pow(K{*}K),\ well\text{-}ord(X,r) \wedge z = ordertype(X,r)\}$

definition
csucc :: $i{\Rightarrow}i$ **where**
— needed because *jump-cardinal*(K) might not be the successor of K
csucc$(K) \equiv \mu\ L.\ Card(L) \wedge K{<}L$

lemma *Card-Union* [*simp*,*intro*,*TC*]:
assumes A: $\bigwedge x.\ x{\in}A \Longrightarrow Card(x)$ **shows** $Card(\bigcup(A))$
⟨*proof*⟩

lemma *Card-UN*: $(\bigwedge x.\ x \in A \Longrightarrow Card(K(x))) \Longrightarrow Card(\bigcup x{\in}A.\ K(x))$
⟨*proof*⟩

lemma *Card-OUN* [*simp*,*intro*,*TC*]:
$(\bigwedge x.\ x \in A \Longrightarrow Card(K(x))) \Longrightarrow Card(\bigcup x{<}A.\ K(x))$
⟨*proof*⟩

lemma *in-Card-imp-lesspoll*: $[\![Card(K);\ b \in K]\!] \Longrightarrow b \prec K$
⟨*proof*⟩

34.1 Cardinal addition

Note: Could omit proving the algebraic laws for cardinal addition and multiplication. On finite cardinals these operations coincide with addition and multiplication of natural numbers; on infinite cardinals they coincide with union (maximum). Either way we get most laws for free.

34.1.1 Cardinal addition is commutative

lemma *sum-commute-eqpoll*: $A{+}B \approx B{+}A$
⟨*proof*⟩

lemma *cadd-commute*: $i \oplus j = j \oplus i$
⟨*proof*⟩

34.1.2 Cardinal addition is associative

lemma *sum-assoc-eqpoll*: $(A{+}B){+}C \approx A{+}(B{+}C)$
⟨*proof*⟩

Unconditional version requires AC

lemma *well-ord-cadd-assoc*:
assumes i: *well-ord*(i,ri) **and** j: *well-ord*(j,rj) **and** k: *well-ord*(k,rk)

shows $(i \oplus j) \oplus k = i \oplus (j \oplus k)$
⟨*proof*⟩

34.1.3 0 is the identity for addition

lemma *sum-0-eqpoll*: $0+A \approx A$
⟨*proof*⟩

lemma *cadd-0* [*simp*]: $Card(K) \Longrightarrow 0 \oplus K = K$
⟨*proof*⟩

34.1.4 Addition by another cardinal

lemma *sum-lepoll-self*: $A \lesssim A+B$
⟨*proof*⟩

lemma *cadd-le-self*:
assumes K: $Card(K)$ **and** L: $Ord(L)$ **shows** $K \le (K \oplus L)$
⟨*proof*⟩

34.1.5 Monotonicity of addition

lemma *sum-lepoll-mono*:
$[\![A \lesssim C;\ B \lesssim D]\!] \Longrightarrow A + B \lesssim C + D$
⟨*proof*⟩

lemma *cadd-le-mono*:
$[\![K' \le K;\ L' \le L]\!] \Longrightarrow (K' \oplus L') \le (K \oplus L)$
⟨*proof*⟩

34.1.6 Addition of finite cardinals is "ordinary" addition

lemma *sum-succ-eqpoll*: $succ(A)+B \approx succ(A+B)$
⟨*proof*⟩

lemma *cadd-succ-lemma*:
assumes $Ord(m)$ $Ord(n)$ **shows** $succ(m) \oplus n = |succ(m \oplus n)|$
⟨*proof*⟩

lemma *nat-cadd-eq-add*:
assumes m: $m \in nat$ **and** [*simp*]: $n \in nat$ **shows**$m \oplus n = m \#+ n$
⟨*proof*⟩

34.2 Cardinal multiplication

34.2.1 Cardinal multiplication is commutative

lemma *prod-commute-eqpoll*: $A*B \approx B*A$
⟨*proof*⟩

lemma *cmult-commute*: $i \otimes j = j \otimes i$
⟨*proof*⟩

34.2.2 Cardinal multiplication is associative

lemma *prod-assoc-eqpoll*: $(A*B)*C \approx A*(B*C)$
⟨*proof*⟩

Unconditional version requires AC

lemma *well-ord-cmult-assoc*:
assumes *i*: *well-ord*(i,ri) **and** *j*: *well-ord*(j,rj) **and** *k*: *well-ord*(k,rk)
shows $(i \otimes j) \otimes k = i \otimes (j \otimes k)$
⟨*proof*⟩

34.2.3 Cardinal multiplication distributes over addition

lemma *sum-prod-distrib-eqpoll*: $(A+B)*C \approx (A*C)+(B*C)$
⟨*proof*⟩

lemma *well-ord-cadd-cmult-distrib*:
assumes *i*: *well-ord*(i,ri) **and** *j*: *well-ord*(j,rj) **and** *k*: *well-ord*(k,rk)
shows $(i \oplus j) \otimes k = (i \otimes k) \oplus (j \otimes k)$
⟨*proof*⟩

34.2.4 Multiplication by 0 yields 0

lemma *prod-0-eqpoll*: $0*A \approx 0$
⟨*proof*⟩

lemma *cmult-0* [*simp*]: $0 \otimes i = 0$
⟨*proof*⟩

34.2.5 1 is the identity for multiplication

lemma *prod-singleton-eqpoll*: $\{x\}*A \approx A$
⟨*proof*⟩

lemma *cmult-1* [*simp*]: $Card(K) \Longrightarrow 1 \otimes K = K$
⟨*proof*⟩

34.3 Some inequalities for multiplication

lemma *prod-square-lepoll*: $A \lesssim A*A$
⟨*proof*⟩

lemma *cmult-square-le*: $Card(K) \Longrightarrow K \leq K \otimes K$
⟨*proof*⟩

34.3.1 Multiplication by a non-zero cardinal

lemma *prod-lepoll-self*: $b \in B \Longrightarrow A \lesssim A{*}B$
⟨*proof*⟩

lemma *cmult-le-self*:
$[\![Card(K);\ \ Ord(L);\ \ 0{<}L]\!] \Longrightarrow K \leq (K \otimes L)$
⟨*proof*⟩

34.3.2 Monotonicity of multiplication

lemma *prod-lepoll-mono*:
$[\![A \lesssim C;\ \ B \lesssim D]\!] \Longrightarrow A * B \ \lesssim\ C * D$
⟨*proof*⟩

lemma *cmult-le-mono*:
$[\![K' \leq K;\ \ L' \leq L]\!] \Longrightarrow (K' \otimes L') \leq (K \otimes L)$
⟨*proof*⟩

34.4 Multiplication of finite cardinals is "ordinary" multiplication

lemma *prod-succ-eqpoll*: $succ(A){*}B \approx B + A{*}B$
⟨*proof*⟩

lemma *cmult-succ-lemma*:
$[\![Ord(m);\ \ Ord(n)]\!] \Longrightarrow succ(m) \otimes n = n \oplus (m \otimes n)$
⟨*proof*⟩

lemma *nat-cmult-eq-mult*: $[\![m \in nat;\ \ n \in nat]\!] \Longrightarrow m \otimes n = m\#{*}n$
⟨*proof*⟩

lemma *cmult-2*: $Card(n) \Longrightarrow 2 \otimes n = n \oplus n$
⟨*proof*⟩

lemma *sum-lepoll-prod*:
assumes C: $2 \lesssim C$ **shows** $B{+}B \lesssim C{*}B$
⟨*proof*⟩

lemma *lepoll-imp-sum-lepoll-prod*: $[\![A \lesssim B;\ 2 \lesssim A]\!] \Longrightarrow A{+}B \lesssim A{*}B$
⟨*proof*⟩

34.5 Infinite Cardinals are Limit Ordinals

lemma *nat-cons-lepoll*: $nat \lesssim A \Longrightarrow cons(u,A) \lesssim A$
⟨*proof*⟩

lemma *nat-cons-eqpoll*: $nat \lesssim A \Longrightarrow cons(u,A) \approx A$
⟨*proof*⟩

lemma *nat-succ-eqpoll*: $nat \subseteq A \Longrightarrow succ(A) \approx A$
⟨*proof*⟩

lemma *InfCard-nat*: $InfCard(nat)$
⟨*proof*⟩

lemma *InfCard-is-Card*: $InfCard(K) \Longrightarrow Card(K)$
⟨*proof*⟩

lemma *InfCard-Un*:
$[\![InfCard(K);\ \ Card(L)]\!] \Longrightarrow InfCard(K \cup L)$
⟨*proof*⟩

lemma *InfCard-is-Limit*: $InfCard(K) \Longrightarrow Limit(K)$
⟨*proof*⟩

lemma *ordermap-eqpoll-pred*:
$[\![well\text{-}ord(A,r);\ \ x \in A]\!] \Longrightarrow ordermap(A,r)`x \approx Order.pred(A,x,r)$
⟨*proof*⟩

34.5.1 Establishing the well-ordering

lemma *well-ord-csquare*:
assumes K: $Ord(K)$ **shows** $well\text{-}ord(K{*}K,\ csquare\text{-}rel(K))$
⟨*proof*⟩

34.5.2 Characterising initial segments of the well-ordering

lemma *csquareD*:
$[\![<\langle x,y\rangle,\ \langle z,z\rangle> \in csquare\text{-}rel(K);\ \ x<K;\ \ y<K;\ \ z<K]\!] \Longrightarrow x \le z \wedge y \le z$
⟨*proof*⟩

lemma *pred-csquare-subset*:
$z<K \Longrightarrow Order.pred(K{*}K,\ \langle z,z\rangle,\ csquare\text{-}rel(K)) \subseteq succ(z){*}succ(z)$
⟨*proof*⟩

lemma *csquare-ltI*:
$[\![x<z;\ \ y<z;\ \ z<K]\!] \Longrightarrow\ <\langle x,y\rangle,\ \langle z,z\rangle> \in \mathit{csquare\text{-}rel}(K)$
⟨*proof*⟩

lemma *csquare-or-eqI*:
$[\![x \le z;\ \ y \le z;\ \ z<K]\!] \Longrightarrow <\langle x,y\rangle,\ \langle z,z\rangle> \in \mathit{csquare\text{-}rel}(K)\ |\ x=z \wedge y=z$
⟨*proof*⟩

34.5.3 The cardinality of initial segments

lemma *ordermap-z-lt*:
$[\![\mathit{Limit}(K);\ \ x<K;\ \ y<K;\ \ z=\mathit{succ}(x \cup y)]\!] \Longrightarrow$
$\mathit{ordermap}(K*K,\ \mathit{csquare\text{-}rel}(K))\ `\ \langle x,y\rangle <$
$\mathit{ordermap}(K*K,\ \mathit{csquare\text{-}rel}(K))\ `\ \langle z,z\rangle$
⟨*proof*⟩

Kunen: "each $\langle x,\ y\rangle \in K \times K$ has no more than $z \times z$ predecessors..." (page 29)

lemma *ordermap-csquare-le*:
assumes K: $\mathit{Limit}(K)$ **and** x: $x<K$ **and** y: $y<K$
defines $z \equiv \mathit{succ}(x \cup y)$
shows $|\mathit{ordermap}(K \times K,\ \mathit{csquare\text{-}rel}(K))\ `\ \langle x,y\rangle| \le |\mathit{succ}(z)| \otimes |\mathit{succ}(z)|$
⟨*proof*⟩

Kunen: "... so the order type is ≤ K"

lemma *ordertype-csquare-le*:
assumes IK: $\mathit{InfCard}(K)$ **and** eq: $\bigwedge y.\ y \in K \Longrightarrow \mathit{InfCard}(y) \Longrightarrow y \otimes y = y$
shows $\mathit{ordertype}(K*K,\ \mathit{csquare\text{-}rel}(K)) \le K$
⟨*proof*⟩

lemma *InfCard-csquare-eq*:
assumes IK: $\mathit{InfCard}(K)$ **shows** $K \otimes K = K$
⟨*proof*⟩

lemma *well-ord-InfCard-square-eq*:
assumes r: $\mathit{well\text{-}ord}(A,r)$ **and** I: $\mathit{InfCard}(|A|)$ **shows** $A \times A \approx A$
⟨*proof*⟩

lemma *InfCard-square-eqpoll*: $\mathit{InfCard}(K) \Longrightarrow K \times K \approx K$
⟨*proof*⟩

lemma *Inf-Card-is-InfCard*: $[\![\mathit{Card}(i);\ \neg\ \mathit{Finite}(i)]\!] \Longrightarrow \mathit{InfCard}(i)$
⟨*proof*⟩

34.5.4 Toward's Kunen's Corollary 10.13 (1)

lemma *InfCard-le-cmult-eq*: $[\![\mathit{InfCard}(K);\ \ L \le K;\ \ 0<L]\!] \Longrightarrow K \otimes L = K$

⟨*proof*⟩

lemma *InfCard-cmult-eq*: $[\![InfCard(K); \ \ InfCard(L)]\!] \Longrightarrow K \otimes L = K \cup L$
⟨*proof*⟩

lemma *InfCard-cdouble-eq*: $InfCard(K) \Longrightarrow K \oplus K = K$
⟨*proof*⟩

lemma *InfCard-le-cadd-eq*: $[\![InfCard(K); \ \ L \le K]\!] \Longrightarrow K \oplus L = K$
⟨*proof*⟩

lemma *InfCard-cadd-eq*: $[\![InfCard(K); \ \ InfCard(L)]\!] \Longrightarrow K \oplus L = K \cup L$
⟨*proof*⟩

34.6 For Every Cardinal Number There Exists A Greater One

This result is Kunen's Theorem 10.16, which would be trivial using AC

lemma *Ord-jump-cardinal*: $Ord(jump\text{-}cardinal(K))$
⟨*proof*⟩

lemma *jump-cardinal-iff*:
$i \in jump\text{-}cardinal(K) \longleftrightarrow$
$(\exists\, r\ X.\ r \subseteq K{*}K \wedge X \subseteq K \wedge well\text{-}ord(X,r) \wedge i = ordertype(X,r))$
⟨*proof*⟩

lemma *K-lt-jump-cardinal*: $Ord(K) \Longrightarrow K < jump\text{-}cardinal(K)$
⟨*proof*⟩

lemma *Card-jump-cardinal-lemma*:
$[\![well\text{-}ord(X,r); \ \ r \subseteq K * K; \ \ X \subseteq K;$
$f \in bij(ordertype(X,r),\ jump\text{-}cardinal(K))]\!]$
$\Longrightarrow jump\text{-}cardinal(K) \in jump\text{-}cardinal(K)$
⟨*proof*⟩

lemma *Card-jump-cardinal*: $Card(jump\text{-}cardinal(K))$
⟨*proof*⟩

34.7 Basic Properties of Successor Cardinals

lemma *csucc-basic*: $Ord(K) \Longrightarrow Card(csucc(K)) \wedge K < csucc(K)$
⟨*proof*⟩

lemmas *Card-csucc* = *csucc-basic* [*THEN conjunct1*]

lemmas *lt-csucc* = *csucc-basic* [*THEN conjunct2*]

lemma *Ord-0-lt-csucc*: $Ord(K) \Longrightarrow 0 < csucc(K)$
⟨*proof*⟩

lemma *csucc-le*: $[\![Card(L);\ K<L]\!] \Longrightarrow csucc(K) \leq L$
⟨*proof*⟩

lemma *lt-csucc-iff*: $[\![Ord(i);\ Card(K)]\!] \Longrightarrow i < csucc(K) \longleftrightarrow |i| \leq K$
⟨*proof*⟩

lemma *Card-lt-csucc-iff*:
$[\![Card(K');\ Card(K)]\!] \Longrightarrow K' < csucc(K) \longleftrightarrow K' \leq K$
⟨*proof*⟩

lemma *InfCard-csucc*: $InfCard(K) \Longrightarrow InfCard(csucc(K))$
⟨*proof*⟩

34.7.1 Removing elements from a finite set decreases its cardinality

lemma *Finite-imp-cardinal-cons* [*simp*]:
assumes *FA*: $Finite(A)$ **and** *a*: $a \notin A$ **shows** $|cons(a,A)| = succ(|A|)$
⟨*proof*⟩

lemma *Finite-imp-succ-cardinal-Diff*:
$[\![Finite(A);\ a \in A]\!] \Longrightarrow succ(|A-\{a\}|) = |A|$
⟨*proof*⟩

lemma *Finite-imp-cardinal-Diff*: $[\![Finite(A);\ a \in A]\!] \Longrightarrow |A-\{a\}| < |A|$
⟨*proof*⟩

lemma *Finite-cardinal-in-nat* [*simp*]: $Finite(A) \Longrightarrow |A| \in nat$
⟨*proof*⟩

lemma *card-Un-Int*:
$[\![Finite(A);\ Finite(B)]\!] \Longrightarrow |A| \ \#+\ |B| = |A \cup B| \ \#+\ |A \cap B|$
⟨*proof*⟩

lemma *card-Un-disjoint*:
$[\![Finite(A);\ Finite(B);\ A \cap B = 0]\!] \Longrightarrow |A \cup B| = |A| \ \#+\ |B|$
⟨*proof*⟩

lemma *card-partition*:
assumes *FC*: $Finite(C)$
shows
$Finite\ (\bigcup\ C) \Longrightarrow$

$(\forall\, c{\in}C.\ |c| = k) \Longrightarrow$
$(\forall\, c1 \in C.\ \forall\, c2 \in C.\ c1 \neq c2 \longrightarrow c1 \cap c2 = 0) \Longrightarrow$
$k \mathbin{\#*} |C| = |\bigcup\ C|$
⟨*proof*⟩

34.7.2 Theorems by Krzysztof Grabczewski, proofs by lcp

lemmas *nat-implies-well-ord* = *nat-into-Ord* [*THEN well-ord-Memrel*]

lemma *nat-sum-eqpoll-sum*:
assumes *m*: $m \in nat$ **and** *n*: $n \in nat$ **shows** $m + n \approx m \mathbin{\#+} n$
⟨*proof*⟩

lemma *Ord-subset-natD* [*rule-format*]: $Ord(i) \Longrightarrow i \subseteq nat \Longrightarrow i \in nat \mid i{=}nat$
⟨*proof*⟩

lemma *Ord-nat-subset-into-Card*: $[\![Ord(i);\ i \subseteq nat]\!] \Longrightarrow Card(i)$
⟨*proof*⟩

end

35 Main ZF Theory: Everything Except AC

theory *ZF* **imports** *List IntDiv CardinalArith* **begin**

35.1 Iteration of the function *F*

consts *iterates* :: [*i*⇒*i*,*i*,*i*] ⇒ *i* (‹(‹notation=‹mixfix iterates››-^- ′(-′))› [*60*,*1000*,*1000*] *60*)

primrec
$F\hat{}0\ (x) = x$
$F\hat{}(succ(n))\ (x) = F(F\hat{}n\ (x))$

definition
iterates-omega :: [*i*⇒*i*,*i*] ⇒ *i* (‹(‹notation=‹mixfix iterates-omega››-^ω ′(-′))› [*60*,*1000*] *60*) **where**
$F\hat{}\omega\ (x) \equiv \bigcup n{\in}nat.\ F\hat{}n\ (x)$

lemma *iterates-triv*:
$[\![n{\in}nat;\ \ F(x) = x]\!] \Longrightarrow F\hat{}n\ (x) = x$
⟨*proof*⟩

lemma *iterates-type* [*TC*]:
$[\![n \in nat;\ \ a \in A;\ \bigwedge x.\ x \in A \Longrightarrow F(x) \in A]\!]$
$\Longrightarrow F\hat{}n\ (a) \in A$
⟨*proof*⟩

lemma *iterates-omega-triv*:

$F(x) = x \Longrightarrow F\hat{}\omega\ (x) = x$
⟨*proof*⟩

lemma *Ord-iterates* [*simp*]:
$[\![n{\in}nat;\ \bigwedge i.\ Ord(i) \Longrightarrow Ord(F(i));\ \ Ord(x)]\!]$
$\Longrightarrow Ord(F\hat{}n\ (x))$
⟨*proof*⟩

lemma *iterates-commute*: $n \in nat \Longrightarrow F(F\hat{}n\ (x)) = F\hat{}n\ (F(x))$
⟨*proof*⟩

35.2 Transfinite Recursion

Transfinite recursion for definitions based on the three cases of ordinals

definition
transrec3 :: $[i,\ i,\ [i,i]{\Rightarrow}i,\ [i,i]{\Rightarrow}i] \Rightarrow i$ **where**
$transrec3(k,\ a,\ b,\ c) \equiv$
$transrec(k,\ \lambda x\ r.$
if $x{=}0$ *then* a
else if $Limit(x)$ *then* $c(x,\ \lambda y{\in}x.\ r`y)$
else $b(Arith.pred(x),\ r\ `\ Arith.pred(x)))$

lemma *transrec3-0* [*simp*]: $transrec3(0,a,b,c) = a$
⟨*proof*⟩

lemma *transrec3-succ* [*simp*]:
$transrec3(succ(i),a,b,c) = b(i,\ transrec3(i,a,b,c))$
⟨*proof*⟩

lemma *transrec3-Limit*:
$Limit(i) \Longrightarrow$
$transrec3(i,a,b,c) = c(i,\ \lambda j{\in}i.\ transrec3(j,a,b,c))$
⟨*proof*⟩

⟨*ML*⟩

end

36 The Axiom of Choice

theory *AC* **imports** *ZF* **begin**

This definition comes from Halmos (1960), page 59.

axiomatization where
AC: $[\![a \in A;\ \bigwedge x.\ x \in A \Longrightarrow (\exists y.\ y \in B(x))]\!] \Longrightarrow \exists z.\ z \in Pi(A,B)$

lemma *AC-Pi*: $[\![\bigwedge x.\ x \in A \Longrightarrow (\exists\, y.\ y \in B(x))]\!] \Longrightarrow \exists\, z.\ z \in Pi(A,B)$
⟨*proof*⟩

lemma *AC-ball-Pi*: $\forall\, x \in A.\ \exists\, y.\ y \in B(x) \Longrightarrow \exists\, y.\ y \in Pi(A,B)$
⟨*proof*⟩

lemma *AC-Pi-Pow*: $\exists f.\ f \in (\prod X \in Pow(C){-}\{0\}.\ X)$
⟨*proof*⟩

lemma *AC-func*:
$[\![\bigwedge x.\ x \in A \Longrightarrow (\exists\, y.\ y \in x)]\!] \Longrightarrow \exists f \in A{-}{>}\bigcup(A).\ \forall\, x \in A.\ f`x \in x$
⟨*proof*⟩

lemma *non-empty-family*: $[\![0 \notin A;\ \ x \in A]\!] \Longrightarrow \exists\, y.\ y \in x$
⟨*proof*⟩

lemma *AC-func0*: $0 \notin A \Longrightarrow \exists f \in A{-}{>}\bigcup(A).\ \forall\, x \in A.\ f`x \in x$
⟨*proof*⟩

lemma *AC-func-Pow*: $\exists f \in (Pow(C){-}\{0\})\ {-}{>}\ C.\ \forall\, x \in Pow(C){-}\{0\}.\ f`x \in x$
⟨*proof*⟩

lemma *AC-Pi0*: $0 \notin A \Longrightarrow \exists f.\ f \in (\prod x \in A.\ x)$
⟨*proof*⟩

end

37 Zorn's Lemma

theory *Zorn* **imports** *OrderArith AC Inductive* **begin**

Based upon the unpublished article "Towards the Mechanization of the Proofs of Some Classical Theorems of Set Theory," by Abrial and Laffitte.

definition
Subset-rel :: $i \Rightarrow i$ **where**
$Subset\text{-}rel(A) \equiv \{z \in A*A\ .\ \exists\, x\ y.\ z{=}\langle x,y\rangle \wedge x{<}{=}y \wedge x{\neq}y\}$

definition
chain :: $i \Rightarrow i$ **where**
$chain(A) \equiv \{F \in Pow(A).\ \forall\, X{\in}F.\ \forall\, Y{\in}F.\ X{<}{=}Y \mid Y{<}{=}X\}$

definition
super :: $[i,i] \Rightarrow i$ **where**
$super(A,c) \equiv \{d \in chain(A).\ c{<}{=}d \wedge c{\neq}d\}$

definition
maxchain :: $i \Rightarrow i$ **where**
$maxchain(A) \equiv \{c \in chain(A).\ super(A,c){=}0\}$

definition
increasing :: $i \Rightarrow i$ **where**
$increasing(A) \equiv \{f \in Pow(A){-}{>}Pow(A).\ \forall x.\ x{<}{=}A \longrightarrow x{<}{=}f`x\}$

Lemma for the inductive definition below

lemma *Union-in-Pow*: $Y \in Pow(Pow(A)) \Longrightarrow \bigcup(Y) \in Pow(A)$
⟨*proof*⟩

We could make the inductive definition conditional on $next \in increasing(S)$ but instead we make this a side-condition of an introduction rule. Thus the induction rule lets us assume that condition! Many inductive proofs are therefore unconditional.

consts
TFin :: $[i,i] \Rightarrow i$

inductive
domains $TFin(S,next) \subseteq Pow(S)$
intros
nextI: $[\![x \in TFin(S,next);\ \ next \in increasing(S)]\!]$
$\Longrightarrow next`x \in TFin(S,next)$

Pow-UnionI: $Y \in Pow(TFin(S,next)) \Longrightarrow \bigcup(Y) \in TFin(S,next)$

monos *Pow-mono*
con-defs *increasing-def*
type-intros *CollectD1* [*THEN apply-funtype*] *Union-in-Pow*

37.1 Mathematical Preamble

lemma *Union-lemma0*: $(\forall x{\in}C.\ x{<}{=}A \mid B{<}{=}x) \Longrightarrow \bigcup(C){<}{=}A \mid B{<}{=}\bigcup(C)$
⟨*proof*⟩

lemma *Inter-lemma0*:
$[\![c \in C;\ \forall x{\in}C.\ A{<}{=}x \mid x{<}{=}B]\!] \Longrightarrow A \subseteq \bigcap(C) \mid \bigcap(C) \subseteq B$
⟨*proof*⟩

37.2 The Transfinite Construction

lemma *increasingD1*: $f \in increasing(A) \Longrightarrow f \in Pow(A){-}{>}Pow(A)$
⟨*proof*⟩

lemma *increasingD2*: $[\![f \in increasing(A);\ x{<}{=}A]\!] \Longrightarrow x \subseteq f`x$
⟨*proof*⟩

lemmas *TFin-UnionI* = *PowI* [*THEN TFin.Pow-UnionI*]

lemmas *TFin-is-subset* = *TFin.dom-subset* [*THEN subsetD*, *THEN PowD*]

Structural induction on $TFin(S, next)$

lemma *TFin-induct*:
$[\![n \in TFin(S,next);$
$\bigwedge x.\ [\![x \in TFin(S,next);\ P(x);\ next \in increasing(S)]\!] \Longrightarrow P(next`x);$
$\bigwedge Y.\ [\![Y \subseteq TFin(S,next);\ \forall y\in Y.\ P(y)]\!] \Longrightarrow P(\bigcup(Y))$
$]\!] \Longrightarrow P(n)$
⟨*proof*⟩

37.3 Some Properties of the Transfinite Construction

lemmas *increasing-trans* = *subset-trans* [*OF* - *increasingD2*,
OF - - *TFin-is-subset*]

Lemma 1 of section 3.1

lemma *TFin-linear-lemma1*:
$[\![n \in TFin(S,next);\ m \in TFin(S,next);$
$\forall x \in TFin(S,next)\ .\ x<=m \longrightarrow x=m\ |\ next`x<=m]\!]$
$\Longrightarrow n<=m\ |\ next`m<=n$
⟨*proof*⟩

Lemma 2 of section 3.2. Interesting in its own right! Requires $next \in increasing(S)$ in the second induction step.

lemma *TFin-linear-lemma2*:
$[\![m \in TFin(S,next);\ next \in increasing(S)]\!]$
$\Longrightarrow \forall n \in TFin(S,next).\ n<=m \longrightarrow n=m\ |\ next`n \subseteq m$
⟨*proof*⟩

a more convenient form for Lemma 2

lemma *TFin-subsetD*:
$[\![n<=m;\ m \in TFin(S,next);\ n \in TFin(S,next);\ next \in increasing(S)]\!]$
$\Longrightarrow n=m\ |\ next`n \subseteq m$
⟨*proof*⟩

Consequences from section 3.3 – Property 3.2, the ordering is total

lemma *TFin-subset-linear*:
$[\![m \in TFin(S,next);\ n \in TFin(S,next);\ next \in increasing(S)]\!]$
$\Longrightarrow n \subseteq m\ |\ m<=n$
⟨*proof*⟩

Lemma 3 of section 3.3

lemma *equal-next-upper*:
$[\![n \in TFin(S,next);\ m \in TFin(S,next);\ m = next`m]\!] \Longrightarrow n \subseteq m$
⟨*proof*⟩

Property 3.3 of section 3.3

lemma *equal-next-Union*:
$[\![m \in TFin(S,next);\ next \in increasing(S)]\!]$
$\Longrightarrow m = next`m <-> m = \bigcup(TFin(S,next))$
⟨*proof*⟩

37.4 Hausdorff's Theorem: Every Set Contains a Maximal Chain

NOTE: We assume the partial ordering is $\subseteq$, the subset relation!

* Defining the "next" operation for Hausdorff's Theorem *

lemma *chain-subset-Pow*: $chain(A) \subseteq Pow(A)$
$\langle proof \rangle$

lemma *super-subset-chain*: $super(A,c) \subseteq chain(A)$
$\langle proof \rangle$

lemma *maxchain-subset-chain*: $maxchain(A) \subseteq chain(A)$
$\langle proof \rangle$

lemma *choice-super*:
$[\![ch \in (\prod X \in Pow(chain(S)) - \{0\}.\ X);\ X \in chain(S);\ \ X \notin maxchain(S)]\!]$
$\Longrightarrow ch\ `\ super(S,X) \in super(S,X)$
$\langle proof \rangle$

lemma *choice-not-equals*:
$[\![ch \in (\prod X \in Pow(chain(S)) - \{0\}.\ X);\ X \in chain(S);\ \ X \notin maxchain(S)]\!]$
$\Longrightarrow ch\ `\ super(S,X) \neq X$
$\langle proof \rangle$

This justifies Definition 4.4

lemma *Hausdorff-next-exists*:
$ch \in (\prod X \in Pow(chain(S)){-}\{0\}.\ X) \Longrightarrow$
$\exists\, next \in increasing(S).\ \forall X \in Pow(S).$
$next`X = if(X \in chain(S){-}maxchain(S),\ ch`super(S,X),\ X)$
$\langle proof \rangle$

Lemma 4

lemma *TFin-chain-lemma4*:
$[\![c \in TFin(S,next);$
$ch \in (\prod X \in Pow(chain(S)){-}\{0\}.\ X);$
$next \in increasing(S);$
$\forall X \in Pow(S).\ next`X =$
$if(X \in chain(S){-}maxchain(S),\ ch`super(S,X),\ X)]\!]$
$\Longrightarrow c \in chain(S)$
$\langle proof \rangle$

theorem *Hausdorff*: $\exists\, c.\ c \in maxchain(S)$
$\langle proof \rangle$

37.5 Zorn's Lemma: If All Chains in S Have Upper Bounds In S, then S contains a Maximal Element

Used in the proof of Zorn's Lemma

lemma *chain-extend*:
$[\![c \in chain(A);\ \ z \in A;\ \ \forall x \in c.\ x{<=}z]\!] \Longrightarrow cons(z,c) \in chain(A)$
$\langle proof \rangle$

lemma *Zorn*: $\forall c \in chain(S).\ \bigcup(c) \in S \Longrightarrow \exists y \in S.\ \forall z \in S.\ y{<=}z \longrightarrow y{=}z$
$\langle proof \rangle$

Alternative version of Zorn's Lemma

theorem *Zorn2*:
$\forall c \in chain(S).\ \exists y \in S.\ \forall x \in c.\ x \subseteq y \Longrightarrow \exists y \in S.\ \forall z \in S.\ y{<=}z \longrightarrow y{=}z$
$\langle proof \rangle$

37.6 Zermelo's Theorem: Every Set can be Well-Ordered

Lemma 5

lemma *TFin-well-lemma5*:
$[\![n \in TFin(S,next);\ \ Z \subseteq TFin(S,next);\ \ z{:}Z;\ \ \neg \bigcap(Z) \in Z]\!]$
$\Longrightarrow \forall m \in Z.\ n \subseteq m$
$\langle proof \rangle$

Well-ordering of $TFin(S,\ next)$

lemma *well-ord-TFin-lemma*: $[\![Z \subseteq TFin(S,next);\ \ z \in Z]\!] \Longrightarrow \bigcap(Z) \in Z$
$\langle proof \rangle$

This theorem just packages the previous result

lemma *well-ord-TFin*:
$next \in increasing(S)$
$\Longrightarrow well\text{-}ord(TFin(S,next),\ Subset\text{-}rel(TFin(S,next)))$
$\langle proof \rangle$

* Defining the "next" operation for Zermelo's Theorem *

lemma *choice-Diff*:
$[\![ch \in (\prod X \in Pow(S) - \{0\}.\ X);\ \ X \subseteq S;\ \ X{\neq}S]\!] \Longrightarrow ch\ `\ (S{-}X) \in S{-}X$
$\langle proof \rangle$

This justifies Definition 6.1

lemma *Zermelo-next-exists*:
$ch \in (\prod X \in Pow(S){-}\{0\}.\ X) \Longrightarrow$
$\exists next \in increasing(S).\ \forall X \in Pow(S).$
$next`X = (if\ X{=}S\ then\ S\ else\ cons(ch`(S{-}X),\ X))$
$\langle proof \rangle$

The construction of the injection

lemma *choice-imp-injection*:
$[\![ch \in (\prod X \in Pow(S){-}\{0\}.\ X);$
$next \in increasing(S);$
$\forall X \in Pow(S).\ next`X = if(X{=}S,\ S,\ cons(ch`(S{-}X),\ X))]\!]$

$\Longrightarrow (\lambda\ x \in S.\ \bigcup(\{y \in TFin(S,next).\ x \notin y\}))$
$\in inj(S,\ TFin(S,next) - \{S\})$
⟨*proof*⟩

The wellordering theorem

theorem *AC-well-ord*: $\exists\, r.\ well\text{-}ord(S,r)$
⟨*proof*⟩

37.7 Zorn's Lemma for Partial Orders

Reimported from HOL by Clemens Ballarin.

definition *Chain* :: $i \Rightarrow i$ **where**
$Chain(r) = \{A \in Pow(field(r)).\ \forall\, a{\in}A.\ \forall\, b{\in}A.\ \langle a,\ b\rangle \in r \mid \langle b,\ a\rangle \in r\}$

lemma *mono-Chain*:
$r \subseteq s \Longrightarrow Chain(r) \subseteq Chain(s)$
⟨*proof*⟩

theorem *Zorn-po*:
assumes *po*: $Partial\text{-}order(r)$
and *u*: $\forall\, C{\in}Chain(r).\ \exists\, u{\in}field(r).\ \forall\, a{\in}C.\ \langle a,\ u\rangle \in r$
shows $\exists\, m{\in}field(r).\ \forall\, a{\in}field(r).\ \langle m,\ a\rangle \in r \longrightarrow a = m$
⟨*proof*⟩

end

38 Cardinal Arithmetic Using AC

theory *Cardinal-AC* **imports** *CardinalArith Zorn* **begin**

38.1 Strengthened Forms of Existing Theorems on Cardinals

lemma *cardinal-eqpoll*: $|A| \approx A$
⟨*proof*⟩

The theorem $||A|| = |A|$

lemmas *cardinal-idem* = *cardinal-eqpoll* [*THEN cardinal-cong*, *simp*]

lemma *cardinal-eqE*: $|X| = |Y| \Longrightarrow X \approx Y$
⟨*proof*⟩

lemma *cardinal-eqpoll-iff*: $|X| = |Y| \longleftrightarrow X \approx Y$
⟨*proof*⟩

lemma *cardinal-disjoint-Un*:
$[\![|A|{=}|B|;\ \ |C|{=}|D|;\ \ A \cap C = 0;\ \ B \cap D = 0]\!]$
$\Longrightarrow |A \cup C| = |B \cup D|$
⟨*proof*⟩

lemma *lepoll-imp-cardinal-le*: $A \lesssim B \Longrightarrow |A| \le |B|$
⟨*proof*⟩

lemma *cadd-assoc*: $(i \oplus j) \oplus k = i \oplus (j \oplus k)$
⟨*proof*⟩

lemma *cmult-assoc*: $(i \otimes j) \otimes k = i \otimes (j \otimes k)$
⟨*proof*⟩

lemma *cadd-cmult-distrib*: $(i \oplus j) \otimes k = (i \otimes k) \oplus (j \otimes k)$
⟨*proof*⟩

lemma *InfCard-square-eq*: $InfCard(|A|) \Longrightarrow A*A \approx A$
⟨*proof*⟩

38.2 The relationship between cardinality and le-pollence

lemma *Card-le-imp-lepoll*:
assumes $|A| \le |B|$ **shows** $A \lesssim B$
⟨*proof*⟩

lemma *le-Card-iff*: $Card(K) \Longrightarrow |A| \le K \longleftrightarrow A \lesssim K$
⟨*proof*⟩

lemma *cardinal-0-iff-0* [*simp*]: $|A| = 0 \longleftrightarrow A = 0$
⟨*proof*⟩

lemma *cardinal-lt-iff-lesspoll*:
assumes *i*: $Ord(i)$ **shows** $i < |A| \longleftrightarrow i \prec A$
⟨*proof*⟩

lemma *cardinal-le-imp-lepoll*: $i \le |A| \Longrightarrow i \lesssim A$
⟨*proof*⟩

38.3 Other Applications of AC

lemma *surj-implies-inj*:
assumes *f*: $f \in surj(X,Y)$ **shows** $\exists g.\ g \in inj(Y,X)$
⟨*proof*⟩

Kunen's Lemma 10.20

lemma *surj-implies-cardinal-le*:
assumes *f*: $f \in surj(X,Y)$ **shows** $|Y| \le |X|$
⟨*proof*⟩

Kunen's Lemma 10.21

lemma *cardinal-UN-le*:
assumes *K*: $InfCard(K)$

shows ($\bigwedge i.\ i{\in}K \Longrightarrow |X(i)| \leq K$) $\Longrightarrow |\bigcup i{\in}K.\ X(i)| \leq K$
⟨*proof*⟩

The same again, using *csucc*

lemma *cardinal-UN-lt-csucc*:
$[\![InfCard(K);\ \bigwedge i.\ i{\in}K \Longrightarrow |X(i)| < csucc(K)]\!]$
$\Longrightarrow |\bigcup i{\in}K.\ X(i)| < csucc(K)$
⟨*proof*⟩

The same again, for a union of ordinals. In use, j(i) is a bit like rank(i), the least ordinal j such that i:Vfrom(A,j).

lemma *cardinal-UN-Ord-lt-csucc*:
$[\![InfCard(K);\ \bigwedge i.\ i{\in}K \Longrightarrow j(i) < csucc(K)]\!]$
$\Longrightarrow (\bigcup i{\in}K.\ j(i)) < csucc(K)$
⟨*proof*⟩

38.4 The Main Result for Infinite-Branching Datatypes

As above, but the index set need not be a cardinal. Work backwards along the injection from W into K, given that $W \neq 0$.

lemma *inj-UN-subset*:
assumes *f*: $f \in inj(A,B)$ **and** *a*: $a \in A$
shows $(\bigcup x{\in}A.\ C(x)) \subseteq (\bigcup y{\in}B.\ C(if\ y \in range(f)\ then\ converse(f)`y\ else\ a))$
⟨*proof*⟩

theorem *le-UN-Ord-lt-csucc*:
assumes *IK*: $InfCard(K)$ **and** *WK*: $|W| \leq K$ **and** *j*: $\bigwedge w.\ w{\in}W \Longrightarrow j(w) < csucc(K)$
shows $(\bigcup w{\in}W.\ j(w)) < csucc(K)$
⟨*proof*⟩

end

39 Infinite-Branching Datatype Definitions

theory *InfDatatype* **imports** *Datatype Univ Finite Cardinal-AC* **begin**

lemmas *fun-Limit-VfromE* =
Limit-VfromE [*OF apply-funtype InfCard-csucc* [*THEN InfCard-is-Limit*]]

lemma *fun-Vcsucc-lemma*:
assumes *f*: $f \in D \rightarrow Vfrom(A,csucc(K))$ **and** *DK*: $|D| \leq K$ **and** *ICK*: $InfCard(K)$
shows $\exists j.\ f \in D \rightarrow Vfrom(A,j) \wedge j < csucc(K)$
⟨*proof*⟩

lemma *subset-Vcsucc*:

$[\![D \subseteq Vfrom(A,csucc(K));\ \ |D| \le K;\ \ InfCard(K)]\!]$
$\Longrightarrow \exists j.\ D \subseteq Vfrom(A,j) \wedge j < csucc(K)$
⟨*proof*⟩

lemma *fun-Vcsucc*:
$[\![|D| \le K;\ \ InfCard(K);\ \ D \subseteq Vfrom(A,csucc(K))]\!] \Longrightarrow$
$D \ {-}{>}\ Vfrom(A,csucc(K)) \subseteq Vfrom(A,csucc(K))$
⟨*proof*⟩

lemma *fun-in-Vcsucc*:
$[\![f\colon D \ {-}{>}\ Vfrom(A,\ csucc(K));\ \ |D| \le K;\ \ InfCard(K);$
$D \subseteq Vfrom(A,csucc(K))]\!]$
$\Longrightarrow f\colon Vfrom(A,csucc(K))$
⟨*proof*⟩

Remove ⊆ from the rule above

lemmas *fun-in-Vcsucc′* = *fun-in-Vcsucc* [*OF* - - - *subsetI*]

lemma *Card-fun-Vcsucc*:
$InfCard(K) \Longrightarrow K \ {-}{>}\ Vfrom(A,csucc(K)) \subseteq Vfrom(A,csucc(K))$
⟨*proof*⟩

lemma *Card-fun-in-Vcsucc*:
$[\![f\colon K \ {-}{>}\ Vfrom(A,\ csucc(K));\ \ InfCard(K)]\!] \Longrightarrow f\colon Vfrom(A,csucc(K))$
⟨*proof*⟩

lemma *Limit-csucc*: $InfCard(K) \Longrightarrow Limit(csucc(K))$
⟨*proof*⟩

lemmas *Pair-in-Vcsucc* = *Pair-in-VLimit* [*OF* - - *Limit-csucc*]
lemmas *Inl-in-Vcsucc* = *Inl-in-VLimit* [*OF* - *Limit-csucc*]
lemmas *Inr-in-Vcsucc* = *Inr-in-VLimit* [*OF* - *Limit-csucc*]
lemmas *zero-in-Vcsucc* = *Limit-csucc* [*THEN zero-in-VLimit*]
lemmas *nat-into-Vcsucc* = *nat-into-VLimit* [*OF* - *Limit-csucc*]

lemmas *InfCard-nat-Un-cardinal* = *InfCard-Un* [*OF InfCard-nat Card-cardinal*]

lemmas *le-nat-Un-cardinal* =
Un-upper2-le [*OF Ord-nat Card-cardinal* [*THEN Card-is-Ord*]]

lemmas *UN-upper-cardinal* = *UN-upper* [*THEN subset-imp-lepoll*, *THEN lepoll-imp-cardinal-le*]

lemmas *Data-Arg-intros* =
SigmaI InlI InrI
Pair-in-univ Inl-in-univ Inr-in-univ
zero-in-univ A-into-univ nat-into-univ UnCI

lemmas *inf-datatype-intros* =
InfCard-nat InfCard-nat-Un-cardinal
Pair-in-Vcsucc Inl-in-Vcsucc Inr-in-Vcsucc
zero-in-Vcsucc A-into-Vfrom nat-into-Vcsucc
Card-fun-in-Vcsucc fun-in-Vcsucc′ UN-I

end
theory *ZFC* **imports** *ZF InfDatatype*
begin

end