ZF

Steven Obua

November 22, 2007

theory *Helper*
imports *Main*
begin

lemma *theI2'* : $?!\ x.\ P\ x \Longrightarrow (!!\ x.\ P\ x \Longrightarrow Q\ x) \Longrightarrow Q\ (\mathit{THE}\ x.\ P\ x)$
⟨*proof*⟩

lemma *in-range-superfluous*: $(z \in \mathit{range}\ f\ \&\ z \in (f\ `\ x)) = (z \in f\ `\ x)$
⟨*proof*⟩

lemma *f-x-in-range-f*: $f\ x \in \mathit{range}\ f$
⟨*proof*⟩

lemma *comp-inj*: $\mathit{inj}\ f \Longrightarrow \mathit{inj}\ g \Longrightarrow \mathit{inj}\ (g\ o\ f)$
⟨*proof*⟩

lemma *comp-image-eq*: $(g\ o\ f)\ `\ x = g\ `\ f\ `\ x$
⟨*proof*⟩

end

theory *HOLZF*
imports *Helper*
begin

typedecl *ZF*

axiomatization
Empty :: *ZF* **and**
Elem :: $\mathit{ZF} \Rightarrow \mathit{ZF} \Rightarrow \mathit{bool}$ **and**
Sum :: $\mathit{ZF} \Rightarrow \mathit{ZF}$ **and**
Power :: $\mathit{ZF} \Rightarrow \mathit{ZF}$ **and**
Repl :: $\mathit{ZF} \Rightarrow (\mathit{ZF} \Rightarrow \mathit{ZF}) \Rightarrow \mathit{ZF}$ **and**
Inf :: *ZF*

constdefs
$Upair$:: $ZF \Rightarrow ZF \Rightarrow ZF$
$Upair\ a\ b == Repl\ (Power\ (Power\ Empty))\ (\%\ x.\ if\ x = Empty\ then\ a\ else\ b)$
$Singleton$:: $ZF \Rightarrow ZF$
$Singleton\ x == Upair\ x\ x$
$union$ :: $ZF \Rightarrow ZF \Rightarrow ZF$
$union\ A\ B == Sum\ (Upair\ A\ B)$
$SucNat$:: $ZF \Rightarrow ZF$
$SucNat\ x == union\ x\ (Singleton\ x)$
$subset$ :: $ZF \Rightarrow ZF \Rightarrow bool$
$subset\ A\ B == !\ x.\ Elem\ x\ A \longrightarrow Elem\ x\ B$

axioms
$Empty$: $Not\ (Elem\ x\ Empty)$
Ext: $(x = y) = (!\ z.\ Elem\ z\ x = Elem\ z\ y)$
Sum: $Elem\ z\ (Sum\ x) = (?\ y.\ Elem\ z\ y\ \&\ Elem\ y\ x)$
$Power$: $Elem\ y\ (Power\ x) = (subset\ y\ x)$
$Repl$: $Elem\ b\ (Repl\ A\ f) = (?\ a.\ Elem\ a\ A\ \&\ b = f\ a)$
$Regularity$: $A \neq Empty \longrightarrow (?\ x.\ Elem\ x\ A\ \&\ (!\ y.\ Elem\ y\ x \longrightarrow Not\ (Elem\ y$
$A)))$
$Infinity$: $Elem\ Empty\ Inf\ \&\ (!\ x.\ Elem\ x\ Inf \longrightarrow Elem\ (SucNat\ x)\ Inf)$

constdefs
Sep:: $ZF \Rightarrow (ZF \Rightarrow bool) \Rightarrow ZF$
$Sep\ A\ p == (if\ (!x.\ Elem\ x\ A \longrightarrow Not\ (p\ x))\ then\ Empty\ else$
$(let\ z = (\epsilon\ x.\ Elem\ x\ A\ \&\ p\ x)\ in$
$let\ f = \%\ x.\ (if\ p\ x\ then\ x\ else\ z)\ in\ Repl\ A\ f))$

thm *Power*[*unfolded subset-def*]

theorem *Sep*: $Elem\ b\ (Sep\ A\ p) = (Elem\ b\ A\ \&\ p\ b)$
⟨*proof*⟩

lemma *subset-empty*: $subset\ Empty\ A$
⟨*proof*⟩

theorem *Upair*: $Elem\ x\ (Upair\ a\ b) = (x = a \mid x = b)$
⟨*proof*⟩

lemma *Singleton*: $Elem\ x\ (Singleton\ y) = (x = y)$
⟨*proof*⟩

constdefs
$Opair$:: $ZF \Rightarrow ZF \Rightarrow ZF$
$Opair\ a\ b == Upair\ (Upair\ a\ a)\ (Upair\ a\ b)$

lemma *Upair-singleton*: $(Upair\ a\ a = Upair\ c\ d) = (a = c\ \&\ a = d)$
⟨*proof*⟩

lemma *Upair-fsteq*: $(Upair\ a\ b = Upair\ a\ c) = ((a = b\ \&\ a = c) \mid (b = c))$
  ⟨*proof*⟩

lemma *Upair-comm*: $Upair\ a\ b = Upair\ b\ a$
  ⟨*proof*⟩

theorem *Opair*: $(Opair\ a\ b = Opair\ c\ d) = (a = c\ \&\ b = d)$
  ⟨*proof*⟩

constdefs
  $Replacement :: ZF \Rightarrow (ZF \Rightarrow ZF\ option) \Rightarrow ZF$
  $Replacement\ A\ f == Repl\ (Sep\ A\ (\%\ a.\ f\ a \neq None))\ (the\ o\ f)$

theorem *Replacement*: $Elem\ y\ (Replacement\ A\ f) = (?\ x.\ Elem\ x\ A\ \&\ f\ x = Some\ y)$
  ⟨*proof*⟩

constdefs
  $Fst :: ZF \Rightarrow ZF$
  $Fst\ q == SOME\ x.\ ?\ y.\ q = Opair\ x\ y$
  $Snd :: ZF \Rightarrow ZF$
  $Snd\ q == SOME\ y.\ ?\ x.\ q = Opair\ x\ y$

theorem *Fst*: $Fst\ (Opair\ x\ y) = x$
  ⟨*proof*⟩

theorem *Snd*: $Snd\ (Opair\ x\ y) = y$
  ⟨*proof*⟩

constdefs
  $isOpair :: ZF \Rightarrow bool$
  $isOpair\ q == ?\ x\ y.\ q = Opair\ x\ y$

lemma *isOpair*: $isOpair\ (Opair\ x\ y) = True$
  ⟨*proof*⟩

lemma *FstSnd*: $isOpair\ x \Longrightarrow Opair\ (Fst\ x)\ (Snd\ x) = x$
  ⟨*proof*⟩

constdefs
  $CartProd :: ZF \Rightarrow ZF \Rightarrow ZF$
  $CartProd\ A\ B == Sum(Repl\ A\ (\%\ a.\ Repl\ B\ (\%\ b.\ Opair\ a\ b)))$

lemma *CartProd*: $Elem\ x\ (CartProd\ A\ B) = (?\ a\ b.\ Elem\ a\ A\ \&\ Elem\ b\ B\ \&\ x = (Opair\ a\ b))$
  ⟨*proof*⟩

constdefs
  $explode :: ZF \Rightarrow ZF\ set$

$explode\ z == \{\ x.\ Elem\ x\ z\ \}$

lemma *explode-Empty*: $(explode\ x = \{\}) = (x = Empty)$
⟨*proof*⟩

lemma *explode-Elem*: $(x \in explode\ X) = (Elem\ x\ X)$
⟨*proof*⟩

lemma *Elem-explode-in*: $[\![\ Elem\ a\ A;\ explode\ A \subseteq B]\!] \Longrightarrow a \in B$
⟨*proof*⟩

lemma *explode-CartProd-eq*: $explode\ (CartProd\ a\ b) = (\%\ (x,y).\ Opair\ x\ y)$ ‘ $((explode\ a) \times (explode\ b))$
⟨*proof*⟩

lemma *explode-Repl-eq*: $explode\ (Repl\ A\ f) = image\ f\ (explode\ A)$
⟨*proof*⟩

constdefs
$Domain :: ZF \Rightarrow ZF$
$Domain\ f == Replacement\ f\ (\%\ p.\ if\ isOpair\ p\ then\ Some\ (Fst\ p)\ else\ None)$
$Range :: ZF \Rightarrow ZF$
$Range\ f == Replacement\ f\ (\%\ p.\ if\ isOpair\ p\ then\ Some\ (Snd\ p)\ else\ None)$

theorem *Domain*: $Elem\ x\ (Domain\ f) = (?\ y.\ Elem\ (Opair\ x\ y)\ f)$
⟨*proof*⟩

theorem *Range*: $Elem\ y\ (Range\ f) = (?\ x.\ Elem\ (Opair\ x\ y)\ f)$
⟨*proof*⟩

theorem *union*: $Elem\ x\ (union\ A\ B) = (Elem\ x\ A\ |\ Elem\ x\ B)$
⟨*proof*⟩

constdefs
$Field :: ZF \Rightarrow ZF$
$Field\ A == union\ (Domain\ A)\ (Range\ A)$

constdefs
$app :: ZF \Rightarrow ZF => ZF$ (**infixl** ´ *90*) — function application
$f\ ´\ x == (THE\ y.\ Elem\ (Opair\ x\ y)\ f)$

constdefs
$isFun :: ZF \Rightarrow bool$
$isFun\ f == (!\ x\ y1\ y2.\ Elem\ (Opair\ x\ y1)\ f\ \&\ Elem\ (Opair\ x\ y2)\ f \longrightarrow y1 = y2)$

constdefs
$Lambda :: ZF \Rightarrow (ZF \Rightarrow ZF) \Rightarrow ZF$
$Lambda\ A\ f == Repl\ A\ (\%\ x.\ Opair\ x\ (f\ x))$

lemma *Lambda-app*: *Elem x A* $\Longrightarrow$ (*Lambda A f*)´*x* = *f x*
⟨*proof*⟩

lemma *isFun-Lambda*: *isFun* (*Lambda A f*)
⟨*proof*⟩

lemma *domain-Lambda*: *Domain* (*Lambda A f*) = *A*
⟨*proof*⟩

lemma *Lambda-ext*: (*Lambda s f* = *Lambda t g*) = (*s* = *t* & (! *x*. *Elem x s* $\longrightarrow$ *f x* = *g x*))
⟨*proof*⟩

constdefs
PFun :: *ZF* $\Rightarrow$ *ZF* $\Rightarrow$ *ZF*
PFun A B == *Sep* (*Power* (*CartProd A B*)) *isFun*
Fun :: *ZF* $\Rightarrow$ *ZF* $\Rightarrow$ *ZF*
Fun A B == *Sep* (*PFun A B*) (λ *f*. *Domain f* = *A*)

lemma *Fun-Range*: *Elem f* (*Fun U V*) $\Longrightarrow$ *subset* (*Range f*) *V*
⟨*proof*⟩

lemma *Elem-Elem-PFun*: *Elem F* (*PFun U V*) $\Longrightarrow$ *Elem p F* $\Longrightarrow$ *isOpair p* & *Elem* (*Fst p*) *U* & *Elem* (*Snd p*) *V*
⟨*proof*⟩

lemma *Fun-implies-PFun*[*simp*]: *Elem f* (*Fun U V*) $\Longrightarrow$ *Elem f* (*PFun U V*)
⟨*proof*⟩

lemma *Elem-Elem-Fun*: *Elem F* (*Fun U V*) $\Longrightarrow$ *Elem p F* $\Longrightarrow$ *isOpair p* & *Elem* (*Fst p*) *U* & *Elem* (*Snd p*) *V*
⟨*proof*⟩

lemma *PFun-inj*: *Elem F* (*PFun U V*) $\Longrightarrow$ *Elem x F* $\Longrightarrow$ *Elem y F* $\Longrightarrow$ *Fst x* = *Fst y* $\Longrightarrow$ *Snd x* = *Snd y*
⟨*proof*⟩

lemma *Fun-total*: $[\![$*Elem F* (*Fun U V*); *Elem a U*$]\!]$ $\Longrightarrow$ $\exists$ *x*. *Elem* (*Opair a x*) *F*
⟨*proof*⟩

lemma *unique-fun-value*: $[\![$*isFun f*; *Elem x* (*Domain f*)$]\!]$ $\Longrightarrow$?! *y*. *Elem* (*Opair x y*) *f*
⟨*proof*⟩

lemma *fun-value-in-range*: $[\![$*isFun f*; *Elem x* (*Domain f*)$]\!]$ $\Longrightarrow$ *Elem* (*f*´*x*) (*Range f*)
⟨*proof*⟩

lemma *fun-range-witness*: $[\![isFun\ f;\ Elem\ y\ (Range\ f)]\!] \Longrightarrow ?\ x.\ Elem\ x\ (Domain\ f)\ \&\ f\,\acute{}\,x = y$
⟨*proof*⟩

lemma *Elem-Fun-Lambda*: $Elem\ F\ (Fun\ U\ V) \Longrightarrow ?\ f.\ F = Lambda\ U\ f$
⟨*proof*⟩

lemma *Elem-Lambda-Fun*: $Elem\ (Lambda\ A\ f)\ (Fun\ U\ V) = (A = U\ \&\ (!\ x.\ Elem\ x\ A \longrightarrow Elem\ (f\ x)\ V))$
⟨*proof*⟩

constdefs
is-Elem-of :: $(ZF * ZF)$ *set*
is-Elem-of $==$ { (a,b) | $a\ b.\ Elem\ a\ b$ }

lemma *cond-wf-Elem*:
assumes *hyps*:$\forall x.\ (\forall y.\ Elem\ y\ x \longrightarrow Elem\ y\ U \longrightarrow P\ y) \longrightarrow Elem\ x\ U \longrightarrow P\ x\ Elem\ a\ U$
shows $P\ a$
⟨*proof*⟩
term P
term *Sep*
⟨*proof*⟩

lemma *cond2-wf-Elem*:
assumes
special-P: $?\ U.\ !\ x.\ Not(Elem\ x\ U) \longrightarrow (P\ x)$
and *P-induct*: $\forall x.\ (\forall y.\ Elem\ y\ x \longrightarrow P\ y) \longrightarrow P\ x$
shows
$P\ a$
⟨*proof*⟩

consts
nat2Nat :: *nat* ⇒ *ZF*

primrec
nat2Nat-0[*intro*]: *nat2Nat 0* = *Empty*
nat2Nat-Suc[*intro*]: *nat2Nat* (*Suc n*) = *SucNat* (*nat2Nat n*)

constdefs
Nat2nat :: *ZF* ⇒ *nat*
Nat2nat == *inv nat2Nat*

lemma *Elem-nat2Nat-inf*[*intro*]: *Elem* (*nat2Nat n*) *Inf*
⟨*proof*⟩

constdefs

Nat :: *ZF*
Nat == *Sep Inf* (λ *N*. *? n*. *nat2Nat n* = *N*)

lemma *Elem-nat2Nat-Nat*[*intro*]: *Elem* (*nat2Nat n*) *Nat*
⟨*proof*⟩

lemma *Elem-Empty-Nat*: *Elem Empty Nat*
⟨*proof*⟩

lemma *Elem-SucNat-Nat*: *Elem N Nat* ⟹ *Elem* (*SucNat N*) *Nat*
⟨*proof*⟩

lemma *no-infinite-Elem-down-chain*:
Not (*? f*. *isFun f* & *Domain f* = *Nat* & (! *N*. *Elem N Nat* ⟶ *Elem* (*f* ´(*SucNat N*)) (*f* ´*N*)))
⟨*proof*⟩

lemma *Upair-nonEmpty*: *Upair a b* ≠ *Empty*
⟨*proof*⟩

lemma *Singleton-nonEmpty*: *Singleton x* ≠ *Empty*
⟨*proof*⟩

lemma *antisym-Elem*: *Not*(*Elem a b* & *Elem b a*)
⟨*proof*⟩

lemma *irreflexiv-Elem*: *Not*(*Elem a a*)
⟨*proof*⟩

lemma *antisym-Elem*: *Elem a b* ⟹ *Not* (*Elem b a*)
⟨*proof*⟩

consts
NatInterval :: *nat* ⇒ *nat* ⇒ *ZF*

primrec
NatInterval n 0 = *Singleton* (*nat2Nat n*)
NatInterval n (*Suc m*) = *union* (*NatInterval n m*) (*Singleton* (*nat2Nat* (*n*+*m*+*1*)))

lemma *n-Elem-NatInterval*[*rule-format*]: ! *q*. *q* <= *m* ⟶ *Elem* (*nat2Nat* (*n*+*q*)) (*NatInterval n m*)
⟨*proof*⟩

lemma *NatInterval-not-Empty*: *NatInterval n m* ≠ *Empty*
⟨*proof*⟩

lemma *increasing-nat2Nat*[*rule-format*]: *0* < *n* ⟶ *Elem* (*nat2Nat* (*n* − *1*)) (*nat2Nat n*)
⟨*proof*⟩

lemma *represent-NatInterval*[*rule-format*]: *Elem* x (*NatInterval* n m) $\longrightarrow$ (*?* u. $n \leq u$ & $u \leq n{+}m$ & *nat2Nat* $u = x$)
⟨*proof*⟩

lemma *inj-nat2Nat*: *inj nat2Nat*
⟨*proof*⟩

lemma *Nat2nat-nat2Nat*[*simp*]: *Nat2nat* (*nat2Nat* n) $= n$
⟨*proof*⟩

lemma *nat2Nat-Nat2nat*[*simp*]: *Elem* n *Nat* $\Longrightarrow$ *nat2Nat* (*Nat2nat* n) $= n$
⟨*proof*⟩

lemma *Nat2nat-SucNat*: *Elem* N *Nat* $\Longrightarrow$ *Nat2nat* (*SucNat* N) $=$ *Suc* (*Nat2nat* N)
⟨*proof*⟩

lemma *Elem-Opair-exists*: *?* z. *Elem* x z & *Elem* y z & *Elem* z (*Opair* x y)
⟨*proof*⟩

lemma *UNIV-is-not-in-ZF*: *UNIV* $\neq$ *explode* R
⟨*proof*⟩

constdefs
SpecialR :: (*ZF* $*$ *ZF*) *set*
SpecialR $\equiv$ { (x, y) . $x \neq$ *Empty* $\wedge$ $y =$ *Empty*}

lemma *wf SpecialR*
⟨*proof*⟩

constdefs
Ext :: ($'a$ $*$ $'b$) *set* $\Rightarrow$ $'b$ $\Rightarrow$ $'a$ *set*
Ext R y $\equiv$ { x . (x, y) $\in R$ }

lemma *Ext-Elem*: *Ext is-Elem-of* $=$ *explode*
⟨*proof*⟩

lemma *Ext SpecialR Empty* $\neq$ *explode* z
⟨*proof*⟩

constdefs
implode :: *ZF set* $\Rightarrow$ *ZF*
implode $==$ *inv explode*

lemma *inj-explode*: *inj explode*

⟨proof⟩

lemma *implode-explode*[*simp*]: *implode* (*explode* *x*) = *x*
⟨proof⟩

constdefs
regular :: (*ZF* ∗ *ZF*) *set* ⇒ *bool*
regular R == ! *A*. *A* ≠ *Empty* ⟶ (*?* *x*. *Elem x A* & (! *y*. (*y*, *x*) ∈ *R* ⟶ *Not* (*Elem y A*)))
set-like :: (*ZF* ∗ *ZF*) *set* ⇒ *bool*
set-like R == ! *y*. *Ext R y* ∈ *range explode*
wfzf :: (*ZF* ∗ *ZF*) *set* ⇒ *bool*
wfzf R == *regular R* & *set-like R*

lemma *regular-Elem*: *regular is-Elem-of*
⟨proof⟩

lemma *set-like-Elem*: *set-like is-Elem-of*
⟨proof⟩

lemma *wfzf-is-Elem-of*: *wfzf is-Elem-of*
⟨proof⟩

constdefs
SeqSum :: (*nat* ⇒ *ZF*) ⇒ *ZF*
SeqSum f == *Sum* (*Repl Nat* (*f o Nat2nat*))

lemma *SeqSum*: *Elem x* (*SeqSum f*) = (*?* *n*. *Elem x* (*f n*))
⟨proof⟩

constdefs
Ext-ZF :: (*ZF* ∗ *ZF*) *set* ⇒ *ZF* ⇒ *ZF*
Ext-ZF R s == *implode* (*Ext R s*)

lemma *Elem-implode*: *A* ∈ *range explode* ⟹ *Elem x* (*implode A*) = (*x* ∈ *A*)
⟨proof⟩

lemma *Elem-Ext-ZF*: *set-like R* ⟹ *Elem x* (*Ext-ZF R s*) = ((*x*,*s*) ∈ *R*)
⟨proof⟩

consts
Ext-ZF-n :: (*ZF* ∗ *ZF*) *set* ⇒ *ZF* ⇒ *nat* ⇒ *ZF*

primrec
Ext-ZF-n R s 0 = *Ext-ZF R s*
Ext-ZF-n R s (*Suc n*) = *Sum* (*Repl* (*Ext-ZF-n R s n*) (*Ext-ZF R*))

constdefs
Ext-ZF-hull :: (*ZF* ∗ *ZF*) *set* ⇒ *ZF* ⇒ *ZF*

$Ext\text{-}ZF\text{-}hull\ R\ s == SeqSum\ (Ext\text{-}ZF\text{-}n\ R\ s)$

lemma *Elem-Ext-ZF-hull*:
assumes *set-like-R*: $set\text{-}like\ R$
shows $Elem\ x\ (Ext\text{-}ZF\text{-}hull\ R\ S) = (?\ n.\ Elem\ x\ (Ext\text{-}ZF\text{-}n\ R\ S\ n))$
⟨*proof*⟩

lemma *Elem-Elem-Ext-ZF-hull*:
assumes *set-like-R*: $set\text{-}like\ R$
and *x-hull*: $Elem\ x\ (Ext\text{-}ZF\text{-}hull\ R\ S)$
and *y-R-x*: $(y, x) \in R$
shows $Elem\ y\ (Ext\text{-}ZF\text{-}hull\ R\ S)$
⟨*proof*⟩

lemma *wfzf-minimal*:
assumes *hyps*: $wfzf\ R\ C \neq \{\}$
shows $\exists x.\ x \in C \wedge (\forall y.\ (y, x) \in R \longrightarrow y \notin C)$
⟨*proof*⟩

lemma *wfzf-implies-wf*: $wfzf\ R \Longrightarrow wf\ R$
⟨*proof*⟩

lemma *wf-is-Elem-of*: $wf\ is\text{-}Elem\text{-}of$
⟨*proof*⟩

lemma *in-Ext-RTrans-implies-Elem-Ext-ZF-hull*:
$set\text{-}like\ R \Longrightarrow x \in (Ext\ (R\hat{}+)\ s) \Longrightarrow Elem\ x\ (Ext\text{-}ZF\text{-}hull\ R\ s)$
⟨*proof*⟩

lemma *implodeable-Ext-trancl*: $set\text{-}like\ R \Longrightarrow set\text{-}like\ (R\hat{}+)$
⟨*proof*⟩

lemma *Elem-Ext-ZF-hull-implies-in-Ext-RTrans*[*rule-format*]:
$set\text{-}like\ R \Longrightarrow !\ x.\ Elem\ x\ (Ext\text{-}ZF\text{-}n\ R\ s\ n) \longrightarrow x \in (Ext\ (R\hat{}+)\ s)$
⟨*proof*⟩

lemma $set\text{-}like\ R \Longrightarrow Ext\text{-}ZF\ (R\hat{}+)\ s = Ext\text{-}ZF\text{-}hull\ R\ s$
⟨*proof*⟩

lemma *wf-implies-regular*: $wf\ R \Longrightarrow regular\ R$
⟨*proof*⟩

lemma *wf-eq-wfzf*: $(wf\ R \wedge set\text{-}like\ R) = wfzf\ R$
⟨*proof*⟩

lemma *wfzf-trancl*: $wfzf\ R \Longrightarrow wfzf\ (R\hat{}+)$
⟨*proof*⟩

lemma *Ext-subset-mono*: $R \subseteq S \Longrightarrow Ext\ R\ y \subseteq Ext\ S\ y$

$\langle proof \rangle$

lemma *set-like-subset*: *set-like* $R \Longrightarrow S \subseteq R \Longrightarrow$ *set-like* S
$\langle proof \rangle$

lemma *wfzf-subset*: *wfzf* $S \Longrightarrow R \subseteq S \Longrightarrow$ *wfzf* R
$\langle proof \rangle$

end

theory *Zet*
imports *HOLZF*
begin

typedef $'a$ *zet* $= \{A :: 'a\ set \mid A\ f\ z.\ inj\text{-}on\ f\ A \wedge f\ `\ A \subseteq explode\ z\}$
$\langle proof \rangle$

constdefs
zin $:: 'a \Rightarrow 'a\ zet \Rightarrow bool$
zin $x\ A == x \in (Rep\text{-}zet\ A)$

lemma *zet-ext-eq*: $(A = B) = (!\ x.\ zin\ x\ A = zin\ x\ B)$
$\langle proof \rangle$

constdefs
zimage $:: ('a \Rightarrow 'b) \Rightarrow 'a\ zet \Rightarrow 'b\ zet$
zimage $f\ A == Abs\text{-}zet\ (image\ f\ (Rep\text{-}zet\ A))$

lemma *zet-def'*: $zet = \{A :: 'a\ set \mid A\ f\ z.\ inj\text{-}on\ f\ A \wedge f\ `\ A = explode\ z\}$
$\langle proof \rangle$

lemma *image-Inv-f-f*: $inj\text{-}on\ f\ B \Longrightarrow A \subseteq B \Longrightarrow (Inv\ B\ f)\ `\ f\ `\ A = A$
$\langle proof \rangle$

lemma *image-zet-rep*: $A \in zet \Longrightarrow ?\ z\ .\ g\ `\ A = explode\ z$
$\langle proof \rangle$

lemma *Inv-f-f-mem*:
assumes $x \in A$
shows $Inv\ A\ g\ (g\ x) \in A$
$\langle proof \rangle$

lemma *zet-image-mem*:
assumes *Azet*: $A \in zet$
shows $g\ `\ A \in zet$
$\langle proof \rangle$

lemma *Rep-zimage-eq*: *Rep-zet* (*zimage f A*) = *image f* (*Rep-zet A*)
⟨*proof*⟩

lemma *zimage-iff*: *zin y* (*zimage f A*) = (*? x. zin x A* & *y* = *f x*)
⟨*proof*⟩

constdefs
zimplode :: *ZF zet* ⇒ *ZF*
zimplode A == *implode* (*Rep-zet A*)
zexplode :: *ZF* ⇒ *ZF zet*
zexplode z == *Abs-zet* (*explode z*)

lemma *Rep-zet-eq-explode*: *? z. Rep-zet A* = *explode z*
⟨*proof*⟩

lemma *zexplode-zimplode*: *zexplode* (*zimplode A*) = *A*
⟨*proof*⟩

lemma *explode-mem-zet*: *explode z* ∈ *zet*
⟨*proof*⟩

lemma *zimplode-zexplode*: *zimplode* (*zexplode z*) = *z*
⟨*proof*⟩

lemma *zin-zexplode-eq*: *zin x* (*zexplode A*) = *Elem x A*
⟨*proof*⟩

lemma *comp-zimage-eq*: *zimage g* (*zimage f A*) = *zimage* (*g o f*) *A*
⟨*proof*⟩

constdefs
zunion :: *'a zet* ⇒ *'a zet* ⇒ *'a zet*
zunion a b ≡ *Abs-zet* ((*Rep-zet a*) ∪ (*Rep-zet b*))
zsubset :: *'a zet* ⇒ *'a zet* ⇒ *bool*
zsubset a b ≡ ! *x. zin x a* ⟶ *zin x b*

lemma *explode-union*: *explode* (*union a b*) = (*explode a*) ∪ (*explode b*)
⟨*proof*⟩

lemma *Rep-zet-zunion*: *Rep-zet* (*zunion a b*) = (*Rep-zet a*) ∪ (*Rep-zet b*)
⟨*proof*⟩

lemma *zunion*: *zin x* (*zunion a b*) = ((*zin x a*) ∨ (*zin x b*))
⟨*proof*⟩

lemma *zimage-zexplode-eq*: *zimage f* (*zexplode z*) = *zexplode* (*Repl z f*)
⟨*proof*⟩

lemma *range-explode-eq-zet*: *range explode* = *zet*

⟨*proof*⟩

lemma *Elem-zimplode*: (*Elem x* (*zimplode z*)) = (*zin x z*)
⟨*proof*⟩

constdefs
zempty :: $'a$ *zet*
zempty ≡ *Abs-zet* {}

lemma *zempty*[*simp*]: ¬ (*zin x zempty*)
⟨*proof*⟩

lemma *zimage-zempty*[*simp*]: *zimage f zempty* = *zempty*
⟨*proof*⟩

lemma *zunion-zempty-left*[*simp*]: *zunion zempty a* = *a*
⟨*proof*⟩

lemma *zunion-zempty-right*[*simp*]: *zunion a zempty* = *a*
⟨*proof*⟩

lemma *zimage-id*[*simp*]: *zimage id A* = *A*
⟨*proof*⟩

lemma *zimage-cong*[*recdef-cong*]: ⟦ *M* = *N*; !! *x*. *zin x N* ⟹ *f x* = *g x* ⟧ ⟹ *zimage f M* = *zimage g N*
⟨*proof*⟩

end

1 Multisets

theory *Multiset*
imports *Main*
begin

1.1 The type of multisets

typedef $'a$ *multiset* = {*f*::$'a$ => *nat*. *finite* {*x* . *f x* > *0*}}
⟨*proof*⟩

lemmas *multiset-typedef* [*simp*] =
Abs-multiset-inverse Rep-multiset-inverse Rep-multiset
and [*simp*] = *Rep-multiset-inject* [*symmetric*]

definition
Mempty :: $'a$ *multiset* ({#}) **where**
{#} = *Abs-multiset* ($\lambda a.\ 0$)

definition
single :: *'a* => *'a multiset* ({#-#}) **where**
{#*a*#} = *Abs-multiset* (λ*b*. *if b* = *a then 1 else 0*)

definition
count :: *'a multiset* => *'a* => *nat* **where**
count = *Rep-multiset*

definition
MCollect :: *'a multiset* => (*'a* => *bool*) => *'a multiset* **where**
MCollect M P = *Abs-multiset* (λ*x*. *if P x then Rep-multiset M x else 0*)

abbreviation
Melem :: *'a* => *'a multiset* => *bool* ((-/ :# -) [*50*, *51*] *50*) **where**
a :# *M* == *count M a* > *0*

syntax
-MCollect :: *pttrn* => *'a multiset* => *bool* => *'a multiset* ((*1*{# - : -./ -#}))
translations
{#*x*:*M*. *P*#} == *CONST MCollect M* (λ*x*. *P*)

definition
set-of :: *'a multiset* => *'a set* **where**
set-of M = {*x*. *x* :# *M*}

instance *multiset* :: (*type*) {*plus*, *minus*, *zero*, *size*}
union-def: *M* + *N* == *Abs-multiset* (λ*a*. *Rep-multiset M a* + *Rep-multiset N a*)
diff-def: *M* − *N* == *Abs-multiset* (λ*a*. *Rep-multiset M a* − *Rep-multiset N a*)
Zero-multiset-def [*simp*]: *0* == {#}
size-def: *size M* == *setsum* (*count M*) (*set-of M*) ⟨*proof*⟩

definition
multiset-inter :: *'a multiset* ⇒ *'a multiset* ⇒ *'a multiset* (**infixl** #∩ *70*) **where**
multiset-inter A B = *A* − (*A* − *B*)

Preservation of the representing set *multiset*.

lemma *const0-in-multiset* [*simp*]: (λ*a*. *0*) ∈ *multiset*
⟨*proof*⟩

lemma *only1-in-multiset* [*simp*]: (λ*b*. *if b* = *a then 1 else 0*) ∈ *multiset*
⟨*proof*⟩

lemma *union-preserves-multiset* [*simp*]:
M ∈ *multiset* ==> *N* ∈ *multiset* ==> (λ*a*. *M a* + *N a*) ∈ *multiset*
⟨*proof*⟩

lemma *diff-preserves-multiset* [*simp*]:
M ∈ *multiset* ==> (λ*a*. *M a* − *N a*) ∈ *multiset*

⟨*proof*⟩

1.2 Algebraic properties of multisets

1.2.1 Union

lemma *union-empty* [*simp*]: $M + \{\#\} = M \wedge \{\#\} + M = M$
⟨*proof*⟩

lemma *union-commute*: $M + N = N + (M::'a\ multiset)$
⟨*proof*⟩

lemma *union-assoc*: $(M + N) + K = M + (N + (K::'a\ multiset))$
⟨*proof*⟩

lemma *union-lcomm*: $M + (N + K) = N + (M + (K::'a\ multiset))$
⟨*proof*⟩

lemmas *union-ac* = *union-assoc union-commute union-lcomm*

instance *multiset* :: (*type*) *comm-monoid-add*
⟨*proof*⟩

1.2.2 Difference

lemma *diff-empty* [*simp*]: $M - \{\#\} = M \wedge \{\#\} - M = \{\#\}$
⟨*proof*⟩

lemma *diff-union-inverse2* [*simp*]: $M + \{\#a\#\} - \{\#a\#\} = M$
⟨*proof*⟩

1.2.3 Count of elements

lemma *count-empty* [*simp*]: $count\ \{\#\}\ a = 0$
⟨*proof*⟩

lemma *count-single* [*simp*]: $count\ \{\#b\#\}\ a = (if\ b = a\ then\ 1\ else\ 0)$
⟨*proof*⟩

lemma *count-union* [*simp*]: $count\ (M + N)\ a = count\ M\ a + count\ N\ a$
⟨*proof*⟩

lemma *count-diff* [*simp*]: $count\ (M - N)\ a = count\ M\ a - count\ N\ a$
⟨*proof*⟩

1.2.4 Set of elements

lemma *set-of-empty* [*simp*]: $set\text{-}of\ \{\#\} = \{\}$
⟨*proof*⟩

lemma *set-of-single* [*simp*]: *set-of* $\{\#b\#\} = \{b\}$
⟨*proof*⟩

lemma *set-of-union* [*simp*]: *set-of* $(M + N) =$ *set-of* $M \cup$ *set-of* N
⟨*proof*⟩

lemma *set-of-eq-empty-iff* [*simp*]: (*set-of* $M = \{\}) = (M = \{\#\})$
⟨*proof*⟩

lemma *mem-set-of-iff* [*simp*]: $(x \in$ *set-of* $M) = (x :\# M)$
⟨*proof*⟩

1.2.5 Size

lemma *size-empty* [*simp*]: *size* $\{\#\} = 0$
⟨*proof*⟩

lemma *size-single* [*simp*]: *size* $\{\#b\#\} = 1$
⟨*proof*⟩

lemma *finite-set-of* [*iff*]: *finite* (*set-of* M)
⟨*proof*⟩

lemma *setsum-count-Int*:
finite A ==> *setsum* (*count* N) ($A \cap$ *set-of* N) = *setsum* (*count* N) A
⟨*proof*⟩

lemma *size-union* [*simp*]: *size* ($M + N$::'*a multiset*) = *size* M + *size* N
⟨*proof*⟩

lemma *size-eq-0-iff-empty* [*iff*]: (*size* $M = 0) = (M = \{\#\})$
⟨*proof*⟩

lemma *size-eq-Suc-imp-elem*: *size* M = *Suc* n ==> $\exists\, a.\ a :\# M$
⟨*proof*⟩

1.2.6 Equality of multisets

lemma *multiset-eq-conv-count-eq*: $(M = N) = (\forall\, a.$ *count* $M\ a$ = *count* $N\ a)$
⟨*proof*⟩

lemma *single-not-empty* [*simp*]: $\{\#a\#\} \neq \{\#\} \wedge \{\#\} \neq \{\#a\#\}$
⟨*proof*⟩

lemma *single-eq-single* [*simp*]: $(\{\#a\#\} = \{\#b\#\}) = (a = b)$
⟨*proof*⟩

lemma *union-eq-empty* [*iff*]: $(M + N = \{\#\}) = (M = \{\#\} \wedge N = \{\#\})$
⟨*proof*⟩

lemma *empty-eq-union* [*iff*]: $(\{\#\} = M + N) = (M = \{\#\} \wedge N = \{\#\})$
⟨*proof*⟩

lemma *union-right-cancel* [*simp*]: $(M + K = N + K) = (M = (N::'a\ multiset))$
⟨*proof*⟩

lemma *union-left-cancel* [*simp*]: $(K + M = K + N) = (M = (N::'a\ multiset))$
⟨*proof*⟩

lemma *union-is-single*:
$(M + N = \{\#a\#\}) = (M = \{\#a\#\} \wedge N=\{\#\} \vee M = \{\#\} \wedge N = \{\#a\#\})$
⟨*proof*⟩

lemma *single-is-union*:
$(\{\#a\#\} = M + N) = (\{\#a\#\} = M \wedge N = \{\#\} \vee M = \{\#\} \wedge \{\#a\#\} = N)$
⟨*proof*⟩

lemma *add-eq-conv-diff*:
$(M + \{\#a\#\} = N + \{\#b\#\}) =$
$(M = N \wedge a = b \vee M = N - \{\#a\#\} + \{\#b\#\} \wedge N = M - \{\#b\#\} + \{\#a\#\})$
⟨*proof*⟩

declare *Rep-multiset-inject* [*symmetric*, *simp del*]

instance *multiset* :: (*type*) *cancel-ab-semigroup-add*
⟨*proof*⟩

1.2.7 Intersection

lemma *multiset-inter-count*:
$count\ (A\ \#\cap\ B)\ x = min\ (count\ A\ x)\ (count\ B\ x)$
⟨*proof*⟩

lemma *multiset-inter-commute*: $A\ \#\cap\ B = B\ \#\cap\ A$
⟨*proof*⟩

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

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

lemmas *multiset-inter-ac* =
multiset-inter-commute
multiset-inter-assoc
multiset-inter-left-commute

lemma *multiset-union-diff-commute*: $B \ \#\cap\ C = \{\#\} \Longrightarrow A + B - C = A - C + B$
⟨*proof*⟩

1.3 Induction over multisets

lemma *setsum-decr*:
$finite\ F ==> (0::nat) < f\ a ==>$
$setsum\ (f\ (a := f\ a - 1))\ F = (if\ a{\in}F\ then\ setsum\ f\ F - 1\ else\ setsum\ f\ F)$
⟨*proof*⟩

lemma *rep-multiset-induct-aux*:
assumes *1*: $P\ (\lambda a.\ (0::nat))$
and *2*: $!!f\ b.\ f \in multiset ==> P\ f ==> P\ (f\ (b := f\ b + 1))$
shows $\forall f.\ f \in multiset\ --> setsum\ f\ \{x.\ f\ x \neq 0\} = n\ --> P\ f$
⟨*proof*⟩

theorem *rep-multiset-induct*:
$f \in multiset ==> P\ (\lambda a.\ 0) ==>$
$(!!f\ b.\ f \in multiset ==> P\ f ==> P\ (f\ (b := f\ b + 1))) ==> P\ f$
⟨*proof*⟩

theorem *multiset-induct* [*case-names empty add*, *induct type*: *multiset*]:
assumes *empty*: $P\ \{\#\}$
and *add*: $!!M\ x.\ P\ M ==> P\ (M + \{\#x\#\})$
shows $P\ M$
⟨*proof*⟩

lemma *MCollect-preserves-multiset*:
$M \in multiset ==> (\lambda x.\ if\ P\ x\ then\ M\ x\ else\ 0) \in multiset$
⟨*proof*⟩

lemma *count-MCollect* [*simp*]:
$count\ \{\#\ x{:}M.\ P\ x\ \#\}\ a = (if\ P\ a\ then\ count\ M\ a\ else\ 0)$
⟨*proof*⟩

lemma *set-of-MCollect* [*simp*]: $set\text{-}of\ \{\#\ x{:}M.\ P\ x\ \#\} = set\text{-}of\ M \cap \{x.\ P\ x\}$
⟨*proof*⟩

lemma *multiset-partition*: $M = \{\#\ x{:}M.\ P\ x\ \#\} + \{\#\ x{:}M.\ \neg\ P\ x\ \#\}$
⟨*proof*⟩

lemma *add-eq-conv-ex*:
$(M + \{\#a\#\} = N + \{\#b\#\}) =$
$(M = N \wedge a = b \vee (\exists K.\ M = K + \{\#b\#\} \wedge N = K + \{\#a\#\}))$
⟨*proof*⟩

declare *multiset-typedef* [*simp del*]

1.4 Multiset orderings

1.4.1 Well-foundedness

definition
$mult1 :: ('a \times 'a)\ set => ('a\ multiset \times 'a\ multiset)\ set$ **where**
$mult1\ r =$
$\{(N, M).\ \exists\, a\ M0\ K.\ M = M0 + \{\#a\#\} \wedge N = M0 + K \wedge$
$(\forall\, b.\ b :\#\ K \dashrightarrow (b, a) \in r)\}$

definition
$mult :: ('a \times 'a)\ set => ('a\ multiset \times 'a\ multiset)\ set$ **where**
$mult\ r = (mult1\ r)^+$

lemma *not-less-empty* [*iff*]: $(M, \{\#\}) \notin mult1\ r$
⟨*proof*⟩

lemma *less-add*: $(N, M0 + \{\#a\#\}) \in mult1\ r ==>$
$(\exists\, M.\ (M, M0) \in mult1\ r \wedge N = M + \{\#a\#\}) \vee$
$(\exists\, K.\ (\forall\, b.\ b :\#\ K \dashrightarrow (b, a) \in r) \wedge N = M0 + K)$
(**is** - $\Longrightarrow$ *?case1* $(mult1\ r) \vee$ *?case2*)
⟨*proof*⟩

lemma *all-accessible*: $wf\ r ==> \forall\, M.\ M \in acc\ (mult1\ r)$
⟨*proof*⟩

theorem *wf-mult1*: $wf\ r ==> wf\ (mult1\ r)$
⟨*proof*⟩

theorem *wf-mult*: $wf\ r ==> wf\ (mult\ r)$
⟨*proof*⟩

1.4.2 Closure-free presentation

lemma *diff-union-single-conv*: $a :\#\ J ==> I + J - \{\#a\#\} = I + (J - \{\#a\#\})$
⟨*proof*⟩

One direction.

lemma *mult-implies-one-step*:
$trans\ r ==> (M, N) \in mult\ r ==>$
$\exists\, I\ J\ K.\ N = I + J \wedge M = I + K \wedge J \neq \{\#\} \wedge$
$(\forall\, k \in set\text{-}of\ K.\ \exists\, j \in set\text{-}of\ J.\ (k, j) \in r)$
⟨*proof*⟩

lemma *elem-imp-eq-diff-union*: $a :\#\ M ==> M = M - \{\#a\#\} + \{\#a\#\}$
⟨*proof*⟩

lemma *size-eq-Suc-imp-eq-union*: $size\ M = Suc\ n ==> \exists\, a\ N.\ M = N + \{\#a\#\}$
⟨*proof*⟩

lemma *one-step-implies-mult-aux*:
$trans\ r\ ==>$
$\forall I\ J\ K.\ (size\ J = n \wedge J \neq \{\#\} \wedge (\forall k \in set\text{-}of\ K.\ \exists j \in set\text{-}of\ J.\ (k, j) \in r))$
$--> (I + K, I + J) \in mult\ r$
⟨*proof*⟩

lemma *one-step-implies-mult*:
$trans\ r\ ==> J \neq \{\#\} ==> \forall k \in set\text{-}of\ K.\ \exists j \in set\text{-}of\ J.\ (k, j) \in r$
$==> (I + K, I + J) \in mult\ r$
⟨*proof*⟩

1.4.3 Partial-order properties

instance *multiset* :: (*type*) *ord* ⟨*proof*⟩

defs (overloaded)
less-multiset-def: $M' < M == (M', M) \in mult\ \{(x', x).\ x' < x\}$
le-multiset-def: $M' <= M == M' = M \vee M' < (M::'a\ multiset)$

lemma *trans-base-order*: $trans\ \{(x', x).\ x' < (x::'a::order)\}$
⟨*proof*⟩

Irreflexivity.

lemma *mult-irrefl-aux*:
$finite\ A ==> (\forall x \in A.\ \exists y \in A.\ x < (y::'a::order)) \Longrightarrow A = \{\}$
⟨*proof*⟩

lemma *mult-less-not-refl*: $\neg\ M < (M::'a::order\ multiset)$
⟨*proof*⟩

lemma *mult-less-irrefl* [*elim*!]: $M < (M::'a::order\ multiset) ==> R$
⟨*proof*⟩

Transitivity.

theorem *mult-less-trans*: $K < M ==> M < N ==> K < (N::'a::order\ multiset)$
⟨*proof*⟩

Asymmetry.

theorem *mult-less-not-sym*: $M < N ==> \neg\ N < (M::'a::order\ multiset)$
⟨*proof*⟩

theorem *mult-less-asym*:
$M < N ==> (\neg\ P ==> N < (M::'a::order\ multiset)) ==> P$
⟨*proof*⟩

theorem *mult-le-refl* [*iff*]: $M <= (M::'a::order\ multiset)$
⟨*proof*⟩

Anti-symmetry.

theorem *mult-le-antisym*:
  $M <= N ==> N <= M ==> M = (N::'a::order\ multiset)$
  ⟨*proof*⟩

Transitivity.

theorem *mult-le-trans*:
  $K <= M ==> M <= N ==> K <= (N::'a::order\ multiset)$
  ⟨*proof*⟩

theorem *mult-less-le*: $(M < N) = (M <= N \wedge M \neq (N::'a::order\ multiset))$
  ⟨*proof*⟩

Partial order.

instance *multiset* :: (*order*) *order*
  ⟨*proof*⟩

1.4.4 Monotonicity of multiset union

lemma *mult1-union*:
  $(B, D) \in mult1\ r ==> trans\ r ==> (C + B, C + D) \in mult1\ r$
  ⟨*proof*⟩

lemma *union-less-mono2*: $B < D ==> C + B < C + (D::'a::order\ multiset)$
  ⟨*proof*⟩

lemma *union-less-mono1*: $B < D ==> B + C < D + (C::'a::order\ multiset)$
  ⟨*proof*⟩

lemma *union-less-mono*:
  $A < C ==> B < D ==> A + B < C + (D::'a::order\ multiset)$
  ⟨*proof*⟩

lemma *union-le-mono*:
  $A <= C ==> B <= D ==> A + B <= C + (D::'a::order\ multiset)$
  ⟨*proof*⟩

lemma *empty-leI* [*iff*]: $\{\#\} <= (M::'a::order\ multiset)$
  ⟨*proof*⟩

lemma *union-upper1*: $A <= A + (B::'a::order\ multiset)$
⟨*proof*⟩

lemma *union-upper2*: $B <= A + (B::'a::order\ multiset)$
  ⟨*proof*⟩

instance *multiset* :: (*order*) *pordered-ab-semigroup-add*
⟨*proof*⟩

1.5 Link with lists

consts
$multiset\text{-}of :: 'a\ list \Rightarrow 'a\ multiset$
primrec
$multiset\text{-}of\ [] = \{\#\}$
$multiset\text{-}of\ (a\ \#\ x) = multiset\text{-}of\ x + \{\#\ a\ \#\}$

lemma *multiset-of-zero-iff* [*simp*]: $(multiset\text{-}of\ x = \{\#\}) = (x = [])$
⟨*proof*⟩

lemma *multiset-of-zero-iff-right* [*simp*]: $(\{\#\} = multiset\text{-}of\ x) = (x = [])$
⟨*proof*⟩

lemma *set-of-multiset-of* [*simp*]: $set\text{-}of(multiset\text{-}of\ x) = set\ x$
⟨*proof*⟩

lemma *mem-set-multiset-eq*: $x \in set\ xs = (x :\#\ multiset\text{-}of\ xs)$
⟨*proof*⟩

lemma *multiset-of-append* [*simp*]:
$multiset\text{-}of\ (xs\ @\ ys) = multiset\text{-}of\ xs + multiset\text{-}of\ ys$
⟨*proof*⟩

lemma *surj-multiset-of*: $surj\ multiset\text{-}of$
⟨*proof*⟩

lemma *set-count-greater-0*: $set\ x = \{a.\ count\ (multiset\text{-}of\ x)\ a > 0\}$
⟨*proof*⟩

lemma *distinct-count-atmost-1*:
$distinct\ x = (!\ a.\ count\ (multiset\text{-}of\ x)\ a = (if\ a \in set\ x\ then\ 1\ else\ 0))$
⟨*proof*⟩

lemma *multiset-of-eq-setD*:
$multiset\text{-}of\ xs = multiset\text{-}of\ ys \Longrightarrow set\ xs = set\ ys$
⟨*proof*⟩

lemma *set-eq-iff-multiset-of-eq-distinct*:
$[\![distinct\ x;\ distinct\ y]\!]$
$\Longrightarrow (set\ x = set\ y) = (multiset\text{-}of\ x = multiset\text{-}of\ y)$
⟨*proof*⟩

lemma *set-eq-iff-multiset-of-remdups-eq*:
$(set\ x = set\ y) = (multiset\text{-}of\ (remdups\ x) = multiset\text{-}of\ (remdups\ y))$
⟨*proof*⟩

lemma *multiset-of-compl-union* [*simp*]:
$multiset\text{-}of\ [x{\leftarrow}xs.\ P\ x] + multiset\text{-}of\ [x{\leftarrow}xs.\ \neg P\ x] = multiset\text{-}of\ xs$
⟨*proof*⟩

lemma *count-filter*:
$count\ (multiset\text{-}of\ xs)\ x = length\ [y \leftarrow xs.\ y = x]$
⟨*proof*⟩

1.6 Pointwise ordering induced by count

definition
$mset\text{-}le :: 'a\ multiset \Rightarrow 'a\ multiset \Rightarrow bool$ (**infix** $\leq\#$ *50*) **where**
$(A \leq\# B) = (\forall a.\ count\ A\ a \leq count\ B\ a)$
definition
$mset\text{-}less :: 'a\ multiset \Rightarrow 'a\ multiset \Rightarrow bool$ (**infix** $<\#$ *50*) **where**
$(A <\# B) = (A \leq\# B \wedge A \neq B)$

lemma *mset-le-refl*[*simp*]: $A \leq\# A$
⟨*proof*⟩

lemma *mset-le-trans*: $\llbracket A \leq\# B;\ B \leq\# C \rrbracket \Longrightarrow A \leq\# C$
⟨*proof*⟩

lemma *mset-le-antisym*: $\llbracket A \leq\# B;\ B \leq\# A \rrbracket \Longrightarrow A = B$
⟨*proof*⟩

lemma *mset-le-exists-conv*:
$(A \leq\# B) = (\exists C.\ B = A + C)$
⟨*proof*⟩

lemma *mset-le-mono-add-right-cancel*[*simp*]: $(A + C \leq\# B + C) = (A \leq\# B)$
⟨*proof*⟩

lemma *mset-le-mono-add-left-cancel*[*simp*]: $(C + A \leq\# C + B) = (A \leq\# B)$
⟨*proof*⟩

lemma *mset-le-mono-add*: $\llbracket A \leq\# B;\ C \leq\# D \rrbracket \Longrightarrow A + C \leq\# B + D$
⟨*proof*⟩

lemma *mset-le-add-left*[*simp*]: $A \leq\# A + B$
⟨*proof*⟩

lemma *mset-le-add-right*[*simp*]: $B \leq\# A + B$
⟨*proof*⟩

lemma *multiset-of-remdups-le*: $multiset\text{-}of\ (remdups\ xs) \leq\# multiset\text{-}of\ xs$
⟨*proof*⟩

interpretation *mset-order*:
$order\ [op \leq\#\ op <\#]$
⟨*proof*⟩

interpretation *mset-order-cancel-semigroup*:
pordered-cancel-ab-semigroup-add [*op* $\leq$# *op* <# *op* +]
⟨*proof*⟩

interpretation *mset-order-semigroup-cancel*:
pordered-ab-semigroup-add-imp-le [*op* $\leq$# *op* <# *op* +]
⟨*proof*⟩

end

theory *LProd*
imports *Multiset*
begin

inductive-set
lprod :: $('a * 'a)$ *set* $\Rightarrow$ $('a$ *list* $* 'a$ *list*$)$ *set*
for R :: $('a * 'a)$ *set*
where
lprod-single[*intro*!]: $(a,\ b) \in R \Longrightarrow ([a],\ [b]) \in$ *lprod* R
| *lprod-list*[*intro*!]: $(ah@at,\ bh@bt) \in$ *lprod* $R \Longrightarrow (a,b) \in R \vee a = b \Longrightarrow (ah@a\#at,$ $bh@b\#bt) \in$ *lprod* R

lemma $(as,bs) \in$ *lprod* $R \Longrightarrow$ *length* as = *length* bs
⟨*proof*⟩

lemma $(as,\ bs) \in$ *lprod* $R \Longrightarrow 1 \leq$ *length* $as \wedge 1 \leq$ *length* bs
⟨*proof*⟩

lemma *lprod-subset-elem*: $(as,\ bs) \in$ *lprod* $S \Longrightarrow S \subseteq R \Longrightarrow (as,\ bs) \in$ *lprod* R
⟨*proof*⟩

lemma *lprod-subset*: $S \subseteq R \Longrightarrow$ *lprod* $S \subseteq$ *lprod* R
⟨*proof*⟩

lemma *lprod-implies-mult*: $(as,\ bs) \in$ *lprod* $R \Longrightarrow$ *trans* $R \Longrightarrow$ (*multiset-of* as, *multiset-of* bs) $\in$ *mult* R
⟨*proof*⟩

lemma *wf-lprod*[*recdef-wf*,*simp*,*intro*]:
assumes *wf-R*: *wf* R
shows *wf* (*lprod* R)
⟨*proof*⟩

constdefs
gprod-2-2 :: $('a * 'a)$ *set* $\Rightarrow$ $(('a * 'a) * ('a * 'a))$ *set*
gprod-2-2 $R \equiv \{\ ((a,b),\ (c,d))\ .\ (a = c \wedge (b,d) \in R) \vee (b = d \wedge (a,c) \in R)\ \}$
gprod-2-1 :: $('a * 'a)$ *set* $\Rightarrow$ $(('a * 'a) * ('a * 'a))$ *set*

gprod-2-1 $R \equiv$ { $((a,b), (c,d))$. $(a = d \land (b,c) \in R) \lor (b = c \land (a,d) \in R)$ }

lemma *lprod-2-3*: $(a,\ b) \in R \Longrightarrow ([a,\ c],\ [b,\ c]) \in$ *lprod* R
⟨*proof*⟩

lemma *lprod-2-4*: $(a,\ b) \in R \Longrightarrow ([c,\ a],\ [c,\ b]) \in$ *lprod* R
⟨*proof*⟩

lemma *lprod-2-1*: $(a,\ b) \in R \Longrightarrow ([c,\ a],\ [b,\ c]) \in$ *lprod* R
⟨*proof*⟩

lemma *lprod-2-2*: $(a,\ b) \in R \Longrightarrow ([a,\ c],\ [c,\ b]) \in$ *lprod* R
⟨*proof*⟩

lemma [*recdef-wf*, *simp*, *intro*]:
assumes *wfR*: *wf* R **shows** *wf* (*gprod-2-1* R)
⟨*proof*⟩

lemma [*recdef-wf*, *simp*, *intro*]:
assumes *wfR*: *wf* R **shows** *wf* (*gprod-2-2* R)
⟨*proof*⟩

lemma *lprod-3-1*: **assumes** $(x',\ x) \in R$ **shows** $([y,\ z,\ x'],\ [x,\ y,\ z]) \in$ *lprod* R
⟨*proof*⟩

lemma *lprod-3-2*: **assumes** $(z',z) \in R$ **shows** $([z',\ x,\ y],\ [x,y,z]) \in$ *lprod* R
⟨*proof*⟩

lemma *lprod-3-3*: **assumes** *xr*: $(xr,\ x) \in R$ **shows** $([xr,\ y,\ z],\ [x,\ y,\ z]) \in$ *lprod* R
⟨*proof*⟩

lemma *lprod-3-4*: **assumes** *yr*: $(yr,\ y) \in R$ **shows** $([x,\ yr,\ z],\ [x,\ y,\ z]) \in$ *lprod* R
⟨*proof*⟩

lemma *lprod-3-5*: **assumes** *zr*: $(zr,\ z) \in R$ **shows** $([x,\ y,\ zr],\ [x,\ y,\ z]) \in$ *lprod* R
⟨*proof*⟩

lemma *lprod-3-6*: **assumes** y': $(y',\ y) \in R$ **shows** $([x,\ z,\ y'],\ [x,\ y,\ z]) \in$ *lprod* R
⟨*proof*⟩

lemma *lprod-3-7*: **assumes** z': $(z',z) \in R$ **shows** $([x,\ z',\ y],\ [x,\ y,\ z]) \in$ *lprod* R
⟨*proof*⟩

constdefs
perm :: $('a \Rightarrow 'a) \Rightarrow 'a$ *set* $\Rightarrow$ *bool*
perm f A $\equiv$ *inj-on f A* $\land$ *f* ‘ $A = A$

lemma $((as,bs) \in$ *lprod* $R) =$
$(\exists\ f.$ *perm f* {*0* ..< (*length as*)} $\land$

$(\forall\ j.\ j < \mathit{length\ as} \longrightarrow ((\mathit{nth\ as}\ j,\ \mathit{nth\ bs}\ (f\ j)) \in R \lor (\mathit{nth\ as}\ j = \mathit{nth\ bs}\ (f\ j))))$
$\land$
$(\exists\ i.\ i < \mathit{length\ as} \land (\mathit{nth\ as}\ i,\ \mathit{nth\ bs}\ (f\ i)) \in R))$
⟨*proof*⟩

lemma $\mathit{trans}\ R \Longrightarrow (\mathit{ah}@a\#\mathit{at},\ \mathit{bh}@b\#\mathit{bt}) \in \mathit{lprod}\ R \Longrightarrow (b,\ a) \in R \lor a = b \Longrightarrow (\mathit{ah}@\mathit{at},\ \mathit{bh}@\mathit{bt}) \in \mathit{lprod}\ R$
⟨*proof*⟩

end

theory *MainZF*
imports *Zet LProd*
begin
end

theory *Games*
imports *MainZF*
begin

constdefs
fixgames :: *ZF set* ⇒ *ZF set*
fixgames A ≡ { *Opair l r* | *l r*. *explode l* ⊆ *A* & *explode r* ⊆ *A*}
games-lfp :: *ZF set*
games-lfp ≡ *lfp fixgames*
games-gfp :: *ZF set*
games-gfp ≡ *gfp fixgames*

lemma *mono-fixgames*: *mono* (*fixgames*)
⟨*proof*⟩

lemma *games-lfp-unfold*: *games-lfp* = *fixgames games-lfp*
⟨*proof*⟩

lemma *games-gfp-unfold*: *games-gfp* = *fixgames games-gfp*
⟨*proof*⟩

lemma *games-lfp-nonempty*: *Opair Empty Empty* ∈ *games-lfp*
⟨*proof*⟩

constdefs
left-option :: *ZF* ⇒ *ZF* ⇒ *bool*
left-option g opt ≡ (*Elem opt* (*Fst g*))
right-option :: *ZF* ⇒ *ZF* ⇒ *bool*
right-option g opt ≡ (*Elem opt* (*Snd g*))

is-option-of :: (*ZF* ∗ *ZF*) *set*
is-option-of ≡ { (*opt*, *g*) | *opt g*. *g* ∈ *games-gfp* ∧ (*left-option g opt* ∨ *right-option g opt*) }

lemma *games-lfp-subset-gfp*: *games-lfp* ⊆ *games-gfp*
⟨*proof*⟩

lemma *games-option-stable*:
assumes *fixgames*: *games* = *fixgames games*
and *g*: *g* ∈ *games*
and *opt*: *left-option g opt* ∨ *right-option g opt*
shows *opt* ∈ *games*
⟨*proof*⟩

lemma *option2elem*: (*opt*,*g*) ∈ *is-option-of* ⟹ ∃ *u v*. *Elem opt u* ∧ *Elem u v* ∧ *Elem v g*
⟨*proof*⟩

lemma *is-option-of-subset-is-Elem-of*: *is-option-of* ⊆ (*is-Elem-of*ˆ+)
⟨*proof*⟩

lemma *wfzf-is-option-of*: *wfzf is-option-of*
⟨*proof*⟩

lemma *games-gfp-imp-lfp*: *g* ∈ *games-gfp* ⟶ *g* ∈ *games-lfp*
⟨*proof*⟩

theorem *games-lfp-eq-gfp*: *games-lfp* = *games-gfp*
⟨*proof*⟩

theorem *unique-games*: (*g* = *fixgames g*) = (*g* = *games-lfp*)
⟨*proof*⟩

lemma *games-lfp-option-stable*:
assumes *g*: *g* ∈ *games-lfp*
and *opt*: *left-option g opt* ∨ *right-option g opt*
shows *opt* ∈ *games-lfp*
⟨*proof*⟩

lemma *is-option-of-imp-games*:
assumes *hyp*: (*opt*, *g*) ∈ *is-option-of*
shows *opt* ∈ *games-lfp* ∧ *g* ∈ *games-lfp*
⟨*proof*⟩

lemma *games-lfp-represent*: *x* ∈ *games-lfp* ⟹ ∃ *l r*. *x* = *Opair l r*
⟨*proof*⟩

typedef *game* = *games-lfp*
⟨*proof*⟩

constdefs
left-options :: *game* ⇒ *game zet*
left-options $g \equiv$ *zimage Abs-game* (*zexplode* (*Fst* (*Rep-game* g)))
right-options :: *game* ⇒ *game zet*
right-options $g \equiv$ *zimage Abs-game* (*zexplode* (*Snd* (*Rep-game* g)))
options :: *game* ⇒ *game zet*
options $g \equiv$ *zunion* (*left-options* g) (*right-options* g)
Game :: *game zet* ⇒ *game zet* ⇒ *game*
Game L R ≡ *Abs-game* (*Opair* (*zimplode* (*zimage Rep-game L*)) (*zimplode* (*zimage Rep-game R*)))

lemma *Repl-Rep-game-Abs-game*: ∀ *e*. *Elem e z* ⟶ *e* ∈ *games-lfp* ⟹ *Repl z* (*Rep-game o Abs-game*) = *z*
⟨*proof*⟩

lemma *game-split*: *g* = *Game* (*left-options g*) (*right-options g*)
⟨*proof*⟩

lemma *Opair-in-games-lfp*:
assumes *l*: *explode l* ⊆ *games-lfp*
and *r*: *explode r* ⊆ *games-lfp*
shows *Opair l r* ∈ *games-lfp*
⟨*proof*⟩

lemma *left-options*[*simp*]: *left-options* (*Game l r*) = *l*
⟨*proof*⟩

lemma *right-options*[*simp*]: *right-options* (*Game l r*) = *r*
⟨*proof*⟩

lemma *Game-ext*: (*Game l1 r1* = *Game l2 r2*) = ((*l1* = *l2*) ∧ (*r1* = *r2*))
⟨*proof*⟩

constdefs
option-of :: (*game* ∗ *game*) *set*
option-of ≡ *image* (λ (*option*, *g*). (*Abs-game option*, *Abs-game g*)) *is-option-of*

lemma *option-to-is-option-of*: ((*option*, *g*) ∈ *option-of*) = ((*Rep-game option*, *Rep-game g*) ∈ *is-option-of*)
⟨*proof*⟩

lemma *wf-is-option-of*: *wf is-option-of*
⟨*proof*⟩

lemma *wf-option-of*[*recdef-wf*, *simp*, *intro*]: *wf option-of*
⟨*proof*⟩

lemma *right-option-is-option*[*simp*, *intro*]: *zin x* (*right-options g*) ⟹ *zin x* (*options*

g)
⟨*proof*⟩

lemma *left-option-is-option*[*simp*, *intro*]: *zin x* (*left-options g*) $\Longrightarrow$ *zin x* (*options g*)
⟨*proof*⟩

lemma *zin-options*[*simp*, *intro*]: *zin x* (*options g*) $\Longrightarrow$ (*x*, *g*) $\in$ *option-of*
⟨*proof*⟩

consts
neg-game :: *game* $\Rightarrow$ *game*

recdef *neg-game option-of*
neg-game g = *Game* (*zimage neg-game* (*right-options g*)) (*zimage neg-game* (*left-options g*))

declare *neg-game.simps*[*simp del*]

lemma *neg-game* (*neg-game g*) = *g*
⟨*proof*⟩

consts
ge-game :: (*game* $*$ *game*) $\Rightarrow$ *bool*

recdef *ge-game* (*gprod-2-1 option-of*)
ge-game (*G*, *H*) = ($\forall$ *x*. *if zin x* (*right-options G*) *then* (
if zin x (*left-options H*) *then* $\neg$ (*ge-game* (*H*, *x*) $\vee$ (*ge-game* (*x*, *G*)))
else $\neg$ (*ge-game* (*H*, *x*)))
else (*if zin x* (*left-options H*) *then* $\neg$ (*ge-game* (*x*, *G*)) *else True*))
(**hints** *simp*: *gprod-2-1-def*)

declare *ge-game.simps* [*simp del*]

lemma *ge-game-def*: *ge-game* (*G*, *H*) = ($\forall$ *x*. (*zin x* (*right-options G*) $\longrightarrow$ $\neg$ *ge-game* (*H*, *x*)) $\wedge$ (*zin x* (*left-options H*) $\longrightarrow$ $\neg$ *ge-game* (*x*, *G*)))
⟨*proof*⟩

lemma *ge-game-leftright-refl*[*rule-format*]:
$\forall$ *y*. (*zin y* (*right-options x*) $\longrightarrow$ $\neg$ *ge-game* (*x*, *y*)) $\wedge$ (*zin y* (*left-options x*) $\longrightarrow$ $\neg$ (*ge-game* (*y*, *x*))) $\wedge$ *ge-game* (*x*, *x*)
⟨*proof*⟩

lemma *ge-game-refl*: *ge-game* (*x*,*x*) ⟨*proof*⟩

lemma $\forall$ *y*. (*zin y* (*right-options x*) $\longrightarrow$ $\neg$ *ge-game* (*x*, *y*)) $\wedge$ (*zin y* (*left-options x*) $\longrightarrow$ $\neg$ (*ge-game* (*y*, *x*))) $\wedge$ *ge-game* (*x*, *x*)

⟨*proof*⟩

constdefs
eq-game :: *game* ⇒ *game* ⇒ *bool*
eq-game G H ≡ *ge-game* (*G*, *H*) ∧ *ge-game* (*H*, *G*)

lemma *eq-game-sym*: (*eq-game G H*) = (*eq-game H G*)
⟨*proof*⟩

lemma *eq-game-refl*: *eq-game G G*
⟨*proof*⟩

lemma *induct-game*: (⋀*x*. ∀ *y*. (*y*, *x*) ∈ *lprod option-of* ⟶ *P y* ⟹ *P x*) ⟹ *P a*
⟨*proof*⟩

lemma *ge-game-trans*:
assumes *ge-game* (*x*, *y*) *ge-game* (*y*, *z*)
shows *ge-game* (*x*, *z*)
⟨*proof*⟩

lemma *eq-game-trans*: *eq-game a b* ⟹ *eq-game b c* ⟹ *eq-game a c*
⟨*proof*⟩

constdefs
zero-game :: *game*
zero-game ≡ *Game zempty zempty*

consts
plus-game :: *game* ∗ *game* ⇒ *game*

recdef *plus-game gprod-2-2 option-of*
plus-game (*G*, *H*) = *Game* (*zunion* (*zimage* (λ *g*. *plus-game* (*g*, *H*)) (*left-options G*))
(*zimage* (λ *h*. *plus-game* (*G*, *h*)) (*left-options H*)))
(*zunion* (*zimage* (λ *g*. *plus-game* (*g*, *H*)) (*right-options G*))
(*zimage* (λ *h*. *plus-game* (*G*, *h*)) (*right-options H*)))
(**hints** *simp add*: *gprod-2-2-def*)

declare *plus-game.simps*[*simp del*]

lemma *plus-game-comm*: *plus-game* (*G*, *H*) = *plus-game* (*H*, *G*)
⟨*proof*⟩

lemma *game-ext-eq*: (*G* = *H*) = (*left-options G* = *left-options H* ∧ *right-options G* = *right-options H*)
⟨*proof*⟩

lemma *left-zero-game*[*simp*]: *left-options* (*zero-game*) = *zempty*

⟨*proof*⟩

lemma *right-zero-game*[*simp*]: *right-options* (*zero-game*) = *zempty*
⟨*proof*⟩

lemma *plus-game-zero-right*[*simp*]: *plus-game* (*G*, *zero-game*) = *G*
⟨*proof*⟩

lemma *plus-game-zero-left*: *plus-game* (*zero-game*, *G*) = *G*
⟨*proof*⟩

lemma *left-imp-options*[*simp*]: *zin opt* (*left-options g*) $\Longrightarrow$ *zin opt* (*options g*)
⟨*proof*⟩

lemma *right-imp-options*[*simp*]: *zin opt* (*right-options g*) $\Longrightarrow$ *zin opt* (*options g*)
⟨*proof*⟩

lemma *left-options-plus*:
left-options (*plus-game* (*u*, *v*)) = *zunion* (*zimage* (λg. *plus-game* (*g*, *v*)) (*left-options u*)) (*zimage* (λh. *plus-game* (*u*, *h*)) (*left-options v*))
⟨*proof*⟩

lemma *right-options-plus*:
right-options (*plus-game* (*u*, *v*)) = *zunion* (*zimage* (λg. *plus-game* (*g*, *v*)) (*right-options u*)) (*zimage* (λh. *plus-game* (*u*, *h*)) (*right-options v*))
⟨*proof*⟩

lemma *left-options-neg*: *left-options* (*neg-game u*) = *zimage neg-game* (*right-options u*)
⟨*proof*⟩

lemma *right-options-neg*: *right-options* (*neg-game u*) = *zimage neg-game* (*left-options u*)
⟨*proof*⟩

lemma *plus-game-assoc*: *plus-game* (*plus-game* (*F*, *G*), *H*) = *plus-game* (*F*, *plus-game* (*G*, *H*))
⟨*proof*⟩

lemma *neg-plus-game*: *neg-game* (*plus-game* (*G*, *H*)) = *plus-game*(*neg-game G*, *neg-game H*)
⟨*proof*⟩

lemma *eq-game-plus-inverse*: *eq-game* (*plus-game* (*x*, *neg-game x*)) *zero-game*
⟨*proof*⟩

lemma *ge-plus-game-left*: *ge-game* (*y*,*z*) = *ge-game*(*plus-game* (*x*, *y*), *plus-game* (*x*, *z*))
⟨*proof*⟩

lemma *ge-plus-game-right*: *ge-game* (y,z) = *ge-game*(*plus-game* (y, x), *plus-game* (z, x))
  ⟨*proof*⟩

lemma *ge-neg-game*: *ge-game* (*neg-game* x, *neg-game* y) = *ge-game* (y, x)
⟨*proof*⟩

constdefs
  eq-game-rel :: (*game* $*$ *game*) *set*
  eq-game-rel $\equiv$ { (p, q) . *eq-game* p q }

typedef *Pg* = *UNIV*//*eq-game-rel*
  ⟨*proof*⟩

lemma *equiv-eq-game*[*simp*]: *equiv UNIV eq-game-rel*
  ⟨*proof*⟩

instance *Pg* :: {*ord*,*zero*,*plus*,*minus*} ⟨*proof*⟩

defs (overloaded)
  Pg-zero-def: *0* $\equiv$ *Abs-Pg* (*eq-game-rel* `` {*zero-game*})
  Pg-le-def: $G \leq H \equiv \exists\ g\ h.\ g \in$ *Rep-Pg* $G \wedge h \in$ *Rep-Pg* $H \wedge$ *ge-game* (h, g)
  Pg-less-def: $G < H \equiv G \leq H \wedge G \neq (H$::*Pg*)
  Pg-minus-def: $-\ G \equiv$ *contents* ($\bigcup\ g \in$ *Rep-Pg* G. {*Abs-Pg* (*eq-game-rel* `` {*neg-game* g}))})
  Pg-plus-def: $G + H \equiv$ *contents* ($\bigcup\ g \in$ *Rep-Pg* G. $\bigcup\ h \in$ *Rep-Pg* H. {*Abs-Pg* (*eq-game-rel* `` {*plus-game* (g,h)}})})
  Pg-diff-def: $G - H \equiv G + (-\ (H$::*Pg*))

lemma *Rep-Abs-eq-Pg*[*simp*]: *Rep-Pg* (*Abs-Pg* (*eq-game-rel* `` {$g$})) = *eq-game-rel* `` {g}
  ⟨*proof*⟩

lemma *char-Pg-le*[*simp*]: (*Abs-Pg* (*eq-game-rel* `` {$g$}) $\leq$ *Abs-Pg* (*eq-game-rel* `` {h})) = (*ge-game* (h, g))
  ⟨*proof*⟩

lemma *char-Pg-eq*[*simp*]: (*Abs-Pg* (*eq-game-rel* `` {$g$}) = *Abs-Pg* (*eq-game-rel* `` {h})) = (*eq-game* g h)
  ⟨*proof*⟩

lemma *char-Pg-plus*[*simp*]: *Abs-Pg* (*eq-game-rel* `` {$g$}) + *Abs-Pg* (*eq-game-rel* `` {h}) = *Abs-Pg* (*eq-game-rel* `` {*plus-game* (g, h)})
⟨*proof*⟩

lemma *char-Pg-minus*[*simp*]: $-$ *Abs-Pg* (*eq-game-rel* `` {$g$}) = *Abs-Pg* (*eq-game-rel* `` {*neg-game* g})
⟨*proof*⟩

lemma *eq-Abs-Pg*[*rule-format*, *cases type*: *Pg*]: $(\forall\ g.\ z = \textit{Abs-Pg}\ (\textit{eq-game-rel}\ ``\ \{g\}) \longrightarrow P) \longrightarrow P$
  ⟨*proof*⟩

instance *Pg* :: *pordered-ab-group-add*
⟨*proof*⟩

end