

ZF

Steven Obua

January 18, 2026

```
theory HOLZF
imports Main
begin
```

```
typedecl ZF
```

```
axiomatization
```

```
  Empty :: ZF and
  Elem :: ZF  $\Rightarrow$  ZF  $\Rightarrow$  bool and
  Sum :: ZF  $\Rightarrow$  ZF and
  Power :: ZF  $\Rightarrow$  ZF and
  Repl :: ZF  $\Rightarrow$  (ZF  $\Rightarrow$  ZF)  $\Rightarrow$  ZF and
  Inf :: ZF
```

```
definition Upair :: ZF  $\Rightarrow$  ZF  $\Rightarrow$  ZF where
```

```
  Upair a b == Repl (Power (Power Empty)) (% x. if x = Empty then a else b)
```

```
definition Singleton:: ZF  $\Rightarrow$  ZF where
```

```
  Singleton x == Upair x x
```

```
definition union :: ZF  $\Rightarrow$  ZF  $\Rightarrow$  ZF where
```

```
  union A B == Sum (Upair A B)
```

```
definition SucNat:: ZF  $\Rightarrow$  ZF where
```

```
  SucNat x == union x (Singleton x)
```

```
definition subset :: ZF  $\Rightarrow$  ZF  $\Rightarrow$  bool where
```

```
  subset A B  $\equiv \forall x. Elem\ x\ A \longrightarrow Elem\ x\ B$ 
```

```
axiomatization where
```

```
  Empty: Not (Elem x Empty) and
  Ext: (x = y) = ( $\forall z. Elem\ z\ x = Elem\ z\ y$ ) and
  Sum: Elem z (Sum x) = ( $\exists y. Elem\ z\ y \wedge Elem\ y\ x$ ) and
  Power: Elem y (Power x) = (subset y x) and
  Repl: Elem b (Repl A f) = ( $\exists a. Elem\ a\ A \wedge b = f\ a$ ) and
  Regularity: A  $\neq Empty \longrightarrow (\exists x. Elem\ x\ A \wedge (\forall y. Elem\ y\ x \longrightarrow Not\ (Elem\ y$ 
```

A))) and

Infinity: Elem Empty Inf $\wedge (\forall x. \text{Elem } x \text{ Inf} \longrightarrow \text{Elem } (\text{SucNat } x) \text{ Inf})$

definition *Sep* :: $ZF \Rightarrow (ZF \Rightarrow \text{bool}) \Rightarrow ZF$ **where**

*Sep A p == (if ($\forall x. \text{Elem } x A \longrightarrow \text{Not } (p x)$) then Empty else
(let $z = (\epsilon x. \text{Elem } x A \ \& \ p x$) in
let $f = \lambda x. (\text{if } p x \text{ then } x \text{ else } z)$ in Repl A f))*

thm *Power*[unfolded subset-def]

theorem *Sep*: $\text{Elem } b (\text{Sep } A p) = (\text{Elem } b A \wedge p b)$
(proof)

lemma *subset-empty*: $\text{subset Empty } A$
(proof)

theorem *Upair*: $\text{Elem } x (\text{Upair } a b) = (x = a \vee x = b)$
(proof)

lemma *Singleton*: $\text{Elem } x (\text{Singleton } y) = (x = y)$
(proof)

definition *Opair* :: $ZF \Rightarrow ZF \Rightarrow ZF$ **where**
Opair a b == Upair (Upair a a) (Upair a b)

lemma *Upair-singleton*: $(\text{Upair } a a = \text{Upair } c d) = (a = c \ \& \ a = d)$
(proof)

lemma *Upair-fstsq*: $(\text{Upair } a b = \text{Upair } a c) = ((a = b \ \& \ a = c) \mid (b = c))$
(proof)

lemma *Upair-comm*: $\text{Upair } a b = \text{Upair } b a$
(proof)

theorem *Opair*: $(\text{Opair } a b = \text{Opair } c d) = (a = c \ \& \ b = d)$
(proof)

definition *Replacement* :: $ZF \Rightarrow (ZF \Rightarrow ZF \text{ option}) \Rightarrow ZF$ **where**
Replacement A f == Repl (Sep A (% a. f a \neq None)) (the o f)

theorem *Replacement*: $\text{Elem } y (\text{Replacement } A f) = (\exists x. \text{Elem } x A \wedge f x = \text{Some } y)$
(proof)

definition *Fst* :: $ZF \Rightarrow ZF$ **where**
Fst q == SOME x. $\exists y. q = \text{Opair } x y$

definition *Snd* :: $ZF \Rightarrow ZF$ **where**
Snd q == SOME y. $\exists x. q = \text{Opair } x y$

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

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

definition *isOpair* :: $ZF \Rightarrow bool$ **where**
isOpair $q == \exists x\ y. q = Opair\ x\ y$

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

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

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

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

definition *explode* :: $ZF \Rightarrow ZF\ set$ **where**
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*: $\llbracket Elem\ a\ A; explode\ A \subseteq B \rrbracket \Longrightarrow a \in B$
 ⟨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⟩

definition *Domain* :: $ZF \Rightarrow ZF$ **where**
Domain $f == Replacement\ f\ (\% p. if\ isOpair\ p\ then\ Some\ (Fst\ p)\ else\ None)$

definition *Range* :: $ZF \Rightarrow ZF$ **where**
Range $f == Replacement\ f\ (\% p. if\ isOpair\ p\ then\ Some\ (Snd\ p)\ else\ None)$

theorem *Domain*: $Elem\ x\ (Domain\ f) = (\exists y. Elem\ (Opair\ x\ y)\ f)$

$\langle \text{proof} \rangle$

theorem *Range*: $\text{Elem } y \ (\text{Range } f) = (\exists x. \text{Elem } (\text{Opair } x \ y) \ f)$
 $\langle \text{proof} \rangle$

theorem *union*: $\text{Elem } x \ (\text{union } A \ B) = (\text{Elem } x \ A \mid \text{Elem } x \ B)$
 $\langle \text{proof} \rangle$

definition *Field* :: $ZF \Rightarrow ZF$ **where**
 $\text{Field } A == \text{union } (\text{Domain } A) \ (\text{Range } A)$

definition *app* :: $ZF \Rightarrow ZF \Rightarrow ZF$ (**infixl** $\langle ' \rangle$ 90) — function application **where**
 $f \ ' \ x == (\text{THE } y. \text{Elem } (\text{Opair } x \ y) \ f)$

definition *isFun* :: $ZF \Rightarrow \text{bool}$ **where**
 $\text{isFun } f == (\forall x \ y1 \ y2. \text{Elem } (\text{Opair } x \ y1) \ f \ \& \ \text{Elem } (\text{Opair } x \ y2) \ f \longrightarrow y1 = y2)$

definition *Lambda* :: $ZF \Rightarrow (ZF \Rightarrow ZF) \Rightarrow ZF$ **where**
 $\text{Lambda } A \ f == \text{Repl } A \ (\% x. \text{Opair } x \ (f \ x))$

lemma *Lambda-app*: $\text{Elem } x \ A \Longrightarrow (\text{Lambda } A \ f) \ ' \ x = f \ x$
 $\langle \text{proof} \rangle$

lemma *isFun-Lambda*: $\text{isFun } (\text{Lambda } A \ f)$
 $\langle \text{proof} \rangle$

lemma *domain-Lambda*: $\text{Domain } (\text{Lambda } A \ f) = A$
 $\langle \text{proof} \rangle$

lemma *Lambda-ext*: $(\text{Lambda } s \ f = \text{Lambda } t \ g) = (s = t \wedge (\forall x. \text{Elem } x \ s \longrightarrow f \ x = g \ x))$
 $\langle \text{proof} \rangle$

definition *PFun* :: $ZF \Rightarrow ZF \Rightarrow ZF$ **where**
 $\text{PFun } A \ B == \text{Sep } (\text{Power } (\text{CartProd } A \ B)) \ \text{isFun}$

definition *Fun* :: $ZF \Rightarrow ZF \Rightarrow ZF$ **where**
 $\text{Fun } A \ B == \text{Sep } (\text{PFun } A \ B) \ (\lambda f. \text{Domain } f = A)$

lemma *Fun-Range*: $\text{Elem } f \ (\text{Fun } U \ V) \Longrightarrow \text{subset } (\text{Range } f) \ V$
 $\langle \text{proof} \rangle$

lemma *Elem-Elem-PFun*: $\text{Elem } F \ (\text{PFun } U \ V) \Longrightarrow \text{Elem } p \ F \Longrightarrow \text{isOpair } p \ \& \ \text{Elem } (\text{Fst } p) \ U \ \& \ \text{Elem } (\text{Snd } p) \ V$
 $\langle \text{proof} \rangle$

lemma *Fun-implies-PFun[simp]*: $\text{Elem } f \ (\text{Fun } U \ V) \Longrightarrow \text{Elem } f \ (\text{PFun } U \ V)$
 $\langle \text{proof} \rangle$

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

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

lemma *Fun-total*: $\llbracket Elem\ F\ (Fun\ U\ V); Elem\ a\ U \rrbracket \implies \exists x. Elem\ (Opair\ a\ x)\ F$
 ⟨proof⟩

lemma *unique-fun-value*: $\llbracket isFun\ f; Elem\ x\ (Domain\ f) \rrbracket \implies \exists! y. Elem\ (Opair\ x\ y)\ f$
 ⟨proof⟩

lemma *fun-value-in-range*: $\llbracket isFun\ f; Elem\ x\ (Domain\ f) \rrbracket \implies Elem\ (f'x)\ (Range\ f)$
 ⟨proof⟩

lemma *fun-range-witness*: $\llbracket isFun\ f; Elem\ y\ (Range\ f) \rrbracket \implies \exists x. Elem\ x\ (Domain\ f) \ \&\ f'x = y$
 ⟨proof⟩

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

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

definition *is-Elem-of* :: $(ZF * ZF)\ set$ **where**
is-Elem-of == $\{ (a,b) \mid a\ b. Elem\ a\ b \}$

lemma *cond-wf-Elem*:

assumes *hyp*s: $\forall x. (\forall y. Elem\ y\ x \longrightarrow Elem\ y\ U \longrightarrow P\ y) \longrightarrow Elem\ x\ U \longrightarrow P\ x$
shows $P\ a$
 ⟨proof⟩

lemma *cond2-wf-Elem*:

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

primrec *nat2Nat* :: *nat* \Rightarrow *ZF* **where**
nat2Nat-0[*intro*]: *nat2Nat* 0 = *Empty*
| *nat2Nat-Suc*[*intro*]: *nat2Nat* (*Suc* n) = *SucNat* (*nat2Nat* n)

definition *Nat2nat* :: *ZF* \Rightarrow *nat* **where**
Nat2nat == *inv nat2Nat*

lemma *Elem-nat2Nat-inf*[*intro*]: *Elem* (*nat2Nat* n) *Inf*
<proof>

definition *Nat* :: *ZF*
where *Nat* == *Sep Inf* ($\lambda N. \exists n. \text{nat2Nat } n = N$)

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* \implies *Elem (SucNat N) Nat*
<proof>

lemma *no-infinite-Elem-down-chain*:
Not ($\exists f. \text{isFun } f \wedge \text{Domain } f = \text{Nat} \wedge (\forall N. \text{Elem } N \text{ Nat} \longrightarrow \text{Elem } (f'(\text{SucNat } N)) (f' N))$)
<proof>

lemma *Upair-nonEmpty*: *Upair a b* \neq *Empty*
<proof>

lemma *Singleton-nonEmpty*: *Singleton x* \neq *Empty*
<proof>

lemma *notsym-Elem*: *Not*(*Elem a b* & *Elem b a*)
<proof>

lemma *irreflexiv-Elem*: *Not*(*Elem a a*)
<proof>

lemma *antisym-Elem*: *Elem a b* \implies *Not (Elem b a)*
<proof>

primrec *NatInterval* :: *nat* \Rightarrow *nat* \Rightarrow *ZF* **where**
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*]: $\forall q. q \leq m \longrightarrow \text{Elem } (\text{nat2Nat } (n+q))$
(*NatInterval* n m)

$\langle proof \rangle$

lemma *NatInterval-not-Empty*: $NatInterval\ n\ m \neq Empty$
 $\langle proof \rangle$

lemma *increasing-nat2Nat[rule-format]*: $0 < n \longrightarrow Elem\ (nat2Nat\ (n - 1))$
 $(nat2Nat\ n)$
 $\langle proof \rangle$

lemma *represent-NatInterval[rule-format]*: $Elem\ x\ (NatInterval\ n\ m) \longrightarrow (\exists\ u.\ n \leq u \wedge u \leq n+m \wedge nat2Nat\ u = x)$
 $\langle proof \rangle$

lemma *inj-nat2Nat*: $inj\ nat2Nat$
 $\langle proof \rangle$

lemma *Nat2nat-nat2Nat[simp]*: $Nat2nat\ (nat2Nat\ n) = n$
 $\langle proof \rangle$

lemma *nat2Nat-Nat2nat[simp]*: $Elem\ n\ Nat \implies nat2Nat\ (Nat2nat\ n) = n$
 $\langle proof \rangle$

lemma *Nat2nat-SucNat*: $Elem\ N\ Nat \implies Nat2nat\ (SucNat\ N) = Suc\ (Nat2nat\ N)$
 $\langle proof \rangle$

lemma *Elem-Opair-exists*: $\exists\ z.\ Elem\ x\ z \ \&\ Elem\ y\ z \ \&\ Elem\ z\ (Opair\ x\ y)$
 $\langle proof \rangle$

lemma *UNIV-is-not-in-ZF*: $UNIV \neq explode\ R$
 $\langle proof \rangle$

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

lemma *wf SpecialR*
 $\langle proof \rangle$

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

lemma *Ext-Elem*: $Ext\ is-Elem-of = explode$
 $\langle proof \rangle$

lemma *Ext SpecialR Empty* $\neq explode\ z$
 $\langle proof \rangle$

definition *implode* :: $ZF \text{ set} \Rightarrow ZF$ **where**

implode == *inv explode*

lemma *inj-explode*: *inj explode*

<proof>

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

<proof>

definition *regular* :: $(ZF * ZF) \text{ set} \Rightarrow \text{bool}$ **where**

regular R == $\forall A. A \neq \text{Empty} \longrightarrow (\exists x. \text{Elem } x A \wedge (\forall y. (y, x) \in R \longrightarrow \text{Not } (\text{Elem } y A)))$

definition *set-like* :: $(ZF * ZF) \text{ set} \Rightarrow \text{bool}$ **where**

set-like R == $\forall y. \text{Ext } R y \in \text{range explode}$

definition *wfzf* :: $(ZF * ZF) \text{ set} \Rightarrow \text{bool}$ **where**

wfzf R == *regular R* \wedge *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>

definition *SeqSum* :: $(\text{nat} \Rightarrow ZF) \Rightarrow ZF$ **where**

SeqSum f == *Sum (Repl Nat (f o Nat2nat))*

lemma *SeqSum*: *Elem x (SeqSum f) = ($\exists n. \text{Elem } x (f n)$)*

<proof>

definition *Ext-ZF* :: $(ZF * ZF) \text{ set} \Rightarrow ZF \Rightarrow ZF$ **where**

Ext-ZF R s == *implode (Ext R s)*

lemma *Elem-implode*: $A \in \text{range explode} \implies \text{Elem } x (\text{implode } A) = (x \in A)$

<proof>

lemma *Elem-Ext-ZF*: $\text{set-like } R \implies \text{Elem } x (\text{Ext-ZF } R s) = ((x, s) \in R)$

<proof>

primrec *Ext-ZF-n* :: $(ZF * ZF) \text{ set} \Rightarrow ZF \Rightarrow \text{nat} \Rightarrow ZF$ **where**

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))*

definition *Ext-ZF-hull* :: $(ZF * ZF) \text{ set} \Rightarrow ZF \Rightarrow ZF$ **where**

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

lemma *Elem-Ext-ZF-hull*:

assumes *set-like-R*: *set-like* R

shows $Elem\ x\ (Ext-ZF-hull\ R\ S) = (\exists\ n.\ Elem\ x\ (Ext-ZF-n\ R\ S\ n))$

<proof>

lemma *Elem-Elem-Ext-ZF-hull*:

assumes *set-like-R*: *set-like* R

and *x-hull*: $Elem\ x\ (Ext-ZF-hull\ R\ S)$

and *y-R-x*: $(y, x) \in R$

shows $Elem\ y\ (Ext-ZF-hull\ R\ S)$

<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-Elem-of*

<proof>

lemma *in-Ext-RTrans-implies-Elem-Ext-ZF-hull*:

set-like $R \Longrightarrow x \in (Ext\ (R^+)\ s) \Longrightarrow Elem\ x\ (Ext-ZF-hull\ R\ s)$

<proof>

lemma *implodeable-Ext-trancl*: *set-like* $R \Longrightarrow set-like\ (R^+)$

<proof>

lemma *Elem-Ext-ZF-hull-implies-in-Ext-RTrans*[*rule-format*]:

set-like $R \Longrightarrow \forall\ x.\ Elem\ x\ (Ext-ZF-n\ R\ s\ n) \longrightarrow x \in (Ext\ (R^+)\ s)$

<proof>

lemma *set-like* $R \Longrightarrow Ext-ZF\ (R^+)\ s = Ext-ZF-hull\ R\ s$

<proof>

lemma *wf-implies-regular*: $wf\ R \Longrightarrow regular\ R$

<proof>

lemma *wf-eq-wfzf*: $(wf\ R \wedge set-like\ R) = wfzf\ R$

<proof>

lemma *wfzf-trancl*: $wfzf\ R \Longrightarrow wfzf\ (R^+)$

<proof>

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

```

    <proof>

lemma set-like-subset: set-like  $R \implies S \subseteq R \implies \text{set-like } S$ 
    <proof>

lemma wfzf-subset: wfzf  $S \implies R \subseteq S \implies \text{wfzf } R$ 
    <proof>

end

theory Zet
imports HOLZF
begin

definition zet =  $\{A :: 'a \text{ set} \mid A \text{ f } z. \text{ inj-on } f \ A \wedge f \ ' \ A \subseteq \text{explode } z\}$ 

typedef  $'a \text{ zet} = \text{zet} :: 'a \text{ set set}$ 
    <proof>

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

lemma zet-ext-eq:  $(A = B) = (\forall x. \text{zin } x \ A = \text{zin } x \ B)$ 
    <proof>

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

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

lemma image-zet-rep:  $A \in \text{zet} \implies \exists z. g \ ' \ A = \text{explode } z$ 
    <proof>

lemma zet-image-mem:
    assumes Azet:  $A \in \text{zet}$ 
    shows  $g \ ' \ A \in \text{zet}$ 
    <proof>

lemma Rep-zimage-eq:  $\text{Rep-zet } (\text{zimage } f \ A) = \text{image } f \ (\text{Rep-zet } A)$ 
    <proof>

lemma zimage-iff:  $\text{zin } y \ (\text{zimage } f \ A) = (\exists x. \text{zin } x \ A \wedge y = f \ x)$ 
    <proof>

definition zimplode ::  $ZF \ \text{zet} \Rightarrow ZF$  where
     $\text{zimplode } A == \text{implode } (\text{Rep-zet } A)$ 

```

definition $zexplode :: ZF \Rightarrow ZF\ zet$ **where**
 $zexplode\ z == Abs\text{-}zet\ (explode\ z)$

lemma $Rep\text{-}zet\text{-}eq\text{-}explode: \exists z. Rep\text{-}zet\ A = explode\ z$
 $\langle proof \rangle$

lemma $zexplode\text{-}zimplode: zexplode\ (zimplode\ A) = A$
 $\langle proof \rangle$

lemma $explode\text{-}mem\text{-}zet: explode\ z \in zet$
 $\langle proof \rangle$

lemma $zimplode\text{-}zexplode: zimplode\ (zexplode\ z) = z$
 $\langle proof \rangle$

lemma $zin\text{-}zexplode\text{-}eq: zin\ x\ (zexplode\ A) = Elem\ x\ A$
 $\langle proof \rangle$

lemma $comp\text{-}zimage\text{-}eq: zimage\ g\ (zimage\ f\ A) = zimage\ (g\ o\ f)\ A$
 $\langle proof \rangle$

definition $zunion :: 'a\ zet \Rightarrow 'a\ zet \Rightarrow 'a\ zet$ **where**
 $zunion\ a\ b \equiv Abs\text{-}zet\ ((Rep\text{-}zet\ a) \cup (Rep\text{-}zet\ b))$

definition $zsubset :: 'a\ zet \Rightarrow 'a\ zet \Rightarrow bool$ **where**
 $zsubset\ a\ b \equiv \forall x. zin\ x\ a \longrightarrow zin\ x\ b$

lemma $explode\text{-}union: explode\ (union\ a\ b) = (explode\ a) \cup (explode\ b)$
 $\langle proof \rangle$

lemma $Rep\text{-}zet\text{-}zunion: Rep\text{-}zet\ (zunion\ a\ b) = (Rep\text{-}zet\ a) \cup (Rep\text{-}zet\ b)$
 $\langle proof \rangle$

lemma $zunion: zin\ x\ (zunion\ a\ b) = ((zin\ x\ a) \vee (zin\ x\ b))$
 $\langle proof \rangle$

lemma $zimage\text{-}zexplode\text{-}eq: zimage\ f\ (zexplode\ z) = zexplode\ (Repl\ z\ f)$
 $\langle proof \rangle$

lemma $range\text{-}explode\text{-}eq\text{-}zet: range\ explode = zet$
 $\langle proof \rangle$

lemma $Elem\text{-}zimplode: (Elem\ x\ (zimplode\ z)) = (zin\ x\ z)$
 $\langle proof \rangle$

definition $zempty :: 'a\ zet$ **where**
 $zempty \equiv Abs\text{-}zet\ \{\}$

lemma $zempty[simp]: \neg (zin\ x\ zempty)$

$\langle \text{proof} \rangle$

lemma *zimage-zempty[simp]*: $\text{zimage } f \text{ zempty} = \text{zempty}$
 $\langle \text{proof} \rangle$

lemma *zunion-zempty-left[simp]*: $\text{zunion zempty } a = a$
 $\langle \text{proof} \rangle$

lemma *zunion-zempty-right[simp]*: $\text{zunion } a \text{ zempty} = a$
 $\langle \text{proof} \rangle$

lemma *zimage-id[simp]*: $\text{zimage id } A = A$
 $\langle \text{proof} \rangle$

lemma *zimage-cong[fundef-cong]*: $\llbracket M = N; !! x. \text{zin } x \ N \implies f \ x = g \ x \rrbracket \implies$
 $\text{zimage } f \ M = \text{zimage } g \ N$
 $\langle \text{proof} \rangle$

end

theory *LProd*
imports *HOL-Library.Multiset*
begin

inductive-set

$\text{lprod} :: ('a * 'a) \text{ set} \Rightarrow ('a \text{ list} * 'a \text{ list}) \text{ set}$
for $R :: ('a * 'a) \text{ set}$

where

$\text{lprod-single[intro!]}: (a, b) \in R \implies ([a], [b]) \in \text{lprod } R$
 $\mid \text{lprod-list[intro!]}: (ah@at, bh@bt) \in \text{lprod } R \implies (a, b) \in R \vee a = b \implies (ah@a\#at,$
 $bh@b\#bt) \in \text{lprod } R$

lemma $(as, bs) \in \text{lprod } R \implies \text{length } as = \text{length } bs$
 $\langle \text{proof} \rangle$

lemma $(as, bs) \in \text{lprod } R \implies 1 \leq \text{length } as \wedge 1 \leq \text{length } bs$
 $\langle \text{proof} \rangle$

lemma *lprod-subset-elem*: $(as, bs) \in \text{lprod } S \implies S \subseteq R \implies (as, bs) \in \text{lprod } R$
 $\langle \text{proof} \rangle$

lemma *lprod-subset*: $S \subseteq R \implies \text{lprod } S \subseteq \text{lprod } R$
 $\langle \text{proof} \rangle$

lemma *lprod-implies-mult*: $(as, bs) \in \text{lprod } R \implies \text{trans } R \implies (\text{mset } as, \text{mset } bs)$
 $\in \text{mult } R$
 $\langle \text{proof} \rangle$

lemma *wf-lprod*[*simp,intro*]:

assumes *wf-R*: *wf R*

shows *wf (lprod R)*

<proof>

definition *gprod-2-2* :: $('a * 'a) \text{ set} \Rightarrow (('a * 'a) * ('a * 'a)) \text{ set}$ **where**

gprod-2-2 R $\equiv \{ ((a,b), (c,d)) . (a = c \wedge (b,d) \in R) \vee (b = d \wedge (a,c) \in R) \}$

definition *gprod-2-1* :: $('a * 'a) \text{ set} \Rightarrow (('a * 'a) * ('a * 'a)) \text{ set}$ **where**

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

lemma *lprod-2-3*: $(a, b) \in R \implies ([a, c], [b, c]) \in \text{lprod } R$

<proof>

lemma *lprod-2-4*: $(a, b) \in R \implies ([c, a], [c, b]) \in \text{lprod } R$

<proof>

lemma *lprod-2-1*: $(a, b) \in R \implies ([c, a], [b, c]) \in \text{lprod } R$

<proof>

lemma *lprod-2-2*: $(a, b) \in R \implies ([a, c], [c, b]) \in \text{lprod } R$

<proof>

lemma [*simp, intro*]:

assumes *wfR*: *wf R* **shows** *wf (gprod-2-1 R)*

<proof>

lemma [*simp, intro*]:

assumes *wfR*: *wf R* **shows** *wf (gprod-2-2 R)*

<proof>

lemma *lprod-3-1*: **assumes** $(x', x) \in R$ **shows** $([y, z, x'], [x, y, z]) \in \text{lprod } R$

<proof>

lemma *lprod-3-2*: **assumes** $(z', z) \in R$ **shows** $([z', x, y], [x, y, z]) \in \text{lprod } R$

<proof>

lemma *lprod-3-3*: **assumes** *xr*: $(xr, x) \in R$ **shows** $([xr, y, z], [x, y, z]) \in \text{lprod } R$

<proof>

lemma *lprod-3-4*: **assumes** *yr*: $(yr, y) \in R$ **shows** $([x, yr, z], [x, y, z]) \in \text{lprod } R$

<proof>

lemma *lprod-3-5*: **assumes** *zr*: $(zr, z) \in R$ **shows** $([x, y, zr], [x, y, z]) \in \text{lprod } R$

<proof>

lemma *lprod-3-6*: **assumes** *y'*: $(y', y) \in R$ **shows** $([x, z, y'], [x, y, z]) \in \text{lprod } R$

<proof>

lemma *lprod-3-7*: **assumes** $z': (z', z) \in R$ **shows** $([x, z', y], [x, y, z]) \in \text{lprod } R$
 <proof>

definition *perm* :: $('a \Rightarrow 'a) \Rightarrow 'a \text{ set} \Rightarrow \text{bool}$ **where**
 $\text{perm } f \ A \equiv \text{inj-on } f \ A \wedge f ` A = A$

lemma $((as, bs) \in \text{lprod } R) =$
 $(\exists f. \text{perm } f \ \{0 ..< (\text{length } as)\} \wedge$
 $(\forall j. j < \text{length } as \longrightarrow ((\text{nth } as \ j, \text{nth } bs \ (f \ j)) \in R \vee (\text{nth } as \ j = \text{nth } bs \ (f \ j))))$
 \wedge
 $(\exists i. i < \text{length } as \wedge (\text{nth } as \ i, \text{nth } bs \ (f \ i)) \in R))$
 <proof>

lemma $\text{trans } R \Longrightarrow (ah@a\#at, bh@b\#bt) \in \text{lprod } R \Longrightarrow (b, a) \in R \vee a = b \Longrightarrow$
 $(ah@at, bh@bt) \in \text{lprod } R$
 <proof>

end

theory *MainZF*
imports *Zet LProd*
begin

end

theory *Games*
imports *MainZF*
begin

definition *fixgames* :: $ZF \text{ set} \Rightarrow ZF \text{ set}$ **where**
 $\text{fixgames } A \equiv \{ \text{Opair } l \ r \mid l \ r. \text{explode } l \subseteq A \ \& \ \text{explode } r \subseteq A \}$

definition *games-lfp* :: $ZF \text{ set}$ **where**
 $\text{games-lfp} \equiv \text{lfp } \text{fixgames}$

definition *games-gfp* :: $ZF \text{ set}$ **where**
 $\text{games-gfp} \equiv \text{gfp } \text{fixgames}$

lemma *mono-fixgames*: $\text{mono } (\text{fixgames})$
 <proof>

lemma *games-lfp-unfold*: $\text{games-lfp} = \text{fixgames } \text{games-lfp}$
 <proof>

lemma *games-gfp-unfold*: $\text{games-gfp} = \text{fixgames } \text{games-gfp}$
 <proof>

lemma *games-lfp-nonempty*: $\text{Opair Empty Empty} \in \text{games-lfp}$
 $\langle \text{proof} \rangle$

definition *left-option* :: $ZF \Rightarrow ZF \Rightarrow \text{bool}$ **where**
 $\text{left-option } g \text{ opt} \equiv (\text{Elem opt } (\text{Fst } g))$

definition *right-option* :: $ZF \Rightarrow ZF \Rightarrow \text{bool}$ **where**
 $\text{right-option } g \text{ opt} \equiv (\text{Elem opt } (\text{Snd } g))$

definition *is-option-of* :: $(ZF * ZF) \text{ set}$ **where**
 $\text{is-option-of} \equiv \{ (opt, g) \mid \text{opt } g. g \in \text{games-gfp} \wedge (\text{left-option } g \text{ opt} \vee \text{right-option } g \text{ opt}) \}$

lemma *games-lfp-subset-gfp*: $\text{games-lfp} \subseteq \text{games-gfp}$
 $\langle \text{proof} \rangle$

lemma *games-option-stable*:
assumes *fixgames*: $\text{games} = \text{fixgames games}$
and *g*: $g \in \text{games}$
and *opt*: $\text{left-option } g \text{ opt} \vee \text{right-option } g \text{ opt}$
shows $\text{opt} \in \text{games}$
 $\langle \text{proof} \rangle$

lemma *option2elem*: $(opt, g) \in \text{is-option-of} \implies \exists u v. \text{Elem opt } u \wedge \text{Elem } u v \wedge \text{Elem } v g$
 $\langle \text{proof} \rangle$

lemma *is-option-of-subset-is-Elem-of*: $\text{is-option-of} \subseteq (\text{is-Elem-of}^+)$
 $\langle \text{proof} \rangle$

lemma *wfzf-is-option-of*: wfzf is-option-of
 $\langle \text{proof} \rangle$

lemma *games-gfp-imp-lfp*: $g \in \text{games-gfp} \longrightarrow g \in \text{games-lfp}$
 $\langle \text{proof} \rangle$

theorem *games-lfp-eq-gfp*: $\text{games-lfp} = \text{games-gfp}$
 $\langle \text{proof} \rangle$

theorem *unique-games*: $(g = \text{fixgames } g) = (g = \text{games-lfp})$
 $\langle \text{proof} \rangle$

lemma *games-lfp-option-stable*:
assumes *g*: $g \in \text{games-lfp}$
and *opt*: $\text{left-option } g \text{ opt} \vee \text{right-option } g \text{ opt}$
shows $\text{opt} \in \text{games-lfp}$
 $\langle \text{proof} \rangle$

lemma *is-option-of-imp-games*:

assumes *hyp*: $(opt, g) \in is-option-of$
shows $opt \in games-lfp \wedge g \in games-lfp$
 $\langle proof \rangle$

lemma *games-lfp-represent*: $x \in games-lfp \implies \exists l r. x = Opair\ l\ r$
 $\langle proof \rangle$

definition *game* = *games-lfp*

typedef *game* = *game*
 $\langle proof \rangle$

definition *left-options* :: *game* \Rightarrow *game zet* **where**
left-options *g* $\equiv zimage\ Abs-game\ (zexplode\ (Fst\ (Rep-game\ g)))$

definition *right-options* :: *game* \Rightarrow *game zet* **where**
right-options *g* $\equiv zimage\ Abs-game\ (zexplode\ (Snd\ (Rep-game\ g)))$

definition *options* :: *game* \Rightarrow *game zet* **where**
options *g* $\equiv zunion\ (left-options\ g)\ (right-options\ g)$

definition *Game* :: *game zet* \Rightarrow *game zet* \Rightarrow *game* **where**
Game *L* *R* $\equiv Abs-game\ (Opair\ (zimplode\ (zimage\ Rep-game\ L))\ (zimplode\ (zimage\ Rep-game\ R)))$

lemma *Repl-Rep-game-Abs-game*: $\forall e. Elem\ e\ z \longrightarrow e \in games-lfp \implies Repl\ z\ (Rep-game\ o\ Abs-game) = z$
 $\langle proof \rangle$

lemma *game-split*: $g = Game\ (left-options\ g)\ (right-options\ g)$
 $\langle proof \rangle$

lemma *Opair-in-games-lfp*:
assumes *l*: $explode\ l \subseteq games-lfp$
and *r*: $explode\ r \subseteq games-lfp$
shows $Opair\ l\ r \in games-lfp$
 $\langle proof \rangle$

lemma *left-options[simp]*: $left-options\ (Game\ l\ r) = l$
 $\langle proof \rangle$

lemma *right-options[simp]*: $right-options\ (Game\ l\ r) = r$
 $\langle proof \rangle$

lemma *Game-ext*: $(Game\ l1\ r1 = Game\ l2\ r2) = ((l1 = l2) \wedge (r1 = r2))$
 $\langle proof \rangle$

definition *option-of* :: (*game* * *game*) *set* **where**
option-of $\equiv image\ (\lambda (option, g). (Abs-game\ option, Abs-game\ g))\ is-option-of$

lemma *option-to-is-option-of*: $((option, g) \in option-of) = ((Rep-game\ option, Rep-game\ g) \in is-option-of)$
 $\langle proof \rangle$

lemma *wf-is-option-of*: *wf is-option-of*
 $\langle proof \rangle$

lemma *wf-option-of*[*simp, intro*]: *wf option-of*
 $\langle proof \rangle$

lemma *right-option-is-option*[*simp, intro*]: $zin\ x\ (right-options\ g) \implies zin\ x\ (options\ g)$
 $\langle proof \rangle$

lemma *left-option-is-option*[*simp, intro*]: $zin\ x\ (left-options\ g) \implies zin\ x\ (options\ g)$
 $\langle proof \rangle$

lemma *zin-options*[*simp, intro*]: $zin\ x\ (options\ g) \implies (x, g) \in option-of$
 $\langle proof \rangle$

function

neg-game :: *game* \Rightarrow *game*

where

[*simp del*]: *neg-game* *g* = *Game* (*zimage* *neg-game* (*right-options* *g*)) (*zimage* *neg-game* (*left-options* *g*))
 $\langle proof \rangle$

termination $\langle proof \rangle$

lemma *neg-game* (*neg-game* *g*) = *g*
 $\langle proof \rangle$

function

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

where

[*simp del*]: *ge-game* (*G*, *H*) = $(\forall\ x.\ if\ zin\ x\ (right-options\ G)\ then\ ($
 $\quad if\ zin\ x\ (left-options\ H)\ then\ \neg\ (ge-game\ (H, x) \vee (ge-game$
 $\quad (x, G)))$
 $\quad else\ \neg\ (ge-game\ (H, x)))$
 $\quad else\ (if\ zin\ x\ (left-options\ H)\ then\ \neg\ (ge-game\ (x, G))\ else$
 $\quad True))$

$\langle proof \rangle$

termination $\langle proof \rangle$

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

lemma *ge-game-leftright-refl*[*rule-format*]:

$\forall y. (zin\ y\ (right-options\ x) \longrightarrow \neg\ ge-game\ (x, y)) \wedge (zin\ y\ (left-options\ x) \longrightarrow \neg\ (ge-game\ (y, x))) \wedge ge-game\ (x, x)$
 $\langle proof \rangle$

lemma *ge-game-refl*: *ge-game* (*x*,*x*) $\langle proof \rangle$

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

definition *eq-game* :: *game* \Rightarrow *game* \Rightarrow *bool* **where**

eq-game *G* *H* $\equiv ge-game\ (G, H) \wedge ge-game\ (H, G)$

lemma *eq-game-sym*: (*eq-game* *G* *H*) = (*eq-game* *H* *G*)
 $\langle proof \rangle$

lemma *eq-game-refl*: *eq-game* *G* *G*
 $\langle proof \rangle$

lemma *induct-game*: ($\bigwedge x. \forall y. (y, x) \in lprod\ option-of \longrightarrow P\ y \implies P\ x$) $\implies P\ a$
 $\langle proof \rangle$

lemma *ge-game-trans*:

assumes *ge-game* (*x*, *y*) *ge-game* (*y*, *z*)

shows *ge-game* (*x*, *z*)

$\langle proof \rangle$

lemma *eq-game-trans*: *eq-game* *a* *b* $\implies eq-game\ b\ c \implies eq-game\ a\ c$
 $\langle proof \rangle$

definition *zero-game* :: *game*

where *zero-game* $\equiv Game\ zempty\ zempty$

function

plus-game :: *game* \Rightarrow *game* \Rightarrow *game*

where

[*simp del*]: *plus-game* *G* *H* = *Game* (*zunion* (*zimage* ($\lambda g. plus-game\ g\ H$) (*left-options* *G*))

(*zimage* ($\lambda h. plus-game\ G\ h$) (*left-options* *H*)))
(*zunion* (*zimage* ($\lambda g. plus-game\ g\ H$) (*right-options* *G*))
(*zimage* ($\lambda h. plus-game\ G\ h$) (*right-options* *H*)))

$\langle proof \rangle$

termination $\langle proof \rangle$

lemma *plus-game-comm*: *plus-game* *G* *H* = *plus-game* *H* *G*

$\langle proof \rangle$

lemma *game-ext-eq*: (*G* = *H*) = (*left-options* *G* = *left-options* *H* \wedge *right-options*

$G = \text{right-options } H$
 $\langle \text{proof} \rangle$

lemma *left-zero-game[simp]*: $\text{left-options } (\text{zero-game}) = \text{zempty}$
 $\langle \text{proof} \rangle$

lemma *right-zero-game[simp]*: $\text{right-options } (\text{zero-game}) = \text{zempty}$
 $\langle \text{proof} \rangle$

lemma *plus-game-zero-right[simp]*: $\text{plus-game } G \text{ zero-game} = G$
 $\langle \text{proof} \rangle$

lemma *plus-game-zero-left*: $\text{plus-game } \text{zero-game } G = G$
 $\langle \text{proof} \rangle$

lemma *left-imp-options[simp]*: $\text{zin opt } (\text{left-options } g) \implies \text{zin opt } (\text{options } g)$
 $\langle \text{proof} \rangle$

lemma *right-imp-options[simp]*: $\text{zin opt } (\text{right-options } g) \implies \text{zin opt } (\text{options } g)$
 $\langle \text{proof} \rangle$

lemma *left-options-plus*:
 $\text{left-options } (\text{plus-game } u \ v) = \text{zunion } (\text{zimage } (\lambda g. \text{plus-game } g \ v) (\text{left-options } u)) (\text{zimage } (\lambda h. \text{plus-game } u \ h) (\text{left-options } v))$
 $\langle \text{proof} \rangle$

lemma *right-options-plus*:
 $\text{right-options } (\text{plus-game } u \ v) = \text{zunion } (\text{zimage } (\lambda g. \text{plus-game } g \ v) (\text{right-options } u)) (\text{zimage } (\lambda h. \text{plus-game } u \ h) (\text{right-options } v))$
 $\langle \text{proof} \rangle$

lemma *left-options-neg*: $\text{left-options } (\text{neg-game } u) = \text{zimage } \text{neg-game } (\text{right-options } u)$
 $\langle \text{proof} \rangle$

lemma *right-options-neg*: $\text{right-options } (\text{neg-game } u) = \text{zimage } \text{neg-game } (\text{left-options } u)$
 $\langle \text{proof} \rangle$

lemma *plus-game-assoc*: $\text{plus-game } (\text{plus-game } F \ G) \ H = \text{plus-game } F \ (\text{plus-game } G \ H)$
 $\langle \text{proof} \rangle$

lemma *neg-plus-game*: $\text{neg-game } (\text{plus-game } G \ H) = \text{plus-game } (\text{neg-game } G) \ (\text{neg-game } H)$
 $\langle \text{proof} \rangle$

lemma *eq-game-plus-inverse*: $\text{eq-game } (\text{plus-game } x \ (\text{neg-game } x)) \ \text{zero-game}$
 $\langle \text{proof} \rangle$

lemma *ge-plus-game-left*: $ge\text{-}game\ (y,z) = ge\text{-}game\ (plus\text{-}game\ x\ y,\ plus\text{-}game\ x\ z)$
 $\langle proof \rangle$

lemma *ge-plus-game-right*: $ge\text{-}game\ (y,z) = ge\text{-}game\ (plus\text{-}game\ y\ x,\ plus\text{-}game\ z\ x)$
 $\langle proof \rangle$

lemma *ge-neg-game*: $ge\text{-}game\ (neg\text{-}game\ x,\ neg\text{-}game\ y) = ge\text{-}game\ (y,\ x)$
 $\langle proof \rangle$

definition *eq-game-rel* :: $(game * game)$ set **where**
 $eq\text{-}game\text{-}rel \equiv \{ (p,\ q) \cdot eq\text{-}game\ p\ q \}$

definition $Pg = UNIV // eq\text{-}game\text{-}rel$

typedef $Pg = Pg$
 $\langle proof \rangle$

lemma *equiv-eq-game[simp]*: $equiv\ UNIV\ eq\text{-}game\text{-}rel$
 $\langle proof \rangle$

instantiation $Pg :: \{ord,\ zero,\ plus,\ minus,\ uminus\}$
begin

definition
 $Pg\text{-}zero\text{-}def: 0 = Abs\text{-}Pg\ (eq\text{-}game\text{-}rel\ \text{“}\ \{zero\text{-}game\}\text{”})$

definition
 $Pg\text{-}le\text{-}def: G \leq H \longleftrightarrow (\exists\ g\ h. g \in Rep\text{-}Pg\ G \wedge h \in Rep\text{-}Pg\ H \wedge ge\text{-}game\ (h,\ g))$

definition
 $Pg\text{-}less\text{-}def: G < H \longleftrightarrow G \leq H \wedge G \neq (H::Pg)$

definition
 $Pg\text{-}minus\text{-}def: -\ G = the\text{-}elem\ (\bigcup g \in Rep\text{-}Pg\ G. \{Abs\text{-}Pg\ (eq\text{-}game\text{-}rel\ \text{“}\ \{neg\text{-}game\ g\}\text{”})\})$

definition
 $Pg\text{-}plus\text{-}def: G + H = the\text{-}elem\ (\bigcup g \in Rep\text{-}Pg\ G. \bigcup h \in Rep\text{-}Pg\ H. \{Abs\text{-}Pg\ (eq\text{-}game\text{-}rel\ \text{“}\ \{plus\text{-}game\ g\ h\}\text{”})\})$

definition
 $Pg\text{-}diff\text{-}def: G - H = G + (-\ (H::Pg))$

instance $\langle proof \rangle$

end

lemma *Rep-Abs-eq-Pg[simp]*: $\text{Rep-Pg } (\text{Abs-Pg } (\text{eq-game-rel } \{g\})) = \text{eq-game-rel } \{g\}$
 <proof>

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

lemma *char-Pg-eq[simp]*: $(\text{Abs-Pg } (\text{eq-game-rel } \{g\})) = \text{Abs-Pg } (\text{eq-game-rel } \{h\}) = (\text{eq-game } g \ h)$
 <proof>

lemma *char-Pg-plus[simp]*: $\text{Abs-Pg } (\text{eq-game-rel } \{g\}) + \text{Abs-Pg } (\text{eq-game-rel } \{h\}) = \text{Abs-Pg } (\text{eq-game-rel } \{\text{plus-game } g \ h\})$
 <proof>

lemma *char-Pg-minus[simp]*: $-\text{Abs-Pg } (\text{eq-game-rel } \{g\}) = \text{Abs-Pg } (\text{eq-game-rel } \{\text{neg-game } g\})$
 <proof>

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

instance *Pg :: ordered-ab-group-add*
 <proof>

end