

ZF

Steven Obua

March 13, 2025

```
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: $Elem\ Empty\ Inf \wedge (\forall x. Elem\ x\ Inf \longrightarrow Elem\ (SucNat\ x)\ Inf)$

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

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

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

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

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

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

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

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

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

lemma *Upair-fsteg*: $(Upair\ a\ b = Upair\ a\ c) = ((a = b \ \&\ a = c) \ | \ (b = c))$
{*proof*}

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

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

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

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

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

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

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*: $[[Elem\ a\ A; explode\ A \subseteq B]] \Longrightarrow a \in B$
<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>

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

<proof>

theorem *Range*: $Elem\ y\ (Range\ f) = (\exists x. Elem\ (Opair\ x\ y)\ f)$
<proof>

theorem *union*: $Elem\ x\ (union\ A\ B) = (Elem\ x\ A\ |\ Elem\ x\ B)$
<proof>

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

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

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

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

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*: $(Lambda\ s\ f = Lambda\ t\ g) = (s = t \wedge (\forall x. Elem\ x\ s \longrightarrow f\ x = g\ x))$
<proof>

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

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

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

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

<proof>

lemma *NatInterval-not-Empty*: $\text{NatInterval } n \ m \neq \text{Empty}$
<proof>

lemma *increasing-nat2Nat*[rule-format]: $0 < n \longrightarrow \text{Elem } (\text{nat2Nat } (n - 1))$
 $(\text{nat2Nat } n)$
<proof>

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

lemma *inj-nat2Nat*: $\text{inj } \text{nat2Nat}$
<proof>

lemma *Nat2nat-nat2Nat*[simp]: $\text{Nat2nat } (\text{nat2Nat } n) = n$
<proof>

lemma *nat2Nat-Nat2nat*[simp]: $\text{Elem } n \ \text{Nat} \implies \text{nat2Nat } (\text{Nat2nat } n) = n$
<proof>

lemma *Nat2nat-SucNat*: $\text{Elem } N \ \text{Nat} \implies \text{Nat2nat } (\text{SucNat } N) = \text{Suc } (\text{Nat2nat } N)$
<proof>

lemma *Elem-Opair-exists*: $\exists z. \text{Elem } x \ z \ \& \ \text{Elem } y \ z \ \& \ \text{Elem } z \ (\text{Opair } x \ y)$
<proof>

lemma *UNIV-is-not-in-ZF*: $\text{UNIV} \neq \text{explode } R$
<proof>

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

lemma *wf SpecialR*
<proof>

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

lemma *Ext-Elem*: $\text{Ext is-Elem-of} = \text{explode}$
<proof>

lemma *Ext SpecialR Empty* $\neq \text{explode } z$
<proof>

definition *implode* :: *ZF 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*) *set* \Rightarrow *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*) *set* \Rightarrow *bool* **where**

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

definition *wfzf* :: (*ZF * ZF*) *set* \Rightarrow *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* :: (*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*) *set* \Rightarrow *ZF* \Rightarrow *ZF* **where**

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

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

<proof>

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

<proof>

primrec *Ext-ZF-n* :: (*ZF * ZF*) *set* \Rightarrow *ZF* \Rightarrow *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*) *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 *hyp*s: $wfzf\ R\ C \neq \{\}$

shows $\exists\ x.\ x \in C \wedge (\forall\ y.\ (y, x) \in R \longrightarrow y \notin C)$

<proof>

lemma *wfzf-implies-wf*: $wfzf\ R \implies wf\ R$

<proof>

lemma *wf-is-Elem-of*: *wf is-Elem-of*

<proof>

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

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

<proof>

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

<proof>

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

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

<proof>

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

<proof>

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

<proof>

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

<proof>

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

<proof>

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

<proof>

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

lemma *wfzf-subset*: $wfzf\ S \implies R \subseteq S \implies wfzf\ R$
<proof>

end

theory *Zet*
imports *HOLZF*
begin

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

typedef *'a zet* = *zet* :: *'a set set*
<proof>

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

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

definition *zimage* :: (*'a* \Rightarrow *'b*) \Rightarrow *'a zet* \Rightarrow *'b zet* **where**
zimage f A == *Abs-zet (image f (Rep-zet A))*

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

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

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

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

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

definition *zimplode* :: *ZF zet* \Rightarrow *ZF* **where**
zimplode A == *implode (Rep-zet A)*

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

<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[fundef-cong]*: $\llbracket M = N; \forall x. zin\ x\ N \implies f\ x = g\ x \rrbracket \implies$
 $zimage\ f\ M = zimage\ g\ N$
<proof>

end

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

inductive-set

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

where

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

lemma $(as, bs) \in lprod\ R \implies length\ as = length\ bs$
<proof>

lemma $(as, bs) \in lprod\ R \implies 1 \leq length\ as \wedge 1 \leq length\ bs$
<proof>

lemma *lprod-subset-elem*: $(as, bs) \in lprod\ S \implies S \subseteq R \implies (as, bs) \in lprod\ R$
<proof>

lemma *lprod-subset*: $S \subseteq R \implies lprod\ S \subseteq lprod\ R$
<proof>

lemma *lprod-implies-mult*: $(as, bs) \in lprod\ R \implies trans\ R \implies (mset\ as, mset\ bs)$
 $\in mult\ R$
<proof>

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

assumes *wf-R*: *wf R*

shows *wf (lprod R)*

<proof>

definition *gprod-2-2* :: $('a * 'a)$ *set* \Rightarrow $(('a * 'a) * ('a * 'a))$ *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* :: $('a * 'a)$ *set* \Rightarrow $(('a * 'a) * ('a * 'a))$ *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*: $(a, b) \in R \Longrightarrow ([a, c], [b, c]) \in$ *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 [*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** $([y, z, x'], [x, y, z]) \in$ *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** $([x, z', y], [x, y, z]) \in \text{lprod } R$
 ⟨*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*: $Opair\ Empty\ Empty \in games-lfp$
<proof>

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

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

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

lemma *games-lfp-subset-gfp*: $games-lfp \subseteq games-gfp$
<proof>

lemma *games-option-stable*:
assumes *fixgames*: $games = fixgames\ games$
and *g*: $g \in games$
and *opt*: $left-option\ g\ opt \vee right-option\ g\ opt$
shows $opt \in games$
<proof>

lemma *option2elem*: $(opt, g) \in is-option-of \implies \exists u\ v.\ Elem\ opt\ u \wedge Elem\ u\ v \wedge Elem\ v\ g$
<proof>

lemma *is-option-of-subset-is-Elem-of*: $is-option-of \subseteq (is-Elem-of^+)$
<proof>

lemma *wfzf-is-option-of*: $wfzf\ is-option-of$
<proof>

lemma *games-gfp-imp-lfp*: $g \in games-gfp \longrightarrow g \in games-lfp$
<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*: $g \in games-lfp$
and *opt*: $left-option\ g\ opt \vee right-option\ g\ opt$
shows $opt \in games-lfp$
<proof>

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

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

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

definition *game* = *games-lfp*

typedef *game* = *game*
 ⟨*proof*⟩

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

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

lemma *Opair-in-games-lfp*:
assumes *l*: $explode\ l \subseteq games-lfp$
and *r*: $explode\ r \subseteq games-lfp$
shows $Opair\ l\ r \in games-lfp$
 ⟨*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*: $(Game\ l1\ r1 = Game\ l2\ r2) = ((l1 = l2) \wedge (r1 = r2))$
 ⟨*proof*⟩

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*: $((option, g) \in option-of) = ((Rep-game\ option, Rep-game\ g) \in is-option-of)$
 ⟨proof⟩

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

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

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

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

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

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*))
 ⟨proof⟩

termination ⟨proof⟩

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

function

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

where

[*simp del*]: *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))

⟨proof⟩

termination ⟨proof⟩

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*)))
 ⟨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>

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*)
<proof>

lemma *eq-game-refl*: *eq-game* *G G*
<proof>

lemma *induct-game*: ($\bigwedge x. \forall y. (y,\ x) \in lprod\ option\ of \longrightarrow P\ y \Longrightarrow P\ x$) $\Longrightarrow P\ a$
<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* $\Longrightarrow eq\ game\ b\ c \Longrightarrow eq\ game\ a\ c$
<proof>

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*)))

<proof>

termination *<proof>*

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* \wedge *right-options*

$G = \text{right-options } H$
(proof)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

typedef $Pg = Pg$
 ⟨*proof*⟩

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

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

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

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

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

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

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

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

instance ⟨*proof*⟩

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