Generalizations in Isabelle/ZF

Victor Porton

December 24, 2010

Contents

1 Link to informal document

This Isabelle/ZF theory is based on the ideas of article "Generalization in ZF".

2 Some addons to Isabelle/ZF

theory *ZF-Addons*
imports *Main-ZF*
begin

We need to add some additional lemmas to Isabelle/ZF before proceed.

lemma *inj-inj-range*: $f \in inj(A,B) \Longrightarrow f \in inj(A,range(f))$
apply (*unfold inj-def*)
apply (*auto simp add*: *Pi-iff function-def*)
done

lemma *inj-bij-range*: $f \in inj(A,B) \Longrightarrow f \in bij(A,range(f))$
apply (*auto simp add*: *bij-def inj-inj-range*)
apply (*blast intro*: *inj-is-fun fun-is-surj elim*:)

done

lemma *range-subset*: f: $A \to B \Longrightarrow range(f) \subseteq B$
proof −
  assume f: $A \to B$
  hence $f \in Pow(A \times B)$ **by** (*simp add*: *Pi-def*)
  hence $f \subseteq A \times B$ **by** *auto*
  moreover
  hence $range(A \times B) \subseteq B$ **by** *auto*
  ultimately show $range(f) \subseteq B$ **by** *auto*
qed

lemma *lam-is-fun*: $f = (lam\ x{:}d.\ b(x)) \Longrightarrow f$: $d \to range(f)$
proof −
  assume *eq*: $f = (lam\ x{:}d.\ b(x))$
  have *dom*: $domain(f) = d$ **using** *domain-lam* **by** (*auto simp add*: *eq*)
  with *eq* **have** $function(f)$ **using** *function-lam* **by** *auto*
  moreover
  with *eq* **have** $relation(f)$ **using** *relation-lam* **by** *auto*
  ultimately have f: $domain(f) \to range(f)$ **by** (*rule function-imp-Pi*)
  with *dom* **show** *?thesis* **by** *auto*
qed

lemma *comp-fun2*:
  $[\![\ g$: $A \to B1;\ f$: $B0 \to C;\ B1 \subseteq B0\ ]\!] \Longrightarrow (f\ O\ g) : A \to C$
proof −
  assume g: $A \to B1$ f: $B0 \to C$ $B1 \subseteq B0$
  with ‹g: $A \to B1$› **have** g: $A \to B0$ **by** *auto* (*rule fun-weaken-type*)
  from ‹g: $A \to B0$› ‹f: $B0 \to C$› **show** $(f\ O\ g) : A \to C$ **using** *comp-fun* **by** *auto*
qed

This lemma superceedes the lemma *comp-eq-id-iff* in Isabelle/ZF:

lemma *comp-eq-id-iff1*:
  $[\![\ g$: $B \to A;\ f$: $A \to C\ ]\!] \Longrightarrow (\forall\, y \in B.\ f`(g`y) = y) \longleftrightarrow f\ O\ g = id(B)$
proof −
  assume g: $B \to A$ f: $A \to C$
  hence $f\ O\ g$: $B \to C$ **by** (*rule comp-fun*)
  moreover
  have $id(B)$: $B \to B$ **by** (*rule id-type*)
  ultimately have m: $(\forall\, y \in B.\ (f\ O\ g)`y = id(B)`y) \longleftrightarrow f\ O\ g = id(B)$ **by** (*rule fun-extension-iff*)
  from ‹g: $B \to A$› **have** c [*simp*]: $\forall\, y \in B.\ (f\ O\ g)`y = f`(g`y)$ **by** *auto*
  have i [*simp*]: $\forall\, y \in B.\ id(B)`y = y$ **by** *auto*
  from m **show** $(\forall\, y \in B.\ f`(g`y) = y) \longleftrightarrow f\ O\ g = id(B)$ **by** *simp*
qed

lemma *right-comp-id-any*: $r\ O\ id(C) = restrict(r, C)$
unfolding *restrict-def* **by** *auto*

end

3 A theory of generalization

theory *Generalization*
imports *Main-ZF ZF-Addons*
begin

This theory formalizes the intuitive notion of *generalization*.

3.1 Generalization situation

locale *generalization-situation* =
fixes *small*::*i* **and** *big*::*i*
fixes *embed*::*i*
assumes *embed-inj*: *embed*∈*inj*(*small*, *big*)
begin

In Isabelle 2009-2 locale params are not visible. As a workaround I define their abbreviations:

abbreviation *small-param* ≡ *small*
abbreviation *big-param* ≡ *big*
abbreviation *embed-param* ≡ *embed*

definition *small2-def*: *small2* ≡ *range*(*embed*)
definition *spec-def*: *spec* ≡ *converse*(*embed*)

lemma *spec-inj*: *spec*∈*inj*(*small2*, *small*)
proof −
from *embed-inj* **have** *converse*(*embed*)∈*inj*(*range*(*embed*), *small*) **by** (*rule inj-converse-inj*)
with *small2-def spec-def* **show** *?thesis* **by** *simp*
qed

lemma *spec-fun*: *spec*: *small2*→*small*
proof −
from *embed-inj* **have** *converse*(*embed*): *range*(*embed*)→*small* **by** (*rule inj-converse-fun*)
with *small2-def spec-def* **show** *?thesis* **by** *simp*
qed

lemma *embed-fun*: *embed*: *small*→*big* **by** (*rule inj-is-fun*[*OF embed-inj*])

lemma *embed-surj*: *embed*∈*surj*(*small*, *small2*)
proof −
have *embed*∈*surj*(*small*, *range*(*embed*)) **by** (*rule fun-is-surj*[*OF embed-fun*])
hence *?thesis embed*∈*surj*(*small*, *small2*) **by** (*unfold small2-def*)
show *?thesis* **by** *fact*
qed

theorem *embed-bij*: *embed*∈*bij*(*small*, *small2*)
proof −
from *embed-inj* **have** *embed*∈*bij*(*small*, *range*(*embed*)) **using** *inj-bij-range* **by** *auto*
thus *?thesis* **by** (*unfold small2-def*)
qed

theorem *small2-sub-big*: *small2*⊆*big* **using** *embed-fun range-subset* **by** (*auto simp add*: *small2-def*)

theorem *spec-bij*: *spec*∈*bij*(*small2*, *small*)
proof −
have *converse*(*embed*)∈*bij*(*small2*, *small*) **by** (*rule bij-converse-bij*[*OF embed-bij*])
thus *?thesis* **by** (*unfold spec-def*)
qed

end

3.2 Arbitrary generalizations

locale *arbitrary-generalization* = *generalization-situation* +
fixes *newbig*::*i*
fixes *move*::*i*
assumes
move-bij: *move*∈*bij*(*big*, *newbig*) **and**
move-embed: *move O embed* = *id*(*small*)
begin

In Isabelle 2009-2 locale params are not visible. As a workaround I define their abbreviations:

abbreviation *newbig-param* ≡ *newbig*
abbreviation *move-param* ≡ *move*

lemma *move-fun*: *move*: *big*→*newbig* **using** *move-bij bij-is-fun* **by** *auto*

lemma *move-inj*: *move*∈*inj*(*big*, *newbig*) **using** *move-bij* **by** (*rule bij-is-inj*)
lemma *move-surj*: *move*∈*surj*(*big*, *newbig*) **using** *move-bij* **by** (*rule bij-is-surj*)

lemma *move-domain*: *domain*(*move*) = *big* **using** *domain-of-fun* [*OF move-fun*]
by *auto*

theorem *move-embed-plain* [*simp*]: *x*∈*small* ⟹ *move*‘(*embed*‘*x*) = *x*
proof −
assume *x*: *small*
with *embed-fun* **have** *move*‘(*embed*‘*x*) = (*move O embed*)‘*x* **using** *comp-fun-apply*
by *auto*
also have ... = *id*(*small*)‘*x* **by** (*simp add*: *move-embed*)
also with ‹*x*∈*small*› **have** *id*(*small*)‘*x* = *x* **by** *simp*
finally show *move*‘(*embed*‘*x*) = *x* **by** *simp*

qed

definition *ret-def*: *ret* ≡ *converse*(*move*)

lemma *ret-bij*: *ret*∈*bij*(*newbig*, *big*) **using** *move-bij* **unfolding** *ret-def* **by** *auto*

lemma *ret-inj*: *ret*∈*inj*(*newbig*, *big*) **using** *ret-bij* **by** (*rule bij-is-inj*)
lemma *ret-surj*: *ret*∈*surj*(*newbig*, *big*) **using** *ret-bij* **by** (*rule bij-is-surj*)

lemma *ret-restrict*: *embed* = *restrict*(*ret*, *small*)
proof −
have *converse*(*move*) *O move O embed* = *converse*(*move*) *O id*(*small*) **using** *move-embed* **by** *auto*
hence (*converse*(*move*) *O move*) *O embed* = *converse*(*move*) *O id*(*small*) **using** *comp-assoc* **by** *auto*
moreover
with *left-comp-inverse move-inj* **have** *i* [*simp*]: *converse*(*move*) *O move* = *id*(*big*) **by** *simp*
ultimately have *a*: *id*(*big*) *O embed* = *converse*(*move*) *O id*(*small*) **by** *simp*
with *embed-fun* **have** *embed*⊆*small*×*big* **using** *fun-is-rel* **by** *auto*
hence *id*(*big*) *O embed* = *embed* **using** *left-comp-id* **by** *auto*
with *a* **have** *embed* = *converse*(*move*) *O id*(*small*) **by** *auto*
hence *embed* = *restrict*(*converse*(*move*), *small*) **using** *right-comp-id-any* **by** *auto*
thus *embed* = *restrict*(*ret*, *small*) **using** *right-comp-id-any* **unfolding** *ret-def* **by** *auto*
qed

end

3.3 ZF generalization

We will need this lemma to assert that ZF generalization is an arbitrary generalization:

lemma *mem-not-refl-2*: {*t*}∉*t*
proof
assume *as*: {*t*}∈*t*
let *?A* = {*t*,{*t*}}
have *nz*: *?A* ≠ *0* **by** *auto*
have ∀ *x*∈*?A*. ∃ *y*∈*x*. *y*∈*?A*
proof
fix *x* **assume** *x*∈*?A*
have *x-in-A*: *x*∈*?A* **by** *fact*
have *ex*: ∃ *y*∈*x*. *y*∈*?A*
proof *cases*
assume *x*=*t*
with *as* **have** {*t*}∈*x* ∧ {*t*}∈*?A* **by** *auto*
thus *?thesis* **by** *auto*
next

assume $x \neq t$
with *x-in-A* **have** $x = \{t\}$ **by** *auto*
with *as* **have** $t \in x \wedge t \in ?A$ **by** *auto*
thus *?thesis* **by** *auto*
qed
thus $\exists\, y \in x.\ y \in ?A$ **by** *auto*
qed
with *foundation* **show** *False* **by** *auto*
qed

locale *ZF-generalization* = *generalization-situation*
begin

definition *token-def*: $token \equiv Pow(\bigcup(\bigcup(small)))$

lemma *token-not-small*: $\langle token,x\rangle \notin small$
proof
assume $\langle token,x\rangle \in small$
hence $\{\{token,token\}, \{token,x\}\} \in small$ **by** (*simp add*: *Pair-def*)
hence $\{\{token\}, \{token,x\}\} \in small$ **by** *auto*
hence $\{token\} \in \bigcup(small)$ **by** *auto*
hence $\{token\} \subseteq \bigcup(\bigcup(small))$ **by** *auto*
hence $\{token\} \in Pow(\bigcup(\bigcup(small)))$ **by** *auto*
hence $\{token\} \in token$ **by** (*simp add*: *token-def*)
with *mem-not-refl-2* **show** *False* **by** *contradiction*
qed

definition *zf-move-fun*::$i \Rightarrow i$ **where** *zf-move-fun-def*: *zf-move-fun*$(x) \equiv$ *if* x: *small2* *then* *spec*‘x *else* $\langle token,x\rangle$
definition *zf-move-def*: *zf-move* $\equiv$ (*lam* x:*big*. *zf-move-fun*(x))

definition *zf-newbig-def*: *zf-newbig* $\equiv$ *range*(*zf-move*)

lemma *zf-move-domain*: *domain*(*zf-move*) = *big* **using** *domain-lam* **by** (*auto simp add*: *zf-move-def*)

lemma *zf-move-fun*: *zf-move*: *big* $\rightarrow$ *zf-newbig* **by** (*simp add*: *lam-is-fun zf-move-def zf-newbig-def*)

theorem *small-less-zf-newbig*: *small* $\subseteq$ *zf-newbig*
proof
fix x
assume *s*: $x \in small$
with *embed-fun* **have** *embed*‘$x \in range(embed)$ **by** (*rule apply-rangeI*)
hence *s1*: *embed*‘$x \in small2$ **by** (*unfold small2-def*)
with *small2-sub-big* **have** *s2*: *embed*‘$x \in big$ **by** *auto*
hence *zf-move*‘(*embed*‘x) = *zf-move-fun*(*embed*‘x) **by** (*auto simp add*: *zf-move-def*)
with *s1* **have** *zf-move*‘(*embed*‘x) = *spec*‘(*embed*‘x) **by** (*simp add*: *zf-move-fun-def*)
with *embed-inj s spec-def* **have** *x-val*: *zf-move*‘(*embed*‘x) = x **using** *left-inverse*

```
by auto
 from zf-move-fun s2 have zf-move‘(embed‘x)∈range(zf-move) by (rule apply-rangeI)
  with x-val have x∈range(zf-move) by auto
  with x-val zf-newbig-def show x∈zf-newbig by auto
qed

theorem zf-move-inj: zf-move∈inj(big, zf-newbig)
proof −
  have ∀ a∈big. ∀ b∈big. zf-move‘a = zf-move‘b ⟶ a=b
  proof −
    {
      fix a b
      assume a∈big b∈big
      assume move-eq: zf-move‘a = zf-move‘b
      have spec1-a: a:small2 ⟹ zf-move‘a = spec‘a
      proof −
        assume a∈small2
        with ‹a∈big› show ?thesis by (simp add: zf-move-def zf-move-fun-def)
      qed
      have spec1-b: b∈small2 ⟹ zf-move‘b = spec‘b
      proof −
        assume b∈small2
        with ‹b∈big› show ?thesis by (simp add: zf-move-def zf-move-fun-def)
      qed
      have spec2-a: a∉small2 ⟹ zf-move‘a = ⟨token,a⟩
      proof −
        assume a∉small2
        with ‹a∈big› show ?thesis by (simp add: zf-move-def zf-move-fun-def)
      qed
      have spec2-b: b∉small2 ⟹ zf-move‘b = ⟨token,b⟩
      proof −
        assume b∉small2
        with ‹b∈big› show ?thesis by (simp add: zf-move-def zf-move-fun-def)
      qed
      have a=b
      proof −
        { assume a∈small2 b∈small2
          from ‹a∈small2› have zf-move‘a = spec‘a by (rule spec1-a)
          moreover
          from ‹b∈small2› have zf-move‘b = spec‘b by (rule spec1-b)
          ultimately
          have spec‘a = spec‘b using move-eq by auto
          with spec-inj ‹a:small2› ‹b:small2› have a=b using inj-def by auto
        }
        moreover
        { assume a∈small2 b∉small2
          from ‹a∈small2› have zf-move‘a = spec‘a by (rule spec1-a)
        with spec-fun ‹a∈small2› have ma-s: zf-move‘a∈small using apply-funtype
by auto
```

```
      from ‹b∉small2› have zf-move‘b = ⟨token,b⟩ by (rule spec2-b)
      hence zf-move‘b∉small using token-not-small by auto
      with move-eq ma-s have False by auto
    }
    moreover
    { assume a∉small2 b∈small2
      from ‹b∈small2› have zf-move‘b = spec‘b by (rule spec1-b)
    with spec-fun ‹b∈small2› have mb-s: zf-move‘b∈small using apply-funtype
by auto
      from ‹a∉small2› have zf-move‘a = ⟨token,a⟩ by (rule spec2-a)
      hence zf-move‘a∉small using token-not-small by auto
      with move-eq mb-s have False by auto
    }
    moreover
    { assume a∉small2 b∉small2
      from ‹a∉small2› have mt-a: zf-move‘a = ⟨token,a⟩ by (rule spec2-a)
      moreover
      from ‹b∉small2› have mt-b: zf-move‘b = ⟨token,b⟩ by (rule spec2-b)
       from move-eq mt-a mt-b have ⟨token,a⟩=⟨token,b⟩ by auto — Order of
from conditions is important to not cause infinite loop
      hence a=b by auto
    }
    ultimately show a=b by auto
  qed
 }
 thus ?thesis by auto
qed
with zf-move-fun show ?thesis using inj-def by simp
qed

theorem zf-move-surj: zf-move∈surj(big, zf-newbig)
proof −
 have zf-move∈surj(big, range(zf-move)) by (rule fun-is-surj[OF zf-move-fun])
 hence ?thesis zf-move∈surj(big, zf-newbig) by (unfold zf-newbig-def)
 show ?thesis by fact
qed

theorem zf-move-bij: zf-move∈bij(big, zf-newbig)
proof −
 from zf-move-inj have zf-move∈bij(big, range(zf-move)) using inj-bij-range by
auto
 thus ?thesis by (unfold zf-newbig-def)
qed

theorem zf-move-embed [simp]: x∈small ⟹ zf-move‘(embed‘x) = x
proof −
 assume x∈small
 with embed-fun have embed‘x∈range(embed) by (rule apply-rangeI)
 hence s1: embed‘x∈small2 by (unfold small2-def)
```

with *small2-sub-big* **have** *embed‘x∈big* **by** *auto*
with *s1* **have** *zf-move‘(embed‘x) = spec‘(embed‘x)* **by** (*auto simp add*: *zf-move-def zf-move-fun-def*)
also from *embed-inj ‹x∈small› spec-def* **have** *spec‘(embed‘x) = x* **by** *auto*
finally show *zf-move‘(embed‘x) = x* **by** *auto*
qed

theorem *zf-embed-move*: *zf-move O embed = id(small)*
proof −
have *∀ y∈small. zf-move‘(embed‘y) = y* **by** *simp*
moreover
have *embed*: *small→big* **by** (*rule embed-fun*)
moreover
have *zf-move*: *big→zf-newbig* **by** (*rule zf-move-fun*)
ultimately show *?thesis* **using** *comp-eq-id-iff1* **by** *blast*
qed

end

Next prove that ZF generalization is an arbitrary generalization:

sublocale *ZF-generalization ⊆ arbitrary-generalization small big embed zf-newbig zf-move*
proof
show *zf-move∈bij(big, zf-newbig)* **using** *zf-move-bij* **by** *auto*
show *zf-move O embed = id(small)* **using** *zf-embed-move* **by** *auto*
qed

end

4 An example of generalization

theory *int-obj-ex*
imports *Generalization*
begin

In this example I show that integers can be considered as a generalization of natural numbers.

interpretation *int-interpr*: *ZF-generalization nat int* (*lam n:nat. int-of(n)*)
proof
show *Lambda(nat, int-of)∈inj(nat, int)* **using** *int-of-inject* **unfolding** *inj-def*
by *auto*
qed

abbreviation *int-obj ≡ int-interpr.newbig-param*

Naturals are a subset of integers.

lemma *nat ⊆ int-obj* **using** *int-interpr.small-less-zf-newbig* **by** *auto*

An example of defining an operation on the generalization set:

definition
  add::[i,i]⇒i
 where $add(x,y) == int\text{-}interpr.move\text{-}param`(int\text{-}interpr.ret`x\ \$+\ int\text{-}interpr.ret`y)$

end