Title: Zermelo-Fraenkel set theory in HOL (part 1)

Ref: DBC/RBJ/138 *Issue:* *Date:* 1988-07-16

Status: Draft *Type:*

Author:

Name	*Location*	*Signature*	*Date*
Roger Bishop Jones			

Abstract: This document consists of a formal axiomatisation in HOL of Zermelo-Fraenkel set theory.

Distribution:

0 DOCUMENT CONTROL

0.1 Contents list

0.2 Document cross references

[1] *Introduction to higher order categorical logic*, L.Lambek and P.J.Scott, Cambridge University Press 1986, ISBN 0-521-24665-2

0.3 Changes history

First version.

0.4 Changes forecast

Under development, highly incomplete, totally volatile.

0.5 Abbreviations and notation

HOL Higher Order Logic

1 INTRODUCTION

This is a minimally modified transcription into HOL of the formalisation of ZF found in **The Logical Foundations of Mathematics** by William S. Hatcher (Pergammon 1982 ISBN $0-08-025800$). The fact that this is an axiomatisation in higher order logic rather than first order logic has some consequences, but doesn't cause any very major changes.

I mention here just two points:

- $\Leftrightarrow$ In HOL logical equivalence is interchangeable with equality over type BOOL(i.e. $\vdash\ulcorner\forall(a : BOOL)(b : BOOL).(a = b)\Leftrightarrow(a\Leftrightarrow b)\urcorner$). Though equivalence is used in the axioms, equivalent lemmas are then proven using equality, and these lemmas are used in subsequent proofs.
- $=$ Equality is in the basic HOL system, and so we do not need to define is as Hatcher does. However, Hatcher has a curious division of labour between his definition of equality and his axiom of extensionality (ZF1). It looks as though his definition of equality in fact gives half of the necessary facts for equality together with extensionality, and his axiom of extensionality is just the other half of the equality axiom. The fact that equality is built into HOL means that Hatcher's ZF1 is provable in HOL, but his definition of equality isn't!

2 ZERMELO-FRAENKEL ZFSET THEORY

2.1 Formalisation of ZF

First we start a new theory for ZF and declare the new type ZFSET.

```
SML
new_theory"zf138";
new_type ("ZFSET",0);
push_proof_context prove_∃_epc;
val asm_ante_tac = undisch_tac;
fun taut_rule t = tac_proof(([],t), REPEAT ∀_tac THEN taut_tac);
usefile"/hat/users/rbj/demo/dtd067";
usefile"/hat/users/rbj/demo/imp067";
val res_tac = REPEAT strip_tac THEN (TRY_T(basic_res_tac 3 []));
val res_tac1 = basic_res_tac 3 [];
val res_tac2 = REPEAT strip_tac THEN (TRY_T((basic_res_tac 3 []) ORELSE (basic_res_tac1 3 [])));
fun res_rule t = tac_proof (([],t), res_tac2);
```

2.1.1 Membership and Extensionality

Next we introduce the membership predicate and the axiom of extensionality. This differs from Hatcher's presentation since the equality rules are already built into HOL, whereas Hatcher's definition of equality is in fact the rule of extensionality.

```
SML
new_const ("∈z",⌜:ZFSET → ZFSET → BOOL⌝);
declare_infix (230,"∈z");
val EXT = new_axiom (["EXT"],⌜
        ∀ (x:ZFSET)(y:ZFSET)•(∀ (z:ZFSET)• z ∈z x ⇔ z ∈z y) ⇒ (x = y)
⌝);
```

Hatchers theorem 1 is simply proved in HOL:

```
SML
push_goal([],⌜
        ∀(x1:ZFSET)(x2:ZFSET)•(x1 = x2) ⇒ ∀(x3:ZFSET)• x1 ∈z x3 ⇔ x2 ∈z x3
⌝);
a (REPEAT_UNTIL (is_⇔ o snd) strip_tac THEN asm_rewrite_tac []);
val ZF_thm1 = save_pop_thm "ZF_thm1";
```

The following rewrite rule will be useful for proving equality of sets.

SML

```
push_goal([],⌜
        ∀(x:ZFSET)(y:ZFSET)• (x = y) ⇔ (∀(z:ZFSET)• z ∈z x ⇔ z ∈z y)
⌝);
a (REPEAT_N 5 strip_tac THEN (TRY_T (asm_rewrite_tac[])));
a(strip_asm_tac(list_∀_elim [⌜x⌝,⌜y⌝] EXT));
a(spec_asm_tac ⌜∀ z• z ∈z x ⇔ z ∈z y⌝⌜z⌝);
a(spec_asm_tac ⌜∀ z• z ∈z x ⇔ z ∈z y⌝⌜z⌝);
val ZF_le1 = save_pop_thm "ZF_le1";
```

Hatcher's theorem 2 is too trivial to be worth saving:

SML

```
val ZF_thm2 = refl_conv ⌜x:ZFSET⌝;
```

Now we define non-membership and inequality.

SML

```
declare_infix (230,"∉z");
```

HOL Constant

```
        $∉z: ZFSET → ZFSET → BOOL
├──────────────────────────────────────────────────
        ∀x y• x ∉z y = ¬(x ∈z y)
```

SML

```
val ∉z = get_specification "∉z";
declare_infix (200,"≠");
```

HOL Constant

```
        $≠: 'a → 'a → BOOL
├──────────────────────────────────────────────────
        ∀x y• x ≠ y ⇔ ¬(x = y)
```

SML

```
val ≠ = get_specification "≠";;
```

2.1.2 Separation

The constant $\Lambda?_z$ is introduced for separation. Though suggestive of abstraction it is not itself a binder.

SML

```
new_const ("Λz",⌜:ZFSET → (ZFSET → BOOL) → ZFSET⌝);
val ZF2 = new_axiom(["ZF2"],⌜
        ∀(A:ZFSET→BOOL)(z:ZFSET)(x:ZFSET)• x ∈z (Λz z A) ⇔  x ∈z z ∧ A x
⌝);
```

2.1.3 The Empty Set

SML
```
new_const ("∅z",⌜:ZFSET⌝);

val ZF3 = new_axiom (["ZF3"],⌜
        (∅z:ZFSET) = Λz ∅z λx1:ZFSET• F
⌝);
```

We now prove that nothing is a member of the empty set.

SML
```
push_goal([],⌜
        ∀(x1:ZFSET)• x1 ∉z ∅z
⌝);
a (pure_rewrite_tac [∉z]);
a (pure_once_rewrite_tac [ZF3]);
a (rewrite_tac [ZF2]);
val ZF_thm3 = save_pop_thm "ZF_thm3";
```

The following form is more convenient.

SML
```
val ZF_le2 = save_thm ("ZF_le2", rewrite_rule [∉z] ZF_thm3);
```

The empty set is unique.

SML
```
push_goal([],⌜
        ∀(x1:ZFSET)•(∀(x2:ZFSET)•x2 ∉z x1) ⇔ (∅z = x1)
⌝);
a (rewrite_tac [∉z,ZF_le1,ZF_le2]);
val ZF_thm4 = save_pop_thm "ZF_thm4";
```

A helpful lemma for proving non-emptyness:

SML
```
push_goal([],⌜
        ∀(x1:ZFSET)(x2:ZFSET)• x2 ∈z x1 ⇒ x1 ≠ ∅z
⌝);
a (rewrite_tac [≠,ZF_le1,ZF_le2]);
a (     REPEAT strip_tac
        THEN ∃_tac ⌜x2⌝
        THEN REPEAT strip_tac);
val ZF_le3 = save_pop_thm "ZF_le3";
```

2.1.4 Power Sets

The power set is defined using the subset relationship:

SML

```
declare_infix (250,"⊆z");
```

HOL Constant

$$\$\subseteq_z : ZFSET \to ZFSET \to BOOL$$

$$\forall a\ b \bullet\ a \subseteq_z b = \forall c \bullet\ c \in_z a \Rightarrow c \in_z b$$

SML

```
val ⊆z = get_specification "⊆z";
```

We prove some obvious properties of $\subseteq_z$:

SML

```
push_goal([],⌜
        ∀(x1:ZFSET)(x2:ZFSET)• (x1 ⊆z x2 ∧ x2 ⊆z x1) ⇒ (x1 = x2)
⌝);
a (rewrite_tac [⊆z, ZF_le1]);
a (REPEAT strip_tac);
a (spec_asm_tac ⌜∀ c• c ∈z x1 ⇒ c ∈z x2⌝ ⌜z⌝);
a (spec_asm_tac ⌜∀ c• c ∈z x2 ⇒ c ∈z x1⌝ ⌜z⌝);
val ZF_thm5 = save_pop_thm "ZF_thm5";

push_goal([],⌜
        ∀(x1:ZFSET)• x1 ⊆z x1
⌝);
a (rewrite_tac [⊆z]);
val ZF_thm6 = save_pop_thm "ZF_thm6";

push_goal([],⌜
        ∀(x1:ZFSET)(x2:ZFSET)(x3:ZFSET)• (x1 ⊆z x2 ∧ x2 ⊆z x3) ⇒ (x1 ⊆z x3)
⌝);
a (rewrite_tac [⊆z] THEN REPEAT strip_tac);
a (spec_asm_tac ⌜∀ c• c ∈z x2 ⇒ c ∈z x3⌝ ⌜c⌝);
a (spec_asm_tac ⌜∀ c• c ∈z x1 ⇒ c ∈z x2⌝ ⌜c⌝);
val ZF_thm7 = save_pop_thm "ZF_thm7";

push_goal([],⌜
        ∀(x1:ZFSET)• ∅z ⊆z x1
⌝);
a (rewrite_tac [⊆z, ZF_le2]);
val ZF_thm8 = save_pop_thm "ZF_thm8";

new_const("ℙz",⌜:ZFSET → ZFSET⌝);
val ZF4 = new_axiom(["ZF4"],⌜
        ∀(y:ZFSET)(x:ZFSET)• x ∈z (ℙz y) ⇔ x ⊆z y
⌝);
```

2.1.5 Pairs

SML

```
new_const("pair",⌜:ZFSET → ZFSET → ZFSET⌝);
val ZF5 = new_axiom(["ZF5"],⌜
        ∀(y:ZFSET)(z:ZFSET)(x:ZFSET)• x ∈z (pair y z) ⇔ (x=y) ∨ (x=z)
⌝);

push_goal([],⌜
        ∀(x:ZFSET)(y:ZFSET)• x ∈z (pair x y) ∧ y ∈z (pair x y)
⌝);
a (REPEAT strip_tac THEN rewrite_tac [ZF5]);
val ZF_le4 = save_pop_thm "ZF_le4";
```

HOL Constant

```
        unit:ZFSET → ZFSET
─────────────────────────────────────────
        ∀x • unit x = pair x x
```

SML

```
val unit = get_specification "unit";
push_goal([],⌜
        ∀x1 x2:ZFSET• x1 ∈z unit x2 = (x1 = x2)
⌝);
a (strip_tac THEN rewrite_tac [unit, ZF5, ZF_le4]);
val ZF_thm9 = save_pop_thm "ZF_thm9";
```

SML

```
push_goal([],⌜
        ∀(x:ZFSET)• unit x ≠ ∅z
⌝);
a (rewrite_tac [≠,ZF_thm9,ZF_le1,ZF_le2]);
a (REPEAT strip_tac);
a (∃_tac ⌜x⌝);
a (REPEAT strip_tac);
val ZF_le2b = save_pop_thm "ZF_le2b";
```

SML

```
push_goal([],⌜
        ∀(x:ZFSET)(y:ZFSET)• (unit x = unit y) = (x = y)
⌝);
a (rewrite_tac [list_∀_elim [⌜unit x⌝,⌜unit y⌝] ZF_le1, ZF_thm9]);
a (REPEAT strip_tac THEN (TRY_T (asm_rewrite_tac[])));
(* why no REPEAT_WHILE? *)
a (spec_asm_tac ⌜∀ z• z = x ⇔ z = y⌝ ⌜x⌝);
(* strip_assume_tac, etc. should apply refl_conv?
        (test for ¬(x=x) and (x=x)?) *)
a (strip_asm_tac (refl_conv ⌜x⌝));
val ZF_le13 = save_pop_thm "ZF_le13";

declare_infix (300,"↦z");
```

HOL Constant

```
$↦z:ZFSET → ZFSET → ZFSET
```

```
        ∀a b• a ↦z b = pair (unit a) (pair a b)
```

SML

```
val ↦z = get_specification "↦z";

push_goal([],⌜
        ∀(x:ZFSET)(y:ZFSET)• (unit x) ∈z (x ↦z y) ∧ (pair x y) ∈z (x ↦z y)
⌝);
a (REPEAT strip_tac THEN rewrite_tac [↦z, ZF_le4]);
val ZF_le5 = save_pop_thm "ZF_le5";
```

2.1.6 Union and Intersection

SML

```
new_const("⋃z",⌜:ZFSET → ZFSET⌝);
val ZF6 = new_axiom(["ZF6"],⌜
        ∀(y:ZFSET)(x:ZFSET)• x ∈z (⋃z y) ⇔ ∃(z:ZFSET)•z ∈z y ∧ x ∈z z
⌝);
fun prove_thm (key,term,tactic) = save_thm(key, tac_proof(([],term),tactic));
(* should we provide "prove_thm"? *)
```

HOL Constant

```
$        ⋂z:ZFSET → ZFSET
├──────────────────────────────────────────────────────
        ∀ x• ⋂z x = Λz (⋃z x) λ y• ∀z• z ∈z x ⇒ y ∈z z
```

SML

```
val ⋂z = get_specification "⋂z";

declare_infix (280,"∪z");
```

HOL Constant

```
$        ∪z:ZFSET→ZFSET→ZFSET
├──────────────────────────────────────────────────────
        ∀ x y • x ∪z y = ⋃z (pair x y)
```

SML

```
val ∪z = get_specification "∪z";

declare_infix (290,"∩z");
```

HOL Constant

```
$        ∩z:ZFSET→ZFSET→ZFSET
├──────────────────────────────────────────────────────
        ∀ x y • x ∩z y = ⋂z (pair x y)
```

SML

```
val ∩z = get_specification "∩z";

push_goal([],⌜
        ∀(x1:ZFSET)(x2:ZFSET)(x3:ZFSET)•x1 ∈z (x2 ∪z x3) ⇔ x1 ∈z x2 ∨ x1 ∈z x3
⌝);
a(rewrite_tac [∪z,ZF6,ZF5]);
a (REPEAT strip_tac);
a (undisch_tac ⌜x1 ∈z z⌝
        THEN asm_rewrite_tac[]
        THEN REPEAT strip_tac);
a (undisch_tac ⌜x1 ∈z z⌝
        THEN asm_rewrite_tac[]
        THEN REPEAT strip_tac);
a (∃_tac ⌜x2⌝ THEN asm_rewrite_tac[]);
a (∃_tac ⌜x3⌝ THEN asm_rewrite_tac[]);
val ZF_thm10 = save_pop_thm "ZF_thm10";

push_goal([],⌜
        ∀(x1:ZFSET)(x2:ZFSET)(x3:ZFSET)•x1 ∈z (x2 ∩z x3) ⇔ x1 ∈z x2 ∧ x1 ∈z x3
⌝);
a(rewrite_tac [∩z,⋂z,ZF2,ZF5,ZF6]);
a (REPEAT strip_tac);
a (undisch_tac ⌜x1 ∈z z⌝ THEN asm_rewrite_tac[]);
a (spec_asm_tac ⌜∀ z• z = x2 ∨ z = x3 ⇒ x1 ∈z z⌝ ⌜x3⌝);
a (strip_asm_tac (refl_conv ⌜x3⌝));
a (spec_asm_tac ⌜∀ z• z = x2 ∨ z = x3 ⇒ x1 ∈z z⌝ ⌜x2⌝);
a (strip_asm_tac (refl_conv ⌜x2⌝));
a (spec_asm_tac ⌜∀ z• z = x2 ∨ z = x3 ⇒ x1 ∈z z⌝ ⌜x3⌝);
a (strip_asm_tac (refl_conv ⌜x3⌝));
a (∃_tac ⌜x2⌝ THEN asm_rewrite_tac[]);
a (asm_rewrite_tac[]);
a (asm_rewrite_tac[]);
val ZF_thm11 = save_pop_thm "ZF_thm11";
```

2.1.7 Natural Numbers

SML

HOL Constant

$$suc\colon ZFSET \to ZFSET$$

$$\forall x \bullet suc\ x = x \cup_z (unit\ x)$$

SML

```
val suc = get_specification "suc";
```

HOL Constant

$$Nat\colon \mathbb{N} \to ZFSET$$

$$\begin{array}{lll} \forall n\bullet & Nat\ 0 & = \varnothing_z \\ \wedge & Nat\ (n+1) & = suc\ (Nat\ n) \end{array}$$

SML

```
val Nat = get_specification "Nat";
```

2.1.8 The Axiom of Regularity

The following axiom Hatcher calls the axiom of regularity, sometimes it is called the axiom of well foundedness.

SML

```
val ZF7 = new_axiom (["ZF7"],⌜
        ∀(x:ZFSET)•x ≠ ∅z ⇒ ∃(y:ZFSET)• y ∈z x ∧ (y ∩z x = ∅z)
⌝);
```

We now prove some consequences of well foundedness.

SML

```
push_goal([],⌜
        ∀x1:ZFSET• x1 ∉z x1
⌝);
a (ante_tac ZF7 THEN rewrite_tac [∉z, ≠, ZF_le1, ZF_le2] THEN c_contr_tac);
a (spec_asm_tac
        ⌜∀ x• ¬ (∀ z• ¬ z ∈z x) ⇒ (∃ y• y ∈z x ∧ (∀ z• ¬ z ∈z y ∩z x))⌝
        ⌜Λz x1 (λx• x = x1)⌝);
a (spec_asm_tac ⌜∀ z• ¬ z ∈z Λz x1 (λ x• x = x1)⌝ ⌜x1⌝);
a (asm_ante_tac ⌜¬ x1 ∈z Λz x1 (λ x• x = x1)⌝);
a (rewrite_tac[ZF2]);
a goal_in_asms_tac;
a (spec_asm_tac ⌜∀ z• ¬ z ∈z y ∩z Λz x1 (λ x• x = x1)⌝ ⌜x1⌝);
a (list_undisch_tac [
        ⌜y ∈z Λz x1 (λ x• x = x1)⌝,
        ⌜¬ x1 ∈z y ∩z Λz x1 (λ x• x = x1)⌝]);
a (rewrite_tac[ZF2, ZF_thm11, taut_rule ⌜∀a b c•a ∧ b ⇒ c ⇔ b ⇒ a ⇒ c⌝]);
a (strip_tac THEN asm_rewrite_tac[]);
val ZF_thm12 = save_pop_thm "ZF_thm12";
```

SML

```
val PURE_NOT_FORALL_TAC = conv_tac (TOP_MAP_C ¬_∀_conv);
val NOT_FORALL_TAC = PURE_NOT_FORALL_TAC THEN (rewrite_tac[]);

val PURE_NOT_EXISTS_TAC = conv_tac (TOP_MAP_C ¬_∃_conv);
val NOT_EXISTS_TAC = PURE_NOT_EXISTS_TAC THEN (rewrite_tac[]);
```

SML

```
repeat drop_main_goal;
push_goal([],⌜
        ∀x1 x2• x1 ∈z x2 ⇒ x2 ∉z x1
⌝);
a (ante_tac ZF7 THEN rewrite_tac [∉z,≠,ZF_le1,ZF_le2, ZF5, ZF_thm11] THEN c_contr_tac);
a (spec_nth_asm_tac 3 ⌜pair x1 x2⌝);
a (spec_nth_asm_tac 1 ⌜x1⌝);
a (undisch_tac ⌜¬ x1 ∈z pair x1 x2⌝ THEN rewrite_tac [ZF5]);
a (undisch_tac ⌜y ∈z pair x1 x2⌝ THEN rewrite_tac [ZF5] THEN c_contr_tac);
a (spec_asm_tac ⌜∀ z• ¬ (z ∈z y ∧ z ∈z pair x1 x2)⌝ ⌜x2⌝);
a (undisch_tac ⌜¬ x2 ∈z y⌝ THEN asm_rewrite_tac []);
a (undisch_tac ⌜¬ x2 ∈z pair x1 x2⌝ THEN asm_rewrite_tac [ZF5]);
a (spec_asm_tac ⌜∀ z• ¬ (z ∈z y ∧ z ∈z pair x1 x2)⌝ ⌜x1⌝);
a (undisch_tac ⌜¬ x1 ∈z y⌝ THEN asm_rewrite_tac []);
a (undisch_tac ⌜¬ x1 ∈z pair x1 x2⌝ THEN asm_rewrite_tac [ZF5]);
val ZF_thm13 = save_pop_thm "ZF_thm13";

(* val map_conv_tac = conv_tac o TOP_MAP_C; *)
```

The following function *set* takes a list of sets and returns the set containing just those elements:

SML

```
val set = new_list_rec_definition("set",⌜
        (set [] = ∅z) ∧
        (set (CONS h t) = (unit h) ∪z (set t))
⌝);


push_proof_context prove_∃_epc;
```

HOL Constant

```
        set: ZFSET LIST → ZFSET
------------------------------------------------------------
        set [] = ∅z ∧
        ∀ h t• set (Cons h t) = (unit h) ∪z (set t)
```

val set = it;

One more consequence of well-foundedness:

SML

```
push_goal([],⌜
        ∀(x1:ZFSET)(x2:ZFSET)(x3:ZFSET)• x1 ∈z x2 ∧ x2 ∈z x3 ⇒ x3 ∉z x1 ∧ x3 ≠ x1
⌝);
a (ante_tac ZF7 THEN rewrite_tac [set, ∉z,≠,ZF_le1, ZF_le2, ZF5, ZF_thm10, ZF_thm11]
        THEN  strip_tac);
a (spec_nth_asm_tac 1 ⌜set[(x1:ZFSET);(x2:ZFSET);(x3:ZFSET)]⌝);
a (undisch_tac  ⌜∀ z• ¬ z ∈z set [x1; x2; x3]⌝
        THEN rewrite_tac [set, ∉z,≠,ZF_le2, ZF5, ZF_thm9, ZF_thm10, ZF_thm11]);
a (strip_tac);
a ( THEN asm_rewrite_tac[]);

a res_tac;
a (ante_tac ZF7 THEN rewrite_tac [set, ∉z,≠,ZF_le1,ZF_le2, ZF5, ZF_thm9, ZF_thm10, ZF_thm11]
        THEN  c_contr_tac);

(* *** Goal "1" *** *)

a (spec_nth_asm_tac 4 ⌜set[(x1:ZFSET);(x2:ZFSET);(x3:ZFSET)]⌝);

(* *** Goal "1.1" *** *)
a (undisch_tac⌜∀ z• ¬ z ∈z set [x1; x2; x3]⌝
        THEN rewrite_tac [ZF_thm9, ZF_thm10, set, ZF_le2]
        THEN strip_tac THEN ∃_tac ⌜x1⌝
        THEN rewrite_tac[]);

(* *** Goal "1.2" *** *)
a (undisch_tac ⌜∀ z• ¬ (z ∈z y ∧ z ∈z set [x1; x2; x3])⌝
        THEN undisch_tac ⌜y ∈z set [x1; x2; x3]⌝
        THEN rewrite_tac [ZF_thm9, ZF_thm10, set, ZF_le2]);
a (strip_tac THEN asm_rewrite_tac[]
        THEN strip_tac);

a (∃_tac ⌜x3⌝ THEN asm_rewrite_tac[]);
a (∃_tac ⌜x1⌝ THEN asm_rewrite_tac[]);
a (∃_tac ⌜x2⌝ THEN asm_rewrite_tac[]);
```

SML
```
(* *** Goal "2" *** *)


a (spec_nth_asm_tac 1 ⌜x1⌝);
a (undisch_tac⌜¬ x1 ∈z set [x1; x2; x3]⌝ THEN rewrite_tac [ZF_thm9, ZF_thm10, set]);

(* *** Goal "1.2" *** *)

a (spec_nth_asm_tac 1 ⌜x1⌝);

(* *** Goal "1.2.1" *** *)

a (undisch_tac ⌜y ∈z set [x1; x2; x3]⌝
        THEN  rewrite_tac [set, ZF_le2, ZF_thm9, ZF_thm10]);
        THEN c_contr_tac);

(* *** Goal "1.2.1.1" *** *)
a (spec_asm_tac  );

(* old proof starts here *)

a (EVERY[
        rewrite_tac[∉z; ≠];
        REPEAT strip_tac;
        MP_TAC ZF7;
        TAUT_rewrite_tac ⌜a ⇒ F = ¬a⌝;
        NOT_FORALL_TAC;
        EXISTS_TAC ⌜set[(x1:ZFSET);(x2:ZFSET);(x3:ZFSET)]⌝;
        rewrite_tac [set;≠;ZF_le1;ZF_thm11;
                ZF_thm10;rewrite_rule[∉z]ZF_thm3;unit];
        TAUT_rewrite_tac ⌜¬ (¬a ⇒ b) = ¬a ∧ ¬b⌝;
        strip_tac ]);
```

Giving four subgoals.

The first:
SML
```
a (EVERY[
        NOT_FORALL_TAC;
        EXISTS_TAC ⌜x1:ZFSET⌝;
        asm_rewrite_tac [ZF5]]);
```

The second:

```
SML
a (asm_rewrite_tac [ZF5]);
a (NOT_EXISTS_TAC);
a (TAUT_rewrite_tac ⌜¬(a ∧ b) = a ⇒ ¬b⌝);
a (EVERY [strip_tac; strip_tac; NOT_FORALL_TAC; asm_rewrite_tac[]]);
a (EXISTS_TAC ⌜x3:ZFSET⌝ THEN asm_rewrite_tac []);
a (EXISTS_TAC ⌜x1:ZFSET⌝ THEN asm_rewrite_tac []);
a (EXISTS_TAC ⌜x2:ZFSET⌝ THEN asm_rewrite_tac []);
```

The third:

```
SML
a NOT_FORALL_TAC;
a (EXISTS_TAC ⌜x1:ZFSET⌝);
a (asm_rewrite_tac [ZF5]);
```

And the fourth:

```
SML
a (EVERY [
        RMP_TAC ⌜F⌝;
        rewrite_tac[];
        MP_TAC ZF_thm13;
        rewrite_tac [∉z];
        NOT_FORALL_TAC;
        NOT_FORALL_TAC;
        EXISTS_TAC ⌜x1:ZFSET⌝;
        EXISTS_TAC ⌜x2:ZFSET⌝;
        asm_rewrite_tac [];
        accept_tac (rewrite_rule [ASSUME ⌜x3 = x1⌝] (ASSUME ⌜x2 ∈z x3⌝))]);

val ZF_thm14 = save_pop_thm "ZF_thm14";
```

2.1.9 Ordinals

Transitive sets:

```
SML
val Trans = new_definition("Trans",⌜
        (Trans:ZFSET → BOOL) x = ∀(y:ZFSET)• y ∈z x ⇒ y ⊆z x
⌝);
```

Connected sets:

SML
```
val Con = new_definition("Con",⌜
        (Con:ZFSET → BOOL) x =
        ∀(y:ZFSET)(z:ZFSET)• (y ∈z x ∧ z ∈z x ∧ z ≠ y) ⇒ (z ∈z y ∨ y ∈z z)
⌝);

val ZF_le6 = save_thm("ZF_le6",
        TAUT_rewrite_rule ⌜a ∧ b ∧ ¬ c ⇒ d = a ∧ b ⇒ c ∨ d⌝
        (rewrite_rule [≠] Con));

val ZF_le7 = save_thm("ZF_le7",
        TAUT_rewrite_rule ⌜a ∧ b ∧ ¬ c ⇒ d = a ⇒ b ⇒ c ∨ d⌝
        (rewrite_rule [≠] Con));

push_goal([],⌜
        Trans(x1:ZFSET) = ∀(x2:ZFSET)(x3:ZFSET)•(x2 ∈z x1 ∧ x3 ∈z x2 ⇒ x3 ∈z x1)
⌝);
a (rewrite_tac [Trans;⊆z]);
a (EQ_TAC THEN REPEAT strip_tac THEN RES_TAC);
val ZF_le8 = save_pop_thm "ZF_le8";
```

Ordinal numbers:

SML
```
val On = new_definition("On",⌜
        (On:ZFSET → BOOL) x = Trans x ∧ Con x
⌝);
```

Hatcher's exercise to prove that an ordinal is transitive under $\in_z$.

SML

```
push_goal([],⌜
        ∀(x1:ZFSET)(x2:ZFSET)(x3:ZFSET)(x4:ZFSET)•
                ((On(x1) ∧ x2 ∈z x1 ∧ x3 ∈z x1 ∧ x4 ∈z x1 ∧ x2 ∈z x3 ∧ x3 ∈z x4)
                        ⇒ x2 ∈z x4)
⌝);
a (rewrite_tac [On;Con;ZF_le8;≠;⊆z] THEN REPEAT strip_tac);

a (LEMMA ⌜¬(x4 ∈z x2) ∧ ¬(x4 = x2)⌝);
a RES_TAC;
a RES_TAC;
a (LEMMA ⌜(x2 ∈z x3) ∧ (x3 ∈z x4)⌝);
a (IMP_RES_TAC (SPECL [⌜x2⌝;⌜x3⌝;⌜x4⌝] (rewrite_rule [≠;∉z] ZF_thm14)));
a (asm_rewrite_tac []);
a (asm_rewrite_tac []);
val ZF_le9 = save_pop_thm "ZF_le9";
```

2.1.10 Ordinals

SML

```
push_goal([],⌜
∀(p:ZFSET→BOOL)(q:ZFSET→BOOL)• (∀(x:ZFSET)• p x ⇒ q x) ⇒ ((∀(x:ZFSET)•p x) ⇒ ∀(x:ZFS
⌝);
a (strip_tac THEN strip_tac THEN strip_tac);
a FORALL_OUT_TAC;
a (asm_rewrite_tac[]);
val ZF_le10 = save_pop_thm "ZF_le10";
```

The ordinals are totally ordered by $\in_z$.

SML

```
push_goal([],⌜
        ∀(x1:ZFSET)(x2:ZFSET)• x1 ⊆z x2 ∧ x1 ≠ ∅z ∧ (On x2)
                ⇒ ∃(x3:ZFSET)• x3 ∈z x1
                        ∧ ∀(x4:ZFSET)• x4 ∈z x1 ⇒ (x4 = x3) ∨ x3 ∈z x4
⌝);
a (rewrite_tac [⊆z;≠;On;ZF_le7]);
a (REPEAT strip_tac);
a (MP_TAC (SPEC ⌜x1:ZFSET⌝ ZF7));
a (asm_rewrite_tac[≠]);
a (TAUT_rewrite_tac ⌜a ⇒ b = ¬b ⇒ ¬a⌝);
a NOT_EXISTS_TAC;
a FORALL_OUT_TAC;
a (TAUT_rewrite_tac ⌜¬(a ∧ b) ⇒ ¬(a ∧ c) = (a ∧ c) ⇒ b⌝);
a (rewrite_tac [SPEC ⌜x ∩z x1⌝ ZF_le1]);
a (rewrite_tac [ZF_thm11; rewrite_rule [∉z] ZF_thm3]);
a (TAUT_rewrite_tac ⌜¬(a ∧ b) = b ⇒ ¬a⌝);
a (REPEAT strip_tac);
a RES_TAC;
a (MP_TAC (SPECL [⌜x:ZFSET⌝;⌜x4:ZFSET⌝] (ASSUME ⌜∀y z•
        y ∈z x2 ⇒
        z ∈z x2 ⇒
        (z = y) ∨ z ∈z y ∨ y ∈z z⌝)));
a (RMP_TAC ⌜(x ∈z x2 ∧ x4 ∈z x2 ∧ ¬(x4 ∈z x))⌝);
a TAUT_TAC;
a (asm_rewrite_tac []);

val ZF_thm15 = save_pop_thm "ZF_thm15";
```

The empty set is an ordinal.

SML

```
val ZF_thm16 = prove_thm ("ZF_thm16",⌜
        On ∅z
⌝,
        rewrite_tac [On;Trans;Con;ZF_le2]);
```

A subset of a connected set is connected.

SML
```
push_goal([],⌜
        ∀(x1:ZFSET)(x2:ZFSET)•Con x1 ∧ x2 ⊆z x1 ⇒ Con x2
⌝);
a(rewrite_tac [Con;≠;⊆z]);
a(TAUT_rewrite_tac ⌜a ∧ b ⇒ c = b ⇒ a ⇒ c⌝);
a (strip_tac THEN strip_tac THEN strip_tac);
a (FORALL_OUT_TAC THEN strip_tac THEN FORALL_OUT_TAC);
a (TAUT_rewrite_tac ⌜a ⇒b ⇒ c = b ⇒ a ⇒ c⌝);
a (strip_tac THEN strip_tac);
a (RES_TAC THEN asm_rewrite_tac []);
val ZF_thm17 = save_pop_thm "ZF_thm17";
```

Every member of an ordinal is an ordinal.

SML
```
push_goal([],⌜
        ∀(x1:ZFSET)(x2:ZFSET)• On x1 ∧ x2 ∈z x1 ⇒ On x2
⌝);
a (rewrite_tac [On]);
a (TAUT_rewrite_tac ⌜a ⇒ b ∧ c = (a ⇒ b) ∧ (a ⇒ c)⌝);
a (strip_tac THEN strip_tac THEN strip_tac);
```

subgoal 1

SML
```
a (rewrite_tac[Trans;⊆z]);
a (REPEAT strip_tac);
a RES_TAC;
a RES_TAC;
a (IMP_RES_TAC (SPECL [⌜c:ZFSET⌝;⌜y:ZFSET⌝]
                    (rewrite_rule [≠;∉z] ZF_thm14)));
a (IMP_RES_TAC (rewrite_rule [Con;≠] (ASSUME ⌜Con (x1:ZFSET)⌝)));
a (IMP_RES_TAC (TAUT_RULE ⌜x2 ∈z c ⇒ ¬x2 ∈z c ⇒ c ∈z x2⌝));
```

subgoal 2

SML
```
a (rewrite_tac [Trans]);
a (REPEAT strip_tac);
a (RES_TAC THEN IMP_RES_TAC ZF_thm17);
val ZF_thm18 = save_pop_thm "ZF_thm18";
```

Successor:

SML
```
val Sc = new_definition("Sc",⌜
        (Sc:ZFSET→BOOL) x = ∃(y:ZFSET)• On y ∧ (suc y = x)
⌝);
```

Limit ordinals:

SML
```
val Lim = new_definition("Lim",⌜
        (Lim:ZFSET→BOOL) x = On x ∧ x ≠ ∅z ∧ ¬ (Sc x)
⌝);
```

Natural numbers:

SML
```
val N = new_definition("N",⌜
        (N:ZFSET→BOOL) x =
                        On x
                ∧       ((x = ∅z) ∨ (Sc x))
                ∧        ∀(y:ZFSET)•(y ∈z x ⇒ (y = ∅z) ∨ (Sc y))
⌝);
```

The empty set is a natural number.

SML
```
val ZF_thm19 = prove_thm("ZF_thm19",⌜
        N ∅z
⌝,
        rewrite_tac [N;ZF_thm16;ZF_le2]);
```

Every non-zero Natural number is a successor.

SML
```
push_goal([],⌜
        ∀(x:ZFSET)• N x ∧ x ≠ ∅z ⇒ Sc x
⌝);
a (REPEAT strip_tac);
a (DEF_RES_TAC N);
a (DEF_RES_TAC ≠);
val ZF_le11 = save_pop_thm "ZF_le11";
```

Every natural number is transitive and connected.

SML
```
push_goal([],⌜
        ∀(x:ZFSET)• N x ⇒ Trans x ∧ Con x
⌝);
a (rewrite_tac[N; On]);
a (REPEAT strip_tac THEN asm_rewrite_tac []);
val ZF_le12 = save_pop_thm "ZF_le12";
```

Zero is not the successor of any set.

SML
```
push_goal([],⌜
        ∀(x1:ZFSET)• ∅z ≠ suc x1
⌝);
a (rewrite_tac [≠; suc; ZF_le1;ZF_le2]);
a (strip_tac THEN NOT_FORALL_TAC);
a (EXISTS_TAC ⌜x1:ZFSET⌝);
a (rewrite_tac [ZF_thm10;ZF_thm9]);
val ZF_thm20 = save_pop_thm "ZF_thm20";
```

The successor of an ordinal is an ordinal.

SML

```
push_goal([],⌜
        ∀(x1:ZFSET)• On x1 ⇒ On (suc x1)
⌝);
a (strip_tac THEN strip_tac);
a (rewrite_tac [On; suc]);
a (DEF_RES_TAC On);
a (DEF_RES_TAC (rewrite_rule [⊆z] Trans));
a strip_tac;
a (rewrite_tac [Trans; ZF_thm9;ZF_thm10]);
a (REPEAT strip_tac);
a (DEF_RES_TAC Trans);
a (IMP_RES_TAC
        (((SPECL [⌜y:ZFSET⌝;⌜x1:ZFSET⌝;⌜(x1:ZFSET) ∪z (unit x1)⌝]) o
        (TAUT_rewrite_rule ⌜a ∧ b ⇒ c = a ⇒ b ⇒ c⌝)) ZF_thm7));
a (RMP_TAC ⌜(x1:ZFSET) ⊆z (x1 ∪z (unit x1))⌝);
a (asm_rewrite_tac[]);
a (pure_rewrite_tac [⊆z; ZF_thm10]);
a (TAUT_SIMP_TAC THEN rewrite_tac[]);
a (asm_rewrite_tac [⊆z; ZF_thm10]
        THEN TAUT_SIMP_TAC THEN rewrite_tac[]);

a(pure_rewrite_tac[ZF_le7;≠;ZF_thm9;ZF_thm10]);
a (DEF_RES_TAC ZF_le7);
a (REPEAT strip_tac);
```

Leaving four sugoals

first

SML

```
a (accept_tac (MP (MP (SPEC_ALL (ASSUME
        ⌜∀y z• y ∈z x1 ⇒ z ∈z x1 ⇒ (z = y) ∨ z ∈z y ∨ y ∈z z⌝))
        (ASSUME ⌜y ∈z x1⌝)) (ASSUME ⌜z ∈z x1⌝)));
```

second

SML

```
a (rewrite_tac [rewrite_rule [SYM (ASSUME ⌜z = x1⌝)] (ASSUME ⌜y ∈z x1⌝)]);
```

third

SML

```
a (rewrite_tac [rewrite_rule [SYM (ASSUME ⌜y = x1⌝)] (ASSUME ⌜z ∈z x1⌝)]);
```

fourth

SML

```
a (rewrite_tac [rewrite_rule [SYM (ASSUME ⌜y = x1⌝)] (ASSUME ⌜z = x1⌝)]);
val ZF_thm21 = save_pop_thm "ZF_thm21";
```

The successor of a natural number is a natural number.

SML

```
push_goal([],⌜
        ∀(x1:ZFSET)• N x1 ⇒ N (suc x1)
⌝);
a (pure_rewrite_tac [N]);
a (strip_tac THEN strip_tac THEN IMP_RES_TAC ZF_thm21
                THEN asm_rewrite_tac[]);

a strip_tac;
a DISJ2_TAC;
a (pure_rewrite_tac [Sc]);
a (EXISTS_TAC ⌜∅z⌝);
a (rewrite_tac[ZF_thm16]);

a strip_tac;
a (rewrite_tac [suc;Sc;ZF_thm10;ZF_le2;ZF_thm9]);
a TAUT_TAC;

a strip_tac;
a DISJ2_TAC;
a (pure_rewrite_tac [suc; Sc]);
a (EXISTS_TAC ⌜x1:ZFSET⌝);
a (asm_rewrite_tac[]);

a (pure_rewrite_tac [suc; ZF_thm9; ZF_thm10]);
a (REPEAT strip_tac);
a (RES_TAC THEN asm_rewrite_tac []);
a DISJ2_TAC;
a (accept_tac (rewrite_rule [SYM (ASSUME ⌜(y:ZFSET) = x1⌝)] (ASSUME ⌜Sc x1⌝)));

val ZF_thm22 = save_pop_thm "ZF_thm22";
```

The successor function is injective.

SML

```
push_goal([],⌜
        ∀(x1:ZFSET)(x2:ZFSET)• (suc x1 = suc x2) ⇒ (x1 = x2)
⌝);
a (TAUT_rewrite_tac ⌜a ⇒ b = ¬b ⇒ ¬a⌝);
a (strip_tac THEN strip_tac THEN strip_tac);
a (pure_rewrite_tac [suc; ZF_le1; ZF_thm10]);
a (pure_rewrite_tac [ZF_thm9]);
a (TAUT_rewrite_tac ⌜(a = b) = (a ⇒ b) ∧ (b ⇒ a)⌝);
a strip_tac;
a (ASSUME_TAC (REFL ⌜x1⌝));
a (ASSUME_TAC (REFL ⌜x2⌝));
a RES_TAC;

a (MP_TAC ZF_thm13);
a (TAUT_rewrite_tac ⌜a ⇒ F = ¬a⌝);
a (NOT_FORALL_TAC THEN NOT_FORALL_TAC);
a (EXISTS_TAC ⌜x1:ZFSET⌝);
a (EXISTS_TAC ⌜x2:ZFSET⌝);
a (asm_rewrite_tac [∉z]);

a (MP_TAC (ASSUME ⌜¬(x1 = x2)⌝) THEN rewrite_tac [ASSUME ⌜x2 = x1⌜]);

val ZF_thm23 = save_pop_thm "ZF_thm23";
```

Every member of a natural number is a natural number.

SML

```
push_goal([],⌜
        ∀(x1:ZFSET)(x2:ZFSET)• N x1 ∧ x2 ∈z x1 ⇒ N x2
⌝);
a (pure_rewrite_tac [N]);
a (strip_tac THEN strip_tac);
a (TAUT_rewrite_tac ⌜a ∧ b ⇒ c = b ⇒ a ⇒ c⌝);
a strip_tac;
a (LEMMA ⌜x2 ∈z x1 ⇒ ¬(x1 = ∅z)⌝);
a RES_TAC;
a (asm_rewrite_tac[]);
a strip_tac;
a (IMP_RES_TAC ZF_thm18);
a (asm_rewrite_tac []);
a strip_tac;
a (IMP_RES_TAC (SPEC ⌜x2:ZFSET⌝
        (ASSUME ⌜∀y• y ∈z x1 ⇒ (y = ∅z) ∨ Sc y⌝))
        THEN asm_rewrite_tac[]);

a (DEF_RES_TAC On);
a (DEF_RES_TAC Trans);

a strip_tac;
a strip_tac;
a RES_TAC;
a (MP_TAC (rewrite_rule [ASSUME ⌜x2 = ∅z⌝] (ASSUME ⌜y ∈z x2⌝)));
a (rewrite_tac [SPEC ⌜y:ZFSET⌝ ZF_le2]);

a (DEF_RES_TAC ⊆z);
a (RES_TAC THEN asm_rewrite_tac[]);

a (TAUT_rewrite_tac ⌜a ⇒ ¬b = b ⇒ ¬a⌝);
a (strip_tac THEN rewrite_tac[ASSUME ⌜x1 = ∅z⌝;ZF_le2]);

val ZF_thm24 = save_pop_thm "ZF_thm24";
```

The principle of induction over the natural numbers.

SML

```
push_goal([],⌜
        ∀(A:ZFSET → BOOL)• A ∅z ∧ (∀(x:ZFSET)• N x ∧ A x ⇒ A (suc x))
                                  ⇒ ∀(x:ZFSET)• N x ⇒ A x
⌝);
a (strip_tac THEN strip_tac);
a (TAUT_rewrite_tac ⌜a = ¬¬a⌝);
a PURE_NOT_FORALL_TAC;
a (TAUT_rewrite_tac ⌜¬(a ⇒ b) = a ∧ ¬b⌝);
a strip_tac;

lemma ⌜∃L:ZFSET• (L = Λz (suc x) λx:ZFSET• ¬ (A x))⌝;

 a (EXISTS_TAC ⌜Λz (suc x) λx:ZFSET• ¬ (A x)⌝);
 a (asm_rewrite_tac[]);

lemma ⌜x ∈z L⌝;

 a (asm_rewrite_tac[ZF2;suc;ZF_thm9;ZF_thm10]);
 a (BETA_TAC THEN asm_rewrite_tac[]);

lemma ⌜∀(y:ZFSET)• y ∈z L ⇒ ¬ ((A:ZFSET→BOOL) y)⌝;

 a (REPEAT strip_tac);
 a (MP_TAC (ASSUME ⌜L = Λz(suc x)(λx• ¬A x)⌝));
 a (pure_rewrite_tac [ZF_le1; ZF2]);
 a BETA_TAC;
 a (TAUT_rewrite_tac ⌜a ⇒ F = ¬a⌝);
 a NOT_FORALL_TAC;
 a (EXISTS_TAC ⌜y:ZFSET⌝);
 a (asm_rewrite_tac[]);

lemma ⌜L ≠ ∅z⌝;

 a (pure_rewrite_tac[≠;ZF_le1]);
 a NOT_FORALL_TAC;
 a (EXISTS_TAC ⌜x:ZFSET⌝);
 a (asm_rewrite_tac[ZF_le2]);

lemma ⌜∃b:ZFSET• b ∈z L ∧ (b ∩z L = ∅z)⌝;

 a (IMP_RES_TAC ZF7);
```



```
lemma ⌜L ⊆z (suc x)⌝;

 a (rewrite_tac[⊆z]);
 a (asm_rewrite_tac[ZF2]);
 a TAUT_SIMP_TAC;
```

No the example of proof by induction.

2.2 The completing axioms

2.2.1 The axiom of infinity

```
SML
|new_const ("ω",⌜:ZFSET⌝);
|val ZF8 = new_axiom ("ZF8",⌜
|        ∀(x1:ZFSET)• x1 ∈z ω ⇔ N x1
|⌝);
|val ZF8b = prove_thm ("ZF8b",
|        ⌜∀(x1:ZFSET)• x1 ∈z ω = N x1⌝,
|        once_rewrite_tac[ZF_le0] THEN accept_tac ZF8);
```

2.2.2 Replacement

```
SML
|val ZF9 = new_axiom ("ZF9",⌜
|        ∀(f:ZFSET→ZFSET→BOOL)(r:ZFSET)•(∀(x:ZFSET)(y:ZFSET)(z:ZFSET)•
|        (f x y ∧ f x z ⇒ (z = y))) ⇒
|                ∃(w:ZFSET)•∀(y:ZFSET)•y ∈z w ⇔ ∃(x:ZFSET)•x ∈z r ∧ f x y
|⌝);
|val ZF9b = prove_thm ("ZF9b",
|        ⌜∀(f:ZFSET→ZFSET→BOOL)(r:ZFSET)•(∀(x:ZFSET)(y:ZFSET)(z:ZFSET)•
|        (f x y ∧ f x z ⇒ (z = y))) ⇒
|                ∃(w:ZFSET)•∀(y:ZFSET)•y ∈z w = ∃(x:ZFSET)•x ∈z r ∧ f x y⌝,
|        once_rewrite_tac[ZF_le0] THEN accept_tac ZF9);
```

2.3 Relations, functions, and simple recursion

Relations:

```
SML
|val relation = new_definition ("relation",⌜
|        (relation:ZFSET → BOOL) x = ∀(y:ZFSET)• y ∈z x ⇒ ∃(w:ZFSET)(z:ZFSET)• y = (w ↦z z)
|⌝);
```

Functions:

SML

```
val function = new_definition ("function",⌜
        (function:ZFSET → BOOL) x =
                relation x ∧
                ∀(y:ZFSET)(w:ZFSET)(z:ZFSET)• (y ↦z z) ∈z x ∧ (y ↦z w) ∈z x ⇒ (z = w)
⌝);

val domain = new_definition ("domain",⌜
        (domain:ZFSET→ZFSET) x = Λz (⋃z (⋃z x))  λ y:ZFSET• ∃ z:ZFSET• (y ↦z z) ∈z x
⌝);

val image = new_definition ("image",⌜
        (image:ZFSET → ZFSET) x = Λz (⋃z (⋃z x)) λ y:ZFSET• ∃ z:ZFSET• (z ↦z y) ∈z x
⌝);

val z = new_infix_definition ("z",⌜
        (z:ZFSET → ZFSET → ZFSET) x z = ⋃z (Λz (image x) λ y:ZFSET• (z ↦z y) ∈z x)
⌝);

val ↔z = new_infix_definition("↔z",⌜
        (↔z:ZFSET → ZFSET → ZFSET) t r =
                Λz (ℙ(ℙ(t ∪z r))) λ(y:ZFSET)•∃(u:ZFSET)(v:ZFSET)• (y = (u ↦z v)) ∧ u ∈z t ∧ v ∈z
⌝);
```

2.4 The Axiom of Choice

The axiom of choice is here introduced as an axiom. This isn't strictly necessary since there is a choice function in HOL already, and the choice function in set theory could be defined using the HOL choice function. However the presentation follows closer to Hatcher by just introducing Hatcher's choice axiom.

SML

```
new_const ("σ",⌜:ZFSET → ZFSET⌝);
val ZF10 = new_axiom ("ZF10",⌜
        ∀ x1:ZFSET • x1 ≠ ∅z ⇒ (σ x1) ∈z x1
⌝);
```

3 REFORMULATION of REPLACEMENT

```
SML
|push_goal([],⌜
|        ∀(f:ZFSET → ZFSET)(d:ZFSET)• ∃(s:ZFSET)• ∀(x:ZFSET)•
|                        x ∈z s = ∃(y:ZFSET)• y ∈z d ∧ (x = f y)
|⌝);
|a (REPEAT GEN_TAC);
|lemma_proof ⌜∃(rel:ZFSET → ZFSET → BOOL)• ∀(x:ZFSET)(y:ZFSET)• rel x y = (y = f x)⌝
|        [EXISTS_TAC ⌜λ(x:ZFSET)(y:ZFSET)• y = f x⌝; REPEAT GEN_TAC;
|        BETA_TAC; REFL_TAC];
|lemma_proof ⌜∀(x:ZFSET)(y:ZFSET)(z:ZFSET)• rel x y ∧ rel x z ⇒ (z = y)⌝
|        [asm_rewrite_tac[] THEN REPEAT strip_tac THEN asm_rewrite_tac[]];
|lemma_proof ⌜∃w• ∀y• y ∈z w = (∃x• x ∈z d ∧ rel x y)⌝
|        [IMP_RES_TAC (SPECL [⌜rel:ZFSET→ZFSET→BOOL⌝;⌜d:ZFSET⌝] ZF9b)];
|a (EXISTS_TAC ⌜w:ZFSET⌝ THEN asm_rewrite_tac[]);
|val ZF_thm26 = save_pop_thm "ZF_thm26";
```

4 THE THEORY