Zermelo-Fraenkel set theory in HOL (part 2)

Roger Bishop Jones

ICL Defence Systems

ABSTRACT

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

1. INTRODUCTION

ZF in HOL part 2. Part 1 is DS/FM/RBJ/120, which includes all Hatcher's definitions and the first twenty-five of his theorems. This theory provides further material as required for applications of the theory.

```
ML
new_theory‘zf122‘;;
new_parent‘zf120‘;;
loadf‘/escher/usr2/projects/infra/pholfiles/TAUT‘;;
load_definitions ‘zf120‘;;
load_theorems ‘zf120‘;;
load_axioms ‘zf120‘;;
delete_cache‘zf120‘;;
let NOT_FORALL_TAC = REWRITE_TAC[NOT_FORALL] THEN BETA_TAC;;
let PURE_NOT_FORALL_TAC = PURE_REWRITE_TAC[NOT_FORALL] THEN BETA_TAC;;
let NOT_EXISTS_TAC = REWRITE_TAC[NOT_EXISTS] THEN BETA_TAC;;
let PURE_NOT_EXISTS_TAC = PURE_REWRITE_TAC[NOT_EXISTS] THEN BETA_TAC;;
```

2. ORDERED PAIRS

```
ML
let field_DEF = new_definition(‘field_DEF‘,"
        (field:SET→SET) s = (domain s) ∪ (image s)
");;
```

2.1. Left projection

```
ML
let fst_DEF = new_definition(‘fst_DEF‘,"
        (fst:SET→SET) s = ∪(∩ s)
");;
```

```
ML
set_goal([],"
        ∀(x:SET)(y:SET)• fst(x ↦ y) = x
");;
expand (PURE_REWRITE_TAC[fst_DEF; ∩_DEF; ↦_DEF; ZF_le1; ZF2; ZF6]);;
expand (REWRITE_TAC [ZF5] THEN BETA_TAC THEN REPEAT STRIP_TAC THEN EQ_TAC);;
expand STRIP_TAC;;
expand (ACCEPT_TAC (
        REWRITE_RULE
                [REWRITE_RULE [ZF_thm9]
                        (REWRITE_RULE [ASSUMP "z’’ = unit x"] (ASSUMP "z’ ∈ z’’"))]
                                (ASSUMP "z ∈ z’")));;
expand (ASSUME_TAC (REFL "unit x"));;
expand RES_TAC;;
expand (ACCEPT_TAC (REWRITE_RULE [REWRITE_RULE [ZF_thm9]
                (ASSUMP "z’ ∈ (unit x)")] (ASSUMP "z ∈ z’")));;
expand (STRIP_TAC THEN EXISTS_TAC "x:SET" THEN ASM_REWRITE_TAC[]);;
expand STRIP_TAC;;
expand (EXISTS_TAC  "unit x");;
expand (ASM_REWRITE_TAC[ZF_thm9]);;
expand (REPEAT STRIP_TAC);;
expand (ASM_REWRITE_TAC [ZF_thm9]);;
expand (ASM_REWRITE_TAC [ZF5]);;
let ZF2_thm1 = save_top_thm ‘ZF2_thm1‘;;
```

2.2. Difference

```
ML
let dif_DEF = new_infix_definition(‘dif_DEF‘,"
        (dif:SET→SET→SET) s t = sep s λx:SET• x ∉ t
");;
```

```
ML
set_goal([],"
        ∀(x:SET)(v:SET)(w:SET)• (pair x v = pair x w) = (v = w)
");;
expand(REWRITE_TAC [SPECL ["pair x v";"pair x w"] ZF_le1]);;
expand (REWRITE_TAC [ZF5]);;
expand (REPEAT STRIP_TAC THEN EQ_TAC
        THEN REPEAT STRIP_TAC THEN ASM_REWRITE_TAC[]);;
lemma_proof "(v = x) ∨ (v = w)"
        [REWRITE_TAC [REWRITE_RULE [] (SPEC "v:SET"
        (ASSUME "∀z• (z = x) ∨ (z = v) = (z = x) ∨ (z = w)"))]];;
lemma_proof "(w = x) ∨ (w = v)"
        [REWRITE_TAC [REWRITE_RULE [] (SPEC "w:SET"
        (ASSUMP "∀z• (z = x) ∨ (z = v) = (z = x) ∨ (z = w)"))]];;
expand (ASM_REWRITE_TAC[]);;
expand (ASM_REWRITE_TAC[]);;
let ZF2_le1 = save_top_thm ‘ZF2_le1‘;;
```

```
ML
set_goal([],"
        ∀(x:SET)(y:SET)(z:SET)•
                z ∈ (x dif y) = z ∈ x ∧ z ∉ y
");;
expand (REWRITE_TAC [dif_DEF; NE_DEF; ∉_DEF; ZF2]);;
expand (BETA_TAC THEN REPEAT STRIP_TAC);;
expand (EQ_TAC THEN REPEAT STRIP_TAC THEN ASM_REWRITE_TAC[] THEN RES_TAC);;
let ZF2_le2 = save_top_thm ‘ZF2_le2‘;;
```

2.3. Right projection

```
ML
let snd_DEF = new_definition(‘snd_DEF‘,"
        (snd:SET→SET) s = (s = unit(unit (fst s))) => fst s
                                | ∪((∪ s) dif (unit(fst s)))
");;
```

```
ML
set_goal([],"
        ∀(x:SET)(y:SET)• snd(x ↦ y) = y
");;
expand (PURE_REWRITE_TAC[snd_DEF; ZF2_thm1] THEN REPEAT STRIP_TAC);;
expand (ASM_CASES_TAC "x = y" THEN ASM_REWRITE_TAC[↦_DEF; unit_DEF]);;
lemma "¬(x = y) ⇒ ¬(pair(pair x x)(pair x y) = pair(pair x x)(pair x x))";;
 expand (TAUT_REWRITE_TAC " ¬a ⇒ ¬b = b ⇒ a");;
 expand STRIP_TAC;;
 lemma "pair x y = pair x x";;
   expand (DEF_RES_TAC (SPECL
        ["pair x x";"pair x y";"pair x x"] ZF2_le1));;
 expand (DEF_RES_TAC (SPECL
        ["x:SET";"y:SET";"x:SET"] ZF2_le1));;
 expand (ACCEPT_TAC (SYM (ASSUMP "y:SET = x")));;
expand RES_TAC;;
expand (ASM_REWRITE_TAC[ZF_le1]);;
expand (REWRITE_TAC [ZF6; ZF5; ZF2_le2; ∉_DEF]);;
expand (STRIP_TAC THEN EQ_TAC THEN STRIP_TAC);;
expand (CONTR_TAC (REWRITE_RULE
                [ASSUMP "z'' = pair x x"; ZF5; ASSUMP "¬(z' = x)"]
                (ASSUMP "z' ∈ z''")));;
expand (ACCEPT_TAC (REWRITE_RULE [REWRITE_RULE
                [ASSUMP "z'' = pair x y"; ZF5; ASSUMP "¬(z' = x)"]
                (ASSUMP "z' ∈ z''")] (ASSUMP "z ∈ z'")));;
expand (EXISTS_TAC "y:SET");;
expand (ASM_REWRITE_TAC[]);;
expand STRIP_TAC;;
expand (EXISTS_TAC "pair x y");;
expand (REWRITE_TAC [ZF5]);;
expand (ASM_REWRITE_TAC []);;
expand (UNDISCH_TAC "¬(x = y)" THEN
        TAUT_REWRITE_TAC "¬a ⇒ ¬b = b ⇒ a" THEN STRIP_TAC THEN ASM_REWRITE_TAC[]);;
let ZF2_thm2 = save_top_thm `ZF2_thm2`;;
```

2.4. Equality of ordered pairs

```
ML
set_goal([],"
        ∀(v:SET)(w:SET)(x:SET)(y:SET)(z:SET)•
                ((v ↦ w) = (x ↦ y)) = (v = x) ∧ (w = y)
");;
expand (REPEAT STRIP_TAC THEN EQ_TAC THEN REPEAT STRIP_TAC);;
expand (ACCEPT_TAC (REWRITE_RULE [ZF2_thm1]
                (AP_TERM "fst" (ASSUME "v ↦ w = x ↦ y"))));;
expand (ACCEPT_TAC (REWRITE_RULE [ZF2_thm2]
                (AP_TERM "snd" (ASSUME "v ↦ w = x ↦ y"))));;
expand (ASM_REWRITE_TAC[]);;
let ZF2_thm3 = save_top_thm ‘ZF2_thm3‘;;
```

3. EXTENSIONALITY OF FUNCTIONS

```
ML
set_goal([],"
        ∀(x:SET)•(∃∀(y:SET)• y ∈ x) ⇒ ∀(y:SET)•y ∈ x ⇒((μy:SET• y ∈ x) = y)
");;
expand (PURE_REWRITE_TAC [EXISTS_UNIQUE_DEF]);;
expand (BETA_TAC THEN BETA_TAC THEN (REPEAT STRIP_TAC));;
expand (IMP_RES_TAC (BETA_RULE (SPEC "λy:SET• y ∈ x"
                (INST_TYPE [":SET",":*"] SELECT_AX))));;
expand (IMP_RES_TAC (SPECL ["μy• y ∈ x";"y"] (ASSUME "∀x’ y• x’ ∈ x ∧ (y ∈ x ⇒ (x’ = y))")));;
let ZF2_l1 = save_top_thm ‘ZF2_l1‘;;
```

```
ML
set_goal([],"
        ∀(x:SET)•(∃∀(y:SET)• y ∈ x)
                        ⇒ (x = unit (μy:SET• y ∈ x))
");;
expand (REPEAT STRIP_TAC);;
expand (IMP_RES_TAC ZF2_l1);;
expand (PURE_REWRITE_TAC [ZF_le1]);;
expand (PURE_REWRITE_TAC [ZF_thm9]);;
expand (STRIP_TAC THEN EQ_TAC);;
expand STRIP_TAC;;
expand RES_TAC;;
expand (ASM_REWRITE_TAC[]);;
expand STRIP_TAC;;
expand (ASM_REWRITE_TAC[]);;
expand (MP_TAC (CONJUNCT1 (BETA_RULE (REWRITE_RULE [EXISTS_UNIQUE_DEF]
                (ASSUME "∃∀ y• y ∈ x")))) THEN STRIP_TAC);;
expand RES_TAC;;
expand (ASM_REWRITE_TAC[]);;
let ZF2_l2 = save_top_thm ‘ZF2_l2‘;;
```

```
ML
set_goal([],"
        ∀(x:SET)(y:SET)(z:SET)• function x ∧ y ∈ domain x
                        ⇒((x © y = z) = (y ↦ z) ∈ x)
");;
expand (PURE_REWRITE_TAC[domain_DEF; ZF2] THEN BETA_TAC);;
expand (REPEAT STRIP_TAC THEN EQ_TAC);;
% 2 subgoals %
```

```
ML
% 1 %
  expand (PURE_REWRITE_TAC [©_DEF]);;
  expand (PURE_REWRITE_TAC [ZF_le1]);;
  expand (PURE_REWRITE_TAC [ZF6;ZF2] THEN BETA_TAC);;
  expand (DEF_RES_TAC function);;
  expand (ASM_CASES_TAC "(z:SET) = z’");;
% 2 subgoals %
```

```
ML
% 1/1 %
    expand (ASM_REWRITE_TAC[]);;
```

```
ML
% 1/2 %
   expand (TAUT_REWRITE_TAC "a ⇒ b = ¬b ⇒ ¬a");;
   expand NOT_FORALL_TAC;;
   expand STRIP_TAC;;
   expand (MP_TAC (ASSUME "¬(z = z')"));;
   expand (SUBST1_TAC (SPECL ["z:SET";"z':SET"] ZF_le1));;
   expand NOT_FORALL_TAC;;
   expand STRIP_TAC;;
   expand (EXISTS_TAC "x':SET");;
   expand (ASM_CASES_TAC "x' ∈ z");;
% 2 subgoals %
```

```
ML
% 1/2/1 %
    expand (ASM_REWRITE_TAC[]);;
    expand NOT_EXISTS_TAC;;
    expand STRIP_TAC;;
    expand (TAUT_REWRITE_TAC "¬ (a ∧ b) = a ⇒ ¬b");;
    expand STRIP_TAC;;
    expand RES_TAC;;
    expand (ONCE_ASM_REWRITE_TAC[]);;
    expand (ACCEPT_TAC (REWRITE_RULE [ASSUME "x' ∈ z"]
            (ASSUME "¬(x' ∈ z = x' ∈ z')")));;
```

```
ML
% 1/2/2 %
    expand (ASM_REWRITE_TAC[]);;
    expand (EXISTS_TAC "z':SET");;
    expand (REWRITE_TAC [REWRITE_RULE [ASSUME "¬x' ∈ z"]
                    (ASSUME "¬(x' ∈ z = x' ∈ z')")]);;
    expand (PURE_REWRITE_TAC[image_DEF;ZF2;ZF6]
            THEN BETA_TAC THEN STRIP_TAC);;
% 2 subgoals %
```

```
ML
% 1/2/2/1 %
     expand STRIP_TAC;;
% 2 subgoals %
```

```
ML
% 1/2/2/1/1 %
        expand (EXISTS_TAC "pair (y:SET) (z':SET)");;
        expand (REWRITE_TAC [ZF5]);;
        expand (EXISTS_TAC "y ↦ z'");;
        expand (ASM_REWRITE_TAC [↦_DEF;ZF5]);;
```

```
ML
% 1/2/2/1/2 %
        expand (EXISTS_TAC "y:SET");;
        expand (ASM_REWRITE_TAC[]);;
```

```
ML
% 1/2/2/2 %
      expand (ASM_REWRITE_TAC[]);;
```

```
ML
% 2 %
 expand (STRIP_TAC THEN PURE_REWRITE_TAC [©_DEF]);;
 expand (PURE_REWRITE_TAC[ZF_le1]);;
 expand (PURE_REWRITE_TAC[ZF6;ZF2] THEN BETA_TAC);;
 expand (DEF_RES_TAC function);;
 expand RES_TAC;;
 expand STRIP_TAC;;
 expand EQ_TAC;;
```

```
ML
% 2/1 %
   expand STRIP_TAC;;
   expand RES_TAC;;
   expand (REWRITE_TAC [ASSUME "z:SET = z''"; ASSUME "z'' ∈ z''"]);;
```

```
ML
% 2/2 %
   expand STRIP_TAC;;
   expand (EXISTS_TAC "z:SET");;
   expand (PURE_ONCE_ASM_REWRITE_TAC []);;
   expand (REWRITE_TAC [image_DEF;ZF2;ZF6] THEN BETA_TAC);;
   expand STRIP_TAC;;
% 2 subgoals %
```

```
ML
% 2/2/1 %
    expand (EXISTS_TAC "pair (y:SET) (z':SET)");;
    expand (REWRITE_TAC [ZF5]);;
    expand (EXISTS_TAC "y ↦ z'");;
    expand (ONCE_ASM_REWRITE_TAC[]);;
    expand (REWRITE_TAC [↦_DEF;ZF5]);;
```

```
ML
% 2/2/2 %
  expand (EXISTS_TAC "y:SET" THEN ONCE_ASM_REWRITE_TAC[]);;
let ZF2_thm4 = save_top_thm 'ZF2_thm4';;
```

```
ML
set_goal([],"
        ∀(w:SET)(x:SET)• (∃(z:SET)• (w ↦ z) ∈ x) ⇒ w ∈ (domain x)
");;
expand (EVERY[
        PURE_REWRITE_TAC [domain_DEF;ZF2;ZF6];
        BETA_TAC;
        REPEAT STRIP_TAC]);;
% 2 subgoals %
```

```
ML
% 1
"∃z• (∃z'• z' ∈ x ∧ z ∈ z') ∧ w ∈ z"
  [ "(w ↦ z) ∈ x" ]
%

expand (EVERY[
        EXISTS_TAC "pair w z";
        REWRITE_TAC [ZF5];
        EXISTS_TAC "w ↦ z";
        ASM_REWRITE_TAC[];
        REWRITE_TAC [↦_DEF;ZF5]]);;
```

```
ML
% 2
"∃z• (w ↦ z) ∈ x"
   [ "(w ↦ z) ∈ x" ]
%
expand (EXISTS_TAC "z:SET" THEN ASM_REWRITE_TAC[]);;
let ZF2_thm6 = save_top_thm ‘ZF2_thm6‘;;
```

```
ML
set_goal([],"
        ∀(x:SET)(y:SET)• x ∈ (domain y) = ∃(z:SET)• (x ↦ z) ∈ y
");;
expand (REPEAT STRIP_TAC THEN EQ_TAC THEN REWRITE_TAC [ZF2_thm6]);;
```

```
sv "HOL output" "x ∈ (domain y) ⇒ (∃z• (x ↦ z) ∈ y)"
```

```
ML
expand (REWRITE_TAC [domain_DEF; ZF2; ZF6]
                THEN BETA_TAC THEN REPEAT STRIP_TAC);;
```

```
"∃z• (x ↦ z) ∈ y"
  [ "z’ ∈ y" ]
  [ "z ∈ z’" ]
  [ "x ∈ z" ]
  [ "(x ↦ z’’) ∈ y" ]
```

```
ML
expand (EXISTS_TAC "z’’:SET" THEN ASM_REWRITE_TAC[]);;
let ZF2_thm11 = save_top_thm ‘ZF2_thm11‘;;
```

```
ML
set_goal([],"
        ∀(x:SET)(y:SET)• x ∈ (image y) = ∃(z:SET)• (z ↦ x) ∈ y
");;
expand (EVERY [REPEAT STRIP_TAC; EQ_TAC;
        REWRITE_TAC [image_DEF; ZF2; ZF6];
        BETA_TAC; STRIP_TAC]);;
```

```
sv "HOL output" 2 subgoals "(∃z• (∃z’• z’ ∈ y ∧ z ∈ z’) ∧ x ∈ z) ∧
 (∃z• (z ↦ x) ∈ y)"
   [ "(z ↦ x) ∈ y" ]
```

```
"∃z• (z ↦ x) ∈ y"
   [ "z' ∈ y" ]
   [ "z ∈ z'" ]
   [ "x ∈ z" ]
   [ "(z'' ↦ x) ∈ y" ]
```

```
ML
expand (EXISTS_TAC "z'':SET" THEN ASM_REWRITE_TAC[]);;
```

```
goal proved
• ⊢ ∃z• (z ↦ x) ∈ y

Previous subproof:
"(∃z• (∃z'• z' ∈ y ∧ z ∈ z') ∧ x ∈ z) ∧
 (∃z• (z ↦ x) ∈ y)"
   [ "(z ↦ x) ∈ y" ]
```

```
ML
expand (STRIP_TAC);;
expand (EXISTS_TAC "pair z x" THEN REWRITE_TAC[ZF5]);;
expand (EXISTS_TAC "z ↦ x" THEN ASM_REWRITE_TAC[↦_DEF; ZF5]);;
expand (EXISTS_TAC "z:SET" THEN ASM_REWRITE_TAC[]);;
let ZF2_thm13 = save_top_thm 'ZF2_thm13';;
```

```
ML
set_goal([],"
        ∀(x:SET)(y:SET)(z:SET)• function x ∧ ¬(y ∈ domain x)
                        ⇒(x © y = ∅)
");;
expand (REWRITE_TAC [ZF_le1; ZF_le2]);;
expand (STRIP_TAC THEN STRIP_TAC);;
expand (TAUT_REWRITE_TAC "a ∧ ¬b ⇒ c = a ⇒ ¬c ⇒ b");;
expand (STRIP_TAC THEN NOT_FORALL_TAC);;
expand (REWRITE_TAC [©_DEF;ZF2;ZF6;domain_DEF] THEN BETA_TAC);;
expand (REPEAT STRIP_TAC);;
expand (EXISTS_TAC "pair (y:SET) (z:SET)");;
expand (REPEAT STRIP_TAC);;
expand (EXISTS_TAC "y ↦ z");;
expand (ASM_REWRITE_TAC[↦_DEF;ZF5]);;
expand (REWRITE_TAC[ZF5]);;
expand (EXISTS_TAC "z:SET" THEN ASM_REWRITE_TAC[]);;
let ZF2_thm5 = save_top_thm 'ZF2_thm5';;
```

```
ML
set_goal([],"
        ∀(x:SET)(y:SET)(z:SET)• (x ↦ y) ∈ z ⇒ x ∈ (domain z) ∧ y ∈ (image z)
");;
expand (PURE_REWRITE_TAC [domain_DEF; image_DEF;ZF2;ZF6]);;
expand BETA_TAC;;
expand (REPEAT STRIP_TAC);;

expand (EVERY[
        EXISTS_TAC "pair x y";
        STRIP_TAC THENL [   EXISTS_TAC "x ↦ y" THEN   ASM_REWRITE_TAC [];
                            ALL_TAC];
        REWRITE_TAC [↦_DEF;ZF5]]);;

expand (EXISTS_TAC "y:SET" THEN ASM_REWRITE_TAC[]);;

expand (EVERY[
        EXISTS_TAC "pair x y";
        STRIP_TAC THENL [   EXISTS_TAC "x ↦ y" THEN   ASM_REWRITE_TAC [];
                            ALL_TAC];
        REWRITE_TAC [↦_DEF;ZF5]]);;

expand (EXISTS_TAC "x:SET" THEN ASM_REWRITE_TAC[]);;

let ZF2_thm7 = save_top_thm ‘ZF2_thm7‘;;
```

```
ML
set_goal([],"
        ∀(x:SET)(y:SET)•
        function x ∧ function y   ∧ ((domain x) = (domain y))
                ⇒ ((x = y) = (∀(z:SET)• x © z = y © z))
");;
expand (EVERY[
        STRIP_TAC; STRIP_TAC;
        SUBST1_TAC (SPECL ["domain x";"domain y"] ZF_le1);
        REPEAT STRIP_TAC;
        EQ_TAC]);;
```

2 subgoals:

```
"(∀z• x © z = y © z) ⇒ (x = y)"
   [ "function x" ]
   [ "function y" ]
   [ "∀z• z ∈ (domain x) = z ∈ (domain y)" ]

"(x = y) ⇒ (∀z• x © z = y © z)"
   [ "function x" ]
   [ "function y" ]
   [ "∀z• z ∈ (domain x) = z ∈ (domain y)" ]
```

```
ML
expand (STRIP_TAC THEN ASM_REWRITE_TAC[]);;
```

```
Previous subproof:
"(∀z• x © z = y © z) ⇒ (x = y)"
   [ "function x" ]
   [ "function y" ]
   [ "∀z• z ∈ (domain x) = z ∈ (domain y)" ]
```

```
ML
expand (EVERY[
        SUBST1_TAC (SPEC_ALL ZF_le1);
        STRIP_TAC; STRIP_TAC; EQ_TAC; STRIP_TAC]);;
```

```
2 subgoals
"z ∈ x"
   [ "function x" ]
   [ "function y" ]
   [ "∀z• z ∈ (domain x) = z ∈ (domain y)" ]
   [ "∀z• x © z = y © z" ]
   [ "z ∈ y" ]

"z ∈ y"
   [ "function x" ]
   [ "function y" ]
   [ "∀z• z ∈ (domain x) = z ∈ (domain y)" ]
   [ "∀z• x © z = y © z" ]
   [ "z ∈ x" ]
```

```
ML
expand (LEMMA_PROOF "∀z• z ∈ (domain x) ⇒ z ∈ (domain y)"
        [ASM_REWRITE_TAC[]]);;

lemma "∃(w:SET)(z':SET)• z = w ↦ z'";;
  expand (DEF_RES_TAC function);;
  expand (DEF_RES_TAC relation);;

expand (ONCE_ASM_REWRITE_TAC[]);;
lemma "(w ↦ z') ∈ x";;
    expand(ONCE_ASM_REWRITE_TAC [SYM (ASSUME"z = w ↦ z'")]);;
    expand(ONCE_ASM_REWRITE_TAC []);;

lemma "w ∈ (domain x)";;
  expand (IMP_RES_TAC ZF2_thm7);;
expand RES_TAC;;

lemma "x © w = z'";;
  expand (IMP_RES_TAC (SPECL ["x:SET";"w";"z':SET"] ZF2_thm4));;
  expand (ASM_REWRITE_TAC []);;

lemma "y © w = z'";;
  expand (REWRITE_TAC[SYM(SPEC"w:SET"(ASSUME "∀z• x © z = y © z"))]);;
  expand (ASM_REWRITE_TAC[]);;

expand (IMP_RES_TAC (SPECL ["y:SET";"w:SET";"z':SET"] ZF2_thm4));;
expand (REWRITE_TAC [SYM (ASSUME "(y © w = z') = (w ↦ z') ∈ y")]);;
expand (ACCEPT_TAC(ASSUME "y © w = z'"));;
```

```
ML
expand (LEMMA_PROOF "∀z• z ∈ (domain y) ⇒ z ∈ (domain x)"
        [ASM_REWRITE_TAC[]]);;

lemma "∃(w:SET)(z':SET)• z = w ↦ z'";;
  expand (DEF_RES_TAC function);;
  expand (DEF_RES_TAC relation);;

expand (ONCE_ASM_REWRITE_TAC[]);;
lemma "(w ↦ z') ∈ y";;
    expand(ONCE_ASM_REWRITE_TAC [SYM (ASSUME"z = w ↦ z'")]);;
    expand(ONCE_ASM_REWRITE_TAC []);;

lemma "w ∈ (domain y)";;
  expand (IMP_RES_TAC ZF2_thm7);;
expand RES_TAC;;

lemma "y © w = z'";;
  expand (IMP_RES_TAC (SPECL ["y:SET";"w";"z':SET"] ZF2_thm4));;
  expand (ASM_REWRITE_TAC []);;

lemma "x © w = z'";;
  expand (REWRITE_TAC[SYM(SPEC"w:SET"(ASSUME "∀z• x © z = y © z"))]);;
  expand (ASM_REWRITE_TAC[]);;

expand (IMP_RES_TAC (SPECL ["x:SET";"w:SET";"z':SET"] ZF2_thm4));;
expand (REWRITE_TAC [SYM (ASSUME "(x © w = z') = (w ↦ z') ∈ x")]);;
expand (ACCEPT_TAC(ASSUME "x © w = z'"));;
```

```
ML
let ZF2_thm8 = save_top_thm ‘ZF2_thm8‘;;
```

```
ML
set_goal([],"
        ∀(x:SET)(y:SET)• function (unit (x ↦ y))
");;
expand (REPEAT STRIP_TAC THEN PURE_REWRITE_TAC [function;relation]);;
expand (PURE_REWRITE_TAC [ZF_thm9]);;
expand (REPEAT STRIP_TAC);;
expand (EXISTS_TAC "x:SET" THEN EXISTS_TAC "y:SET" THEN ASM_REWRITE_TAC[]);;
expand (ASSUME_TAC (REWRITE_RULE [SYM (ASSUMP "y' ↦ w = x ↦ y")]
                (ASSUMP "y' ↦ z = x ↦ y")));;
expand (DEF_RES_TAC
                (SPECL ["y':SET";"z:SET";"y':SET";"w:SET";"z:SET"] ZF2_thm3));;
let ZF2_thm9 = save_top_thm 'ZF2_thm9';;
```

4. RELATIONAL OVERRIDE

```
ML
let ⊕_DEF = new_infix_definition('⊕_DEF',"
        (⊕:SET→SET→SET) r1 r2 = sep (r1 ∪ r2)
        (λp:SET• p ∈ r2 ∨
        (p ∈ r1 ∧ (¬(∃(x:SET)(y:SET)(z:SET)• (p = (x ↦ y)) ∧ (x ↦ z) ∈ r2)))
        )
");;
```

```
ML
set_goal([],"
        ∀(x:SET)(y:SET)• function x ∧ function y ⇒ function (x ⊕ y)
");;
expand (REPEAT STRIP_TAC THEN PURE_REWRITE_TAC [function;⊕_DEF]);;
expand STRIP_TAC;;
expand (PURE_REWRITE_TAC [relation; ZF2]);;
expand BETA_TAC;;
expand (PURE_REWRITE_TAC [ZF_thm10]);;
expand (EVERY[
                REPEAT STRIP_TAC;
                DEF_RES_TAC function;
                DEF_RES_TAC relation]);;
```

```
ML
expand (PURE_REWRITE_TAC [ZF2;ZF_thm10] THEN BETA_TAC);;
expand (EVERY[
            REPEAT STRIP_TAC;
            DEF_RES_TAC function;
            DEF_RES_TAC relation;
            RES_TAC]);;
expand (MP_TAC (ASSUMP
    "¬(∃x' y'' z'• (y' ↦ z = x' ↦ y'') ∧ (x' ↦ z') ∈ y)"));;
expand (TAUT_REWRITE_TAC "¬a ⇒ b = ¬b ⇒ a" THEN STRIP_TAC);;
expand (EVERY (map EXISTS_TAC ["y':SET"; "z:SET"; "w:SET"]));;
expand (ASM_REWRITE_TAC[]);;
```

```
ML
expand (MP_TAC (ASSUMP
    "¬(∃x' y'' z• (y' ↦ w = x' ↦ y'') ∧ (x' ↦ z) ∈ y)"));;
expand (TAUT_REWRITE_TAC "¬a ⇒ b = ¬b ⇒ a" THEN STRIP_TAC);;
expand (EVERY (map EXISTS_TAC ["y':SET"; "w:SET"; "z:SET"]));;
expand (ASM_REWRITE_TAC[]);;
let ZF2_thm10 = save_top_thm `ZF2_thm10`;;
```

```
ML
set_goal([],"
    ∀(x:SET)(y:SET)• function x ∧ function y ⇒
        (domain (x ⊕ y) = domain x ∪ (domain y))
");;
expand (EVERY[
                REPEAT STRIP_TAC;
                PURE_REWRITE_TAC [ZF_le1];
                PURE_REWRITE_TAC [⊕_DEF; ZF_thm10; ZF2; ZF6; ZF2_thm11]]);;
expand (EVERY[BETA_TAC; STRIP_TAC; EQ_TAC; STRIP_TAC]);;
```

```
6 subgoals
"∃z'•
 ((z ↦ z') ∈ x ∨ (z ↦ z') ∈ y) ∧
 ((z ↦ z') ∈ y ∨
  (z ↦ z') ∈ x ∧
  ¬(∃x' y' z''• (z ↦ z' = x' ↦ y') ∧ (x' ↦ z'') ∈ y))"
   [ "function x" ]
   [ "function y" ]
   [ "(z ↦ z') ∈ y" ]

"∃z'•
 ((z ↦ z') ∈ x ∨ (z ↦ z') ∈ y) ∧
 ((z ↦ z') ∈ y ∨
  (z ↦ z') ∈ x ∧
  ¬(∃x' y' z''• (z ↦ z' = x' ↦ y') ∧ (x' ↦ z'') ∈ y))"
   [ "function x" ]
   [ "function y" ]
   [ "(z ↦ z') ∈ x" ]

"(∃z'• (z ↦ z') ∈ x) ∨ (∃z'• (z ↦ z') ∈ y)"
   [ "function x" ]
   [ "function y" ]
   [ "(z ↦ z') ∈ y" ]
   [ "(z ↦ z') ∈ x" ]
   [ "¬(∃x' y' z''• (z ↦ z' = x' ↦ y') ∧ (x' ↦ z'') ∈ y)" ]

"(∃z'• (z ↦ z') ∈ x) ∨ (∃z'• (z ↦ z') ∈ y)"
   [ "function x" ]
   [ "function y" ]
   [ "(z ↦ z') ∈ y" ]

"(∃z'• (z ↦ z') ∈ x) ∨ (∃z'• (z ↦ z') ∈ y)"
   [ "function x" ]
   [ "function y" ]
   [ "(z ↦ z') ∈ x" ]
   [ "¬(∃x' y' z''• (z ↦ z' = x' ↦ y') ∧ (x' ↦ z'') ∈ y)" ]

"(∃z'• (z ↦ z') ∈ x) ∨ (∃z'• (z ↦ z') ∈ y)"
   [ "function x" ]
   [ "function y" ]
   [ "(z ↦ z') ∈ x" ]
   [ "(z ↦ z') ∈ y" ]
```

```
ML
expand (DISJ1_TAC THEN EXISTS_TAC "z':SET" THEN ASM_REWRITE_TAC[]);;
expand (DISJ1_TAC THEN EXISTS_TAC "z':SET" THEN ASM_REWRITE_TAC[]);;
expand (DISJ2_TAC THEN EXISTS_TAC "z':SET" THEN ASM_REWRITE_TAC[]);;
expand (DISJ2_TAC THEN EXISTS_TAC "z':SET" THEN ASM_REWRITE_TAC[]);;
```

```
2 subgoals
"∃z'•
 ((z ↦ z') ∈ x ∨ (z ↦ z') ∈ y) ∧
 ((z ↦ z') ∈ y ∨
  (z ↦ z') ∈ x ∧
  ¬(∃x' y' z''• (z ↦ z' = x' ↦ y') ∧ (x' ↦ z'') ∈ y))"
   [ "function x" ]
   [ "function y" ]
   [ "(z ↦ z') ∈ y" ]

"∃z'•
 ((z ↦ z') ∈ x ∨ (z ↦ z') ∈ y) ∧
 ((z ↦ z') ∈ y ∨
  (z ↦ z') ∈ x ∧
  ¬(∃x' y' z''• (z ↦ z' = x' ↦ y') ∧ (x' ↦ z'') ∈ y))"
   [ "function x" ]
   [ "function y" ]
   [ "(z ↦ z') ∈ x" ]
```

```
ML
expand (ASM_CASES_TAC "∃x' y' z''• (z ↦ z' = x' ↦ y') ∧ (x' ↦ z'') ∈ y");;
expand (UNDISCH_TAC "∃x' y' z''• (z ↦ z' = x' ↦ y') ∧ (x' ↦ z'') ∈ y"
        THEN STRIP_TAC);;
expand (EXISTS_TAC "z'':SET");;
expand (DEF_RES_TAC (SPECL ["z:SET";"z':SET";"x':SET";"y':SET";"z:SET"] ZF2_thm3));;
expand (ASM_REWRITE_TAC[]);;
expand (EXISTS_TAC "z':SET" THEN ASM_REWRITE_TAC[]);;
```

```
Previous subproof:
"∃z'•
 ((z ↦ z') ∈ x ∨ (z ↦ z') ∈ y) ∧
 ((z ↦ z') ∈ y ∨
  (z ↦ z') ∈ x ∧
  ¬(∃x' y' z''• (z ↦ z' = x' ↦ y') ∧ (x' ↦ z'') ∈ y))"
   [ "function x" ]
   [ "function y" ]
   [ "(z ↦ z') ∈ y" ]
```

```
ML
expand (ASM_CASES_TAC "∃x' y' z''• (z ↦ z' = x' ↦ y') ∧ (x' ↦ z'') ∈ y");;
expand (UNDISCH_TAC "∃x' y' z''• (z ↦ z' = x' ↦ y') ∧ (x' ↦ z'') ∈ y"
        THEN STRIP_TAC);;
expand (EXISTS_TAC "z'':SET");;
expand (DEF_RES_TAC (SPECL ["z:SET";"z':SET";"x':SET";"y':SET";"z:SET"] ZF2_thm3));;
expand (ASM_REWRITE_TAC[]);;
expand (EXISTS_TAC "z':SET" THEN ASM_REWRITE_TAC[]);;
let ZF2_thm12 = save_top_thm `ZF2_thm12`;;
```

```
ML
set_goal([],"
        ∀(x:SET)(y:SET)• function x ∧ function y ⇒
                image (x ⊕ y) ⊆ (image x) ∪ (image y)
");;
expand (EVERY[
        REWRITE_TAC [⊆_DEF; ZF2; ZF_thm10; ZF2_thm13; ⊕_DEF];
        BETA_TAC; REPEAT STRIP_TAC]);;
```

```
4 subgoals
"(∃z• (z ↦ c) ∈ x) ∨ (∃z• (z ↦ c) ∈ y)"
  [ "function x" ]
  [ "function y" ]
  [ "(z ↦ c) ∈ y" ]
  [ "(z ↦ c) ∈ x" ]
  [ "¬(∃x' y' z'• (z ↦ c = x' ↦ y') ∧ (x' ↦ z') ∈ y)" ]

"(∃z• (z ↦ c) ∈ x) ∨ (∃z• (z ↦ c) ∈ y)"
  [ "function x" ]
  [ "function y" ]
  [ "(z ↦ c) ∈ y" ]

"(∃z• (z ↦ c) ∈ x) ∨ (∃z• (z ↦ c) ∈ y)"
  [ "function x" ]
  [ "function y" ]
  [ "(z ↦ c) ∈ x" ]
  [ "¬(∃x' y' z'• (z ↦ c = x' ↦ y') ∧ (x' ↦ z') ∈ y)" ]

"(∃z• (z ↦ c) ∈ x) ∨ (∃z• (z ↦ c) ∈ y)"
  [ "function x" ]
  [ "function y" ]
  [ "(z ↦ c) ∈ x" ]
  [ "(z ↦ c) ∈ y" ]
```

```
ML
expand (DISJ1_TAC THEN EXISTS_TAC "z:SET" THEN ASM_REWRITE_TAC[]);;
expand (DISJ1_TAC THEN EXISTS_TAC "z:SET" THEN ASM_REWRITE_TAC[]);;
expand (DISJ2_TAC THEN EXISTS_TAC "z:SET" THEN ASM_REWRITE_TAC[]);;
expand (DISJ2_TAC THEN EXISTS_TAC "z:SET" THEN ASM_REWRITE_TAC[]);;
let ZF2_thm14 = save_top_thm 'ZF2_thm14';;
```

```
ML
set_goal([],"
        ∀(x:SET)(y:SET)(z:SET)• function x ∧ function y ⇒
        ¬ (z ∈ domain (x ⊕ y)) ⇒ ¬ (z ∈ (domain x)) ∧ ¬ (z ∈ (domain y))
");;
expand (EVERY[
        TAUT_REWRITE_TAC "¬a ⇒ ¬b ∧ ¬c = b ∨ c ⇒ a";
        REPEAT STRIP_TAC;
        LEMMA_PROOF "domain(x ⊕ y) = domain x ∪ domain y"
                                        [IMP_RES_TAC ZF2_thm12];
        ASM_REWRITE_TAC [ZF_thm10]]);;
let ZF2_thm15 = save_top_thm ‘ZF2_thm15‘;;
```

```
ML
set_goal([],"
        ∀(x:SET)(y:SET)(z:SET)• function x ∧ function y ⇒
        (z ∈ domain (x ⊕ y) = (z ∈ (domain x)) ∨ (z ∈ (domain y)))
");;
expand (EVERY[
        REPEAT STRIP_TAC;
        IMP_RES_TAC ZF2_thm12;
        ASM_REWRITE_TAC [ZF_thm10]]);;
let ZF2_thm16 = save_top_thm ‘ZF2_thm16‘;;
```

```
ML
set_goal([],"
        ∀(x:SET)(y:SET)(z:SET)• function x ∧ function y ⇒
        ((x © z = y © z) = (x © z = ∅) ∧ (y © z = ∅) ∨
                        ∃(w:SET)• (z ↦ w) ∈ x ∧ (z ↦ w) ∈ y)
");;
expand (REPEAT STRIP_TAC THEN EQ_TAC THEN STRIP_TAC);;
expand (TAUT_REWRITE_TAC "a ∨ b = ¬a ⇒ b");;
expand (REPEAT STRIP_TAC);;
expand (EXISTS_TAC "x © z" THEN STRIP_TAC);;
```

```
ML
lemma_proof "¬(x © z = ∅)"
        [UNDISCH_TAC "¬((x © z = ∅) ∧ (y © z = ∅))";
        ASM_REWRITE_TAC[]];;
lemma_proof "z ∈ (domain x)"
        [IMP_RES_TAC (TAUT_REWRITE_RULE "a ∧ ¬b ⇒ c = a ⇒ ¬c ⇒ b"
                (SPECL ["x:SET";"z:SET";"y:SET"] ZF2_thm5))];;
expand (IMP_RES_TAC (SPECL ["x:SET";"z:SET";"x © z"] ZF2_thm4));;
expand (REWRITE_TAC [SYM (ASSUMP "(x © z = x © z) = (z ↦ (x © z)) ∈ x")]);;
```

```
ML
expand (ASM_REWRITE_TAC[]);;
lemma_proof "¬(y © z = ∅)"
        [UNDISCH_TAC "¬((x © z = ∅) ∧ (y © z = ∅))";
        REWRITE_TAC[SYM (ASSUMP "x © z = y © z")]];;
lemma_proof "z ∈ (domain y)"
        [IMP_RES_TAC (TAUT_REWRITE_RULE "a ∧ ¬b ⇒ c = a ⇒ ¬c ⇒ b"
                (SPECL ["y:SET";"z:SET";"x:SET"] ZF2_thm5))];;
expand (IMP_RES_TAC (SPECL ["y:SET";"z:SET";"y © z"] ZF2_thm4));;
expand (REWRITE_TAC [SYM (ASSUMP "(y © z = y © z) = (z ↦ (y © z)) ∈ y")]);;
```

```
ML
expand (ASM_REWRITE_TAC[]);;
lemma_proof "z ∈ domain x"
        [IMP_RES_TAC (SPECL ["z:SET";"w:SET";"x:SET"] ZF2_thm7)];;
expand (IMP_RES_TAC (SPECL ["x:SET";"z:SET";"w"] ZF2_thm4));;
expand (DEF_RES_TAC (SYM (ASSUMP "(x © z = w) = (z ↦ w) ∈ x")));;
expand (ASM_REWRITE_TAC[]);;
lemma_proof "z ∈ domain y"
        [IMP_RES_TAC (SPECL ["z:SET";"w:SET";"y:SET"] ZF2_thm7)];;
expand (IMP_RES_TAC (SPECL ["y:SET";"z:SET";"w"] ZF2_thm4));;
expand (DEF_RES_TAC (SYM (ASSUMP "(y © z = w) = (z ↦ w) ∈ y")));;
expand (ASM_REWRITE_TAC[]);;
let ZF2_thm17 = save_top_thm ‘ZF2_thm17‘;;
```

```
ML
set_goal([],"
        ∀(x:SET)(y:SET)• y ⊆ (x ⊕ y)
");;
expand (PURE_REWRITE_TAC [⊆_DEF; ⊕_DEF]);;
expand (REPEAT STRIP_TAC);;
expand (PURE_REWRITE_TAC [ZF2; ZF_thm10]);;
expand (BETA_TAC THEN ASM_REWRITE_TAC[]);;
let ZF2_thm18 = save_top_thm ‘ZF2_thm18‘;;
```

```
ML
set_goal([],"
        ∀(x:SET)(y:SET)(z:SET)• function x ∧ function y ⇒
                ((x ⊕ y) © z = (z ∈ (domain y) => y © z | x © z))
");;
expand (EVERY[
                REPEAT STRIP_TAC]);;
expand (ASM_CASES_TAC "z ∈ domain (y)" THEN ASM_REWRITE_TAC[]);;
```

```
ML
lemma_proof "function (x ⊕ y)" [IMP_RES_TAC ZF2_thm10];;
expand (IMP_RES_THEN (λx•PURE_REWRITE_TAC[x]) (SPECL ["(x ⊕ y)"] ZF2_thm17));;
expand DISJ2_TAC;;
lemma_proof "∃z’• (z ↦ z’) ∈ y"
        [DEF_RES_TAC (SPECL ["z:SET";"y:SET"] ZF2_thm11)];;
expand (EXISTS_TAC "z’:SET" THEN ASM_REWRITE_TAC[]);;
expand (IMP_RES_TAC (SPECL ["x:SET";"y:SET";"(z ↦ z’)"]
                                (PURE_REWRITE_RULE [⊆_DEF] ZF2_thm18)));;
expand (ASM_CASES_TAC "z ∈ (domain x)");;
```

```
ML
lemma_proof "∃z'• (z ↦ z') ∈ x"
        [DEF_RES_TAC (SPECL ["z:SET";"y:SET"] ZF2_thm11)];;
lemma "(z ↦ z') ∈ (x ⊕ y)";;
expand (PURE_REWRITE_TAC [⊕_DEF; ZF2; ZF_thm10]);;
expand BETA_TAC;;
expand (ASM_REWRITE_TAC[]);;
expand (DISJ2_TAC);;
expand (REPEAT (CHANGED_TAC NOT_EXISTS_TAC));;
expand (REWRITE_TAC [ZF2_thm3]);;
expand (TAUT_REWRITE_TAC "¬((a ∧ b) ∧ c) = (a ⇒ b ⇒ ¬c)");;
expand (EVERY[STRIP_TAC;STRIP_TAC;STRIP_TAC;STRIP_TAC;STRIP_TAC]);;
expand (REWRITE_TAC [SYM (ASSUMP "z = x'")]);;
```

```
ML
lemma "∀(x:SET)• ¬(z ↦ x) ∈ y";;
expand (DEF_RES_TAC (TAUT_REWRITE_RULE "(a = b) = (¬a = ¬b)"
                                (SPECL ["z:SET";"y:SET"] ZF2_thm11)));;
expand (STRIP_TAC THEN REWRITE_TAC
        [CONV_RULE NOT_EXISTS_CONV (ASSUMP "¬(∃z'• (z ↦ z') ∈ y)")]);;
expand (ASM_REWRITE_TAC[]);;
lemma_proof "z ∈ domain (x ⊕ y)"
        [IMP_RES_THEN (λx•ASM_REWRITE_TAC[x; ZF_thm10]) ZF2_thm12];;
lemma_proof "function (x ⊕ y)" [IMP_RES_TAC ZF2_thm10];;
expand (IMP_RES_THEN (DEF_RES_TAC o SYM)
        (SPECL ["x:SET";"z:SET";"z':SET"] ZF2_thm4));;
expand (ASM_REWRITE_TAC[]);;
```

```
ML
lemma_proof "¬z ∈ (domain (x ⊕ y))"
                [IMP_RES_THEN (λx•PURE_REWRITE_TAC[x]) ZF2_thm12;
                ASM_REWRITE_TAC [ZF_thm10]];;
lemma_proof "function (x ⊕ y)" [IMP_RES_TAC ZF2_thm10];;
lemma_proof "((x ⊕ y) © z = ∅) ∧ (x © z = ∅)"
        [IMP_RES_TAC ZF2_thm5;
        IMP_RES_TAC (SPECL ["(x ⊕ y)";"z:SET";"z:SET"] ZF2_thm5);
        ASM_REWRITE_TAC[]];;
expand (ASM_REWRITE_TAC[]);;
let ZF2_thm19 = save_top_thm `ZF2_thm19`;;
```

```
ML
set_goal([],"∀(x:SET)• x ∪ x = x");;
expand (REWRITE_TAC [ZF_le1; ZF_thm10]
        THEN REPEAT STRIP_TAC THEN TAUT_TAC);;
let ZF2_thm34 = save_top_thm ‘ZF2_thm34‘;;
```

```
ML
set_goal([],"
    ∀(x:SET)(y:SET)• x ∪ y = y ∪ x
");;
expand (REWRITE_TAC [ZF_le1; ZF_thm10]
        THEN REPEAT STRIP_TAC THEN TAUT_TAC);;
let ZF2_thm20 = save_top_thm ‘ZF2_thm20‘;;
```

```
ML
set_goal([],"
    ∀(x:SET)(y:SET)(z:SET)• (x ∪ y) ∪ z = x ∪ (y ∪ z)
");;
expand (REWRITE_TAC [ZF_le1; ZF_thm10]
        THEN REPEAT STRIP_TAC THEN TAUT_TAC);;
let ZF2_thm21 = save_top_thm ‘ZF2_thm21‘;;
```

```
ML
set_goal([],"
    ∀(x:SET)(y:SET)(z:SET)• x ∪ (y ∪ z) = (x ∪ y) ∪ z
");;
expand (REWRITE_TAC [ZF_le1; ZF_thm10]
        THEN REPEAT STRIP_TAC THEN TAUT_TAC);;
let ZF2_thm25 = save_top_thm ‘ZF2_thm25‘;;
```

```
ML
set_goal([],"∀(x:SET)• x ⊆ x");;
expand (PURE_REWRITE_TAC[⊆_DEF] THEN REPEAT GEN_TAC THEN TAUT_TAC);;
let ZF2_thm33 = save_top_thm ‘ZF2_thm33‘;;
```

```
ML
set_goal([],"
        ∀(x:SET)(y:SET)(z:SET)• x ⊆ y ∧ y ⊆ z ⇒ x ⊆ z
");;
expand (EVERY[
                PURE_REWRITE_TAC [⊆_DEF];
                REPEAT STRIP_TAC; RES_TAC; RES_TAC]);;
let ZF2_thm22 = save_top_thm ‘ZF2_thm22‘;;
```

```
ML
set_goal([],"
        ∀(x:SET)(y:SET)(z:SET)• x ⊆ y ⇒ (x ∪ z) ⊆ (y ∪ z)
");;
expand (EVERY[
                PURE_REWRITE_TAC [⊆_DEF; ZF_thm10];    REPEAT STRIP_TAC;
                RES_TAC; RES_TAC; ASM_REWRITE_TAC[]]);;
let ZF2_thm23 = save_top_thm ‘ZF2_thm23‘;;
```

```
ML
set_goal([],"
        ∀(x:SET)(y:SET)(z:SET)• x ⊆ y ⇒ (z ∪ x) ⊆ (z ∪ y)
");;
expand (EVERY[
                PURE_REWRITE_TAC [⊆_DEF; ZF_thm10];    REPEAT STRIP_TAC;
                RES_TAC; RES_TAC; ASM_REWRITE_TAC[]]);;
let ZF2_thm24 = save_top_thm ‘ZF2_thm24‘;;
```

```
ML
set_goal([],"
        ∀(x:SET)(y:SET)• function x ∧ function y ⇒
                field (x ⊕ y) ⊆ field x ∪ field y
");;
expand (REWRITE_TAC [field_DEF]);;
expand (REPEAT STRIP_TAC);;
expand (IMP_RES_THEN (λx•PURE_REWRITE_TAC[x]) ZF2_thm12);;
expand (PURE_REWRITE_TAC [ZF2_thm21]);;
expand (TMP_TAC (SPECL
                ["(domain y) ∪ (image(x ⊕ y))";
                "(image x) ∪ ((domain y) ∪ (image y))";
                "domain x"] ZF2_thm24));;
expand (PURE_REWRITE_TAC [ZF2_thm25]);;
expand (PURE_ONCE_REWRITE_TAC [ZF2_thm20]);;
expand (PURE_REWRITE_TAC [ZF2_thm25]);;
expand (TMP_TAC (SPECL
                ["image(x ⊕ y)";"(image y) ∪ (image x)";"domain y"]
                ZF2_thm23));;
expand (PURE_ONCE_REWRITE_TAC [ZF2_thm20]);;
expand (IMP_RES_TAC ZF2_thm14);;
let ZF2_thm26 = save_top_thm ‘ZF2_thm26‘;;
```

5. CARTESIAN PRODUCT

```
ML
set_goal([],"
        ∀(x:SET)(v:SET)(w:SET)• x ∈ v ª w =
                ∃(y:SET)(z:SET)• y ∈ v ∧ z ∈ w ∧ (x = y ↦ z)
");;
expand (REPEAT GEN_TAC THEN REWRITE_TAC [ª_DEF; ZF2; ZF4] THEN BETA_TAC);;
expand (EQ_TAC THEN STRIP_TAC);;
expand (EXISTS_TAC "u:SET" THEN EXISTS_TAC "v’:SET" THEN ASM_REWRITE_TAC[]);;
expand STRIP_TAC;;
expand (ASM_REWRITE_TAC [⊆_DEF; ↦_DEF; ZF4; ZF5; ZF_thm9; ZF_thm10]);;
expand (EVERY[GEN_TAC; STRIP_TAC; ASM_REWRITE_TAC []; REPEAT STRIP_TAC]);;
expand (DEF_RES_TAC ZF_thm9 THEN ASM_REWRITE_TAC[]);;
expand (DEF_RES_TAC ZF5 THEN ASM_REWRITE_TAC[]);;
expand (EXISTS_TAC "y:SET" THEN EXISTS_TAC "z:SET");;
expand (ASM_REWRITE_TAC[]);;
let ZF2_thm27 = save_top_thm ‘ZF2_thm27‘;;
```

6. FUNCTIONAL ABSTRACTION

First we define functional abstraction.

```
ML
let zfabs_DEF = new_definition(‘zfabs‘,"
        (zfabs:SET→(SET→SET)→SET) d val =
                μx:SET• ∀y:SET•
        y ∈ x = ∃(v:SET)(w:SET)• (y = v ↦ w) ∧ (w = val v) ∧ (v ∈ d)
");;
```

The following property characterises functional abstraction.

```
ML
set_goal([],"
        ∀(d:SET)(val:SET→SET)(y:SET)•   y ∈ (zfabs d val) =
                ∃(v:SET)(w:SET)• (y = v ↦ w) ∧ (w = val v) ∧ (v ∈ d)");;
expand (PURE_REWRITE_TAC [zfabs_DEF] THEN REPEAT GEN_TAC);;
lemma "∃(x:SET)•
        ∀y• y ∈ x = (∃v w• (y = v ↦ w) ∧ (w = val v) ∧ v ∈ d)";;
 expand(REPEAT STRIP_TAC);;
 lemma_proof "∃(s:SET)• ∀(x:SET)• x ∈ s = ∃(y:SET)• y ∈ d ∧ (x = val y)"
        [REWRITE_TAC [ZF_thm26]];;
 expand (EXISTS_TAC "sep (d ^ a s) (λ(x:SET)• val (fst x) = snd x)");;
 expand (REWRITE_TAC [ZF2; ZF2_thm27]);;
 expand (BETA_TAC THEN REPEAT STRIP_TAC);;
 expand (EQ_TAC THEN REPEAT STRIP_TAC);;

 expand (EXISTS_TAC "y’:SET" THEN EXISTS_TAC "z:SET" THEN ASM_REWRITE_TAC[]);;
 expand (UNDISCH_TAC "val(fst y) = snd y"
                THEN ASM_REWRITE_TAC[ZF2_thm1; ZF2_thm2]);;
 expand (STRIP_TAC THEN ASM_REWRITE_TAC[]);;

 expand (EXISTS_TAC "v:SET" THEN EXISTS_TAC "w:SET" THEN ASM_REWRITE_TAC[]);;
 expand (EXISTS_TAC "v:SET");;
 expand (ASM_REWRITE_TAC[]);;

 expand (ASM_REWRITE_TAC[ZF2_thm1; ZF2_thm2]);;
expand (IMP_RES_TAC (BETA_RULE (SPECL
        ["λx:SET•∀y• y ∈ x = (∃v w• (y = v ↦ w) ∧ (w = val v) ∧ v ∈ d)";"x:SET"]        (INST_TYPE [":SET",":*"] S
expand (ASM_REWRITE_TAC[]);;
let ZF2_thm28 = save_top_thm ‘ZF2_thm28‘;;
```

We prove that functional abstraction always yields a function.

```
ML
set_goal([],"∀(d:SET)(val:SET→SET)• function (zfabs d val)");;
expand (PURE_REWRITE_TAC[ZF2_thm28; function; relation]);;
expand (REPEAT STRIP_TAC);;
expand (EXISTS_TAC "v:SET" THEN EXISTS_TAC "w:SET" THEN ASM_REWRITE_TAC[]);;
expand (UNDISCH_TAC "y ↦ z = v ↦ w'"
        THEN UNDISCH_TAC "y ↦ w = v' ↦ w'"
        THEN REWRITE_TAC[ZF2_thm3]);;
expand (REPEAT STRIP_TAC THEN ASM_REWRITE_TAC[]);;
expand (REWRITE_TAC (map (SYM o ASSUMP) ["y:SET = v";"y:SET = v'"]));;
let ZF2_thm29 = save_top_thm 'ZF2_thm29';;
```

A lemma giving the effect of abstracting over the empty set.

```
ML
set_goal([],"∀(f:SET→SET)• zfabs ∅ f = ∅");;
expand (PURE_REWRITE_TAC[ZF_le1] THEN REWRITE_TAC[ZF2_thm28]);;
expand (REPEAT STRIP_TAC THEN EQ_TAC THEN REPEAT STRIP_TAC
        THEN IMP_RES_TAC ZF_le2);;
let ZF2_thm30 = save_top_thm 'ZF2_thm30';;
```

The following theorem says that the domain of a function constructed using zfabs is just the first argument.

```
ML
set_goal([],"∀(d:SET)(f:SET→SET)• domain(zfabs d f) = d");;
expand (REPEAT GEN_TAC THEN PURE_REWRITE_TAC[ZF_le1]
        THEN PURE_REWRITE_TAC[domain_DEF; ZF2; ZF6; ZF2_thm28; ZF2_thm3]
        THEN BETA_TAC);;
expand (REPEAT STRIP_TAC THEN EQ_TAC THEN REPEAT STRIP_TAC);;
expand (ASM_REWRITE_TAC[]);;
expand (EXISTS_TAC "pair z z" THEN REWRITE_TAC[ZF5]);;
expand (EXISTS_TAC "z ↦ (f z)" THEN ASM_REWRITE_TAC[]);;
expand (STRIP_TAC);;
expand (EVERY (map EXISTS_TAC ["z:SET";"(f:SET→SET) z"])
                THEN ASM_REWRITE_TAC[]);;
expand (REWRITE_TAC[↦_DEF; ZF5; unit_DEF]);;
expand (EVERY (map EXISTS_TAC ["(f:SET→SET) z";"z:SET";"(f:SET→SET) z"])
        THEN ASM_REWRITE_TAC[]);;
let ZF2_thm31 = save_top_thm 'ZF2_thm31';;
```

Now the equivalent of beta reduction, showing the interaction between functional abstraction and function application.

```
ML
set_goal([],"
        ∀(d:SET)(f:SET→SET)(x:SET)•
                x ∈ d ⇒ ((zfabs d f) © x = f x)
");;
expand (REPEAT STRIP_TAC);;
lemma_proof "function (zfabs d f)" [REWRITE_TAC[ZF2_thm29]];;
lemma_proof "x ∈ (domain (zfabs d f))" [ASM_REWRITE_TAC[ZF2_thm31]];;
expand (IMP_RES_TAC (SPECL ["zfabs d f";"x:SET";"(f:SET→SET) x"] ZF2_thm4));;
expand (ASM_REWRITE_TAC[ZF2_thm28]);;
expand (EXISTS_TAC "x:SET" THEN EXISTS_TAC "(f:SET→SET) x"
        THEN ASM_REWRITE_TAC[]);;
let ZF2_thm32 = save_top_thm ‘ZF2_thm32‘;;
```

7. THE THEORY ZF2

```
ML
close_theory‘zf122‘;;
```