Pure function theory in HOL

Roger Bishop Jones

ICL Defence Systems

ABSTRACT

This document consists of the introduction into HOL of a pure theory of (partial) functions.

1. INTRODUCTION

```
ML
new_theory‘pf123‘;;
new_parent‘zf122‘;;
loadf‘/escher/usr2/projects/infra/pholfiles/TAUT‘;;
map load_definitions [‘zf120‘;‘zf122‘];;
map load_theorems [‘zf120‘;‘zf122‘];;
map load_axioms [‘zf120‘;‘zf122‘];;
map delete_cache [‘zf120‘;‘zf122‘];;
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;;
let NEW_GOAL_TAC t = RMP_TAC t THENL [STRIP_TAC; ALL_TAC];;
let new_goal t = expand (NEW_GOAL_TAC t);;
let NEW_GOAL_PROOF_TAC t p = RMP_TAC t THENL [STRIP_TAC THEN p; ALL_TAC];;
let new_goal_proof t p = expand (NEW_GOAL_PROOF_TAC t p);;
```

2. INTRODUCING THE NEW TYPE

2.1. The Defining Property

The following property selects those sets which are obtainable from the empty set by iterating the construction of functions.

The property (of properties) of being "function hereditary" is used to define the collection of functions.

A property is function hereditary if it holds of some set under the following conditions:

1 The set is a function (a many-one relation).

2 Bottom is not in the range of the function.

3 Every element in the domain or range of the function has the property.

```
ML
let ´_DEF = new_definition(‘´_DEF‘,"´:SET = unit(¤  ¤)");;

let function_hereditary_DEF = new_definition(‘function_hereditary_DEF‘,"
        (function_hereditary:(SETbool)bool) p =
        (f:SET) function f  ´  (image f)
                ((x:SET) x  (field f) p x)
         p f
");;
```

A function is any set which has every function-hereditary property.

```
ML
let pure_function_DEF = new_definition(‘pure_function_DEF‘,"
        (pure_function:SETbool) s = (p:SETbool) function_hereditary p  p s
");;
```

We will need to use the following lemma stating that pure_function is function hereditary:

```
ML
set_goal([],"
        function_hereditary pure_function
");;
expand(REWRITE_TAC [function_hereditary_DEF]);;
expand (REPEAT STRIP_TAC);;
expand (REWRITE_TAC [pure_function_DEF]);;
expand (REPEAT STRIP_TAC);;
expand (DEF_RES_TAC function_hereditary_DEF);;
expand RES_TAC;;
```

```
ML
lemma "x x  (field f)  p x";;
 expand (REPEAT STRIP_TAC);;
 expand RES_TAC;;
 expand (DEF_RES_TAC pure_function_DEF);;

expand RES_TAC;;

let PF_t01 = save_top_thm ‘PF_t01‘;;
```

We now obtain the inductive principle that any function_hereditary property is true of all the pure functions:

```
ML
set_goal([],"
        (p:SETbool) function_hereditary p
                (x:SET) pure_function x  p x
");;
expand (REPEAT STRIP_TAC);;
expand (DEF_RES_TAC function_hereditary_DEF);;
expand (DEF_RES_TAC pure_function_DEF);;
let PF_t02 = save_top_thm ‘PF_t02‘;;
```

```
ML
set_goal([],"
        function_hereditary function
");;
expand(REWRITE_TAC [function_hereditary_DEF]);;
expand (REPEAT STRIP_TAC THEN ASM_REWRITE_TAC[]);;
let PF_t03 = save_top_thm ‘PF_t03‘;;
```

```
ML
set_goal([],"
        function_hereditary x:SET(´  image x)
");;
expand(REWRITE_TAC [function_hereditary_DEF] THEN BETA_TAC);;
expand (REPEAT STRIP_TAC THEN ASM_REWRITE_TAC[]);;
let PF_t04 = save_top_thm ‘PF_t04‘;;
```

```
ML
set_goal([],"
        x:SET pure_function x  function x  (´  image x)
");;
expand (STRIP_TAC THEN STRIP_TAC THEN STRIP_TAC);;
expand (ASSUME_TAC PF_t03);;
expand (IMP_RES_TAC PF_t02);;
expand (ASSUME_TAC PF_t04);;
expand (IMP_RES_TAC PF_t02);;
expand (ACCEPT_TAC (BETA_RULE (ASSUME "(x ´  (image x))x")));;
let PF_t05 = save_top_thm ‘PF_t05‘;;
```

```
ML
set_goal([],"
        x:SET pure_function x
                y:SET y  (field x) pure_function y
");;
expand (STRIP_TAC THEN STRIP_TAC THEN STRIP_TAC);;

lemma "function_hereditary (x:SET)
                function x
                ´  (image x)
                (y:SET) y  (field x)  pure_function y";;
  expand (PURE_REWRITE_TAC [function_hereditary_DEF] THEN BETA_TAC);;
  expand (REPEAT STRIP_TAC);;

  expand (ASM_REWRITE_TAC[]);;

  expand (ASM_REWRITE_TAC[]);;

  expand RES_TAC;;
  expand (ASSUME_TAC PF_t01);;
  expand (DEF_RES_TAC function_hereditary_DEF);;
  expand RES_TAC;;

expand (DEF_RES_TAC pure_function_DEF);;
expand RES_TAC;;
expand (ASSUME_TAC (BETA_RULE (ASSUMP "(x
        function x
        ´  (image x)
        (y y  (field x)  pure_function y))
       x")));;
expand (ASM_REWRITE_TAC[]);;
let PF_t06 = save_top_thm ‘PF_t06‘;;
```

2.2. Non-emptyness

Before introducing the type determined by this property we must prove that it is non-empty.

```
ML
set_goal([],"
        pure_function ¤
");;
expand (REWRITE_TAC [pure_function_DEF; function_hereditary_DEF]);;
expand (REPEAT STRIP_TAC);;

lemma "function ¤";;
 expand (REWRITE_TAC [function;relation;ZF_le2]);;

lemma "´  (image ¤)";;
 expand (REWRITE_TAC[_DEF;image_DEF;´_DEF;ZF_le2;ZF2]);;

lemma "(x x  (field ¤)  p x)";;
 expand (REWRITE_TAC [ZF_thm10; field_DEF; domain_DEF; image_DEF; ZF_le2; ZF2]);;

expand RES_TAC;;

let PF_t07 = save_top_thm ‘PF_t07‘;;
```

```
ML
set_goal([],"
        (x:SET) pure_function x
");;
expand (EXISTS_TAC "¤");;
expand (ACCEPT_TAC PF_t07);;

let PF_t08 = save_top_thm ‘PF_t08‘;;
```

2.3. Type introduction

Now we introduce the new type:

```
ML
let DEF_PF = new_type_definition(‘PF‘,"pure_function",PF_t08);;
```

```
ML
let PF_ONE_ONE = REWRITE_RULE [definition ‘bool‘ ‘ONE_ONE_DEF‘]
                                                (CONJUNCT1 DEF_PF);;
```

```
ML
set_goal([],"
        (x:PF)(y:PF) (x=y) = (REP_PF x = REP_PF y)
");;
expand (REPEAT STRIP_TAC THEN EQ_TAC);;
expand (STRIP_TAC THEN ASM_REWRITE_TAC[]);;
expand (REWRITE_TAC [PF_ONE_ONE]);;
let PF_t09 = save_top_thm ‘PF_t09‘;;
```

```
ML
let  ABS_PF_DEF = new_definition(‘ABS_PF_DEF‘,"
        (ABS_PF:SETPF) s = p:PF REP_PF p = s
");;
```

```
ML
let PF_l3 = INST_TYPE [":PF",":*"] SELECT_AX;;

set_goal([],"
        pure_function x  (REP_PF (ABS_PF x) = x)
");;
expand (REWRITE_TAC [ABS_PF_DEF;DEF_PF]);;
expand (STRIP_TAC);;
lemma "REP_PF x’ = x";;
expand (ASM_REWRITE_TAC[]);;
expand (IMP_RES_TAC (BETA_RULE (SPEC "(x’:PF) REP_PF x’ = (x:SET)" PF_l3)));;
let PF_t10 = save_top_thm ‘PF_t10‘;;
```

```
ML
set_goal([],"
        (x:PF) ABS_PF(REP_PF x) = x
");;
expand (REWRITE_TAC [ABS_PF_DEF]);;
lemma "(x:PF)(y:PF) (REP_PF x = REP_PF y) = (x = y)";;
expand (REPEAT STRIP_TAC THEN EQ_TAC);;
expand (MATCH_ACCEPT_TAC PF_ONE_ONE);;
expand (STRIP_TAC THEN (ASM_REWRITE_TAC[]));;
expand (STRIP_TAC THEN (ASM_REWRITE_TAC[]));;
expand (ACCEPT_TAC (MP
        (BETA_RULE (SPECL ["p:PF p = x";"x:PF"]
                (INST_TYPE [":PF",":*"] SELECT_AX)))
        (REFL "x:PF")));;
let PF_t11 = save_top_thm ‘PF_t11‘;;
```

ABS_PF mat be thought of as a function which lifts a value from the representation type to the new ‘abstract type’. Analogues for operators are introduced here:

```
ML
let lift_monop_DEF = new_definition(‘lift_monop_DEF‘,"
        (lift_monop:(SETSET)(PFPF)) f = x:PF ABS_PF(f (REP_PF x))
");;
let lift_binop_DEF = new_definition(‘lift_binop_DEF‘,"
        (lift_binop:(SETSETSET)(PFPFPF)) f =
                (x:PF)(y:PF) ABS_PF(f (REP_PF x)(REP_PF y))
");;
```

Before defining abstraction it is necessary to identify the values "´²" and "T²".

```
ML
let ý_DEF = new_definition(‘ý_DEF‘,"
        (ý:PF) = ABS_PF ¤
");;
```

```
ML
let unit_map_DEF = new_definition(‘unit_map_DEF‘,"
        (unit_map:SETSETSET) x y = ((y = ´) => ¤ | unit(x  y))
");;

let ²_DEF = new_infix_definition(‘²_DEF‘,"
        (²:PFPFPF) = lift_binop unit_map
");;
```

```
ML
let ´²_DEF = new_definition(‘´²_DEF‘,"
        (´²:PF) = ý ² ý
");;

let T²_DEF = new_definition(‘T²_DEF‘,"
        (T²:PF) = ´² ² ý
");;
```

```
ML
let truef_DEF = new_definition(‘truef_DEF‘,"
        (truef:SET) = REP_PF T²
");;
```

```
ML
let set_to_type_DEF = new_definition(‘set_to_type_DEF‘,"
        (set_to_type:SETSET) s = s ª (unit (REP_PF T²))
");;
let monop_set_to_type_DEF = new_definition(‘monop_set_to_type_DEF‘,"
        (monop_set_to_type:(SETSET)(SETSET)) o = s:SET set_to_type (o s)
");;
```

```
ML
let dom_DEF = new_definition(‘dom_DEF‘,"
        dom   = lift_monop (monop_set_to_type domain)
");;
let ran_DEF = new_definition(‘ran_DEF‘,"
        ran     = lift_monop (monop_set_to_type image)
");;
let fie_DEF = new_definition(‘fie_DEF‘,"
        fie      = lift_monop (monop_set_to_type field)
");;
```

Some general lemmas would be useful showing how properties are lifted.

3. TRUTH

```
let F²_DEF = new_definition(‘F²_DEF‘,"
        (F²:PF) = T² ² (ý ² ý)
");;
```

4. Abstraction and Application

The most fundamental operations concerned with functions are functional abstraction and function application.

Functional Application

First we define the appropriate operation on SET and then we lift it to PF.

```
ML
let function_application_DEF = new_definition(‘function_application_DEF‘,"
        (function_application:SETSETSET) f a =
                a  (domain f) => f © a | ´
");;
```

```
ML
let ² _DEF = new_infix_definition(‘²_DEF‘,"
        ² = lift_binop function_application
");;
```

```
ML
%
let PF_hereditary_DEF = new_definition(‘PF_hereditary_DEF‘,"
        (PF_hereditary_DEF:(PFbool)bool) p =
                (x:PF)((y:PF)       (fie x) ² y p y)  p x
");;
%
```

We should be able to prove an axiom of extensionality.

First we prove the property of sets corresponding to the required property of pure functions.

```
ML
let func_extensional_DEF = new_definition(‘func_extensional_DEF‘,"
        func_extensional x y =
                ((x = y) =     ((z:SET)
                (function_application x z = (function_application y z))
                                )
                )
");;
```

```
ML
set_goal([],"
        (x:SET)(y:SET) pure_function x
                (y  (domain x) = (function_application x y  ´))
");;
expand (EVERY[
                REWRITE_TAC [function_application_DEF; NE_DEF];
                REPEAT STRIP_TAC; EQ_TAC]);;
```

```
2 subgoals
"((y  (domain x) => x © y | ´) = ´)  y  (domain x)"
   [ "pure_function x" ]

"y  (domain x)  ((y  (domain x) => x © y | ´) = ´)"
   [ "pure_function x" ]
```

```
ML
expand (EVERY[
        STRIP_TAC;
        ASM_REWRITE_TAC[];
        IMP_RES_TAC PF_t05;
        IMP_RES_TAC (SPECL ["x:SET";"y:SET";"´"]ZF2_thm4);
        ASM_REWRITE_TAC[];
        STRIP_TAC;
        IMP_RES_TAC (SPECL ["y:SET";"´:SET";"x:SET"] ZF2_thm7);
        ASSUME_TAC (REWRITE_RULE [_DEF] (ASSUME "´  (image x)"));
        RES_TAC]);;

expand (EVERY[
        TAUT_REWRITE_TAC "a  b = b  a";
        STRIP_TAC;
        ASM_REWRITE_TAC[]]);;
let PF_t12 = save_top_thm ‘PF_t12‘;;
```

```
ML
set_goal([],"
        (x:SET)(y:SET)(z:SET) function x  y  (domain x)
         ((function_application x y = z) = ((y  z)  x))
");;
expand (EVERY[
        REWRITE_TAC [function_application_DEF];
        REPEAT STRIP_TAC;
        IMP_RES_TAC ZF2_thm4;
        EQ_TAC; ASM_REWRITE_TAC[]]);;
let PF_t13 = save_top_thm ‘PF_t13‘;;
```

```
ML
set_goal([],"
        (x:SET)(y:SET)(z:SET) pure_function x  y  (domain x)
         ((function_application x y = z) = ((y  z)  x))
");;
expand (EVERY[
        REWRITE_TAC [function_application_DEF];
        REPEAT STRIP_TAC;
        IMP_RES_TAC PF_t05;
        IMP_RES_TAC ZF2_thm4;
        EQ_TAC; ASM_REWRITE_TAC[]]);;
let PF_t14 = save_top_thm ‘PF_t14‘;;
```

```
ML
set_goal([],"
        (x:SET)(y:SET) pure_function x  pure_function y
                        func_extensional x y
");;
expand (EVERY[
        REPEAT STRIP_TAC;
        PURE_ONCE_REWRITE_TAC[func_extensional_DEF];
        EQ_TAC; REPEAT STRIP_TAC]);;
```

```
2 subgoals
"x = y"
   [ "pure_function x" ]
   [ "pure_function y" ]
   [ "z function_application x z = function_application y z" ]

"function_application x z = function_application y z"
   [ "pure_function x" ]
   [ "pure_function y" ]
   [ "x = y" ]
```

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

```
goal proved
 function_application x z = function_application y z

Previous subproof:
"x = y"
   [ "pure_function x" ]
   [ "pure_function y" ]
   [ "z function_application x z = function_application y z" ]
```

```
ML
expand (EVERY[
        PURE_REWRITE_TAC [ZF_le1];
        REPEAT STRIP_TAC; EQ_TAC; REPEAT STRIP_TAC]);;
```

```
2 subgoals
"z  x"
   [ "pure_function x" ]
   [ "pure_function y" ]
   [ "z function_application x z = function_application y z" ]
   [ "z  y" ]

"z  y"
   [ "pure_function x" ]
   [ "pure_function y" ]
   [ "z function_application x z = function_application y z" ]
   [ "z  x" ]
```

```
ML
lemma "(v:SET)(w:SET) z = (v  w)";;
expand       (IMP_RES_TAC PF_t05);;
expand       (DEF_RES_TAC function);;
expand       (DEF_RES_TAC relation);;
expand (ASM_REWRITE_TAC[]);;
```

```
ML
%
lemma_proof "(v:SET)(w:SET) z = (v  w)"
        [IMP_RES_TAC PF_t05;
        DEF_RES_TAC function;
        DEF_RES_TAC relation];;
expand (ASM_REWRITE_TAC[]);;
%
```

```
"(v  w)  y"
  [ "pure_function x" ]
  [ "pure_function y" ]
  [ "z function_application x z = function_application y z" ]
  [ "z  x" ]
  [ "z = v  w" ]
```

```
ML
lemma_proof "(v  w)  x"
        [ACCEPT_TAC    (REWRITE_RULE [ASSUMP "z = v  w"] (ASSUMP "z  x"))];;

lemma_proof       "v  (domain x)"
        [IMP_RES_TAC ZF2_thm7];;

lemma_proof "function_application x v = w"
        [IMP_RES_TAC (SPECL ["x:SET";"v:SET";"w:SET"] PF_t14);
        ASM_REWRITE_TAC[]];;

lemma_proof "function_application y v = w"
        [ASM_REWRITE_TAC[SYM (SPEC "v:SET" (ASSUMP
        "z function_application x z = function_application y z"))]];;
```

```
ML
lemma_proof "function_application x v  ´"
        [IMP_RES_THEN DEF_RES_TAC (SPECL["x:SET";"v:SET"]PF_t12)];;

lemma_proof "function_application y v  ´"
        [ACCEPT_TAC (REWRITE_RULE [SPEC "v:SET"
        (ASSUMP "z function_application x z = function_application y z")]
        (ASSUMP "(function_application x v)  ´"))];;

lemma_proof        "v  (domain y)"
        [IMP_RES_TAC (SPECL ["y:SET";"v:SET"] PF_t12);
        ASM_REWRITE_TAC[]];;
```

```
"(v  w)  y"
   [ "pure_function x" ]
   [ "pure_function y" ]
   [ "z function_application x z = function_application y z" ]
   [ "z  x" ]
   [ "z = v  w" ]
   [ "(v  w)  x" ]
   [ "v  (domain x)" ]
   [ "function_application x v = w" ]
   [ "function_application y v = w" ]
   [ "(function_application x v)  ´" ]
   [ "(function_application y v)  ´" ]
   [ "v  (domain y)" ]
```

```
ML
expand (IMP_RES_TAC (SPECL ["y:SET";"v:SET";"w:SET"] PF_t14));;
expand (ACCEPT_TAC (REWRITE_RULE
                [ASSUMP "(function_application y v = w) = (v  w)  y"]
                (ASSUMP "function_application y v = w")));;
```

```
Previous subproof:
"z  x"
   [ "pure_function x" ]
   [ "pure_function y" ]
   [ "z function_application x z = function_application y z" ]
   [ "z  y" ]
```

```
ML
lemma "(v:SET)(w:SET) z = (v  w)";;
expand (EVERY
        [IMP_RES_TAC PF_t05;
        DEF_RES_TAC function;
        DEF_RES_TAC relation]);;
expand (ASM_REWRITE_TAC[]);;
```

```
"(v  w)  x"
  [ "pure_function x" ]
  [ "pure_function y" ]
  [ "z function_application x z = function_application y z" ]
  [ "z  y" ]
  [ "z = v  w" ]
```

```
ML
lemma_proof "(v  w)  y"
        [ACCEPT_TAC    (REWRITE_RULE [ASSUMP "z = v  w"] (ASSUMP "z  y"))];;

lemma_proof        "v  (domain y)"
        [IMP_RES_TAC ZF2_thm7];;

lemma_proof "function_application x v = w"
        [IMP_RES_TAC (SPECL ["y:SET";"v:SET";"w:SET"] PF_t14);
        ASM_REWRITE_TAC[]];;

lemma_proof "function_application x v = w"
        [ASM_REWRITE_TAC[SYM (SPEC "v:SET" (ASSUMP
        "z function_application x z = function_application y z"))]];;
```

```
ML
lemma_proof "function_application y v  ´"
        [IMP_RES_THEN DEF_RES_TAC (SPECL["y:SET";"v:SET"]PF_t12)];;

lemma_proof "function_application x v  ´"
        [ACCEPT_TAC (REWRITE_RULE [SYM (SPEC "v:SET"
        (ASSUMP "z function_application x z = function_application y z"))]
        (ASSUMP "(function_application y v)  ´"))];;

lemma_proof        "v  (domain x)"
        [IMP_RES_TAC (SPECL ["x:SET";"v:SET"] PF_t12);
        ASM_REWRITE_TAC[]];;
```

```
"(v  w)  x"
   [ "pure_function x" ]
   [ "pure_function y" ]
   [ "z function_application x z = function_application y z" ]
   [ "z  y" ]
   [ "z = v  w" ]
   [ "(v  w)  y" ]
   [ "v  (domain y)" ]
   [ "function_application x v = w" ]
   [ "(function_application y v)  ´" ]
   [ "(function_application x v)  ´" ]
   [ "v  (domain x)" ]
```

```
ML
expand (IMP_RES_TAC (SPECL ["x:SET";"v:SET";"w:SET"] PF_t14));;
expand (ACCEPT_TAC (REWRITE_RULE
        [ASSUMP "(function_application x v = w) = (v  w)  x"]
        (ASSUMP "function_application x v = w")));;
```

```
goal proved
 x y
   pure_function x  pure_function y  func_extensional x y
```

```
ML
let PF_t15 = save_top_thm ‘PF_t15‘;;
```

```
ML
set_goal([],"
        (x:SET)(y:SET) pure_function x  pure_function y  y  ´
                pure_function (unit (x  y))
");;
expand (PURE_REWRITE_TAC [NE_DEF] THEN REPEAT STRIP_TAC);;
expand (TMP_TAC (SPEC "unit(x  y)" (REWRITE_RULE [function_hereditary_DEF] PF_t01)));;
expand (REPEAT STRIP_TAC);;

expand (ACCEPT_TAC (SPEC_ALL ZF2_thm9));;
```

```
ML
expand (PURE_REWRITE_TAC [_DEF; image_DEF; ZF2]);;
expand (TAUT_REWRITE_TAC "(a  b) = a  b" THEN DISJ2_TAC THEN BETA_TAC);;
expand (PURE_REWRITE_TAC [ZF_thm9; ZF2_thm3]);;
expand (ASM_REWRITE_TAC[]);;
expand (UNDISCH_TAC "(y = ´)");;
expand (TAUT_REWRITE_TAC "a  b = b  a");;
expand (REPEAT STRIP_TAC THEN ASM_REWRITE_TAC []);;
```

```
ML
expand (RMP_TAC "(x' = x)  (x' = y)");;
expand (STRIP_TAC THEN ASM_REWRITE_TAC[]);;
expand (STRIP_ASSUME_TAC (BETA_RULE
        (REWRITE_RULE [field_DEF; image_DEF; domain_DEF;
                                ZF2; ZF_thm9; ZF_thm10; ZF2_thm3]
        (ASSUMP "x'  (field(unit(x  y)))"))))
        THEN ASM_REWRITE_TAC[]);;
let PF_t16 = save_top_thm 'PF_t16';;
```

```
ML
set_goal([],"
        pure_function ´
");;
expand (PURE_REWRITE_TAC [´_DEF]);;
expand (TMP_TAC (SPECL ["¤";"¤"] PF_t16));;
expand (REWRITE_TAC [PF_t07; ´_DEF; NE_DEF; ZF_le1]);;
expand (NOT_FORALL_TAC THEN EXISTS_TAC "¤  ¤");;
expand (REWRITE_TAC [ZF_le2; ZF_thm9]);;
let PF_t17 = save_top_thm 'PF_t17';;
```

```
ML
%
set_goal([],"
        (x:SET)(y:SET) pure_function x  (y  field x)
                pure_function y
");;
expand (REPEAT STRIP_TAC);;
%
```

```
ML
set_goal([],"
        (x:SET)(y:SET) pure_function x  pure_function y
                pure_function (function_application x y)
");;
expand (REPEAT STRIP_TAC);;
expand (ASM_CASES_TAC "y  (domain x)" THEN ASM_REWRITE_TAC[]);;

lemma_proof "z:SET function_application x y = z"
        [EXISTS_TAC "function_application x y"; REWRITE_TAC[]];;

lemma_proof "(function_application x y = z)  (y  z)  x"
        [IMP_RES_TAC PF_t14; ASM_REWRITE_TAC[]];;

expand RES_TAC;;
expand (ASM_REWRITE_TAC[]);;
expand (IMP_RES_TAC (SPECL ["y:SET";"z:SET";"x:SET"] ZF2_thm7));;

lemma_proof "z  (field x)" [ASM_REWRITE_TAC [field_DEF; ZF_thm10]];;

expand (IMP_RES_TAC PF_t06);;

lemma_proof "function_application x y = ´"
        [IMP_RES_THEN (DEF_RES_TAC o (TAUT_REWRITE_RULE
                "(a = b) = (a = b)") o (REWRITE_RULE [NE_DEF])) PF_t12];;

expand (ASM_REWRITE_TAC [PF_t17]);;

let PF_t18 = save_top_thm ‘PF_t18‘;;
```

```
ML
set_goal([],"
        x:PF pure_function (REP_PF x)
");;
expand (REWRITE_TAC [CONJUNCT2 DEF_PF]);;
expand (STRIP_TAC THEN EXISTS_TAC "x:PF" THEN REFL_TAC);;
let PF_t19 = save_top_thm ‘PF_t19‘;;
```

```
ML
set_goal([],"
        (x:SET)(y:SET) pure_function x  pure_function y
                ((z:PF) function_application x (REP_PF z)
                                = function_application y (REP_PF z))
                (x = y)
");;
expand (REPEAT STRIP_TAC);;
lemma "(w:SET) function_application x w = function_application y w";;
expand (STRIP_TAC THEN ASM_CASES_TAC "pure_function w");;
lemma "x' w = REP_PF x'";;
 expand (DEF_RES_TAC (SPEC "w:SET" (CONJUNCT2 DEF_PF)));;
expand (ASM_REWRITE_TAC[]);;
```

```
ML
lemma "function_application x w = ´";;
 lemma " (w  domain x)";;
   expand (IMP_RES_TAC PF_t06);;
   lemma "w  (field x)";;
     expand (STRIP_TAC THEN RES_TAC);;
   expand (UNDISCH_TAC "w  (field x)"
        THEN REWRITE_TAC [field_DEF; ZF_thm10]);;
   expand TAUT_TAC;;
 expand (IMP_RES_THEN (DEF_RES_TAC o
        (TAUT_REWRITE_RULE "(a = b) = (a = b)") o
        (REWRITE_RULE [NE_DEF])) (SPECL ["x:SET";"w:SET"] PF_t12));;
expand (ASM_REWRITE_TAC[]);;
```

```
ML
lemma "function_application y w = ´";;
 lemma " (w  domain y)";;
   expand (IMP_RES_TAC PF_t06);;
   lemma "w  (field y)";;
     expand (STRIP_TAC THEN RES_TAC);;
   expand (UNDISCH_TAC "w  (field y)"
        THEN REWRITE_TAC [field_DEF; ZF_thm10]);;
   expand TAUT_TAC;;
 expand (IMP_RES_THEN (DEF_RES_TAC o
        (TAUT_REWRITE_RULE "(a = b) = (a = b)") o
        (REWRITE_RULE [NE_DEF])) (SPECL ["y:SET";"w:SET"] PF_t12));;
expand (ASM_REWRITE_TAC[]);;
```

```
ML
lemma_proof "func_extensional x y" [IMP_RES_TAC PF_t15];;
expand (DEF_RES_TAC func_extensional_DEF);;
expand (ASM_REWRITE_TAC[]);;
let PF_t20 = save_top_thm ‘PF_t20‘;;
```

```
ML
set_goal([],"
        (x:PF)(y:PF) (x = y) = ((z:PF) x ² z = y ² z)
");;
expand (REPEAT STRIP_TAC THEN EQ_TAC);;
```

```
2 subgoals
"(z x ² z = y ² z)  (x = y)"

"(x = y)  (z x ² z = y ² z)"
```

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

```
goal proved
 (x = y)  (z x ² z = y ² z)

Previous subproof:
"(z x ² z = y ² z)  (x = y)"
```

```
ML
expand (REWRITE_TAC [PF_t09;²_DEF;²_DEF;´²_DEF;lift_binop_DEF]);;
expand BETA_TAC;;
```

```
ML
lemma "((y:PF)(z:PF)REP_PF(ABS_PF(function_application(REP_PF y)(REP_PF z))) =
     function_application(REP_PF y)(REP_PF z))
";;
 expand (STRIP_TAC THEN STRIP_TAC);;
 lemma_proof "pure_function (REP_PF x)" [REWRITE_TAC [PF_t19]];;
 lemma_proof "pure_function (REP_PF y)" [REWRITE_TAC [PF_t19]];;
 lemma_proof "pure_function (REP_PF z)" [REWRITE_TAC [PF_t19]];;
 lemma_proof "pure_function (function_application(REP_PF x)(REP_PF z))"
       [IMP_RES_TAC PF_t18];;
 lemma_proof "pure_function (function_application(REP_PF y)(REP_PF z))"
       [IMP_RES_TAC PF_t18];;
 expand (IMP_RES_TAC PF_t10);;
```

```
ML
expand (ASM_REWRITE_TAC[]);;
lemma_proof "pure_function (REP_PF x)" [REWRITE_TAC [PF_t19]];;
lemma_proof "pure_function (REP_PF y)" [REWRITE_TAC [PF_t19]];;
expand (IMP_RES_TAC PF_t20);;
let PF_t21 = save_top_thm ‘PF_t21‘;;
```

5. ý, the everywhere undefined function

This is a primitive.

```
ML
set_goal([],"
        (x:PF) ý ² x = ´²
");;
expand (REWRITE_TAC[ý_DEF; ²_DEF; lift_binop_DEF;´²_DEF; ²_DEF; ´_DEF; unit_map_DEF]
                THEN BETA_TAC THEN STRIP_TAC);;
lemma_proof       "pure_function ¤"          [ACCEPT_TAC PF_t07];;
expand (IMP_RES_TAC PF_t10 THEN ASM_REWRITE_TAC[]);;
lemma_proof       "(REP_PF x)  (domain ¤)"
                [REWRITE_TAC [domain_DEF; ZF2; ZF_le2]];;
lemma_proof "function_application ¤(REP_PF x) = unit(¤  ¤)"
        [IMP_RES_THEN
                (DEF_RES_TAC o TAUT_REWRITE_RULE "(a = b) = (a = b)")
                (REWRITE_RULE [NE_DEF; ´_DEF] PF_t12)];;
expand (ASM_REWRITE_TAC []);;
expand (ASM_CASES_TAC "unit(¤  ¤) = ¤" THEN ASM_REWRITE_TAC[]);;
lemma "(¤ = unit(¤  ¤))";;
 expand (UNDISCH_TAC"(unit(¤  ¤) = ¤)"
        THEN TAUT_REWRITE_TAC "a  b = b  a"
        THEN STRIP_TAC);;
expand (CONV_TAC SYM_CONV THEN ACCEPT_TAC (ASSUMP "¤ = unit(¤  ¤)"));;
expand (ASM_REWRITE_TAC[]);;

let PF_t22 = save_top_thm ‘PF_t22‘;;
```

6. ², the unit map

This is also primitive.

The following lemma will be useful in establishing the important properties.

```
ML
set_goal([],"
        (x:SET)(y:SET)(z:SET)  pure_function x  pure_function y
         pure_function (unit_map x y)
");;
expand (EVERY[  REPEAT STRIP_TAC;
                        REWRITE_TAC [unit_map_DEF]]);;
```

```
"pure_function((y = ´) => ¤ | unit(x  y))"
   [ "pure_function x" ]
   [ "pure_function y" ]
```

```
ML
expand (ASM_CASES_TAC "y = ´" THEN ASM_REWRITE_TAC[]);;
```

```
2 subgoals
"pure_function(unit(x  y))"
   [ "pure_function x" ]
   [ "pure_function y" ]
   [ "(y = ´)" ]

"pure_function ¤"
   [ "pure_function x" ]
   [ "pure_function y" ]
   [ "y = ´" ]
```

```
ML
expand (ACCEPT_TAC PF_t07);;
```

```
goal proved
 pure_function ¤

Previous subproof:
"pure_function(unit(x  y))"
   [ "pure_function x" ]
   [ "pure_function y" ]
   [ "(y = ´)" ]
```

```
ML
expand (IMP_RES_TAC (REWRITE_RULE [NE_DEF] PF_t16));;
```

```
goal proved
 x y z
   pure_function x  pure_function y
   pure_function(unit_map x y)
```

```
ML
let PF_t23 = save_top_thm ‘PF_t23‘;;
```

```
ML
set_goal([],"
        (x:SET)(y:SET)(z:SET) (y  (domain x))
         (function_application x y = ´)
");;
expand (REPEAT STRIP_TAC THEN ASM_REWRITE_TAC [function_application_DEF]);;
let PF_t24 = save_top_thm ‘PF_t24‘;;
```

```
ML
set_goal([],"
        (x:SET)(y:SET)(z:SET)  pure_function x  pure_function y
         (function_application (unit_map x y) z = ((z = x) => y | ´))
");;
expand (REPEAT STRIP_TAC);;
expand (ASM_CASES_TAC "z  domain(unit_map x y)");;
```

```
2 subgoals
"function_application(unit_map x y)z = ((z = x) => y | ´)"
   [ "pure_function x" ]
   [ "pure_function y" ]
   [ "z  (domain(unit_map x y))" ]

"function_application(unit_map x y)z = ((z = x) => y | ´)"
   [ "pure_function x" ]
   [ "pure_function y" ]
   [ "z  (domain(unit_map x y))" ]
```

```
ML
lemma "(function_application (unit_map x y) z = ((z = x) => y | ´))
                = (z  ((z = x) => y | ´))  (unit_map x y)";;
lemma_proof "pure_function (unit_map x y)" [IMP_RES_TAC PF_t23];;
lemma_proof "function (unit_map x y)" [IMP_RES_TAC PF_t05];;
expand (IMP_RES_TAC (SPECL ["unit_map x y";"z";"(z = x) => y | ´"] PF_t13));;
```

```
goal proved
Previous subproof:
"function_application(unit_map x y)z = ((z = x) => y | ´)"
   [ "pure_function x" ]
   [ "pure_function y" ]
   [ "z  (domain(unit_map x y))" ]
   [ "(function_application(unit_map x y)z = ((z = x) => y | ´)) =
      (z  ((z = x) => y | ´))  (unit_map x y)" ]
```

```
ML
expand (ASM_REWRITE_TAC[]);;
expand (REWRITE_TAC [unit_map_DEF]);;
lemma "(y = ´)";;
        expand STRIP_TAC;;
        lemma_proof "unit_map x y = ¤" [ASM_REWRITE_TAC [unit_map_DEF]];;
        expand (ACCEPT_TAC ((BETA_RULE o REWRITE_RULE
                [ASSUMP "unit_map x y = ¤"; domain_DEF; ZF2; ZF_le2])
                (ASSUMP "z  (domain(unit_map x y))")));;
```

```
ML
expand (ASM_REWRITE_TAC[ZF_thm9; ZF2_thm3]);;
expand (MP_TAC ((CONJUNCT2 o BETA_RULE o REWRITE_RULE
        [domain_DEF; ZF2; ZF_le2; ZF_thm9; ZF2_thm3;
                unit_map_DEF; ASSUMP "(y = ´)"])
                (ASSUMP "z  (domain(unit_map x y))")));;
expand (STRIP_TAC THEN ASM_REWRITE_TAC[]);;

expand (IMP_RES_TAC PF_t24 THEN ASM_REWRITE_TAC[]);;
expand (ASM_CASES_TAC "(z = x)");;
expand (ASM_REWRITE_TAC[]);;
expand (UNDISCH_TAC "z  (domain(unit_map x y))"
        THEN TAUT_REWRITE_TAC "a  b = b  a"
        THEN ASM_REWRITE_TAC
        [domain_DEF; unit_map_DEF; ZF2]);;
expand (SUBST1_TAC (SYM_CONV "´ = y"));;
expand (STRIP_TAC THEN ASM_REWRITE_TAC[]);;
expand (BETA_TAC THEN ASM_REWRITE_TAC[unit_map_DEF;ZF6]);;
expand STRIP_TAC;;
expand (REWRITE_TAC[ZF6; ZF_thm9; ZF2_thm3]);;
expand (EXISTS_TAC "pair x y" THEN REWRITE_TAC[]);;
expand (REWRITE_TAC[ZF5] THEN EXISTS_TAC "x  y");;
expand (REWRITE_TAC [_DEF; ZF5]);;

expand (EXISTS_TAC "y:SET" THEN ASM_REWRITE_TAC[unit_map_DEF;ZF_thm9]);;

expand (ASM_REWRITE_TAC[]);;
let PF_t25 = save_top_thm ‘PF_t25‘;;
```

```
ML
set_goal([],"
        ´² = ABS_PF ´
");;
expand (REWRITE_TAC[´²_DEF;²_DEF; ý_DEF; unit_map_DEF;
                lift_binop_DEF; ´_DEF]);;
expand BETA_TAC;;
expand (ASSUME_TAC PF_t07);;
expand (IMP_RES_TAC PF_t10);;
expand (ASM_REWRITE_TAC[]);;
lemma  " (¤ = unit(¤ ¤))";;
        expand STRIP_TAC;;
        lemma "(¤ ¤) ¤";;
                lemma "(¤ ¤)  unit(¤ ¤)";;
                        expand (REWRITE_TAC [ZF_thm9]);;
                expand (ACCEPT_TAC (REWRITE_RULE
                        [SYM (ASSUMP "¤ = unit(¤ ¤)")]
                        (ASSUMP "(¤ ¤)  (unit(¤ ¤))")));;
                expand (MP_TAC (ASSUMP "(¤ ¤) ¤") THEN REWRITE_TAC [ZF_le2]);;
expand (ASM_REWRITE_TAC[]);;
let PF_t26 = save_top_thm ‘PF_t26‘;;
```

```
ML
set_goal([],"
        (x:PF)(y:PF)(z:PF) (x ² y) ² z = ((z = x) => y | ´²)
");;
expand (EVERY [ REPEAT STRIP_TAC;
                        REWRITE_TAC [²_DEF; ²_DEF; lift_binop_DEF];
                        BETA_TAC]);;
lemma_proof "pure_function (REP_PF x)"
        [REWRITE_TAC [DEF_PF]; EXISTS_TAC "x"; REFL_TAC];;
lemma_proof "pure_function (REP_PF y)"
        [REWRITE_TAC [DEF_PF]; EXISTS_TAC "y"; REFL_TAC];;
lemma_proof "function_application(unit_map(REP_PF x)(REP_PF y))(REP_PF z) =
      ((REP_PF z = REP_PF x) => REP_PF y | ´)"
        [IMP_RES_TAC (SPECL ["REP_PF x";"REP_PF y";"REP_PF z"] PF_t25)];;
lemma "REP_PF (ABS_PF (unit_map(REP_PF x)(REP_PF y)))
                = unit_map(REP_PF x)(REP_PF y)";;
        lemma_proof "pure_function (unit_map(REP_PF x)(REP_PF y))"
                        [IMP_RES_TAC PF_t23];;
        expand (IMP_RES_TAC PF_t10);;
expand (ASM_REWRITE_TAC[]);;
expand (ASM_CASES_TAC "z = x");;
expand (ASM_REWRITE_TAC[PF_t11]);;
expand (ASM_REWRITE_TAC[]);;
expand (ASM_REWRITE_TAC [SYM (SPECL ["z:PF";"x:PF"] PF_t09); PF_t26]);;
let PF_t27 = save_top_thm ‘PF_t27‘;;
```

7. THE END

```
ML
close_theory‘pf123‘;;
print_theory‘pf123‘;;
```

8. THE THEORY pf123

The Theory pf123
Parents -- ZF2 ZF2
Types -- ":PF"
Constants --
´ ":SET" function_hereditary ":(SET bool) bool"
pure_function ":SET bool" REP_PF ":PF SET"
ABS_PF ":SET PF" lift_monop ":(SET SET) (PF PF)"
lift_binop ":(SET (SET SET)) (PF (PF PF))"
ý ":PF" unit_map ":SET (SET SET)" ´² ":PF"
T² ":PF" truef ":SET" set_to_type ":SET SET"
monop_set_to_type ":(SET SET) (SET SET)"
dom ":PF PF" ran ":PF PF" fie ":PF PF"
function_application ":SET (SET SET)"
func_extensional ":SET (SET bool)"
Curried Infixes --
² ":PF (PF PF)" ² ":PF (PF PF)"
Axioms --
DEF_PF
ONE_ONE REP_PF (x pure_function x = (x' x = REP_PF x'))

Definitions --
´_DEF ´ = unit(¤ ¤)
function_hereditary_DEF
function_hereditary p =
(f
function f
´ (image f)
(x x (field f) p x)
p f)
pure_function_DEF
pure_function s = (p function_hereditary p p s)
ABS_PF_DEF ABS_PF s = (p REP_PF p = s)
lift_monop_DEF lift_monop f = (x ABS_PF(f(REP_PF x)))
lift_binop_DEF
lift_binop f = (x y ABS_PF(f(REP_PF x)(REP_PF y)))
ý_DEF ý = ABS_PF ¤
unit_map_DEF unit_map x y = ((y = ´) => ¤ | unit(x y))
²_DEF $² = lift_binop unit_map
´²_DEF ´² = ý ² ý
T²_DEF T² = ´² ² ý
truef_DEF truef = REP_PF T²
set_to_type_DEF set_to_type s = s ª (unit(REP_PF T²))
monop_set_to_type_DEF monop_set_to_type o = (s set_to_type(o s))
dom_DEF dom = lift_monop(monop_set_to_type domain)
ran_DEF ran = lift_monop(monop_set_to_type image)

```
 fie_DEF   fie = lift_monop(monop_set_to_type field)
 function_application_DEF
   function_application f a = (a  (domain f) => f © a | ´)
 ²_DEF   $² = lift_binop function_application
 func_extensional_DEF
   func_extensional x y =
    ((x = y) =
     (z function_application x z = function_application y z))
```

Theorems --

```
 PF_t01   function_hereditary pure_function
 PF_t02   p function_hereditary p  (x pure_function x  p x)
 PF_t03   function_hereditary function
 PF_t04   function_hereditary(x ´  (image x))
 PF_t05   x pure_function x  function x  ´  (image x)
 PF_t06
   x
     pure_function x  (y y  (field x)  pure_function y)
 PF_t07   pure_function ¤
 PF_t08   x pure_function x
 PF_t09   x y (x = y) = (REP_PF x = REP_PF y)
 PF_t10   pure_function x  (REP_PF(ABS_PF x) = x)
 PF_t11   x ABS_PF(REP_PF x) = x
 PF_t12
   x y
     pure_function x
     (y  (domain x) = (function_application x y)  ´)
 PF_t13
   x y z
     function x  y  (domain x)
     ((function_application x y = z) = (y  z)  x)
 PF_t14
   x y z
     pure_function x  y  (domain x)
     ((function_application x y = z) = (y  z)  x)
 PF_t15
   x y
     pure_function x  pure_function y  func_extensional x y
 PF_t16
   x y
     pure_function x  pure_function y  y  ´
     pure_function(unit(x  y))
 PF_t17   pure_function ´
 PF_t18
   x y
     pure_function x
     pure_function y
```

```
    pure_function(function_application x y)
PF_t19   x pure_function(REP_PF x)
PF_t20
  x y
    pure_function x  pure_function y
    (z
     function_application x(REP_PF z) =
     function_application y(REP_PF z))
    (x = y)
PF_t21   x y (x = y) = (z x ² z = y ² z)
PF_t22   x ý ² x = ´²
PF_t23
  x y z
    pure_function x  pure_function y
    pure_function(unit_map x y)
PF_t24
  x y z y  (domain x)  (function_application x y = ´)
PF_t25
  x y z
    pure_function x  pure_function y
    (function_application(unit_map x y)z = ((z = x) => y | ´))
PF_t26   ´² = ABS_PF ´
PF_t27   x y z (x ² y) ² z = ((z = x) => y | ´²)
```