Pure function theory part 3

Roger Bishop Jones

ICL Defence Systems

ABSTRACT

This document consists of the third part of a pure theory of (partial) functions.

1. INTRODUCTION

ML

```
new_theory‘pf129‘;;
new_parent‘pf128‘;;
loadf‘/escher/usr2/projects/infra/pholfiles/TAUT‘;;
map load_definitions [‘zf120‘;‘zf122‘;‘pf123‘;‘pf128‘];;
map load_theorems [‘zf120‘;‘zf122‘;‘pf123‘;‘pf128‘];;
map load_axioms [‘zf120‘;‘zf122‘;‘pf123‘;‘pf128‘];;
map delete_cache [‘zf120‘;‘zf122‘;‘pf123‘;‘pf128‘];;
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);;
let MASSUMP = map ASSUMP;;
let OREWRITE_TAC t = REWRITE_TAC[t];;
```

2. UNION AND CHOICE

The following definition of $\cup_f$ serves two roles. If applied to a set of sets, i.e. something which might have had type ":*→bool→bool", it will yield the union of the sets. If applied to a set of arbitrary functions, (":*→**→bool), it will yield some unspecified function which is defined over the union of the domains of the functions and makes an arbitrary choice between the functions defined at any point in the domain. i.e., an arbitrary functional subset of the union of the domain, defined over the domain of that union.

```
ML
let sid_DEF = new_definition(‘sid_DEF‘,"
        (sid:SET→SET) s = sep (s ª s) λx:SET• ∃(y:SET)• x = (y ↦ y)
");;
```

```
ML
let funion_DEF = new_definition(‘funion_DEF‘,"
        (funion:PF→SET) f = sabs (ABS_PF (sid (domain (∪ (domain (REP_PF f))))))
                λx:PF• µy:PF• (∃z:PF• (f f z) ≠ ⊥f ∧ (y = z f x))
                                ∨ (∀z:PF• ((f f z) = ⊥f) ∧ (y = ⊥f))
");;

let ∪f_DEF = new_definition(‘∪f_DEF‘,"
        (∪f:PF → PF) f = ABS_PF (funion f)
");;
```

3. LEMMAS BELONGING ELSEWHERE

3.1. Function Application

```
ML
set_goal([],"
        ∀(f:PF)(x:PF)• f f x = ABS_PF (function_application(REP_PF f)(REP_PF x))
");;
expand (REWRITE_TAC [f_DEF; lift_binop_DEF] THEN BETA_TAC THEN REWRITE_TAC[]);;
let LBE1 = save_top_thm ‘LBE1‘;;
```

```
ML
set_goal([],"
        ∀(f:PF)(x:PF)• pure_function (function_application(REP_PF f)(REP_PF x))
");;
expand (REPEAT GEN_TAC);;
lemma_proof "pure_function (REP_PF f) ∧ pure_function (REP_PF x)"
                [REWRITE_TAC [PF_t19]];;
expand (IMP_RES_TAC PF_t18);;
let LBE2 = save_top_thm ‘LBE2‘;;
```

```
ML
set_goal([],"
        ∀(f:SET)(x:SET)• pure_function f ⇒
                (x ∈ (domain f) =>
                ((function_application f x = z) = (x ↦ z) ∈ f)        |
                (function_application f x = ⊥))
");;
expand (REPEAT STRIP_TAC);;
expand (ASM_CASES_TAC "x ∈ (domain f)" THEN ASM_REWRITE_TAC[]);;
expand (IMP_RES_TAC PF_t14);;
expand (IMP_RES_TAC PF_t24);;
let LBE3 = save_top_thm ‘LBE3‘;;
```

4. LEMMAS FOR THE UNION AXIOM

4.1. Funion yields a pure function

```
ML
set_goal([],"
        ∀(f:PF)• pure_function (funion f)
");;
expand (REWRITE_TAC [funion_DEF;PFB23]);;
let PFU_l1 = save_top_thm ‘PFU_l1‘;;
```

ML
```
set_goal([],"
        ∀(f:PF)(g:PF)•((function_application(funion f)(REP_PF g)) = ⊥)
        ⇒     ∀(i:PF)•
                ((function_application(REP_PF f)(REP_PF i)) = ⊥)
                ∨       ((function_application(REP_PF i)(REP_PF g)) = ⊥)
");;
expand (REPEAT GEN_TAC);;
lemma "((function_application(funion f)(REP_PF g)) = ⊥)
                = ¬(REP_PF g) ∈ (domain (funion f))";;
  expand (ASSUME_TAC (SPEC_ALL PFU_l1)
                THEN IMP_RES_TAC (SPECL ["funion f";"REP_PF g"] PF_t12));;
expand (REWRITE_TAC [funion_DEF]);;
```

5. THE UNION AXIOM

The following theorem characterises union over pure functions:

ML
```
set_goal([],"
        ∀(f:PF)(g:PF)• (((∪_f f) _f g) ≠ ⊥_f) =>
                (∃(i:PF)• (f _f i) ≠ ⊥_f ∧ (i _f g = (∪_f f) _f g))
        |       (∀(i:PF)• (f _f i) ≠ ⊥_f ⇒ (i _f g = ⊥_f))
");;
expand (REPEAT GEN_TAC);;
expand (REWRITE_TAC [∪_f_DEF; LBE1; NE_DEF]);;
lemma_proof "pure_function (funion f)" [REWRITE_TAC[PFU_l1]];;
expand (IMP_RES_THEN OREWRITE_TAC PF_t10);;
expand (ASM_CASES_TAC
        "(ABS_PF(function_application(funion f)(REP_PF g)))
                = ⊥_f"
        THEN ASM_REWRITE_TAC[]);;
```

6. THE END

ML
```
close_theory‘pf129‘;;
```