*Project:* DRA FRONT END FILTER PROJECT

*Title:* Proofs About Labelling

*Ref:* DS/FMU/FEF/044 *Issue: Revision : 1.5* *Date:* 5 June 2016

*Status:* Approved *Type:* Specification

*Keywords:*

*Author:*

| *Name* | *Location* | *Signature* | *Date* |
|---|---|---|---|
| R. D. Arthan | WIN01 | | |

*Authorisation for Issue:*

| *Name* | *Function* | *Signature* | *Date* |
|---|---|---|---|
| R.B. Jones | HAT Manager | | |

*Abstract:* Formal proofs establishing various results about the multi-level policy and the labelling property for SWORD (for DRA Front End Filter project RSRE 1C/6130).

*Distribution*: HAT FEF File
Simon Wiseman

# 0 DOCUMENT CONTROL

## 0.1 Contents List



## 0.2 Document Cross References

[1] DS/FMU/FEF/003. *Formal Security Policy.* G.M. Prout, ICL Secure Systems, WIN01.

[2] DS/FMU/FEF/040. *Multi-level Formal Security Policy.* R.D. Arthan, ICL Secure Systems, WIN01.

[3] DS/FMU/FEF/042. *Multi-level Architectural Model.* R.D. Arthan, ICL Secure Systems, WIN01.

[4] DS/FMU/FEF/044. *Proofs About Labelling.* R.B. Jones, ICL Secure Systems, WIN01.

## 0.3 Changes History

**Issue** *Revision* : *1.5* **(5 June 2016)** First draft.

**Issue 1.6** Removed dependency on ICL logo font

## 0.4 Changes Forecast

None.

# 1 GENERAL

## 1.1 Scope

This document contains proofs relating to the specifications in documnets [2, 3, 4].

## 1.2 Introduction

The proofs are presented in sections which correspond to the earliest theory in which they can be conducted, which is the theory in which the proven theorems are stored, with one subsection for each result proven.

# 2 PROOFS FOR FEF040

The following ProofPower instructions open the theory *fef040*.

SML

```
open_theory "fef040";
map delete_thm (map (hd o fst) (get_thms "fef040"));
push_pc "hol";
```

## 2.1 Proof of thm_040_1

First a general result about filtering:

SML

```
set_pc"hol";
set_goal([], ⌜∀l a b• (l ↾ a) ↾ b = l ↾ (a ∩ b)⌝);
a(REPEAT strip_tac);
a(list_induction_tac ⌜l⌝ THEN asm_rewrite_tac[get_spec⌜$↾⌝]);
a(REPEAT strip_tac);
a(cases_tac⌜x ∈ a⌝ THEN asm_rewrite_tac[get_spec⌜$↾⌝]);
(* *** Goal "1" *** *)
a(cases_tac⌜x ∈ b⌝ THEN asm_rewrite_tac[get_spec⌜$↾⌝]);
(* *** Goal "1.1" *** *)
a(LEMMA_T ⌜x ∈ a ∩ b⌝ rewrite_thm_tac THEN REPEAT strip_tac);
(* *** Goal "1.2" *** *)
a(LEMMA_T ⌜¬x ∈ a ∩ b⌝ rewrite_thm_tac THEN REPEAT strip_tac);
(* *** Goal "2" *** *)
a(LEMMA_T ⌜¬x ∈ a ∩ b⌝ rewrite_thm_tac THEN REPEAT strip_tac);
val ↾_↾_thm = save_pop_thm"↾_↾_thm";
```

If a lattice has more than one element then *lattice_top* differs from *lattice_bottom*:

SML
```
set_goal([], ⌜∀c• ¬c = lattice_top ⇒ ¬lattice_top = lattice_bottom⌝);
a(contr_tac);
a(lemma_tac⌜lattice_top dominates c⌝
        THEN1 rewrite_tac[get_spec⌜lattice_top⌝]);
a(lemma_tac⌜c dominates lattice_top⌝
        THEN1 asm_rewrite_tac[get_spec⌜lattice_top⌝]);
a(all_fc_tac[get_spec⌜lattice_top⌝]);
val not_lattice_top_thm = save_pop_thm"not_lattice_top_thm";
```

The down-set from the top and the up-set from the bottom both include all elements of the type:

SML
```
set_goal([], ⌜
        lattice_bottom ↑ = Universe
∧       lattice_top ↓ = Universe
⌝);
a(PC_T1 "sets_ext" prove_tac(map get_spec
        [⌜$↑⌝, ⌜$↓⌝, ⌜lattice_bottom⌝]));
val up_down_thm1 = save_pop_thm"up_down_thm1";
```

Any class is a member of its own up-set and down-set. *lattice_bottom* is a member of every down-set and *lattice_top* is amember of every up-set.

SML
```
set_goal([], ⌜
∀c•     c ∈ c ↓
∧       c ∈ c ↑
∧       lattice_bottom ∈ c ↓
∧       lattice_top ∈ c ↑
⌝);
a(PC_T1 "sets_ext" prove_tac(map get_spec
        [⌜$↑⌝, ⌜$↓⌝, ⌜lattice_bottom⌝]));
val up_down_thm2 = save_pop_thm"up_down_thm2";
```

These are combined together to form a useful package of results:

SML
```
val up_down_clauses = save_thm("up_down_clauses",
        (all_∀_intro o list_∧_intro o map all_∀_elim)[up_down_thm1, up_down_thm2]);
```

If class $d$ does not dominate class $c$ then the down-set from $d$ is contained in the complement of the up-set from $c$.

SML
```
set_goal([], ⌜
∀c d• ¬ d dominates c ⇒ d ↓ ⊆ ∼(c ↑)
⌝);
a(PC_T1 "sets_ext" prove_tac(map get_spec
        [⌜∼⌝, ⌜$↑⌝, ⌜$↓⌝, ⌜lattice_bottom⌝]));
a(contr_tac THEN all_fc_tac[get_spec⌜lattice_bottom⌝]);
val up_down_thm3 = save_pop_thm"up_down_thm3";
```

If $R$ is a family of equivalence relations indexed by classes then *LiftRel R* is an *IndexedEquiv*.

SML
```
set_merge_pcs["'bin_rel", "hol2"];
set_goal([], ⌜
∀R•    (∀c• R c ∈ Equivalence) ⇒
       (LiftRel R) ∈ IndexedEquiv
⌝);
a(rewrite_tac(map get_spec[⌜LiftRel⌝, ⌜IndexedEquiv⌝, ⌜Equivalence⌝]) THEN REPEAT strip_tac);
(* *** Goal "1" *** *)
a(asm_rewrite_tac[]);
(* *** Goal "2" *** *)
a(asm_rewrite_tac[]);
a(spec_nth_asm_tac 3 ⌜s⌝ THEN all_asm_fc_tac[] THEN all_asm_fc_tac[]);
(* *** Goal "3" *** *)
a(asm_rewrite_tac[]);
a(spec_nth_asm_tac 4 ⌜s⌝ THEN1 all_asm_fc_tac[]);
a(spec_nth_asm_tac 4 ⌜s⌝ THEN all_asm_fc_tac[] THEN all_asm_fc_tac[]);
(* *** Goal "4" *** *)
a(spec_nth_asm_tac 3 ⌜s⌝ THEN1 all_asm_fc_tac[]);
val lift_rel_indexed_equiv_thm = save_pop_thm"lift_rel_indexed_equiv_thm";
```

*same_ins* is an indexed family of equivalence relations.

SML
```
set_goal([], ⌜
       ∀c• same_ins c ∈ Equivalence
⌝);
a(rewrite_tac(map get_spec[⌜same_ins⌝, ⌜Equivalence⌝, ⌜Let⌝]) THEN
       REPEAT strip_tac THEN asm_rewrite_tac[]);
val same_ins_equiv_thm = save_pop_thm"same_ins_equiv_thm";
```

*same_outs* is an indexed family of equivalence relations.

SML
```
set_goal([], ⌜
        ∀c• same_outs c ∈ Equivalence
⌝);
a(rewrite_tac(map get_spec[⌜same_outs⌝, ⌜Equivalence⌝, ⌜Let⌝]) THEN
        REPEAT strip_tac THEN asm_rewrite_tac[]);
val same_outs_equiv_thm = save_pop_thm"same_outs_equiv_thm";
```

When lifted they are therefore both *IndexedEquiv*s.

SML
```
set_goal([], ⌜
        LiftRel same_ins ∈ IndexedEquiv
∧       LiftRel same_outs ∈ IndexedEquiv
⌝);
a(REPEAT strip_tac THEN
        bc_tac[lift_rel_indexed_equiv_thm] THEN
        rewrite_tac[same_ins_equiv_thm, same_outs_equiv_thm]);
val same_ins_same_outs_indexed_equiv_thm =
        save_pop_thm"same_ins_same_outs_indexed_equiv_thm";
```

In uses of the following result $R$ will be something like *same_ins* or *same_outs* from fef003 i.e. a family of equivalence relations indexed by individual classes where R c denotes the relation "same at or below c". The result says that, under a certain anti-monotonicity condition, the family $R$ embeds in *LiftRel* $R$ as the subfamily indexed by the set of all down-sets.

SML
```
set_pc"hol";
set_goal([], ⌜
∀R•     (∀c• R c ∈ Equivalence) ∧
        (∀c d•c dominates d ⇒ R c ⊆ R d) ⇒
        ∀c• (LiftRel R)(c ↓) = R c
⌝);
a(rewrite_tac[get_spec⌜LiftRel⌝] THEN REPEAT strip_tac);
a(PC_T1 "sets_ext" REPEAT strip_tac);
(* *** Goal "1" *** *)
a(spec_nth_asm_tac 1 ⌜R c⌝);
a(spec_nth_asm_tac 1 ⌜c⌝);
a(all_fc_tac[up_down_clauses]);
(* *** Goal "2" *** *)
a(asm_ante_tac ⌜c' ∈ c ↓⌝ THEN asm_rewrite_tac[get_spec⌜$↓⌝]);
a(strip_tac THEN all_asm_fc_tac[] THEN PC_T1 "sets_ext" all_asm_fc_tac[]);
val equiv_anti_mono_lift_rel_thm = save_pop_thm"equiv_anti_mono_lift_rel_thm";
```

*same_ins* is anti-montonic in an appropriate sense.

SML

```
set_pc"hol2";
set_goal([], ⌜
        ∀c d• c dominates d ⇒ same_ins c ⊆ same_ins d
⌝);
a(rewrite_tac(map get_spec[⌜same_ins⌝, ⌜Let⌝]) THEN
        REPEAT strip_tac THEN rename_tac[]);
a(LEMMA_T⌜∀x•
        x ↾ {(q, c')|d dominates c'} =
        (x ↾ {(q, c')|c dominates c'}) ↾ {(q, c')|d dominates c'}⌝
        (fn th => once_rewrite_tac [th] THEN asm_rewrite_tac[]));
a(rewrite_tac[↑_↾_thm]);
a(LEMMA_T⌜{(q, c')|c dominates c'} ∩ {(q, c')|d dominates c'} =
        {(q, c')|d dominates c'}⌝ rewrite_thm_tac);
(* *** Goal "1" *** *)
a(REPEAT strip_tac);
a(all_fc_tac[get_spec⌜$dominates⌝]);
val same_ins_anti_mono_thm = save_pop_thm"same_ins_anti_mono_thm";
```

*same_outs* is anti-montonic in an appropriate sense.

SML

```
set_pc"hol2";
set_goal([], ⌜
        ∀c d• c dominates d ⇒ same_outs c ⊆ same_outs d
⌝);
a(rewrite_tac(map get_spec[⌜same_outs⌝, ⌜Let⌝]) THEN
        REPEAT strip_tac THEN rename_tac[]);
a(LEMMA_T⌜∀x•
        x ↾ {(c', d')|d dominates c'} =
        (x ↾ {(c', d')|c dominates c'}) ↾ {(c', d')|d dominates c'}⌝
        (fn th => once_rewrite_tac [th] THEN asm_rewrite_tac[]));
a(rewrite_tac[↑_↾_thm]);
a(LEMMA_T⌜{(c', d')|c dominates c'} ∩ {(c', d')|d dominates c'} =
        {(c', d')|d dominates c'}⌝ rewrite_thm_tac);
(* *** Goal "1" *** *)
a(REPEAT strip_tac);
a(all_fc_tac[get_spec⌜$dominates⌝]);
val same_outs_anti_mono_thm = save_pop_thm"same_outs_anti_mono_thm";
```

The result says that the family *same_ins* embeds in *LiftRel same_ins* as the subfamily indexed by the set of all down-sets and similarly for *same_outs*

SML
```
set_pc"hol";
set_goal([], ⌜
∀c•     (LiftRel same_ins)(c ↓) = same_ins c
∧       (LiftRel same_outs)(c ↓) = same_outs c
⌝);
a(REPEAT strip_tac THEN intro_∀_tac(⌜c:Class⌝, ⌜c:Class⌝) THEN
        bc_tac[equiv_anti_mono_lift_rel_thm] THEN
        rewrite_tac
        [same_ins_equiv_thm, same_outs_equiv_thm,
        same_ins_anti_mono_thm, same_outs_anti_mono_thm]);
val lift_rel_same_ins_same_outs_down_set_thm =
        save_pop_thm"lift_rel_same_ins_same_outs_down_set_thm";
```

We are now in a position to prove *conj_040_1* which shows that the original security policy in [1] is a special case of the more generic flow policy given in [2].

SML
```
set_pc"hol";
set_goal([], conj_040_1);
a(rewrite_tac(map get_spec[⌜secure⌝, ⌜ml_secure⌝, ⌜x_ml_secure⌝]) THEN
        REPEAT strip_tac);
(* *** Goal "1" *** *)
a(rewrite_tac[same_ins_same_outs_indexed_equiv_thm]);
(* *** Goal "2" *** *)
a(rewrite_tac[same_ins_same_outs_indexed_equiv_thm]);
(* *** Goal "3" *** *)
a(POP_ASM_T ante_tac THEN
        asm_rewrite_tac[lift_rel_same_ins_same_outs_down_set_thm]);
(* *** Goal "4" *** *)
a(swap_nth_asm_concl_tac 2 THEN
        asm_rewrite_tac[lift_rel_same_ins_same_outs_down_set_thm]);
a(contr_tac THEN all_asm_fc_tac[]);
val thm_040_1 = save_pop_thm"thm_040_1";
```

## 2.2 Proof of thm_040_2

First an elementary result about disrtibuted union.

SML
```
val ⋃_⊆_thm = pc_rule1 "sets_ext" prove_rule[]⌜∀a b• a ⊆ b ⇒ ⋃a ⊆ ⋃b⌝;
```

SML
```
set_merge_pcs["'bin_rel", "hol2"];
set_goal([], ⌜∀c• SameLabVal c ∈ Equivalence⌝);
a(rewrite_tac(map get_spec[⌜SameLabVal⌝, ⌜Equivalence⌝])
        THEN REPEAT strip_tac THEN PC_T1 "prop_eq" asm_prove_tac[]);
val same_lab_val_equiv_thm = save_pop_thm"same_lab_val_equiv_thm";
```

SML
```
set_pc"hol";
set_goal([], ⌜LiftRel SameLabVal ∈ IndexedEquiv⌝);
a(bc_tac[lift_rel_indexed_equiv_thm] THEN rewrite_tac[same_lab_val_equiv_thm]);
val lift_rel_same_lab_val_equiv_thm =
        save_pop_thm"lift_rel_same_lab_val_equiv_thm";
```

SML
```
set_pc"hol2";
set_goal([], ⌜
        ∀c d•c dominates d ⇒ SameLabVal c ⊆ SameLabVal d
⌝);
a(rewrite_tac(map get_spec[⌜SameLabVal⌝]) THEN REPEAT strip_tac);
a(all_fc_tac[get_spec⌜$dominates⌝]);
val same_lab_val_anti_mono_thm = save_pop_thm"same_lab_val_anti_mono_thm";
```

SML
```
set_pc"hol";
set_goal([], ⌜
∀c•     (LiftRel SameLabVal)(c ↓) = SameLabVal c
⌝);
a(bc_tac[equiv_anti_mono_lift_rel_thm] THEN
        rewrite_tac[same_lab_val_equiv_thm, same_lab_val_anti_mono_thm]);
val lift_rel_same_lab_val_down_set_thm =
        save_pop_thm"lift_rel_same_lab_val_down_set_thm";
```

SML

```
set_pc"hol";
set_goal([], conj_040_2);
a(rewrite_tac(lift_rel_same_lab_val_down_set_thm:: map get_spec
        [⌜BoundedObs⌝, ⌜ObservedValue⌝, ⌜SameLabVal⌝,
        ⌜x_ml_secure⌝, ⌜Influenced⌝]) THEN
        PC_T1 "hol2" REPEAT strip_tac);
(* *** Goal "1" *** *)
a(rename_tac[(⌜x'⌝, "d")] THEN rewrite_tac[get_spec⌜$↓⌝]);
a(cases_tac⌜∀e•e dominates d⌝ THEN1 asm_rewrite_tac[]);
a(list_spec_nth_asm_tac 4 [⌜e⌝, ⌜x⌝, ⌜y⌝]);
a(all_fc_tac[up_down_thm3]);
a(LIST_GET_NTH_ASM_T [8]
        (ALL_FC_T (PC_T1 "sets_ext" all_fc_tac) o
                map (rewrite_rule[get_spec⌜IndexedEquiv⌝]))));
(* *** Goal "2" *** *)
a(rename_tac[(⌜x'⌝, "d")] THEN rewrite_tac[get_spec⌜$↓⌝]);
a(contr_tac);
a(all_fc_tac[up_down_thm3]);
a(LIST_GET_NTH_ASM_T [7]
        (ALL_FC_T (PC_T1 "sets_ext" all_fc_tac) o
                map (rewrite_rule[get_spec⌜IndexedEquiv⌝]))));
a(list_spec_nth_asm_tac 6 [⌜C x⌝, ⌜x⌝, ⌜y⌝]);
a(swap_nth_asm_concl_tac 1 THEN rewrite_tac[get_spec⌜lattice_bottom⌝]);
val thm_040_2 = save_pop_thm"thm_040_2";
```

### 2.3 Proof of Theorem thm_040_3

SML
```
set_pc"hol";
set_goal([], ⌜∀c•c ↓ = ⋂{A | ∃d•¬c dominates d ∧ A = ∼(d ↑)}⌝);
a(rewrite_tac(map get_spec[⌜$↓⌝, ⌜$↑⌝]) THEN
        PC_T1 "sets_ext" REPEAT strip_tac);
(* *** Goal "1" *** *)
a(asm_rewrite_tac[get_spec⌜∼⌝]);
a(contr_tac THEN all_fc_tac[get_spec⌜$dominates⌝]);
(* *** Goal "2" *** *)
a(cases_tac⌜c dominates x⌝);
a(spec_nth_asm_tac 2 ⌜∼{y | y dominates x}⌝);
(* *** Goal "2.1" *** *)
a(spec_nth_asm_tac 1 ⌜x⌝);
(* *** Goal "2.2" *** *)
a(POP_ASM_T ante_tac THEN rewrite_tac(map get_spec[⌜∼⌝, ⌜$dominates⌝]));
val down_⋂_comp_up_thm = save_pop_thm"down_⋂_comp_up_thm";
```

# 3 PROOFS FOR FEF042

The following ProofPower instructions open the theory *fef042*.

SML
```
open_theory "fef042";
map delete_thm ((flat o (map fst) o get_thms) "fef042");
set_pc "hol";
```

SML
```
(delete_pc"'tree" handle Fail _ => ());
new_pc"'tree";
set_∃_cd_thms[tree_prim_rec_thm] "'tree";
set_merge_pcs ["'tree", "hol"];
```

SML
```
push_goal([], ⌜∀x y• MkTree x = MkTree y ⇒ x = y⌝);
a(REPEAT strip_tac);
a(lemma_tac⌜∃f• ∀z• f(MkTree z) = z⌝ THEN1 prove_∃_tac);
a(lemma_tac⌜y = f(MkTree x)⌝ THEN1
        (GET_NTH_ASM_T 2 rewrite_thm_tac THEN POP_ASM_T rewrite_thm_tac));
a(POP_ASM_T rewrite_thm_tac THEN POP_ASM_T rewrite_thm_tac);
val mk_tree_one_one_thm = save_pop_thm"mk_tree_one_one_thm";
```

SML

```
push_goal([], ⌜∀t• ∃x• t = MkTree x⌝);
a(REPEAT strip_tac);
a(gen_induction_tac tree_induction_thm ⌜t⌝);
a(∃_tac⌜x⌝ THEN REPEAT strip_tac);
val mk_tree_onto_thm = save_pop_thm"mk_tree_onto_thm";
```

SML

```
push_consistency_goal⌜MkObj⌝;
a(lemma_tac⌜∃mk_obj•∀ c: Class; t : Text; os : Obj LIST•
        mk_obj (c, t, os) = MkTree ((c, t), os)⌝
        THEN1 prove_∃_tac);
a(lemma_tac⌜∃object_class object_text object_refers•
        ∀ctos: (Class × Text) × Obj LIST•
        object_class (MkTree (ctos)) = Fst (Fst ctos)
∧       object_text (MkTree (ctos)) = Snd(Fst ctos)
∧       object_refers (MkTree (ctos)) = Snd ctos⌝
        THEN1 prove_∃_tac);
a(∃_tac ⌜(mk_obj, object_class, object_text, object_refers)⌝);
a(asm_rewrite_tac[]);
save_consistency_thm⌜MkObj⌝(pop_thm());
```

SML

```
push_goal([], ⌜∀ d• ∃1 h• ∀ c t os•
        h (MkObj (c, t, os)) = d c t (Map h os)⌝);
a(REPEAT strip_tac);
a(strip_asm_tac (∀_elim ⌜(λx:(Class × Text) × Obj LIST; y•
        d (Fst(Fst x)) (Snd(Fst x)) y)⌝ tree_prim_rec_thm));
a(∃1_tac⌜h⌝ THEN asm_rewrite_tac[get_spec⌜MkObj⌝]);
a(REPEAT strip_tac);
a(lemma_tac⌜∀ ctos• h' (MkTree ctos) =
                d (Fst (Fst ctos)) (Snd (Fst ctos)) (Map h' (Snd ctos))⌝
        THEN REPEAT strip_tac);
(* *** Goal "1" *** *)
a(LEMMA_T⌜ctos = ((Fst (Fst ctos), Snd (Fst ctos)), Snd ctos)⌝
        once_rewrite_thm_tac THEN1 rewrite_tac[]);
a(TOP_ASM_T pure_rewrite_thm_tac);
a(rewrite_tac[]);
(* *** Goal "2" *** *)
a(lemma_tac⌜∀ x
          • h' (MkTree x)
              = (λ x y• d (Fst (Fst x)) (Snd (Fst x)) y) x (Map h' (Snd x))⌝
        THEN1 asm_rewrite_tac[]);
a(all_asm_fc_tac[]);
val obj_prim_rec_thm = save_pop_thm"obj_prim_rec_thm";
```

SML

```
push_goal([], ⌜
(∀ x• MkObj x = MkTree ((Fst x, Fst(Snd x)), Snd(Snd x))) ∧
∀ ctos• objectClass(MkTree ctos) = Fst (Fst ctos)
∧       objectContains(MkTree ctos) = Snd (Fst ctos)
∧       objectRefers(MkTree ctos) = Snd ctos
⌝);
a(lemma_tac ⌜∀ x• MkObj x = MkTree ((Fst x, Fst(Snd x)), Snd(Snd x))⌝);
(* *** Goal "1" *** *)
a(strip_tac);
a(LEMMA_T ⌜x = (Fst x, (Fst(Snd x)), Snd(Snd x))⌝ once_rewrite_thm_tac
        THEN1 rewrite_tac[]);
a(pure_rewrite_tac[get_spec⌜MkObj⌝]);
a(rewrite_tac[]);
(* *** Goal "2" *** *)
a(ante_tac (get_spec⌜MkObj⌝) THEN asm_rewrite_tac[]);
a(strip_tac THEN strip_tac);
a(LEMMA_T ⌜ctos = ((Fst (Fst ctos), Snd (Fst ctos)), Snd ctos)⌝ once_rewrite_thm_tac
        THEN1 rewrite_tac[]);
a(pure_asm_rewrite_tac[]);
a(rewrite_tac[]);
val object_clauses = save_pop_thm"object_clauses";
```

SML

### 3.1 Consistency of *list_machine*

SML

```
push_consistency_goal⌜lift_machine⌝;
a (prove_∃_tac THEN strip_tac);
a (lemma_tac ⌜∃ lm •
        (lm [] = λs•([], s))
 ∧ (∀ r rl •
        lm (Cons r rl) =
        λ s • (let out = Output mch' (s, r) and s' = Next mch' (s, r)
            in let (outl, final_state) = lm rl s'
               in (Cons out outl, final_state)))⌝
        THEN1 prove_∃_tac);
a (∃_tac ⌜λs rl• lm rl s⌝);
a (asm_rewrite_tac[]);
save_consistency_thm⌜lift_machine⌝(pop_thm());
```

Following is lengthy but not really difficult. The basic idea is to define a propositional function by primitive recursion over trees which represents the set function required for the witness. The proof is then just a check that the formula defining the propositional function is essentially the same as the one the witness is required to satisfy. This involves (inter alia) two inductive proofs of two little lemmas about the formulae which are used to "combine the recursive applications".

SML

```
push_consistency_goal⌜identicalObj⌝;
a(lemma_tac ⌜∃io : Obj → Class → Obj → BOOL• ∀ctos c o2•
        io (MkTree ctos) c o2 ⇔
        ∃ c1 t1 os1 c2 t2 os2•
                (MkTree ctos) = MkObj (c1, t1, os1)
        ∧       o2 = MkObj (c2, t2, os2)
        ∧       c1 = c2
        ∧       (       c dominates c1
                ⇒       t1 = t2
                ∧       Length os1 = Length os2
                ∧       ∀p o2• (p, o2) ∈
                        Elems (Combine (Map io (Snd ctos)) os2)
                        ⇒       p c o2)⌝
        THEN1 prove_∃_tac);
a(∃_tac⌜λc• {(o1, o2) | io o1 c o2}⌝ THEN
        PC_T1 "sets_ext1" asm_rewrite_tac[object_clauses]);
a(PC_T1 "sets_ext1" REPEAT strip_tac);
(* *** Goal "1" *** *)
a(strip_asm_tac(∀_elim⌜x1⌝ mk_tree_onto_thm) THEN var_elim_nth_asm_tac 1);
a(strip_asm_tac(∀_elim⌜x2⌝ mk_tree_onto_thm) THEN var_elim_nth_asm_tac 1);
a(MAP_EVERY ∃_tac[
        ⌜Fst(Fst x)⌝, ⌜Snd(Fst x)⌝, ⌜Snd x⌝,
        ⌜Fst(Fst x')⌝, ⌜Snd(Fst x')⌝, ⌜Snd x'⌝]
        THEN rewrite_tac[]);
a(POP_ASM_T ante_tac THEN asm_rewrite_tac[object_clauses]);
a(strip_tac);
(* *** Goal "1.1" *** *)
a(all_fc_tac[mk_tree_one_one_thm]);
a(MAP_EVERY var_elim_nth_asm_tac [1,1,2]);
a(asm_rewrite_tac[]);
(* *** Goal "1.2" *** *)
a(all_fc_tac[mk_tree_one_one_thm]);
a(MAP_EVERY var_elim_nth_asm_tac [1,1,3,3]);
a(POP_ASM_T ante_tac THEN asm_rewrite_tac[]);
a(POP_ASM_T ante_tac);
a(intro_∀_tac(⌜os1⌝, ⌜os1⌝));
a(list_induction_tac⌜os2⌝);
(* *** Goal "1.2.1" *** *)
a(rewrite_tac[get_spec⌜Length⌝, length_0_thm]);
a(REPEAT_N 2 strip_tac THEN asm_rewrite_tac(map get_spec
        [⌜Combine⌝, ⌜Map⌝, ⌜Elems⌝]));
(* *** Goal "1.2.2" *** *)
```



```
a(rewrite_tac[get_spec⌜Length⌝, length_0_thm]);
a(REPEAT ∀_tac);
a(strip_asm_tac(∀_elim⌜os1⌝ list_cases_thm) THEN var_elim_nth_asm_tac 1
        THEN rewrite_tac[get_spec⌜Length⌝]);
a(PC_T1 "sets_ext1" rewrite_tac(map get_spec [⌜Combine⌝, ⌜Map⌝, ⌜Elems⌝]));
```

SML

```
push_consistency_goal⌜FilterObj⌝;
a(rewrite_tac[object_clauses]);
a(lemma_tac ⌜∃f : Class → Obj → Obj• ∀ c dtos•
        f c (MkTree dtos) =
        let (d, t) = Fst dtos
        in      if c dominates d
                  then MkTree ((d, t), Map (f c) (Snd dtos))
                  else MkTree ((d, Arbitrary), Arbitrary)⌝
        THEN1 prove_∃_tac);
a(∃_tac⌜f⌝ THEN asm_rewrite_tac[let_def]);
save_consistency_thm⌜FilterObj⌝(pop_thm());
```

SML

```
set_merge_pcs["'bin_rel", "hol2"];
```

SML

```
val conj_042_1_1 = ⌜
∀c ob1 ob2•
        (ob1, ob2) ∈ identicalObj c
⇔       FilterObj c ob1 = FilterObj c ob2⌝;
```

SML

```
push_goal([], ⌜∀l x y• (x, y) ∈ Elems(Combine l l) ⇔  y ∈ Elems l ∧ x = y⌝);
a(strip_tac);
a(list_induction_tac⌜l⌝ THEN asm_rewrite_tac(map get_spec[⌜Elems⌝, ⌜Combine⌝]));
a(REPEAT strip_tac THEN_TRY all_var_elim_asm_tac1 THEN REPEAT strip_tac);
val elems_combine_thm = save_pop_thm"elems_combine_thm";
```

SML

```
push_goal([], ⌜∀l1 l2 x y•
        Length l1 = Length l2 ∧ (x, y) ∈ Elems(Combine l1 l2)
        ⇒ x ∈ Elems l1⌝);
a(strip_tac);
a(list_induction_tac⌜l1⌝);
(* *** Goal "1" *** *)
a(REPEAT strip_tac);
a(all_asm_ante_tac THEN
        strip_asm_tac(∀_elim⌜l2⌝ list_cases_thm) THEN var_elim_nth_asm_tac 1
        THEN rewrite_tac(map get_spec[⌜Elems⌝, ⌜Combine⌝, ⌜Length⌝]));
(* *** Goal "2" *** *)
a(REPEAT strip_tac);
a(all_asm_ante_tac THEN
        strip_asm_tac(∀_elim⌜l2⌝ list_cases_thm) THEN var_elim_nth_asm_tac 1
        THEN rewrite_tac(map get_spec[⌜Elems⌝, ⌜Combine⌝, ⌜Length⌝]));
a(REPEAT strip_tac THEN all_asm_fc_tac[]);
val elems_combine_elems_thm = save_pop_thm"elems_combine_elems_thm";
```

SML

```
push_goal([], ⌜∀l1 l2 x1 x2•
        Length l1 = Length l2 ∧
        (x1, x2) ∈ Elems(Combine l1 l2) ⇒ (x2, x1) ∈ Elems(Combine l2 l1)  ⌝);
a(strip_tac);
a(list_induction_tac⌜l1⌝);
(* *** Goal "1" *** *)
a(REPEAT strip_tac);
a(all_asm_ante_tac THEN
        strip_asm_tac(∀_elim⌜l2⌝ list_cases_thm) THEN var_elim_nth_asm_tac 1
        THEN rewrite_tac(map get_spec[⌜Elems⌝, ⌜Combine⌝, ⌜Length⌝]));
(* *** Goal "2" *** *)
a(REPEAT strip_tac);
a(all_asm_ante_tac THEN
        strip_asm_tac(∀_elim⌜l2⌝ list_cases_thm) THEN var_elim_nth_asm_tac 1
        THEN rewrite_tac(map get_spec[⌜Elems⌝, ⌜Combine⌝, ⌜Length⌝]));
a(REPEAT strip_tac THEN all_asm_fc_tac[]);
val elems_combine_swap_thm = save_pop_thm"elems_combine_swap_thm";
```

SML

```
push_goal([], ⌜∀f l1 l2 x1 x2•
        Length l1 = Length l2 ∧
        (x1, x2) ∈ Elems(Combine l1 l2) ∧
        Map f l1 = Map f l2 ⇒
        f x1 = f x2⌝);
a(REPEAT_N 2 strip_tac);
a(list_induction_tac⌜l1⌝);
(* *** Goal "1" *** *)
a(REPEAT strip_tac);
a(all_asm_ante_tac THEN
        strip_asm_tac(∀_elim⌜l2⌝ list_cases_thm) THEN var_elim_nth_asm_tac 1
        THEN rewrite_tac(map get_spec[⌜Elems⌝, ⌜Combine⌝, ⌜Length⌝]));
(* *** Goal "2" *** *)
a(REPEAT strip_tac);
a(all_asm_ante_tac THEN
        strip_asm_tac(∀_elim⌜l2⌝ list_cases_thm) THEN var_elim_nth_asm_tac 1
        THEN rewrite_tac(map get_spec[⌜Map⌝, ⌜Elems⌝, ⌜Combine⌝, ⌜Length⌝]));
a(REPEAT strip_tac THEN all_asm_fc_tac[]);
a(all_var_elim_asm_tac1);
val elems_combine_map_thm = save_pop_thm"elems_combine_map_thm";
```

SML

```
push_goal([], ⌜∀f l1 l2•
        Length l1 = Length l2 ∧
        (∀y1 y2• (y1, y2) ∈ Elems(Combine l1 l2) ⇒ f y1 = f y2) ⇒
        Map f l1 = Map f l2⌝);
a(REPEAT_N 2 strip_tac);
a(list_induction_tac⌜l1⌝);
(* *** Goal "1" *** *)
a(REPEAT strip_tac);
a(all_asm_ante_tac THEN
        strip_asm_tac(∀_elim⌜l2⌝ list_cases_thm) THEN var_elim_nth_asm_tac 1
        THEN rewrite_tac(map get_spec[⌜Elems⌝, ⌜Combine⌝, ⌜Length⌝]));
(* *** Goal "2" *** *)
a(REPEAT strip_tac);
a(all_asm_ante_tac THEN
        strip_asm_tac(∀_elim⌜l2⌝ list_cases_thm) THEN var_elim_nth_asm_tac 1
        THEN rewrite_tac(map get_spec[⌜Map⌝, ⌜Elems⌝, ⌜Combine⌝, ⌜Length⌝]));
a(REPEAT strip_tac THEN all_asm_fc_tac[]);
(* *** Goal "2.1" *** *)
a(list_spec_nth_asm_tac 1 [⌜x⌝, ⌜x'⌝]);
(* *** Goal "2.2" *** *)
a(list_spec_nth_asm_tac 3 [⌜list2⌝]);
a(all_asm_fc_tac[]);
val elems_combine_map_thm1 = save_pop_thm"elems_combine_map_thm1";
```

SML

```
push_goal([], ⌜∀ P
  • (∀ c t ts• (∀ t• t ∈ Elems ts ⇒ P t) ⇒ P (MkObj(c, t, ts))) ⇒ (∀ t• P t)⌝);
a(rewrite_tac[object_clauses] THEN REPEAT strip_tac);
a(gen_induction_tac tree_induction_thm ⌜t⌝);
a(POP_ASM_T ante_tac);
a(PC_T1 "predicates" lemma_tac⌜∃c t ts• x = ((c, t), ts)⌝
        THEN1 (MAP_EVERY ∃_tac[⌜Fst (Fst x)⌝, ⌜Snd (Fst x)⌝, ⌜Snd x⌝]
                        THEN REPEAT strip_tac));
a(asm_rewrite_tac[]);
val obj_induction_thm = save_pop_thm"obj_induction_thm";
```

SML

```
set_goal([], ⌜∀ c ob1 ob2•
        (ob1, ob2) ∈ identicalObj c ⇒ FilterObj c ob1 = FilterObj c ob2⌝);
a(REPEAT_N 2 strip_tac);
a(gen_induction_tac obj_induction_thm ⌜ob1⌝);
a(rewrite_tac(map get_spec[⌜FilterObj⌝]));
a(once_rewrite_tac(map get_spec[⌜identicalObj⌝]));
a(REPEAT strip_tac);
(* *** Goal "1" *** *)
a(all_var_elim_asm_tac1);
a(rewrite_tac(map get_spec[⌜FilterObj⌝]));
a(DROP_NTH_ASM_T 2 ante_tac
        THEN rewrite_tac[object_clauses]);
a(REPEAT strip_tac THEN all_fc_tac[mk_tree_one_one_thm]);
a(all_var_elim_asm_tac1);
a(asm_rewrite_tac[]);
(* *** Goal "2" *** *)
a(all_var_elim_asm_tac1);
a(rewrite_tac(map get_spec[⌜FilterObj⌝]));
a(DROP_NTH_ASM_T 3 ante_tac
        THEN rewrite_tac[object_clauses]);
a(REPEAT strip_tac THEN all_fc_tac[mk_tree_one_one_thm]);
a(all_var_elim_asm_tac1);
a(cases_tac ⌜c dominates c₂⌝ THEN asm_rewrite_tac[]);
a(LEMMA_T ⌜Map (FilterObj c) ob1s = Map (FilterObj c) os₂⌝ rewrite_thm_tac);
a(POP_ASM_T discard_tac THEN all_asm_ante_tac THEN
        intro_∀_tac(⌜os₂⌝, ⌜os₂⌝)
        THEN list_induction_tac ⌜ob1s⌝);
(* *** Goal "2.1" *** *)
a(strip_tac);
a(strip_asm_tac(∀_elim⌜os₂⌝ list_cases_thm) THEN var_elim_nth_asm_tac 1
        THEN rewrite_tac(map get_spec[⌜Elems⌝, ⌜Combine⌝, ⌜Length⌝]));
(* *** Goal "2.2" *** *)
a(REPEAT_N 2 strip_tac);
a(strip_asm_tac(∀_elim⌜os₂⌝ list_cases_thm) THEN var_elim_nth_asm_tac 1
        THEN rewrite_tac(map get_spec[⌜Elems⌝, ⌜Combine⌝, ⌜Length⌝]));
a(REPEAT strip_tac);
a(rewrite_tac[get_spec⌜Map⌝]);
a(list_spec_nth_asm_tac 1 [⌜x⌝, ⌜x'⌝]);
a(list_spec_nth_asm_tac 4 [⌜x⌝]);
a(list_spec_nth_asm_tac 1 [⌜x'⌝]);
a(POP_ASM_T rewrite_thm_tac);
a(list_spec_nth_asm_tac 6 [⌜list2⌝] THEN all_asm_fc_tac[]);
```

SML

```
set_goal([], ⌜∀ c ob1 ob2•
        FilterObj c ob1 = FilterObj c ob2 ⇒ (ob1, ob2) ∈ identicalObj c⌝);
a(REPEAT_N 2 strip_tac);
a(gen_induction_tac obj_induction_thm ⌜ob1⌝);
a(rewrite_tac(map get_spec[⌜FilterObj⌝]));
a(once_rewrite_tac(map get_spec[⌜identicalObj⌝]));
a(REPEAT strip_tac);
a(rewrite_tac[object_clauses]);
a(strip_asm_tac(∀_elim⌜ob2⌝ mk_tree_onto_thm) THEN var_elim_nth_asm_tac 1);
a(PC_T1 "predicates" lemma_tac⌜∃d u us• x = ((d, u), us)⌝
        THEN1 (MAP_EVERY ∃_tac[⌜Fst (Fst x)⌝, ⌜Snd (Fst x)⌝, ⌜Snd x⌝]
                THEN REPEAT strip_tac));
a(var_elim_nth_asm_tac 1);
a(POP_ASM_T (strip_asm_tac o rewrite_rule[object_clauses,
        rewrite_rule[object_clauses](get_spec⌜FilterObj⌝)]));
a(lemma_tac⌜c' = d⌝);
(* *** Goal "1" *** *)
a(POP_ASM_T ante_tac THEN cases_tac ⌜c dominates c'⌝
        THEN cases_tac⌜c dominates d⌝ THEN asm_rewrite_tac[]
        THEN REPEAT strip_tac THEN all_fc_tac[mk_tree_one_one_thm]);
(* *** Goal "2" *** *)
a(var_elim_nth_asm_tac 1);
a(MAP_EVERY ∃_tac[⌜d⌝, ⌜ob1⌝, ⌜ob1s⌝, ⌜d⌝, ⌜u⌝, ⌜us⌝]);
a(POP_ASM_T ante_tac THEN cases_tac⌜c dominates d⌝ THEN asm_rewrite_tac[]);
a(strip_tac THEN all_fc_tac[mk_tree_one_one_thm]);
a(var_elim_nth_asm_tac 2);
a(LEMMA_T⌜Length(Map (FilterObj c) ob1s) = Length(Map (FilterObj c) us)⌝
        (strip_asm_tac o rewrite_rule[length_map_thm]) THEN1 asm_rewrite_tac[]);
a(REPEAT strip_tac);
a(all_fc_tac[elems_combine_map_thm, elems_combine_elems_thm]);
a(all_asm_fc_tac[]);
val identical_obj_filter_obj_thm2 = save_pop_thm"identical_obj_filter_obj_thm2";
```

SML

```
set_goal([], ⌜
        ∀ c• identicalObj c = {(ob1, ob2) | FilterObj c ob1 = FilterObj c ob2} ⌝);
a(REPEAT strip_tac THEN
        all_fc_tac[identical_obj_filter_obj_thm1, identical_obj_filter_obj_thm2]);
val identical_obj_filter_obj_thm = save_pop_thm"identical_obj_filter_obj_thm";
```

SML
```
push_goal([], ⌜∀c ob• (ob, ob) ∈ identicalObj c⌝);
a(rewrite_tac[identical_obj_filter_obj_thm]);
val identical_obj_refl_thm = save_pop_thm"identical_obj_refl_thm";
```

SML
```
push_goal([], ⌜∀c ob1 ob2•
        (ob1, ob2) ∈ identicalObj c ⇒ (ob2, ob1) ∈ identicalObj c⌝);
a(rewrite_tac[identical_obj_filter_obj_thm]);
a(PC_T1 "prop_eq" prove_tac[]);
val identical_obj_sym_thm = save_pop_thm"identical_obj_sym_thm";
```

SML
```
push_goal([], ⌜∀c ob1 ob2 ob3•
        (ob1, ob2) ∈ identicalObj c ∧ (ob2, ob3) ∈ identicalObj c
        ⇒ (ob1, ob3) ∈ identicalObj c⌝);
a(rewrite_tac[identical_obj_filter_obj_thm]);
a(PC_T1 "prop_eq" prove_tac[]);
val identical_obj_trans_thm = save_pop_thm"identical_obj_trans_thm";
```

SML
```
push_goal([], ⌜∀ c
          • identicalObj c ∈ Reflexive
              ∧ identicalObj c ∈ Symmetric
              ∧ identicalObj c ∈ Transitive⌝);
a (rewrite_tac ([identical_obj_refl_thm, identical_obj_sym_thm,
        identical_obj_trans_thm] @ (map get_spec
        [⌜Reflexive⌝, ⌜Symmetric⌝, ⌜Transitive⌝])));
val identical_obj_rft_thm = save_pop_thm "identical_obj_rft_thm";
```

SML
```
push_goal([], ⌜∀ c• identicalObj c ∈ Equivalence⌝);
a(PC_T1 "hol2" rewrite_tac[get_spec ⌜Equivalence⌝, identical_obj_rft_thm]);
val identical_obj_equiv_thm = save_pop_thm"identical_obj_equiv_thm";
```

SML

```
set_goal([], ⌜∀c•
        sameRequests c =
        {       (rs1, rs2)
        |       Map (FilterObj c o reqSsql) (rs1 ↾ VisibleReq c)
        =       Map (FilterObj c o reqSsql) (rs2 ↾ VisibleReq c)}
⌝);
a(rewrite_tac(identical_obj_filter_obj_thm::
        map get_spec[⌜sameRequests⌝, ⌜sameRequest⌝, ⌜Let⌝]));
a(REPEAT strip_tac);
a(lemma_tac ⌜∀ x1' x2'
          • (x1', x2') ∈ Elems (Combine (x1 ↾ VisibleReq c) (x2 ↾ VisibleReq c))
             ⇒ (FilterObj c o reqSsql) x1' = (FilterObj c o reqSsql) x2'⌝);
(* *** Goal "1.1" *** *)
a(rewrite_tac[] THEN REPEAT strip_tac THEN all_asm_fc_tac[]);
(* *** Goal "1.2" *** *)
a(all_fc_tac[elems_combine_map_thm1]);
(* *** Goal "2" *** *)
a(LEMMA_T⌜Length(Map (FilterObj c o reqSsql) (x1 ↾ VisibleReq c)) =
        Length(Map (FilterObj c o reqSsql) (x2 ↾ VisibleReq c))⌝
        (strip_asm_tac o rewrite_rule[length_map_thm]) THEN1 asm_rewrite_tac[]);
(* *** Goal "3" *** *)
a(LEMMA_T⌜(FilterObj c o reqSsql) x1' = (FilterObj c o reqSsql) x2'⌝
        (rewrite_thm_tac o rewrite_rule[]));
a(LEMMA_T⌜Length(Map (FilterObj c o reqSsql) (x1 ↾ VisibleReq c)) =
        Length(Map (FilterObj c o reqSsql) (x2 ↾ VisibleReq c))⌝
        (strip_asm_tac o rewrite_rule[length_map_thm]) THEN1 asm_rewrite_tac[]);
a(all_fc_tac[elems_combine_map_thm]);
val same_requests_filter_obj_thm = save_pop_thm "same_requests_filter_obj_thm";
```

SML

```
set_goal([], ⌜∀c• sameRequests c ∈ Equivalence⌝);
a(rewrite_tac(same_requests_filter_obj_thm::map get_spec[⌜Equivalence⌝])
        THEN REPEAT strip_tac
        THEN asm_rewrite_tac[]);
val same_requests_equiv_thm = save_pop_thm "same_requests_equiv_thm";
```

SML

```
set_goal([], ⌜∀c• sameOutputs c ∈ Equivalence⌝);
a(rewrite_tac(map get_spec[⌜sameOutputs⌝, ⌜Equivalence⌝])
        THEN REPEAT strip_tac
        THEN asm_rewrite_tac[]);
val same_outputs_equiv_thm = save_pop_thm "same_outputs_equiv_thm";
```

SML
```
set_goal([], ⌜LiftRel sameRequests ∈ IndexedEquiv⌝);
a(bc_tac[lift_rel_indexed_equiv_thm] THEN rewrite_tac[same_requests_equiv_thm]);
val same_requests_indexed_equiv_thm = save_pop_thm "same_requests_indexed_equiv_thm";
```

SML
```
set_goal([], ⌜LiftRel sameOutputs ∈ IndexedEquiv⌝);
a(bc_tac[lift_rel_indexed_equiv_thm] THEN rewrite_tac[same_outputs_equiv_thm]);
val same_outputs_indexed_equiv_thm = save_pop_thm "same_outputs_indexed_equiv_thm";
```

# 4 PROOFS ABOUT FEF043

The following ProofPower instructions open the theory *fef043*.

SML
```
open_theory "fef043";
map delete_thm (map (hd o fst) (get_thms "fef043"));
set_pc "hol2";
```

## 4.1 Consistency Proof

The simplest consistency proof is obtained by using a constant function as follows.

HOL Constant
```
        simplest_witness : 'State FactoredMachine
─────────────────────────────────────────────────

    simplest_witness =
        let next = λ sr • Arbitrary
        and output = λ sr •  Arbitrary
        and init = Arbitrary
        and f0 = λ c ob s • Arbitrary
        in      let (f1,f2,f3) = (f0,f0,f0)
        in      let     mach = MkMachine next output init
                and     facs = MkFactorisation f0 f1 f2 f3
                in      MkFactoredMachine mach facs
```

A less trivial witness is given by the following factored machine, which copies its input to its output:

HOL Constant

**identity_witness** : *'State FactoredMachine*

---

```
identity_witness =
    let next = Fst
    and output = reqSsql o Snd
    and init = Arbitrary
    and f0 = λ c ob s • ob
    in      let (f1,f2,f3) = (f0,f0,f0)
    in      let     mach = MkMachine next output init
            and     facs = MkFactorisation f0 f1 f2 f3
            in      MkFactoredMachine mach facs
```

The following function is used in the proofs that the above two candidates are *label_secure*.

HOL Constant

**purge_above_level** : $\mathbb{N} \to Class \to Obj \to Obj$

---

$$(\forall c\ ob \bullet \quad purge\_above\_level\ 0\ c\ ob = (FilterObj\ c\ ob)) \wedge$$
$$(\forall n\ c\ d\ t\ os \bullet$$
$$\quad purge\_above\_level\ (n+1)\ c\ (MkObj(d, t, os)) =$$
$$\quad MkObj(Arbitrary, Arbitrary, Map\ (purge\_above\_level\ n\ c)\ os))$$

SML

```
push_consistency_goal⌜purge_above_level⌝;
a(lemma_tac⌜
∃p:ℕ → Class → Obj → Obj•
(∀ob c•         p 0 c ob = FilterObj c ob) ∧
(∀n c tr•
        p (n + 1) c (MkTree tr) =
        MkTree((Arbitrary, Arbitrary), Map (p n c) (Snd tr)))⌝
        THEN1 MERGE_PCS_T ["'tree", "hol2"] prove_∃_tac);
a(∃_tac⌜p⌝ THEN asm_rewrite_tac[get_spec⌜MkObj⌝]);
save_consistency_thm⌜purge_above_level⌝ (pop_thm());
```

SML

```
set_goal([], ⌜∀ n c ob•
        purge_above_level (n + 1) c ob =
        MkTree((Arbitrary, Arbitrary), Map (purge_above_level n c) (objectRefers ob))⌝);
a(REPEAT strip_tac);
a(strip_asm_tac(∀_elim⌜ob⌝ mk_tree_onto_thm) THEN var_elim_nth_asm_tac 1);
a(LEMMA_T⌜MkTree x = MkObj(Fst (Fst x), Snd (Fst x), Snd x)⌝ rewrite_thm_tac
    THEN1 rewrite_tac[get_spec⌜MkObj⌝]);
a(rewrite_tac[get_spec⌜purge_above_level⌝]);
a(rewrite_tac[get_spec⌜MkObj⌝, mk_tree_one_one_thm]);
val purge_above_level_thm = save_pop_thm"purge_above_level_thm";
```

SML

```
set_goal([], ⌜∀ n c ob1 ob2•
    (ob1, ob2) ∈ same_at_c_below_level n c ⇒
        purge_above_level n c ob1 = purge_above_level n c ob2⌝);
a(strip_tac);
a(induction_tac⌜n⌝);
(* *** Goal "1" *** *)
a(rewrite_tac(map get_spec[⌜same_at_c_below_level⌝, ⌜purge_above_level⌝]));
a(rewrite_tac[identical_obj_filter_obj_thm]);
(* *** Goal "2" *** *)
a(rewrite_tac[ get_spec⌜same_at_c_below_level⌝, purge_above_level_thm ]);
a(REPEAT strip_tac);
a(LEMMA_T ⌜Map (purge_above_level n c) (objectRefers ob1) =
    Map (purge_above_level n c) (objectRefers ob2)⌝ rewrite_thm_tac);
a(all_asm_fc_tac[]);
a(lemma_tac⌜∀ x1 x2•
    (x1, x2) ∈ Elems(Combine(objectRefers ob1) (objectRefers ob2))
    ⇒ purge_above_level n c x1 = purge_above_level n c x2⌝);
(* *** Goal "2.1" *** *)
a(REPEAT strip_tac THEN all_asm_fc_tac[] THEN all_asm_fc_tac[]);
(* *** Goal "2.2" *** *)
a(all_fc_tac[elems_combine_map_thm1]);
val same_at_c_below_level_purge_above_level_thm1 =
    save_pop_thm"same_at_c_below_level_purge_above_level_thm1";
```

SML

```
set_goal([], ⌜∀ n c ob1 ob2•
        purge_above_level n c ob1 = purge_above_level n c ob2 ⇒
        (ob1, ob2) ∈ same_at_c_below_level n c⌝);
a(strip_tac);
a(induction_tac⌜n⌝);
(* *** Goal "1" *** *)
a(rewrite_tac(map get_spec[⌜same_at_c_below_level⌝, ⌜purge_above_level⌝]));
a(rewrite_tac[identical_obj_filter_obj_thm]);
(* *** Goal "2" *** *)
a(rewrite_tac[ get_spec⌜same_at_c_below_level⌝, purge_above_level_thm ]);
a(REPEAT strip_tac);
(* *** Goal "2.1" *** *)
a(all_fc_tac[mk_tree_one_one_thm]);
a(LEMMA_T⌜        Length(Map(purge_above_level n c) (objectRefers ob1)) =
                Length(Map(purge_above_level n c) (objectRefers ob2))⌝ ante_tac
        THEN1 asm_rewrite_tac[]);
a(rewrite_tac[length_map_thm]);
(* *** Goal "2.1" *** *)
a(all_fc_tac[mk_tree_one_one_thm]);
a(LEMMA_T⌜        Length(Map(purge_above_level n c) (objectRefers ob1)) =
                Length(Map(purge_above_level n c) (objectRefers ob2))⌝ ante_tac
        THEN1 asm_rewrite_tac[]);
a(rewrite_tac[length_map_thm] THEN strip_tac);
a(all_fc_tac[elems_combine_map_thm]);
a(all_asm_fc_tac[]);
val same_at_c_below_level_purge_above_level_thm2 =
        save_pop_thm"same_at_c_below_level_purge_above_level_thm2";
```

SML

```
set_goal([], ⌜∀ n c ob1 ob2•
        (ob1, ob2) ∈ same_at_c_below_level n c ⇔
        purge_above_level n c ob1 = purge_above_level n c ob2⌝);
a(REPEAT strip_tac THEN
        all_fc_tac[same_at_c_below_level_purge_above_level_thm1,
                same_at_c_below_level_purge_above_level_thm2]);
val same_at_c_below_level_purge_above_level_thm =
        save_pop_thm"same_at_c_below_level_purge_above_level_thm";
```

SML
```
set_goal([],⌜
        machine simplest_witness =
                MkMachine (λ sr •  Arbitrary) (λ sr •  Arbitrary) Arbitrary
 ∧      factors simplest_witness =
                MkFactorisation (λ c ob s • Arbitrary) (λ c ob s • Arbitrary)
                (λ c ob s • Arbitrary) (λ c ob s • Arbitrary)
 ⌝);
a (rewrite_tac (let_def::(map get_spec [⌜simplest_witness⌝,⌜machine⌝])));
val simple_witness_thm = save_pop_thm "simple_witness_thm";
```

SML
```
set_goal([],⌜
        machine identity_witness =
                MkMachine Fst (reqSsql o Snd) Arbitrary
 ∧      factors identity_witness =
                MkFactorisation (λ c ob s • ob) (λ c ob s • ob)
                (λ c ob s • ob) (λ c ob s • ob)
 ⌝);
a (rewrite_tac (let_def::(map get_spec [⌜identity_witness⌝,⌜machine⌝])));
val identity_witness_thm = save_pop_thm "identity_witness_thm";
```

SML
```
set_goal([],⌜∀n c• (same_at_c_below_level n c) ∈ Equivalence⌝);
a(MERGE_PCS_T1 ["'bin_rel", "hol2"] rewrite_tac[get_spec⌜Equivalence⌝,
        same_at_c_below_level_purge_above_level_thm]);
a(PC_T1 "prop_eq" prove_tac[]);
val same_at_c_below_level_equiv_thm = save_pop_thm "same_at_c_below_level_equiv_thm";
```

SML
```
set_goal([], ⌜∀n• LiftRel (same_at_c_below_level n) ∈ IndexedEquiv⌝);
a(strip_tac THEN bc_tac[lift_rel_indexed_equiv_thm]);
a(rewrite_tac[same_at_c_below_level_equiv_thm]);
val lift_rel_same_at_c_below_level_indexed_equiv_thm =
        save_pop_thm "lift_rel_same_at_c_below_level_indexed_equiv_thm";
set_goal([], ⌜∀n C x• (x, x) ∈ LiftRel (same_at_c_below_level n) C⌝);
a(REPEAT strip_tac);
a(ante_tac(∀_elim⌜n⌝lift_rel_same_at_c_below_level_indexed_equiv_thm));
a(rewrite_tac(map get_spec[⌜IndexedEquiv⌝, ⌜Equivalence⌝,  ⌜Reflexive⌝]));
a(REPEAT strip_tac THEN asm_rewrite_tac[]);
val lift_rel_same_at_c_below_level_refl_thm =
        save_pop_thm "lift_rel_same_at_c_below_level_refl_thm";
```

SML
```
set_goal([],⌜simplest_witness ∈ label_secure⌝);
a (rewrite_tac ((map get_spec [
        ⌜label_secure⌝ , ⌜label_secure_to⌝, ⌜factor0⌝
        ])@[let_def, simple_witness_thm]));
a (rewrite_tac ([let_def, same_requests_indexed_equiv_thm]@(map get_spec [
        ⌜ml_secure⌝,⌜x_ml_secure⌝,⌜special_machine⌝,⌜machine⌝,⌜Next⌝,⌜$∘f⌝])));
a (rewrite_tac[lift_rel_same_at_c_below_level_indexed_equiv_thm,
        lift_rel_same_at_c_below_level_refl_thm]);
val simplest_witness_label_secure_thm = save_pop_thm "simplest_witness_label_secure_thm";
```

SML
```
set_goal([],⌜identity_witness ∈ label_secure⌝);
a (rewrite_tac ((map get_spec [
        ⌜label_secure⌝ , ⌜label_secure_to⌝, ⌜factor0⌝
        ])@[let_def, identity_witness_thm]));
a (rewrite_tac ([let_def, same_requests_indexed_equiv_thm]@(map get_spec [
        ⌜ml_secure⌝,⌜x_ml_secure⌝,⌜special_machine⌝,⌜machine⌝,⌜Next⌝,⌜$∘f⌝])));
a (rewrite_tac[lift_rel_same_at_c_below_level_indexed_equiv_thm,
        lift_rel_same_at_c_below_level_refl_thm]);
val identity_witness_label_secure_thm = save_pop_thm "identity_witness_label_secure_thm";
```

# 5 CLOSING DOWN

The following ProofPower instruction restores the previous proof context.

SML
```
pop_pc();
```

# 6 INDEX