Assured Reconfiguration: Specification, Proofs, and Example

Elisabeth A. Strunk and Xiang Yin

Technical Report CS-2005-05
Department of Computer Science
University of Virginia
April 2005

Preface

This technical report contains the formal elements of a theory and infrastructure for assured reconfiguration. These elements are:

- the PVS specification of an abstract reconfiguration architecture
- the proof obligations for the architecture specification that were generated by the PVS type checker
- proofs of the obligations for the architecture specification and theorems contained in the architecture specification
- the specification of an example instantiation of the architecture that reflects typical avionics systems on an unmanned aerial vehicle
- the proof obligations for the instantiation that were generated by the PVS type checker
- proofs of the obligations for the instantiation and theorems contained in the instantiation.

For a thorough discussion of the architecture and example instantiation, see:

Strunk, Elisabeth A. “Reconfiguration Assurance in Embedded System Software.” Ph.D. dissertation, Department of Computer Science, University of Virginia, May 2005.

All running times listed in this document were generated on a Sun Fire 280R with two 1.2 GHz UltraSPARC III processors and 4 GB of RAM.

Contents

Part I

Architecture Specification

1. State and Operations

```
% In order to put requirements on data manipulation in an abstract manner,
% an abstract type system is created here. Ideally, a theory interpretation
% of this type system, including interpretations of uninterpreted types,
% will be built for each system.

% The data over which the system operates is modeled as a set of mappings from
% identifiers to values. This corresponds to global state for a system.

state : THEORY
BEGIN

% all system variables
data_id: NONEMPTY_TYPE

% all values that can be taken by any variable
data_value: TYPE = int

env_id: NONEMPTY_TYPE
env_param: NONEMPTY_TYPE

% implies no type rules, can restrict if needed
data_state: TYPE = [data_id -> data_value]

% the predicate and function types defined here allow restriction of access
% to global state, and essentially, support information hiding for the
% abstract model.

predicate(scope: set[data_id]) : TYPE =
{p: pred[data_state] | FORALL (st: data_state) :
    (p(st) =>
       FORALL (st2: data_state) :
           (FORALL (id: (scope)) : st2(id) = st(id)) => p(st2))}

func(scope: set[data_id]) : TYPE =
[# pre: predicate(scope),
    f: {f: [data_state -> data_state] |
       FORALL (d: data_state, id: data_id) :
          NOT scope(id) => f(d)(id) = d(id)}
#]

END state
```

2. Module Framework

```
% The module is the basic software component. Its predicates are defined,
% but no functions are defined specifically: the composition of module functions
% that is equivalent to application execution is modeled at the application level.

module : THEORY
BEGIN
IMPORTING state

% values for the module's service level parameter
module_svc: NONEMPTY_TYPE

% interpretation of implication over a module's restricted scope
implies(scope: set[data_id], sv: set[module_svc],
   p1, p2: [(sv) -> predicate((scope))]) : bool =
   FORALL (service: (sv), st: data_state) : p1(service)(st) => p2(service)(st)

module_spec: TYPE =
[#
   % data elements the module can change
   scope: set[data_id],

   % service levels the module provides
   sv: set[module_svc],

   % data element that holds the current value of the module's service
   % level parameter
   svclvl_parm: (scope),

   % module invariant for each service level
   inv: [(sv) -> predicate((scope))],

   % module precondition for each service level
   pre: {p: [(sv) -> predicate((scope))] | implies(scope, sv, p, inv)},

   % module transition condition for each service level
   trans: {p: [(sv) -> predicate((scope))] | implies(scope, sv, p, inv)},

   % module postcondition for each service level
   post: {p: [(sv) -> predicate((scope))] | implies(scope, sv, p, inv)}
#]

END module
```

3. Application

```
application : THEORY
BEGIN
IMPORTING module

% configurations the applications can be in
app_svclvl : NONEMPTY_TYPE

% application scope: conjunction of module scopes
app_scope(modules: set[module_spec]) : set[data_id] =
    {d: data_id | EXISTS (m: (modules)) : m`scope(d)}

% mapping from module specs to module service levels
service_map(modules: set[module_spec]) : TYPE =
    {m: [(modules) -> module_svc] | FORALL (mod: (modules)) : mod`sv(m(mod))}
```

3.1. Predicates

```
% application postcondition: conjunction of module postconditions for the
% application's configuration
post(modules: set[module_spec], map: service_map(modules), st: data_state) :
       bool =
    FORALL (m: (modules)) : m`post(map(m))(st)

% application transition condition: conjunction of module transition conditions
% for the application's configuration
trans(modules: set[module_spec], map: service_map(modules), st: data_state) :
       bool =
    FORALL (m: (modules)) : m`trans(map(m))(st)

% application precondition: conjunction of module preconditions for the
% application's configuration
pre(modules: set[module_spec], map: service_map(modules), st: data_state) : bool =
    FORALL (m: (modules)) : m`pre(map(m))(st)

% application invariant: conjunction of module invariants for the
% application's configuration
inv(modules: set[module_spec], map: service_map(modules), st: data_state) : bool =
    FORALL (m: (modules)) : m`inv(map(m))(st)
```

3.2. Application type

```
% application specification
app_spec: TYPE =
[#
    % application configuration
    svcs: set[app_svclvl],

    % application's constituent modules
    modules: {f: finite_set[module_spec] |
```

```
        nonempty?(f) AND
        FORALL (m1, m2: (f)) : NOT (m1 = m2) =>
           NOT EXISTS (d: data_id) : m1`scope(d) AND m2`scope(d)},

    % mapping from application configurations to module service levels
    svcmap: [(svcs) -> service_map(modules)],

    % entry point for application functionality
    execute: {f: [(svcs) -> func(app_scope(modules))] |
              FORALL (sv: (svcs), st: data_state) :
                 inv(modules, svcmap(sv), f(sv)`f(st))},

    % application functionality in preparation for a possible reconfiguration
    exec_halt: {f: [(svcs) -> func(app_scope(modules))] |
              FORALL (sv: (svcs), st: data_state) :
                 inv(modules, svcmap(sv), f(sv)`f(st))},

    % preparation for reconfiguration if this application is
    % the reconfiguration trigger
    halt: {f: [(svcs) -> func(app_scope(modules))] |
              FORALL (sv: (svcs), st: data_state) :
                 post(modules, svcmap(sv), f(sv)`f(st))},

    % function for the application to meet its transition condition
    prep: {f: [(svcs) -> func(app_scope(modules))] |
              FORALL (sv: (svcs), st: data_state) :
                 f(sv)`pre(st) = post(modules, svcmap(sv), st) AND
                 trans(modules, svcmap(sv), f(sv)`f(st))}
#]

END application
```

4. Reconfiguration Specification

```
reconf_spec: THEORY
BEGIN

IMPORTING SCRAM

% used to represent an inconsistent set of application configurations
indeterminate: speclvl

% ordering of applications, used to express execution dependencies
app_sequence(apps: finite_set[app_spec]) : TYPE =
    {s: finseq[(apps)] | s`length = card(apps) AND injective?(s`seq)}
```

4.1. Type

```
% reconfiguration specification
reconf_spec: TYPE =
[# % set of applications in the system
    apps: {appsp: finite_set[app_spec] |
        nonempty?(appsp) AND
        FORALL (app1, app2: (appsp)) : NOT (app1 = app2) =>
            % application scopes are disjoint
            NOT EXISTS (d: data_id) :
                app_scope(app1`modules)(d) AND app_scope(app2`modules)(d)},
    % system dependencies
    app_seq: app_sequence(apps),
    % set of configuration labels for the system
    S: set[{sp: speclvl | NOT sp = indeterminate}],
    % system environment type system
    E: valid_env,
    % possible environmental states
    R: reachable_env(E),
    % system-specific information on configurations and transitions
    SCRAM_info: {t: SCRAM_table(apps, S, E, R) | nonempty?(t`txns)},
    % function to choose a new configuration, if one is needed
    choose: {c: [(S), env((E)) -> (S)] |
        FORALL (s: (S), e: env((E))) :
            IF (NOT R`D(e)) THEN c(s, e) = s
            ELSIF NOT (reachable_state(apps, S, E, R, SCRAM_info, s, e)) THEN
                c(s, e) = s
            ELSE EXISTS (t: (SCRAM_info`txns)) :
                t`source = s AND t`trigger = e AND t`target = c(s, e)
            ENDIF},
    % time constraints on transitions
    T: [(S), (S) -> {t: real_time | t >= 4 * cycle_time}]
#]
```

4.2. Predicates

```
% composition functions for system predicates from application predicates
post(r: reconf_spec, svc: (r`S), st: data_state) : bool =
```

```
    FORALL (app: (r`apps)) :
       post(app`modules, app`svcmap(r`SCRAM_info`configs(svc)(app)), st)

trans(r: reconf_spec, svc: (r`S), st: data_state) : bool =
    FORALL (app: (r`apps)) :
       trans(app`modules, app`svcmap(r`SCRAM_info`configs(svc)(app)), st)

pre(r: reconf_spec, svc: (r`S), st: data_state) : bool =
    FORALL (app: (r`apps)) :
       pre(app`modules, app`svcmap(r`SCRAM_info`configs(svc)(app)), st)

inv(r: reconf_spec, svc: (r`S), st: data_state) : bool =
    FORALL (app: (r`apps)) :
       inv(app`modules, app`svcmap(r`SCRAM_info`configs(svc)(app)), st)

END reconf_spec
```

5. SCRAM

```
% Defines the application-specific data that must be input to the SCRAM.
% Timing of the SCRAM is encoded in the required sequence of states
% in a system trace.

SCRAM: THEORY
BEGIN
IMPORTING application
```

5.1. Environment

```
% Basic type system for environmental factors and their possible values
valid_env: TYPE = [env_id ->set[env_param]]
env(v: valid_env) : TYPE =
    {f: [env_id -> env_param] | FORALL (e: env_id) : member(f(e), v(e))}

% Possible transitions from one reachable environmental state to another
env_txn(E: valid_env, D: set[env(E)]): TYPE =
[# source: (D),
   target: (D)
#]

% Collection of possible environmental states and transitions that affect which
% system transition is appropriate
reachable_env(E: valid_env): TYPE =
[# D: set[env(E)],
   txns: set[env_txn(E, D)]
#]
```

5.2. Transitions & SCRAM input

```
% Possible system configurations
speclvl: NONEMPTY_TYPE

% Model of real time
real_time: TYPE = nat

% Length of an execution cycle. System synchrony means that real time is only
% necessary in the environment model.
cycle: TYPE = nat
% Correspondence between execution cycles and the real time a cycle takes
cycle_time: real_time

% application - service level mapping that ensures the evaluation of the
% function is a member of the set of application configurations for the
% domain application
valid_app_svcs(apps: set[app_spec]) : TYPE =
    {f: [(apps) -> app_svclvl] | FORALL (app: (apps)) : app`svcs(f(app))}

% mapping from system configurations to application configurations
sys_configs(S: set[speclvl], apps: set[app_spec]) : TYPE =
```

```
   [(S) -> valid_app_svcs(apps)]

% system transitions
transition(apps: set[app_spec], S: set[speclvl], E: valid_env,
   R: reachable_env(E)) : TYPE =
[# source: (S),
   target: (S),
   % if need a precondition on state for trigger to activate, encode it as
   % an environment variable
   trigger: (R`D)
#]

% predicate ensuring a transition can be taken under any hypothesized condition
covering_txns(apps: set[app_spec], S: set[speclvl], E: valid_env, R:
reachable_env(E),
   s: set[transition(apps, S, E, R)], primary: (S), start_env: set[(R`D)]): bool =
   % Have to cover all transitions out of any start state
   (FORALL (e: (start_env), t: (R`txns)) : t`source = e =>
         EXISTS (txn: (s)) : txn`source = primary AND txn`trigger = t`target) AND
   % include start state here so that there can be self-transitions from it
   (FORALL (source: (S), t_s: (s), d: (R`D)) :
      t_s`target = source AND t_s`trigger = d =>
         FORALL (t_e: (R`txns)) : t_e`source = d =>
            EXISTS (t_t: (s)) : t_t`trigger = t_e`target) AND
   % transitions have to be deterministic
   FORALL (source: (S), trigger: (R`D)) : NOT EXISTS (t1, t2: (s)) :
      t1 /= t2 AND t1`source = source AND t1`trigger = trigger AND
      t2`source = source AND t2`trigger = trigger

% collection of system configuration and transition data that must be input to
% the SCRAM
SCRAM_table(apps: set[app_spec], S: set[speclvl], E: valid_env, R:
reachable_env(E)) :
   TYPE =
[# configs: sys_configs(S, apps),
   primary: (S),
   % set of configurations where an interrupt signal may not occur during
   % operation. Note that this set can be empty (e.g., in the case that interrupts
   % go between two configurations of equal value, depending on circumstances).
   safe: set[(S)],
   start_env: set[(R`D)],
   txns: {s: set[transition(apps, S, E, R)] |
         covering_txns(apps, S, E, R, s, primary, start_env)}
#]

% possible combinations of system and environmental states
sys_env_state(S: set[speclvl], E: valid_env, D: set[env(E)]) : TYPE =
[# s: (S),
   e: (D)
#]

% restricts the type above to states that could occur during operation
reachable_state(apps: set[app_spec], S: set[speclvl], E: valid_env,
   R: reachable_env(E), s_table: SCRAM_table(apps, S, E, R),
   s: (S), e: (R`D)) : bool =
      (s = s_table`primary AND s_table`start_env(e)) OR
      EXISTS (t: (s_table`txns)) : t`target = s AND t`trigger = e

END SCRAM
```

6. Monitor

```
% defines application executions.
monitor: THEORY
BEGIN

IMPORTING reconf_spec
```

6.1. System State

```
% Reconfiguration status of an application
reconf_state: TYPE =
    {normal, interrupted, halting, exec_halting, prepping, training}

% Overall system state
sys_state : TYPE =
[# % reconfiguration specification for the system
    sp: reconf_spec,
    % system persistent storage
    st: data_state,
    % reconfiguration status for all applications
    reconf_st: [(sp`apps) -> reconf_state],
    % application configurations
    app_svclvls: valid_app_svcs(sp`apps),
    % last configuration each application was in
    app_last_svcs: valid_app_svcs(sp`apps),
    % system configuration
    svclvl: (sp`S),
    % system configuration during last SFTA
    last_svc: (sp`S),
    % time the state is true of the system
    t: real_time
#]

% lemma to help with TCC and other proofs
nonempty_apps: LEMMA
    FORALL (st: sys_state) :
        card(st`sp`apps) - 1 >= 0
```

6.2. Application Functions

```
% allows the system to be interrupted during execution.
success_exec?(st: sys_state) : {r: reconf_state | r = normal OR r = interrupted}
success_halt?(st: sys_state) : {r: reconf_state | r = halting OR r = interrupted}
success_eh?(st: sys_state) : {r: reconf_state | r = exec_halting OR
    r = interrupted}
success_prep?(st: sys_state) : {r: reconf_state | r = prepping OR r = interrupted}
success_train?(st: sys_state) : {r: reconf_state | r = training OR r = interrupted}

% check each function to see whether the current configuration is a safe one
% (i.e., no interrupt signals allowed)
execute(st: sys_state, app: (st`sp`apps)) : sys_state =
```

```
    LET new_st : data_state =
       app`execute(st`sp`SCRAM_info`configs(st`svclvl)(app))`f(st`st) IN
    IF (st`sp`SCRAM_info`safe(st`svclvl)) THEN st WITH [`st := new_st]
    ELSIF st`reconf_st(app) = training THEN
       st WITH [`st := new_st, `reconf_st(app) := success_train?(st)]
    ELSE
       st WITH [`st := new_st, `reconf_st(app) := success_exec?(st)]
    ENDIF

halt(st: sys_state, app: (st`sp`apps)) : sys_state =
    LET new_st : data_state =
       app`halt(st`sp`SCRAM_info`configs(st`svclvl)(app))`f(st`st) IN
    IF (st`sp`SCRAM_info`safe(st`svclvl)) THEN st WITH [`st := new_st]
    ELSE
       st WITH [`st := new_st, `reconf_st(app) := success_halt?(st)]
    ENDIF

exec_halt(st: sys_state, app: (st`sp`apps)) : sys_state =
    LET new_st : data_state =
       app`exec_halt(st`sp`SCRAM_info`configs(st`svclvl)(app))`f(st`st) IN
    IF (st`sp`SCRAM_info`safe(st`svclvl)) THEN st WITH [`st := new_st]
    ELSE st WITH [`st := new_st, `reconf_st(app) := success_eh?(st)]
    ENDIF

prep(st: sys_state, app: (st`sp`apps)) : sys_state =
    LET new_st : data_state =
       app`prep(st`sp`SCRAM_info`configs(st`svclvl)(app))`f(st`st) IN
    IF (st`sp`SCRAM_info`safe(st`svclvl)) THEN st WITH [`st := new_st]
    ELSE st WITH [`st := new_st, `reconf_st(app) := success_prep?(st)]
    ENDIF

% application monitoring layer function
monitor(st: sys_state, app: (st`sp`apps)) : sys_state =
    CASES st`reconf_st(app) OF
       normal: execute(st, app),
       interrupted: st,
       halting: halt(st, app),
       exec_halting: exec_halt(st, app),
       prepping: prep(st, app),
       training: execute(st, app)
    ENDCASES

% lemmas to assist TCC proofs
monitor_apps_constant : LEMMA
    FORALL (st: sys_state, app: (st`sp`apps)) :
       monitor(st, app)`sp`apps = st`sp`apps

monitor_svcs_constant : LEMMA
    FORALL (st: sys_state, app: (st`sp`apps)) :
       monitor(st, app)`sp`S = st`sp`S
```

6.3. System Functions

```
% added to discharge recursive_monitor's TCC that is too strong and thus unprovable
% this could be shown of recursive_monitor if the function were not universally
% quantified (so monitor_apps_constant cannot be used in the proof)
```

```
rm_apps_const: AXIOM
  FORALL (st: sys_state, apps: finseq[(st`sp`apps)], n: below[apps`length],
          v: [{z: [st: sys_state, apps: finseq[(st`sp`apps)], below[apps`length]]
                  | z`3 < n} -> sys_state]):
    NOT (n = 0) IMPLIES
     v(st, apps, n - 1)`sp`apps = st`sp`apps

% sequence of application executions that satisfy dependency requirements
recursive_monitor(st: sys_state, apps: finseq[(st`sp`apps)],
   n: below[apps`length]) :
RECURSIVE sys_state =
        IF (n = 0) THEN monitor(st, apps(0))
        ELSE monitor(recursive_monitor(st, apps, n-1), apps(n))
        ENDIF
MEASURE n

END monitor
```

7. Trace

```
% defines sequencing functions over system executions.
trace: THEORY
BEGIN

IMPORTING reconf_spec
```

7.1. System Functions

```
% lemmas used in TCC proofs
recursive_monitor_apps: LEMMA
   FORALL (st: sys_state, n: nat) :
      n >= card(st`sp`apps) OR
      recursive_monitor(st, st`sp`app_seq, n)`sp`apps = st`sp`apps

recursive_monitor_svclvl: LEMMA
   FORALL (st: sys_state, n: nat) :
      n >= card(st`sp`apps) OR
      recursive_monitor(st, st`sp`app_seq, n)`sp`S = st`sp`S

% function to define subsequent reconfiguration status of an application
% called when active configuration will remain the same
next_config(st: sys_state) : sys_state =
   st WITH [`reconf_st :=
      % if there's a signal, set everything to halt;
      % take mismatched stages as a signal
      IF (EXISTS (app: (st`sp`apps)) : st`reconf_st(app) = interrupted OR
          (EXISTS (app1, app2: (st`sp`apps)) :
             st`reconf_st(app1) /= normal AND
             st`reconf_st(app1) /= normal AND
             st`reconf_st(app1) /= st`reconf_st(app1))) THEN
          LAMBDA (app: (st`sp`apps)) :
             IF st`reconf_st(app) /= normal THEN halting
             ELSE exec_halting
             ENDIF
      ELSE
          % leave as is if halting since system_monitor will take care of it
          IF (EXISTS (app: (st`sp`apps)) : st`reconf_st(app) = halting) THEN
             LAMBDA (app: (st`sp`apps)) : st`reconf_st(app)
          % change from prepping to training
          ELSIF (EXISTS (app: (st`sp`apps)) : st`reconf_st(app) = prepping) THEN
             LAMBDA (app: (st`sp`apps)) :
                IF (st`reconf_st(app) = prepping) THEN training
                ELSE normal
                ENDIF
          % change from training to normal, or everything is normal
          ELSE LAMBDA (app: (st`sp`apps)) : normal
          ENDIF
      ENDIF]

% coordination of application execution and reconfiguration status
system_monitor(st: sys_state, e: env(st`sp`E)) : sys_state =
   IF ((EXISTS (app: (st`sp`apps)) : st`reconf_st(app) = halting) AND
```

```
       (NOT EXISTS (app: (st`sp`apps)) : st`reconf_st(app) = interrupted))
       THEN LET (next_svc: (st`sp`S)) = st`sp`choose(st`svclvl, e) IN
          recursive_monitor(st WITH [
              `reconf_st :=
                 (LAMBDA (app: (st`sp`apps)) :
                    IF st`sp`SCRAM_info`configs(next_svc)(app) /=
                       st`sp`SCRAM_info`configs(st`svclvl)(app)
                    THEN prepping
                    ELSE normal ENDIF),
              `app_svclvls :=
                 (LAMBDA (app: (st`sp`apps)) :
                    st`sp`SCRAM_info`configs(next_svc)(app)),
              `app_last_svcs :=
                 (LAMBDA (app: (st`sp`apps)) : st`app_svclvls(app)),
              `last_svc := st`svclvl,
              `svclvl := next_svc], st`sp`app_seq, card(st`sp`apps)-1)
   ELSE recursive_monitor(next_config(st), st`sp`app_seq, card(st`sp`apps)-1)
   ENDIF
```

7.2. State Trace

```
% possible environmental sequences
valid_env_trace(E: valid_env, R: reachable_env(E)) : TYPE =
{e: [real_time -> env(E)] |
   FORALL (t1, t2: real_time) : t1 + 1 = t2 AND e(t1) /= e(t2) =>
      R`txns((# source := e(t1), target := e(t2) #))}

% A valid sequence of system states is one where:
% (1) the beginning state is a non-reconfiguration state;
% (2) the system always eventually reaches a non-reconfiguration state;
% (3) any state is equal to the function application of system_monitor
%     to the previous state; and
% (4) the system state is synchronized with the environment.
sys_trace : TYPE =
[# sp: reconf_spec,
   env: valid_env_trace(sp`E, sp`R),
   tr: {c: [cycle -> {s: sys_state | s`sp = sp}] |
          (FORALL (c1, c2 : cycle) :
             c1 + 1 = c2 => c(c2) =
                system_monitor(c(c1), env(c1 * cycle_time))) AND
          (FORALL (app: (sp`apps)) : c(0)`reconf_st(app) = normal) AND
          inv(sp, c(0)`svclvl, c(0)`st) AND
          (FORALL (cyc1: cycle) : EXISTS (cyc2: cycle) : cyc2 > cyc1 AND
                FORALL (app: (sp`apps)) : c(cyc2)`reconf_st(app) = normal)}
#]

% encoding of the assumption that initialization takes only one execution cycle
train_time: AXIOM
   FORALL (s: sys_trace, c: cycle) :
      FORALL (app: (s`sp`apps)) :
          (s`tr(c)`reconf_st(app) = training =>
             pre(app`modules,
                app`svcmap(s`sp`SCRAM_info`configs(s`tr(c)`svclvl)(app)),
                s`tr(c)`st))

END trace
```

8. Lemmas

```
lemmas: THEORY
BEGIN

IMPORTING trace

reconfiguration: TYPE = [# start_c: cycle, end_c: cycle #]

% a reconfiguration is either bounded on one side by an interrupt
% and on the other by "normal" or bounded on both sides by an interrupt

reconfig_start?(s: sys_trace, c: cycle) : bool =
   EXISTS (app: (s`sp`apps)) : s`tr(c)`reconf_st(app) = interrupted

reconfig_end?(s: sys_trace, c: cycle) : bool =
   (FORALL (app: (s`sp`apps)) :
      (s`tr(c)`reconf_st(app) = normal OR s`tr(c)`reconf_st(app) = training)) OR
   (EXISTS (app: (s`sp`apps)) : s`tr(c)`reconf_st(app) = interrupted)

% whether the time is inside some reconfiguration round
reconfiguring?(s: sys_trace, c: cycle) : bool =
   EXISTS (app: (s`sp`apps)) : s`tr(c)`reconf_st(app) /= normal

get_reconfigs(s: sys_trace) : set[reconfiguration] =
{r: reconfiguration |
   r`start_c < r`end_c AND
   reconfig_start?(s, r`start_c) AND
   reconfig_end?(s, r`end_c) AND
   FORALL (c: cycle) : (r`start_c < c AND c < r`end_c => NOT reconfig_end?(s, c))}

change_to_interrupt: LEMMA
   FORALL (st: sys_state, app: (st`sp`apps)) :
      FORALL (app2: (st`sp`apps)) :
         monitor(st, app)`reconf_st(app2) = st`reconf_st(app2) OR
         (app2 = app AND monitor(st, app)`reconf_st(app2) = interrupted)

change_to_interrupt_rec: LEMMA
   FORALL (st: sys_state, app: (st`sp`apps), n: nat) :
      (n >= card(st`sp`apps) OR
      recursive_monitor(st, st`sp`app_seq, n)`reconf_st(app) =
         st`reconf_st(app) OR
      recursive_monitor(st, st`sp`app_seq, n)`reconf_st(app) = interrupted)

m_speclvl_const : LEMMA
   FORALL (st: sys_state, app: (st`sp`apps)) : monitor(st, app)`svclvl = st`svclvl

rm_speclvl_const: LEMMA
   FORALL (st: sys_state, n: nat) :
      n >= card(st`sp`apps) OR
      recursive_monitor(st, st`sp`app_seq, n)`svclvl = st`svclvl

interrupt_st: LEMMA
   FORALL (s: sys_trace, c: cycle) :
      (EXISTS (app: (s`sp`apps)) : s`tr(c)`reconf_st(app) = interrupted) =>
         (FORALL (app: (s`sp`apps)) :
            s`tr(c+1)`reconf_st(app) = halting OR
```

```
            s`tr(c+1)`reconf_st(app) = exec_halting OR
            s`tr(c+1)`reconf_st(app) = interrupted)

reconf_halt: LEMMA
   FORALL (s: sys_trace, r: (get_reconfigs(s))) :
      (FORALL (app: (s`sp`apps)) :
         (s`tr(r`start_c+1)`reconf_st(app) = halting OR
          s`tr(r`start_c+1)`reconf_st(app) = exec_halting OR
          s`tr(r`start_c+1)`reconf_st(app) = interrupted))

int_halt_st: LEMMA
   FORALL (s: sys_trace, r: (get_reconfigs(s))) :
          FORALL (app: (s`sp`apps)) :
             s`sp`SCRAM_info`configs(s`tr(r`start_c)`svclvl)(app) =
                s`sp`SCRAM_info`configs(s`tr(r`start_c+1)`svclvl)(app)

int_halt_len: LEMMA
   FORALL (s: sys_trace, r: (get_reconfigs(s))):
      (EXISTS (app: (s`sp`apps)) :
            s`tr(r`start_c+1)`reconf_st(app) = interrupted) =>
         r`end_c - r`start_c = 1

halt_st: LEMMA
   FORALL (s: sys_trace, c: cycle) :
      (EXISTS (app: (s`sp`apps)) : s`tr(c)`reconf_st(app) = halting) =>
         ((FORALL (app: (s`sp`apps)) :
         s`tr(c+1)`reconf_st(app) = prepping OR
         s`tr(c+1)`reconf_st(app) = normal OR
         s`tr(c+1)`reconf_st(app) = interrupted) OR
      EXISTS (app: (s`sp`apps)) : s`tr(c)`reconf_st(app) = interrupted)

reconf_prep: LEMMA
   FORALL (s: sys_trace, r: (get_reconfigs(s))) :
      r`end_c - r`start_c > 1 =>
      (FORALL (app: (s`sp`apps)) :
         ((s`sp`SCRAM_info`configs(s`tr(r`start_c)`svclvl)(app) /=
               s`sp`SCRAM_info`configs(s`tr(r`start_c+2)`svclvl)(app) AND
            s`tr(r`start_c+2)`reconf_st(app) = prepping) OR
          (s`sp`SCRAM_info`configs(s`tr(r`start_c)`svclvl)(app) =
               s`sp`SCRAM_info`configs(s`tr(r`start_c+2)`svclvl)(app) AND
            s`tr(r`start_c+2)`reconf_st(app) = normal) OR
           s`tr(r`start_c+2)`reconf_st(app) = interrupted))

prep_st: LEMMA
   FORALL (s: sys_trace, r: (get_reconfigs(s))) :
      r`end_c - r`start_c > 2 =>
      (FORALL (app: (s`sp`apps)) :
         (s`tr(r`start_c+2)`reconf_st(app) = prepping =>
            (s`tr(r`start_c+3)`reconf_st(app) = training OR
            s`tr(r`start_c+3)`reconf_st(app) = interrupted)))

int_prep_len: LEMMA
   FORALL (s: sys_trace, r: (get_reconfigs(s))):
      r`end_c - r`start_c > 1 =>
         ((EXISTS (app: (s`sp`apps)) :
            s`tr(r`start_c+2)`reconf_st(app) = interrupted) =>
               r`end_c - r`start_c = 2)

reconf_train: LEMMA
```

```
   FORALL (s: sys_trace, r: (get_reconfigs(s))) :
      r`end_c - r`start_c > 2 =>
      (FORALL (app: (s`sp`apps)) :
         s`sp`SCRAM_info`configs(s`tr(r`start_c+3)`svclvl)(app) =
                s`sp`SCRAM_info`configs(s`tr(r`start_c+2)`svclvl)(app))

train_st: LEMMA
   FORALL (s: sys_trace, r: (get_reconfigs(s))) :
      r`end_c - r`start_c > 2 =>
      (FORALL (app: (s`sp`apps)) :
         (s`tr(r`start_c+3)`reconf_st(app) = training OR
         s`tr(r`start_c+3)`reconf_st(app) = normal OR
         s`tr(r`start_c+3)`reconf_st(app) = interrupted))

reconf_length: LEMMA
   FORALL (s: sys_trace, r: (get_reconfigs(s))) : r`end_c - r`start_c <= 3

invariant_monitor: LEMMA
   FORALL (st: sys_state, app: (st`sp`apps)) :
      (inv(st`sp, st`svclvl, st`st) =>
         inv(st`sp, st`svclvl, monitor(st, app)`st)) AND
      (NOT st`reconf_st(app) = interrupted =>
         inv(app`modules,
            app`svcmap(st`sp`SCRAM_info`configs(st`svclvl)(app)),
            monitor(st, app)`st))

invariant_monitor_rec: LEMMA
   FORALL (st: sys_state, n: nat) :
      n >= card(st`sp`apps) OR
      (inv(st`sp, st`svclvl, st`st) =>
         inv(st`sp, st`svclvl,
            recursive_monitor(st, st`sp`app_seq, n)`st))

cycle_time: LEMMA
   FORALL (c: cycle) : c > 0 => c*cycle_time - cycle_time >= 0

same_conf_or_pre: LEMMA
   FORALL (s: sys_trace, r: (get_reconfigs(s)), app: (s`sp`apps)) :
      s`sp`SCRAM_info`configs(s`tr(r`start_c)`svclvl)(app) =
         s`sp`SCRAM_info`configs(s`tr(r`end_c)`svclvl)(app) OR
      pre(app`modules, app`svcmap(s`sp`SCRAM_info`configs
         (s`tr(r`end_c)`svclvl)(app)), s`tr(r`end_c)`st) OR
      (r`end_c - r`start_c < 3 AND
         EXISTS (app: (s`sp`apps)) :
            s`tr(r`end_c)`reconf_st(app) = interrupted) OR
       s`tr(r`end_c)`reconf_st(app) = interrupted
END lemmas
```

9. Invariant Lemmas

```
lemmas_inv: THEORY
BEGIN

IMPORTING lemmas

rm_spec_constant: LEMMA
   FORALL (st: sys_state, n: nat) :
      n >= card(st`sp`apps) OR
      recursive_monitor(st, st`sp`app_seq, n)`sp = st`sp

change_to_interrupt_rec_app: LEMMA
   FORALL (st: sys_state, app: (st`sp`apps), n: nat) :
      n >= card(st`sp`apps) OR
      ((NOT EXISTS (m: nat) : m <= n AND st`sp`app_seq(m) = app)  =>
         recursive_monitor(st, st`sp`app_seq, n)`reconf_st(app) =
            st`reconf_st(app))

monitor_not_equal_nc: LEMMA
       FORALL (st: sys_state, app: (st`sp`apps), id: data_id) :
              (NOT app_scope(app`modules)(id) =>
                  monitor(st, app)`st(id) = st`st(id))

monitor_inv_equal: LEMMA
       FORALL (st: sys_state, app: (st`sp`apps)) :
              st`reconf_st(app) /= interrupted =>
                     (inv(app`modules,
                        app`svcmap(st`sp`SCRAM_info`configs(st`svclvl)(app)),
                             monitor(st, app)`st))

inv_not_equal: LEMMA
   FORALL (st: sys_state, app: (st`sp`apps), n: posnat) :
      n >= card(st`sp`apps) OR
      (st`sp`app_seq(n) /= app =>
         (inv(app`modules,
            app`svcmap(st`sp`SCRAM_info`configs(st`svclvl)(app)),
            recursive_monitor(st, st`sp`app_seq, n-1)`st) =>
         inv(app`modules,
            app`svcmap(st`sp`SCRAM_info`configs(st`svclvl)(app)),
            recursive_monitor(st, st`sp`app_seq, n)`st)))

invariant_monitor_middle_app: LEMMA
   FORALL (st: sys_state, app: (st`sp`apps), n: nat) :
      n >= card(st`sp`apps) OR
      (((EXISTS (m: nat) : m <= n AND st`sp`app_seq(m) = app) AND
        st`reconf_st(app) /= interrupted) =>
         inv(app`modules,
            app`svcmap(st`sp`SCRAM_info`configs(st`svclvl)(app)),
            recursive_monitor(st, st`sp`app_seq, n)`st))


invariant_monitor_rec_app: LEMMA
   FORALL (st: sys_state, app: (st`sp`apps)) :
      (NOT st`reconf_st(app) = interrupted) =>
         inv(app`modules,
            app`svcmap(st`sp`SCRAM_info`configs(st`svclvl)(app)),
```

```
                recursive_monitor(st, st`sp`app_seq, card(st`sp`apps)-1)`st)

END lemmas_inv
```

10. Reconfiguration Properties

```
assured_reconfig: THEORY
BEGIN

IMPORTING lemmas_inv

% defines what makes up a reconfiguration. Essentially a repetition of the
% get_reconfigs function, repeated for clarity.
CP1: THEOREM
   FORALL (s: sys_trace, r: (get_reconfigs(s))) :
      r`start_c < r`end_c AND
      reconfig_start?(s, r`start_c) AND
      reconfig_end?(s, r`end_c) AND
      FORALL (c: cycle) : (r`start_c < c AND c < r`end_c =>
         NOT reconfig_end?(s, c))

% either: (1) a signal was generated before applications were notified of the new
% configuration and the reconfiguration ends with the system in the same
% configuration it was in when the reconfiguration began; or
% (2) the reconfiguration reached the notification stage; at the end of the
% reconfiguration, the system will be in an appropriate new configuration
CP2: THEOREM
   FORALL (s: sys_trace, r: (get_reconfigs(s))) :
      (r`end_c - r`start_c = 1 AND s`tr(r`end_c)`svclvl =
         s`tr(r`start_c)`svclvl) OR
      EXISTS (c: cycle) :
         r`start_c <= c AND c <= r`end_c AND
         s`tr(r`end_c)`svclvl =
            s`sp`choose(s`tr(c)`svclvl, s`env(c*cycle_time))

% The reconfiguration takes less than or equal to its allotted time
CP3: THEOREM
   FORALL (s: sys_trace, r: (get_reconfigs(s))) :
      (r`end_c - r`start_c + 1)*cycle_time <=
         s`sp`T(s`tr(r`start_c)`svclvl, s`tr(r`end_c)`svclvl)

% The function invariant holds
CP4: THEOREM
   FORALL (s: sys_trace, c: cycle) :
      % The function invariant holds
      inv(s`sp, s`tr(c)`svclvl, s`tr(c)`st) OR

      % A signal was generated in a function that was reconfiguring before
      % it met the transition condition for the new specification,
      % but while all applications were preparing to transition, so that
      % any application that generated a signal during that cycle will maintain
      % the invariant for the previous configuration, while any application that
      % did not generate a signal will maintain the invariant for the
      % new specification. Note that this case is always false if all applications
      % delay processing of signals during the cycle the transition condition
      % is being met.
      (c > 0 AND
       FORALL (app: (s`sp`apps)):

         % the application received a signal during the preparation stage and
         % still meets the last configuration's invariant
```

```
            (s`tr(c)`reconf_st(app) = interrupted AND
               (s`tr(c-1)`reconf_st(app) = halting OR
                  s`tr(c-1)`reconf_st(app) = exec_halting) AND
               (s`sp`SCRAM_info`configs(s`tr(c-1)`svclvl)(app) /=
                  s`sp`SCRAM_info`configs(s`tr(c)`svclvl)(app)) AND
               inv(app`modules,
                  app`svcmap(s`sp`SCRAM_info`configs(s`tr(c-1)`svclvl)(app)),
                  s`tr(c)`st)) OR

            % The application did not receive a signal during the preparation stage
            % and meets the new invariant
            inv(app`modules,
               app`svcmap(s`sp`SCRAM_info`configs(s`tr(c)`svclvl)(app)),
                     s`tr(c)`st))


% set of predicates, one of which must be true at the end of a reconfiguration
CP5: THEOREM
   FORALL (s: sys_trace, r: (get_reconfigs(s))) :
      % the reconfiguration was not interrupted and some application reconfigured
      (r`end_c - r`start_c = 3 AND
         FORALL (app: (s`sp`apps)) :
            (s`sp`SCRAM_info`configs(s`tr(r`start_c)`svclvl)(app) /=
                  s`sp`SCRAM_info`configs(s`tr(r`end_c)`svclvl)(app) AND
               pre(app`modules,
                  app`svcmap(s`sp`SCRAM_info`configs
                  (s`tr(r`end_c)`svclvl)(app)), s`tr(r`end_c)`st) OR
            (s`sp`SCRAM_info`configs(s`tr(r`start_c)`svclvl)(app) =
                  s`sp`SCRAM_info`configs(s`tr(r`end_c)`svclvl)(app) AND
               inv(app`modules,
                  app`svcmap(s`sp`SCRAM_info`configs
                  (s`tr(r`end_c)`svclvl)(app)), s`tr(r`end_c)`st))))
      OR
      % the reconfiguration was not interrupted but no application reconfigured
      (r`end_c - r`start_c = 2 AND
         FORALL (app: (s`sp`apps)) :
            (s`sp`SCRAM_info`configs(s`tr(r`start_c)`svclvl)(app) =
                  s`sp`SCRAM_info`configs(s`tr(r`end_c)`svclvl)(app) AND
               inv(app`modules,
                  app`svcmap(s`sp`SCRAM_info`configs
                  (s`tr(r`end_c)`svclvl)(app)), s`tr(r`end_c)`st)))
      OR
      % the reconfiguration was interrupted
      EXISTS (app: (s`sp`apps)) : s`tr(r`end_c)`reconf_st(app) = interrupted

END assured_reconfig
```

Part II

Architecture TCCs

1. State

The state theory has no TCCs.

2. Module

The module theory has no TCCs.

3. Application

```
% Subtype TCC generated (at line 19, column 32) for  map(m)
    % expected type  (m`sv)
  % proved - complete
post_TCC1: OBLIGATION
  FORALL (modules: set[module_spec], m: (modules), map: service_map(modules)):
    m`sv(map(m));

% The subtype TCC (at line 24, column 33) in decl trans for  map(m)
    %expected type  (m`sv)
  % is subsumed by post_TCC1

% The subtype TCC (at line 29, column 31) in decl pre for  map(m)
    %expected type  (m`sv)
  % is subsumed by post_TCC1

% The subtype TCC (at line 34, column 31) in decl inv for  map(m)
    %expected type  (m`sv)
  % is subsumed by post_TCC1
```

4. SCRAM

The SCRAM theory has no TCCs.

5. Reconfiguration Specification

```
% Subtype TCC generated (at line 24, column 57) for  s
    % expected type  (extend[speclvl,
                     %         {sp: speclvl | NOT sp = indeterminate}, bool,
                     %         FALSE]
                     %      (S))
  % proved - complete
reconf_spec_TCC1: OBLIGATION
  FORALL (E: valid_env, R: reachable_env(E),
          S: set[{sp: speclvl | NOT sp = indeterminate}],
          c: [(S), env(E) -> (S)], e: env(E), s: (S)):
    R`D(e) IMPLIES
```

```
     extend[speclvl, {sp: speclvl | NOT sp = indeterminate}, bool, FALSE](S)(s);

% The subtype TCC (at line 24, column 60) in decl reconf_spec for  e
    %expected type  (R`D)
  % was not generated because it simplifies to TRUE.

% The subtype TCC (at line 24, column 57) in decl reconf_spec for  s
    %expected type  (extend[speclvl,
                    %         {sp: speclvl | NOT sp = indeterminate}, bool,
                    %         FALSE]
                    %      (S))
  % is subsumed by reconf_spec_TCC1

% The subtype TCC (at line 24, column 60) in decl reconf_spec for  e
    %expected type  (R`D)
  % was not generated because it simplifies to TRUE.

% Subtype TCC generated (at line 33, column 52) for  svc
    % expected type  (extend[speclvl,
                    %         {sp: speclvl | NOT sp = indeterminate}, bool,
                    %         FALSE]
                    %      (r`S))
  % proved - complete
post_TCC1: OBLIGATION
  FORALL (r: reconf_spec, app: (r`apps), svc: (r`S)):
    extend[speclvl, {sp: speclvl | NOT sp = indeterminate}, bool, FALSE]
        (r`S)(svc);

% Subtype TCC generated (at line 33, column 31) for
    % r`SCRAM_info`configs(svc)(app)
    % expected type  (app`svcs)
  % proved - complete
post_TCC2: OBLIGATION
  FORALL (r: reconf_spec, app: (r`apps), svc: (r`S)):
    app`svcs(r`SCRAM_info`configs(svc)(app));

% The subtype TCC (at line 37, column 53) in decl trans for  svc
    %expected type  (extend[speclvl,
                    %         {sp: speclvl | NOT sp = indeterminate}, bool,
                    %         FALSE]
                    %      (r`S))
  % is subsumed by post_TCC1

% The subtype TCC (at line 37, column 32) in decl trans for
    %  r`SCRAM_info`configs(svc)(app)
    %expected type  (app`svcs)
  % is subsumed by post_TCC2

% The subtype TCC (at line 41, column 51) in decl pre for  svc
    %expected type  (extend[speclvl,
                    %         {sp: speclvl | NOT sp = indeterminate}, bool,
                    %         FALSE]
                    %      (r`S))
  % is subsumed by post_TCC1

% The subtype TCC (at line 41, column 30) in decl pre for
    %  r`SCRAM_info`configs(svc)(app)
    %expected type  (app`svcs)
  % is subsumed by post_TCC2
```

```
% The subtype TCC (at line 45, column 51) in decl inv for  svc
    %expected type  (extend[speclvl,
                    %         {sp: speclvl | NOT sp = indeterminate}, bool,
                    %         FALSE]
                    %      (r`S))
  % is subsumed by post_TCC1

% The subtype TCC (at line 45, column 30) in decl inv for
    %  r`SCRAM_info`configs(svc)(app)
    %expected type  (app`svcs)
  % is subsumed by post_TCC2
```

6. Monitor

```
% Existence TCC generated (at line 24, column 0) for
    % success_exec?(st: sys_state):
    % {r: reconf_state | r = normal OR r = interrupted}
  % proved - complete
success_exec?_TCC1: OBLIGATION
  EXISTS (x: [sys_state -> {r: reconf_state | r = normal OR r = interrupted}]):
    TRUE;

% Existence TCC generated (at line 25, column 0) for
    % success_halt?(st: sys_state):
    % {r: reconf_state | r = halting OR r = interrupted}
  % proved - complete
success_halt?_TCC1: OBLIGATION
  EXISTS (x:
            [sys_state -> {r: reconf_state | r = halting OR r = interrupted}]):
    TRUE;

% Existence TCC generated (at line 26, column 0) for
    % success_eh?(st: sys_state):
    % {r: reconf_state | r = exec_halting OR r = interrupted}
  % proved - complete
success_eh?_TCC1: OBLIGATION
  EXISTS (x:
            [sys_state ->
               {r: reconf_state | r = exec_halting OR r = interrupted}]):
    TRUE;

% Existence TCC generated (at line 27, column 0) for
    % success_prep?(st: sys_state):
    % {r: reconf_state | r = prepping OR r = interrupted}
  % proved - complete
success_prep?_TCC1: OBLIGATION
  EXISTS (x:
            [sys_state ->
               {r: reconf_state | r = prepping OR r = interrupted}]):
    TRUE;

% Existence TCC generated (at line 28, column 0) for
    % success_train?(st: sys_state):
    % {r: reconf_state | r = training OR r = interrupted}
  % proved - complete
```

```
success_train?_TCC1: OBLIGATION
  EXISTS (x:
            [sys_state ->
               {r: reconf_state | r = training OR r = interrupted}]):
    TRUE;

% Subtype TCC generated (at line 34, column 39) for  st`svclvl
    % expected type  (extend[speclvl,
                    %         {sp: speclvl | NOT sp = indeterminate}, bool,
                    %         FALSE]
                    %      (sp(st)`S))
  % proved - complete
execute_TCC1: OBLIGATION
  FORALL (st: sys_state, app: (st`sp`apps)):
    extend[speclvl, {sp: speclvl | NOT sp = indeterminate}, bool, FALSE]
        (st`sp`S)(st`svclvl);

% Subtype TCC generated (at line 34, column 14) for
    % st`sp`SCRAM_info`configs(st`svclvl)(app)
    % expected type  (app`svcs)
  % proved - complete
execute_TCC2: OBLIGATION
  FORALL (st: sys_state, app: (st`sp`apps)):
    app`svcs(st`sp`SCRAM_info`configs(st`svclvl)(app));

% The subtype TCC (at line 35, column 27) in decl execute for  st`svclvl
    %expected type  (extend[speclvl,
                    %         {sp: speclvl | NOT sp = indeterminate}, bool,
                    %         FALSE]
                    %      (sp(st)`S))
  % is subsumed by execute_TCC1

% The subtype TCC (at line 44, column 36) in decl halt for  st`svclvl
    %expected type  (extend[speclvl,
                    %         {sp: speclvl | NOT sp = indeterminate}, bool,
                    %         FALSE]
                    %      (sp(st)`S))
  % is subsumed by execute_TCC1

% The subtype TCC (at line 44, column 11) in decl halt for
    %  st`sp`SCRAM_info`configs(st`svclvl)(app)
    %expected type  (app`svcs)
  % is subsumed by execute_TCC2

% The subtype TCC (at line 45, column 27) in decl halt for  st`svclvl
    %expected type  (extend[speclvl,
                    %         {sp: speclvl | NOT sp = indeterminate}, bool,
                    %         FALSE]
                    %      (sp(st)`S))
  % is subsumed by execute_TCC1

% The subtype TCC (at line 52, column 41) in decl exec_halt for  st`svclvl
    %expected type  (extend[speclvl,
                    %         {sp: speclvl | NOT sp = indeterminate}, bool,
                    %         FALSE]
                    %      (sp(st)`S))
  % is subsumed by execute_TCC1

% The subtype TCC (at line 52, column 16) in decl exec_halt for
```

```
    %  st`sp`SCRAM_info`configs(st`svclvl)(app)
    %expected type  (app`svcs)
  % is subsumed by execute_TCC2

% The subtype TCC (at line 53, column 27) in decl exec_halt for  st`svclvl
    %expected type  (extend[speclvl,
                    %         {sp: speclvl | NOT sp = indeterminate}, bool,
                    %         FALSE]
                    %      (sp(st)`S))
  % is subsumed by execute_TCC1

% The subtype TCC (at line 59, column 36) in decl prep for  st`svclvl
    %expected type  (extend[speclvl,
                    %         {sp: speclvl | NOT sp = indeterminate}, bool,
                    %         FALSE]
                    %      (sp(st)`S))
  % is subsumed by execute_TCC1

% The subtype TCC (at line 59, column 11) in decl prep for
    %  st`sp`SCRAM_info`configs(st`svclvl)(app)
    %expected type  (app`svcs)
  % is subsumed by execute_TCC2

% The subtype TCC (at line 60, column 27) in decl prep for  st`svclvl
    %expected type  (extend[speclvl,
                    %         {sp: speclvl | NOT sp = indeterminate}, bool,
                    %         FALSE]
                    %      (sp(st)`S))
  % is subsumed by execute_TCC1

% Subtype TCC generated (at line 88, column 17) for  n - 1
    % expected type  below[length(apps)]
  % proved - complete
rm_apps_const_TCC1: OBLIGATION
  FORALL (st: sys_state, apps: finseq[(st`sp`apps)], n: below[apps`length],
          v:
            [{z:
                [st: sys_state, apps: finseq[(st`sp`apps)],
                 below[apps`length]] |
                      z`3 < n} ->
               sys_state]):
    NOT (n = 0) IMPLIES n - 1 >= 0 AND n - 1 < apps`length;

% Subtype TCC generated (at line 88, column 7) for  (st, apps, n - 1)
    % expected type  {z:
                    %    [st: sys_state, apps: finseq[(st`sp`apps)],
                    %     below[apps`length]] |
                    %          z`3 < n}
  % proved - complete
rm_apps_const_TCC2: OBLIGATION
  FORALL (st: sys_state, apps: finseq[(st`sp`apps)], n: below[apps`length],
          v:
            [{z:
                [st: sys_state, apps: finseq[(st`sp`apps)],
                 below[apps`length]] |
                      z`3 < n} ->
               sys_state]):
    NOT (n = 0) IMPLIES n - 1 < n;
```

```
% Subtype TCC generated (at line 92, column 34) for  0
    % expected type  below[length(apps)]
  % proved - complete
recursive_monitor_TCC1: OBLIGATION
  FORALL (st: sys_state, apps: finseq[(st`sp`apps)], n: below[apps`length]):
    (n = 0) IMPLIES 0 < apps`length;

% Subtype TCC generated (at line 93, column 42) for  n - 1
    % expected type  below[length(apps)]
  % proved - complete
recursive_monitor_TCC2: OBLIGATION
  FORALL (st: sys_state, apps: finseq[(st`sp`apps)], n: below[apps`length]):
    NOT (n = 0) IMPLIES n - 1 >= 0 AND n - 1 < apps`length;

% Termination TCC generated (at line 93, column 14) for
    % recursive_monitor(st, apps, n - 1)
  % proved - complete
recursive_monitor_TCC3: OBLIGATION
  FORALL (st: sys_state, apps: finseq[(st`sp`apps)], n: below[apps`length]):
    NOT (n = 0) IMPLIES n - 1 < n;

% Subtype TCC generated (at line 93, column 48) for
    % finseq_appl[((st`sp`apps))](apps)(n)
    % expected type  (recursive_monitor(st, apps, n - 1)`sp`apps)
  % proved - complete
recursive_monitor_TCC4: OBLIGATION
  FORALL (st: sys_state, apps: finseq[(st`sp`apps)], n: below[apps`length],
          v:
            [{z:
                [st: sys_state, apps: finseq[(st`sp`apps)],
                 below[apps`length]] |
                      z`3 < n} ->
               sys_state]):
    NOT (n = 0) IMPLIES
     v(st, apps, n - 1)`sp`apps(finseq_appl[((st`sp`apps))](apps)(n));
```

7. Trace

```
% Subtype TCC generated (at line 9, column 39) for  n
    % expected type  below[length(st`sp`app_seq)]
  % proved - complete
recursive_monitor_apps_TCC1: OBLIGATION
  FORALL (n: nat, st: sys_state):
    NOT n >= card(st`sp`apps) IMPLIES n < st`sp`app_seq`length;

% The subtype TCC (at line 14, column 39) in decl recursive_monitor_svclvl for  n
    %expected type  below[length(st`sp`app_seq)]
  % is subsumed by recursive_monitor_apps_TCC1

% Subtype TCC generated (at line 55, column 60) for  next_svc
    % expected type  (extend[speclvl,
                    %         {sp: speclvl | NOT sp = indeterminate}, bool,
                    %         FALSE]
                    %      (sp(st)`S))
  % proved - complete
system_monitor_TCC1: OBLIGATION
```

```
  FORALL (st: sys_state, e: env(st`sp`E)):
    (NOT (EXISTS (app: (st`sp`apps)): st`reconf_st(app) = interrupted)) AND
     (EXISTS (app: (st`sp`apps)): st`reconf_st(app) = halting)
     IMPLIES
     (FORALL (next_svc: (st`sp`S)):
        next_svc = st`sp`choose(st`svclvl, e) IMPLIES
         (FORALL (app1: (st`sp`apps)):
            extend[speclvl, {sp: speclvl | NOT sp = indeterminate}, bool, FALSE]
                (st`sp`S)(next_svc)));

% Subtype TCC generated (at line 51, column 32) for  st`svclvl
    % expected type  (extend[speclvl,
                    %         {sp: speclvl | NOT sp = indeterminate}, bool,
                    %         FALSE]
                    %      (sp(st)`S))
  % proved - complete
system_monitor_TCC2: OBLIGATION
  FORALL (st: sys_state, e: env(st`sp`E)):
    (NOT (EXISTS (app: (st`sp`apps)): st`reconf_st(app) = interrupted)) AND
     (EXISTS (app: (st`sp`apps)): st`reconf_st(app) = halting)
     IMPLIES
     (FORALL (next_svc: (st`sp`S)):
        next_svc = st`sp`choose(st`svclvl, e) IMPLIES
         (FORALL (app1: (st`sp`apps)):
            extend[speclvl, {sp: speclvl | NOT sp = indeterminate}, bool, FALSE]
                (st`sp`S)(st`svclvl)));

% Subtype TCC generated (at line 58, column 41) for  card(st`sp`apps) - 1
    % expected type  below[length(st`sp`app_seq)]
  % proved - complete
system_monitor_TCC3: OBLIGATION
  FORALL (st: sys_state, e: env(st`sp`E)):
    (NOT (EXISTS (app: (st`sp`apps)): st`reconf_st(app) = interrupted)) AND
     (EXISTS (app: (st`sp`apps)): st`reconf_st(app) = halting)
     IMPLIES
     (FORALL (next_svc: (st`sp`S)):
        next_svc = st`sp`choose(st`svclvl, e) IMPLIES
         card[app_spec](st`sp`apps) - 1 >= 0 AND
          card[app_spec](st`sp`apps) - 1 < st`sp`app_seq`length);

% Subtype TCC generated (at line 59, column 41) for  st`sp`app_seq
    % expected type  finseq[((next_config(st)`sp`apps))]
  % proved - complete
system_monitor_TCC4: OBLIGATION
  FORALL (st: sys_state, e: env(st`sp`E)):
    NOT ((EXISTS (app: (st`sp`apps)): st`reconf_st(app) = halting) AND
          (NOT (EXISTS (app: (st`sp`apps)): st`reconf_st(app) = interrupted)))
     IMPLIES
     (FORALL (x1: below[st`sp`app_seq`length]):
        (next_config(st)`sp`apps)(st`sp`app_seq`seq(x1)));

% Subtype TCC generated (at line 59, column 56) for  card(st`sp`apps) - 1
    % expected type  below[length(st`sp`app_seq)]
  % proved - complete
system_monitor_TCC5: OBLIGATION
  FORALL (st: sys_state, e: env(st`sp`E)):
    NOT ((EXISTS (app: (st`sp`apps)): st`reconf_st(app) = halting) AND
          (NOT (EXISTS (app: (st`sp`apps)): st`reconf_st(app) = interrupted)))
     IMPLIES
```

```
     card[app_spec](st`sp`apps) - 1 >= 0 AND
      card[app_spec](st`sp`apps) - 1 < st`sp`app_seq`length;

% The subtype TCC (at line 50, column 34) in decl system_monitor for  next_svc
    %expected type  (extend[speclvl,
                    %         {sp: speclvl | NOT sp = indeterminate}, bool,
                    %         FALSE]
                    %      (sp(st)`S))
  % is subsumed by system_monitor_TCC1

% Subtype TCC generated (at line 24, column 57) for  s
    % expected type  (extend[speclvl,
                    %         {sp: speclvl | NOT sp = indeterminate}, bool,
                    %         FALSE]
                    %      (S))
  % proved - complete
sys_trace_TCC1: OBLIGATION
  FORALL (E: valid_env, R: reachable_env(E),
          S: set[{sp: speclvl | NOT sp = indeterminate}], e: env(E), s1: (S)):
    R`D(e) IMPLIES
     extend[speclvl, {sp: speclvl | NOT sp = indeterminate}, bool, FALSE]
         (S)(s1);

% Subtype TCC generated (at line 73, column 27) for  env(c1 * cycle_time)
    % expected type  env(c(c1)`sp`E)
  % proved - complete
sys_trace_TCC2: OBLIGATION
  FORALL (sp: reconf_spec, env: valid_env_trace(sp`E, sp`R),
          c: [cycle -> {s: sys_state | s`sp = sp}], c1, c2: cycle):
    c1 + 1 = c2 IMPLIES
     (FORALL (e: env_id):
        member[env_param](env(c1 * cycle_time)(e), c(c1)`sp`E(e)));

% Subtype TCC generated (at line 74, column 45) for  app
    % expected type  (c(0)`sp`apps)
  % proved - complete
sys_trace_TCC3: OBLIGATION
  FORALL (sp: reconf_spec, env: valid_env_trace(sp`E, sp`R),
          c: [cycle -> {s: sys_state | s`sp = sp}]):
    (FORALL (c1, c2: cycle):
       c1 + 1 = c2 => c(c2) = system_monitor(c(c1), env(c1 * cycle_time)))
     IMPLIES (FORALL (app: (sp`apps)): c(0)`sp`apps(app));

% Subtype TCC generated (at line 75, column 11) for  c(0)`svclvl
    % expected type  (sp`S)
  % proved - complete
sys_trace_TCC4: OBLIGATION
  FORALL (sp: reconf_spec, env: valid_env_trace(sp`E, sp`R),
          c: [cycle -> {s: sys_state | s`sp = sp}]):
    ((FORALL (c1, c2: cycle):
        c1 + 1 = c2 => c(c2) = system_monitor(c(c1), env(c1 * cycle_time)))
      AND (FORALL (app: (sp`apps)): c(0)`reconf_st(app) = normal))
     IMPLIES sp`S(c(0)`svclvl);

% Subtype TCC generated (at line 77, column 49) for  app
    % expected type  (c(cyc2)`sp`apps)
  % proved - complete
sys_trace_TCC5: OBLIGATION
  FORALL (sp: reconf_spec, env: valid_env_trace(sp`E, sp`R),
```

```
          c: [cycle -> {s: sys_state | s`sp = sp}]):
     (FORALL (c1, c2: cycle):
        c1 + 1 = c2 => c(c2) = system_monitor(c(c1), env(c1 * cycle_time)))
      AND
      (FORALL (app: (sp`apps)): c(0)`reconf_st(app) = normal) AND
       inv(sp, c(0)`svclvl, c(0)`st)
      IMPLIES
      (FORALL (cyc1: cycle, cyc2: cycle):
         cyc2 > cyc1 IMPLIES (FORALL (app1: (sp`apps)): c(cyc2)`sp`apps(app1)));

% The subtype TCC (at line 24, column 60) in decl sys_trace for  e
    %expected type  (R`D)
  % was not generated because it simplifies to TRUE.

% The subtype TCC (at line 24, column 57) in decl sys_trace for  s
    %expected type  (extend[speclvl,
                   %          {sp: speclvl | NOT sp = indeterminate}, bool,
                   %          FALSE]
                   %       (S))
  % is subsumed by sys_trace_TCC1

% The subtype TCC (at line 24, column 60) in decl sys_trace for  e
    %expected type  (R`D)
  % was not generated because it simplifies to TRUE.

% The subtype TCC (at line 24, column 57) in decl sys_trace for  s
    %expected type  (extend[speclvl,
                   %          {sp: speclvl | NOT sp = indeterminate}, bool,
                   %          FALSE]
                   %       (S))
  % is subsumed by sys_trace_TCC1

% The subtype TCC (at line 24, column 60) in decl sys_trace for  e
    %expected type  (R`D)
  % was not generated because it simplifies to TRUE.

% The subtype TCC (at line 24, column 57) in decl sys_trace for  s
    %expected type  (extend[speclvl,
                   %          {sp: speclvl | NOT sp = indeterminate}, bool,
                   %          FALSE]
                   %       (S))
  % is subsumed by sys_trace_TCC1

% The subtype TCC (at line 24, column 60) in decl sys_trace for  e
    %expected type  (R`D)
  % was not generated because it simplifies to TRUE.

% The subtype TCC (at line 73, column 27) in decl sys_trace for  env(c1 *
cycle_time)
    %expected type  env(c(c1)`sp`E)
  % is subsumed by sys_trace_TCC2

% The subtype TCC (at line 74, column 45) in decl sys_trace for  app
    %expected type  (c(0)`sp`apps)
  % is subsumed by sys_trace_TCC3

% The subtype TCC (at line 75, column 11) in decl sys_trace for  c(0)`svclvl
    %expected type  (sp`S)
  % is subsumed by sys_trace_TCC4
```

```
% The subtype TCC (at line 77, column 49) in decl sys_trace for  app
    %expected type  (c(cyc2)`sp`apps)
  % is subsumed by sys_trace_TCC5

% Subtype TCC generated (at line 83, column 22) for  app
    % expected type  (tr(s)(c)`sp`apps)
  % proved - complete
train_time_TCC1: OBLIGATION
  FORALL (c: cycle, s: sys_trace, app: (s`sp`apps)): s`tr(c)`sp`apps(app);

% Subtype TCC generated (at line 85, column 40) for  s`tr(c)`svclvl
    % expected type  (extend[speclvl,
                    %         {sp: speclvl | NOT sp = indeterminate}, bool,
                    %         FALSE]
                    %      (sp(s)`S))
  % proved - complete
train_time_TCC2: OBLIGATION
  FORALL (c: cycle, s: sys_trace, app: (s`sp`apps)):
    s`tr(c)`reconf_st(app) = training IMPLIES
     extend[speclvl, {sp: speclvl | NOT sp = indeterminate}, bool, FALSE]
         (s`sp`S)(s`tr(c)`svclvl);

% Subtype TCC generated (at line 85, column 16) for
    % s`sp`SCRAM_info`configs(s`tr(c)`svclvl)(app)
    % expected type  (app`svcs)
  % proved - complete
train_time_TCC3: OBLIGATION
  FORALL (c: cycle, s: sys_trace, app: (s`sp`apps)):
    s`tr(c)`reconf_st(app) = training IMPLIES
     app`svcs(s`sp`SCRAM_info`configs(s`tr(c)`svclvl)(app));
```

8. Lemmas

```
% Subtype TCC generated (at line 12, column 47) for  app
    % expected type  (tr(s)(c)`sp`apps)
  % proved - complete
reconfig_start?_TCC1: OBLIGATION
  FORALL (c: cycle, s: sys_trace, app: (s`sp`apps)): s`tr(c)`sp`apps(app);

% Subtype TCC generated (at line 17, column 48) for  app
    % expected type  (tr(s)(c)`sp`apps)
  % proved - complete
reconfig_end?_TCC1: OBLIGATION
  FORALL (c: cycle, s: sys_trace):
    NOT (FORALL (app: (s`sp`apps)):
           (s`tr(c)`reconf_st(app) = normal OR
             s`tr(c)`reconf_st(app) = training))
     IMPLIES (FORALL (app1: (s`sp`apps)): s`tr(c)`sp`apps(app1));

% The subtype TCC (at line 16, column 21) in decl reconfig_end? for  app
    %expected type  (tr(s)(c)`sp`apps)
  % is subsumed by reconfig_start?_TCC1

% The subtype TCC (at line 16, column 56) in decl reconfig_end? for  app
    %expected type  (tr(s)(c)`sp`apps)
```

```
  % is subsumed by reconfig_start?_TCC1

% The subtype TCC (at line 21, column 47) in decl reconfiguring? for  app
    %expected type  (tr(s)(c)`sp`apps)
  % is subsumed by reconfig_start?_TCC1

% Subtype TCC generated (at line 33, column 30) for  app2
    % expected type  (monitor(st, app)`sp`apps)
  % proved - complete
change_to_interrupt_TCC1: OBLIGATION
  FORALL (st: sys_state, app: (st`sp`apps), app2: (st`sp`apps)):
    monitor(st, app)`sp`apps(app2);

% The subtype TCC (at line 34, column 46) in decl change_to_interrupt for  app2
    %expected type  (monitor(st, app)`sp`apps)
  % is subsumed by change_to_interrupt_TCC1

% Subtype TCC generated (at line 39, column 52) for  app
    % expected type  (recursive_monitor(st, app_seq(sp(st)), n)`sp`apps)
  % proved - complete
change_to_interrupt_rec_TCC1: OBLIGATION
  FORALL (n: nat, st: sys_state, app: (st`sp`apps)):
    NOT n >= card(st`sp`apps) IMPLIES
     recursive_monitor(st, st`sp`app_seq, n)`sp`apps(app);

% Subtype TCC generated (at line 39, column 39) for  n
    % expected type  below[length(st`sp`app_seq)]
  % proved - complete
change_to_interrupt_rec_TCC2: OBLIGATION
  FORALL (n: nat, st: sys_state, app: (st`sp`apps)):
    NOT n >= card(st`sp`apps) IMPLIES n < st`sp`app_seq`length;

% The subtype TCC (at line 40, column 52) in decl change_to_interrupt_rec for  app
    %expected type  (recursive_monitor(st, app_seq(sp(st)), n)`sp`apps)
  % is subsumed by change_to_interrupt_rec_TCC1

% The subtype TCC (at line 40, column 39) in decl change_to_interrupt_rec for  n
    %expected type  below[length(st`sp`app_seq)]
  % is subsumed by change_to_interrupt_rec_TCC2

% Subtype TCC generated (at line 48, column 39) for  n
    % expected type  below[length(st`sp`app_seq)]
  % proved - complete
rm_speclvl_const_TCC1: OBLIGATION
  FORALL (n: nat, st: sys_state):
    NOT n >= card(st`sp`apps) IMPLIES n < st`sp`app_seq`length;

% Subtype TCC generated (at line 54, column 24) for  app
    % expected type  (tr(s)(c + 1)`sp`apps)
  % proved - complete
interrupt_st_TCC1: OBLIGATION
  FORALL (c: cycle, s: sys_trace):
    (EXISTS (app: (s`sp`apps)): s`tr(c)`reconf_st(app) = interrupted) IMPLIES
     (FORALL (app1: (s`sp`apps)): s`tr(1 + c)`sp`apps(app1));

% The subtype TCC (at line 52, column 49) in decl interrupt_st for  app
    %expected type  (tr(s)(c)`sp`apps)
  % is subsumed by reconfig_start?_TCC1
```

```
% The subtype TCC (at line 55, column 24) in decl interrupt_st for  app
    %expected type  (tr(s)(c + 1)`sp`apps)
  % is subsumed by interrupt_st_TCC1

% The subtype TCC (at line 56, column 24) in decl interrupt_st for  app
    %expected type  (tr(s)(c + 1)`sp`apps)
  % is subsumed by interrupt_st_TCC1

% Subtype TCC generated (at line 61, column 32) for  app
    % expected type  (tr(s)(start_c(r) + 1)`sp`apps)
  % proved - complete
reconf_halt_TCC1: OBLIGATION
  FORALL (s: sys_trace, app: (s`sp`apps), r: (get_reconfigs(s))):
    s`tr(1 + r`start_c)`sp`apps(app);

% The subtype TCC (at line 62, column 32) in decl reconf_halt for  app
    %expected type  (tr(s)(start_c(r) + 1)`sp`apps)
  % is subsumed by reconf_halt_TCC1

% The subtype TCC (at line 63, column 32) in decl reconf_halt for  app
    %expected type  (tr(s)(start_c(r) + 1)`sp`apps)
  % is subsumed by reconf_halt_TCC1

% Subtype TCC generated (at line 68, column 28) for  s`tr(r`start_c)`svclvl
    % expected type  (extend[speclvl,
                  %         {sp: speclvl | NOT sp = indeterminate}, bool,
                  %         FALSE]
                  %      (sp(s)`S))
  % proved - complete
int_halt_st_TCC1: OBLIGATION
  FORALL (s: sys_trace, app: (s`sp`apps), r: (get_reconfigs(s))):
    extend[speclvl, {sp: speclvl | NOT sp = indeterminate}, bool, FALSE]
        (s`sp`S)(s`tr(r`start_c)`svclvl);

% Subtype TCC generated (at line 69, column 29) for
    % s`tr(r`start_c + 1)`svclvl
    % expected type  (extend[speclvl,
                  %         {sp: speclvl | NOT sp = indeterminate}, bool,
                  %         FALSE]
                  %      (sp(s)`S))
  % proved - complete
int_halt_st_TCC2: OBLIGATION
  FORALL (s: sys_trace, app: (s`sp`apps), r: (get_reconfigs(s))):
    extend[speclvl, {sp: speclvl | NOT sp = indeterminate}, bool, FALSE]
        (s`sp`S)(s`tr(r`start_c + 1)`svclvl);

% The subtype TCC (at line 73, column 59) in decl int_halt_len for  app
    %expected type  (tr(s)(start_c(r) + 1)`sp`apps)
  % is subsumed by reconf_halt_TCC1

% Subtype TCC generated (at line 80, column 23) for  app
    % expected type  (tr(s)(c + 1)`sp`apps)
  % proved - complete
halt_st_TCC1: OBLIGATION
  FORALL (c: cycle, s: sys_trace):
    (EXISTS (app: (s`sp`apps)): s`tr(c)`reconf_st(app) = halting) IMPLIES
     (FORALL (app1: (s`sp`apps)): s`tr(1 + c)`sp`apps(app1));

% Subtype TCC generated (at line 83, column 48) for  app
```

```
    % expected type  (tr(s)(c)`sp`apps)
  % proved - complete
halt_st_TCC2: OBLIGATION
  FORALL (c: cycle, s: sys_trace):
    (EXISTS (app: (s`sp`apps)): s`tr(c)`reconf_st(app) = halting) AND
     NOT (FORALL (app: (s`sp`apps)):
            s`tr(c + 1)`reconf_st(app) = prepping OR
             s`tr(c + 1)`reconf_st(app) = normal OR
              s`tr(c + 1)`reconf_st(app) = interrupted)
     IMPLIES (FORALL (app1: (s`sp`apps)): s`tr(c)`sp`apps(app1));

% The subtype TCC (at line 78, column 49) in decl halt_st for  app
    %expected type  (tr(s)(c)`sp`apps)
  % is subsumed by reconfig_start?_TCC1

% The subtype TCC (at line 81, column 23) in decl halt_st for  app
    %expected type  (tr(s)(c + 1)`sp`apps)
  % is subsumed by halt_st_TCC1

% The subtype TCC (at line 82, column 23) in decl halt_st for  app
    %expected type  (tr(s)(c + 1)`sp`apps)
  % is subsumed by halt_st_TCC1

% Subtype TCC generated (at line 90, column 29) for
    % s`tr(r`start_c + 2)`svclvl
    % expected type  (extend[speclvl,
                  %         {sp: speclvl | NOT sp = indeterminate}, bool,
                  %         FALSE]
                  %      (sp(s)`S))
  % proved - complete
reconf_prep_TCC1: OBLIGATION
  FORALL (s: sys_trace, r: (get_reconfigs(s))):
    r`end_c - r`start_c > 1 IMPLIES
     (FORALL (app: (s`sp`apps)):
        extend[speclvl, {sp: speclvl | NOT sp = indeterminate}, bool, FALSE]
            (s`sp`S)(s`tr(r`start_c + 2)`svclvl));

% Subtype TCC generated (at line 91, column 32) for  app
    % expected type  (tr(s)(start_c(r) + 2)`sp`apps)
  % proved - complete
reconf_prep_TCC2: OBLIGATION
  FORALL (s: sys_trace, r: (get_reconfigs(s))):
    r`end_c - r`start_c > 1 IMPLIES
     (FORALL (app: (s`sp`apps)):
        s`sp`SCRAM_info`configs(s`tr(r`start_c)`svclvl)(app) /=
         s`sp`SCRAM_info`configs(s`tr(r`start_c + 2)`svclvl)(app)
         IMPLIES s`tr(2 + r`start_c)`sp`apps(app));

% Subtype TCC generated (at line 94, column 32) for  app
    % expected type  (tr(s)(start_c(r) + 2)`sp`apps)
  % proved - complete
reconf_prep_TCC3: OBLIGATION
  FORALL (s: sys_trace, r: (get_reconfigs(s))):
    r`end_c - r`start_c > 1 IMPLIES
     (FORALL (app: (s`sp`apps)):
        NOT (s`sp`SCRAM_info`configs(s`tr(r`start_c)`svclvl)(app) /=
              s`sp`SCRAM_info`configs(s`tr(r`start_c + 2)`svclvl)(app)
              AND s`tr(r`start_c + 2)`reconf_st(app) = prepping)
         AND
```

```
          s`sp`SCRAM_info`configs(s`tr(r`start_c)`svclvl)(app) =
           s`sp`SCRAM_info`configs(s`tr(r`start_c + 2)`svclvl)(app)
          IMPLIES s`tr(2 + r`start_c)`sp`apps(app));

% Subtype TCC generated (at line 95, column 32) for  app
    % expected type  (tr(s)(start_c(r) + 2)`sp`apps)
  % proved - complete
reconf_prep_TCC4: OBLIGATION
  FORALL (s: sys_trace, r: (get_reconfigs(s))):
    r`end_c - r`start_c > 1 IMPLIES
     (FORALL (app: (s`sp`apps)):
        NOT (s`sp`SCRAM_info`configs(s`tr(r`start_c)`svclvl)(app) /=
              s`sp`SCRAM_info`configs(s`tr(r`start_c + 2)`svclvl)(app)
              AND s`tr(r`start_c + 2)`reconf_st(app) = prepping)
         AND
         NOT (s`sp`SCRAM_info`configs(s`tr(r`start_c)`svclvl)(app) =
               s`sp`SCRAM_info`configs(s`tr(r`start_c + 2)`svclvl)(app)
               AND s`tr(r`start_c + 2)`reconf_st(app) = normal)
         IMPLIES s`tr(2 + r`start_c)`sp`apps(app));

% The subtype TCC (at line 89, column 29) in decl reconf_prep for
s`tr(r`start_c)`svclvl
    %expected type  (extend[speclvl,
                    %         {sp: speclvl | NOT sp = indeterminate}, bool,
                    %         FALSE]
                    %      (sp(s)`S))
  % is subsumed by int_halt_st_TCC1

% The subtype TCC (at line 92, column 29) in decl reconf_prep for
s`tr(r`start_c)`svclvl
    %expected type  (extend[speclvl,
                    %         {sp: speclvl | NOT sp = indeterminate}, bool,
                    %         FALSE]
                    %      (sp(s)`S))
  % is subsumed by int_halt_st_TCC1

% The subtype TCC (at line 93, column 29) in decl reconf_prep for
    %  s`tr(r`start_c + 2)`svclvl
    %expected type  (extend[speclvl,
                    %         {sp: speclvl | NOT sp = indeterminate}, bool,
                    %         FALSE]
                    %      (sp(s)`S))
  % is subsumed by reconf_prep_TCC1

% Subtype TCC generated (at line 101, column 32) for  app
    % expected type  (tr(s)(start_c(r) + 2)`sp`apps)
  % proved - complete
prep_st_TCC1: OBLIGATION
  FORALL (s: sys_trace, r: (get_reconfigs(s))):
    r`end_c - r`start_c > 2 IMPLIES
     (FORALL (app: (s`sp`apps)): s`tr(2 + r`start_c)`sp`apps(app));

% Subtype TCC generated (at line 102, column 33) for  app
    % expected type  (tr(s)(start_c(r) + 3)`sp`apps)
  % proved - complete
prep_st_TCC2: OBLIGATION
  FORALL (s: sys_trace, r: (get_reconfigs(s))):
    r`end_c - r`start_c > 2 IMPLIES
     (FORALL (app: (s`sp`apps)):
```

```
        s`tr(r`start_c + 2)`reconf_st(app) = prepping IMPLIES
         s`tr(3 + r`start_c)`sp`apps(app));

% The subtype TCC (at line 103, column 32) in decl prep_st for  app
    %expected type  (tr(s)(start_c(r) + 3)`sp`apps)
  % is subsumed by prep_st_TCC2

% Subtype TCC generated (at line 109, column 32) for  app
    % expected type  (tr(s)(start_c(r) + 2)`sp`apps)
  % proved - complete
int_prep_len_TCC1: OBLIGATION
  FORALL (s: sys_trace, r: (get_reconfigs(s))):
    r`end_c - r`start_c > 1 IMPLIES
     (FORALL (app: (s`sp`apps)): s`tr(2 + r`start_c)`sp`apps(app));

% Subtype TCC generated (at line 116, column 27) for
    % s`tr(r`start_c + 3)`svclvl
    % expected type  (extend[speclvl,
                  %         {sp: speclvl | NOT sp = indeterminate}, bool,
                  %         FALSE]
                  %      (sp(s)`S))
  % proved - complete
reconf_train_TCC1: OBLIGATION
  FORALL (s: sys_trace, r: (get_reconfigs(s))):
    r`end_c - r`start_c > 2 IMPLIES
     (FORALL (app: (s`sp`apps))):
        extend[speclvl, {sp: speclvl | NOT sp = indeterminate}, bool, FALSE]
            (s`sp`S)(s`tr(r`start_c + 3)`svclvl));

% Subtype TCC generated (at line 117, column 29) for
    % s`tr(r`start_c + 2)`svclvl
    % expected type  (extend[speclvl,
                  %         {sp: speclvl | NOT sp = indeterminate}, bool,
                  %         FALSE]
                  %      (sp(s)`S))
  % proved - complete
reconf_train_TCC2: OBLIGATION
  FORALL (s: sys_trace, r: (get_reconfigs(s))):
    r`end_c - r`start_c > 2 IMPLIES
     (FORALL (app: (s`sp`apps))):
        extend[speclvl, {sp: speclvl | NOT sp = indeterminate}, bool, FALSE]
            (s`sp`S)(s`tr(r`start_c + 2)`svclvl));

% Subtype TCC generated (at line 123, column 32) for  app
    % expected type  (tr(s)(start_c(r) + 3)`sp`apps)
  % proved - complete
train_st_TCC1: OBLIGATION
  FORALL (s: sys_trace, r: (get_reconfigs(s))):
    r`end_c - r`start_c > 2 IMPLIES
     (FORALL (app: (s`sp`apps)): s`tr(3 + r`start_c)`sp`apps(app));

% The subtype TCC (at line 124, column 31) in decl train_st for  app
    %expected type  (tr(s)(start_c(r) + 3)`sp`apps)
  % is subsumed by train_st_TCC1

% The subtype TCC (at line 125, column 31) in decl train_st for  app
    %expected type  (tr(s)(start_c(r) + 3)`sp`apps)
  % is subsumed by train_st_TCC1
```

```
% Subtype TCC generated (at line 136, column 40) for  st`svclvl
    % expected type  (extend[speclvl,
                     %         {sp: speclvl | NOT sp = indeterminate}, bool,
                     %         FALSE]
                     %      (sp(st)`S))
  % proved - complete
invariant_monitor_TCC1: OBLIGATION
  FORALL (st: sys_state, app: (st`sp`apps)):
    (inv(st`sp, st`svclvl, st`st) =>
      inv(st`sp, st`svclvl, monitor(st, app)`st))
     AND NOT st`reconf_st(app) = interrupted
     IMPLIES
     extend[speclvl, {sp: speclvl | NOT sp = indeterminate}, bool, FALSE]
         (st`sp`S)(st`svclvl);

% Subtype TCC generated (at line 136, column 15) for
    % st`sp`SCRAM_info`configs(st`svclvl)(app)
    % expected type  (app`svcs)
  % proved - complete
invariant_monitor_TCC2: OBLIGATION
  FORALL (st: sys_state, app: (st`sp`apps)):
    (inv(st`sp, st`svclvl, st`st) =>
      inv(st`sp, st`svclvl, monitor(st, app)`st))
     AND NOT st`reconf_st(app) = interrupted
     IMPLIES app`svcs(st`sp`SCRAM_info`configs(st`svclvl)(app));

% The subtype TCC (at line 144, column 41) in decl invariant_monitor_rec for  n
    %expected type  below[length(st`sp`app_seq)]
  % is subsumed by rm_speclvl_const_TCC1

% Subtype TCC generated (at line 152, column 27) for  s`tr(r`end_c)`svclvl
    % expected type  (extend[speclvl,
                     %         {sp: speclvl | NOT sp = indeterminate}, bool,
                     %         FALSE]
                     %      (sp(s)`S))
  % proved - complete
same_conf_or_pre_TCC1: OBLIGATION
  FORALL (s: sys_trace, app: (s`sp`apps), r: (get_reconfigs(s))):
    extend[speclvl, {sp: speclvl | NOT sp = indeterminate}, bool, FALSE]
        (s`sp`S)(s`tr(r`end_c)`svclvl);

% Subtype TCC generated (at line 153, column 30) for
    % s`sp`SCRAM_info`configs(s`tr(r`end_c)`svclvl)(app)
    % expected type  (app`svcs)
  % proved - complete
same_conf_or_pre_TCC2: OBLIGATION
  FORALL (s: sys_trace, app: (s`sp`apps), r: (get_reconfigs(s))):
    NOT s`sp`SCRAM_info`configs(s`tr(r`start_c)`svclvl)(app) =
        s`sp`SCRAM_info`configs(s`tr(r`end_c)`svclvl)(app)
     IMPLIES app`svcs(s`sp`SCRAM_info`configs(s`tr(r`end_c)`svclvl)(app));

% Subtype TCC generated (at line 156, column 55) for  app
    % expected type  (tr(s)(end_c(r))`sp`apps)
  % proved - complete
same_conf_or_pre_TCC3: OBLIGATION
  FORALL (s: sys_trace, app: (s`sp`apps), r: (get_reconfigs(s))):
    NOT s`sp`SCRAM_info`configs(s`tr(r`start_c)`svclvl)(app) =
        s`sp`SCRAM_info`configs(s`tr(r`end_c)`svclvl)(app)
     AND
```

```
     NOT pre(app`modules,
             app`svcmap(s`sp`SCRAM_info`configs(s`tr(r`end_c)`svclvl)(app)),
             s`tr(r`end_c)`st)
      AND r`end_c - r`start_c < 3
     IMPLIES (FORALL (app1: (s`sp`apps)): s`tr(r`end_c)`sp`apps(app1));

% Subtype TCC generated (at line 157, column 32) for  app
    % expected type  (tr(s)(end_c(r))`sp`apps)
  % proved - complete
same_conf_or_pre_TCC4: OBLIGATION
  FORALL (s: sys_trace, app: (s`sp`apps), r: (get_reconfigs(s))):
    NOT s`sp`SCRAM_info`configs(s`tr(r`start_c)`svclvl)(app) =
         s`sp`SCRAM_info`configs(s`tr(r`end_c)`svclvl)(app)
     AND
     NOT pre(app`modules,
             app`svcmap(s`sp`SCRAM_info`configs(s`tr(r`end_c)`svclvl)(app)),
             s`tr(r`end_c)`st)
      AND
      NOT (r`end_c - r`start_c < 3 AND
            (EXISTS (app: (s`sp`apps)):
               s`tr(r`end_c)`reconf_st(app) = interrupted))
     IMPLIES s`tr(r`end_c)`sp`apps(app);

% The subtype TCC (at line 151, column 26) in decl same_conf_or_pre for
s`tr(r`start_c)`svclvl
    %expected type  (extend[speclvl,
                    %         {sp: speclvl | NOT sp = indeterminate}, bool,
                    %         FALSE]
                    %      (sp(s)`S))
  % is subsumed by int_halt_st_TCC1

% The subtype TCC (at line 153, column 54) in decl same_conf_or_pre for
s`tr(r`end_c)`svclvl
    %expected type  (extend[speclvl,
                    %         {sp: speclvl | NOT sp = indeterminate}, bool,
                    %         FALSE]
                    %      (sp(s)`S))
  % is subsumed by same_conf_or_pre_TCC1
```

9. Invariant Lemmas

```
% Subtype TCC generated (at line 9, column 53) for  n
    % expected type  below[length(st`sp`app_seq)]
  % proved - complete
rm_spec_constant_TCC1: OBLIGATION
  FORALL (n: nat, st: sys_state):
    NOT n >= card(st`sp`apps) IMPLIES n < st`sp`app_seq`length;

% Subtype TCC generated (at line 14, column 65) for  m
    % expected type  below[length(app_seq(sp(st)))]
  % proved - complete
change_to_interrupt_rec_app_TCC1: OBLIGATION
  FORALL (n: nat, st: sys_state, app: (st`sp`apps)):
    NOT n >= card(st`sp`apps) IMPLIES
     (FORALL (m: nat): m <= n IMPLIES m < st`sp`app_seq`length);
```

```
% Subtype TCC generated (at line 15, column 74) for  app
    % expected type  (recursive_monitor(st, app_seq(sp(st)), n)`sp`apps)
  % proved - complete
change_to_interrupt_rec_app_TCC2: OBLIGATION
  FORALL (n: nat, st: sys_state, app: (st`sp`apps)):
    NOT n >= card(st`sp`apps) AND
     (NOT (EXISTS (m: nat): m <= n AND finseq_appl(st`sp`app_seq)(m) = app))
     IMPLIES recursive_monitor(st, st`sp`app_seq, n)`sp`apps(app);

% The subtype TCC (at line 15, column 61) in decl change_to_interrupt_rec_app for
n
    %expected type  below[length(st`sp`app_seq)]
  % is subsumed by rm_spec_constant_TCC1

% Subtype TCC generated (at line 25, column 68) for  st`svclvl
    % expected type  (extend[speclvl,
                    %         {sp: speclvl | NOT sp = indeterminate}, bool,
                    %         FALSE]
                    %      (sp(st)`S))
  % proved - complete
monitor_inv_equal_TCC1: OBLIGATION
  FORALL (st: sys_state, app: (st`sp`apps)):
    st`reconf_st(app) /= interrupted IMPLIES
     extend[speclvl, {sp: speclvl | NOT sp = indeterminate}, bool, FALSE]
         (st`sp`S)(st`svclvl);

% Subtype TCC generated (at line 25, column 43) for
    % st`sp`SCRAM_info`configs(st`svclvl)(app)
    % expected type  (app`svcs)
  % proved - complete
monitor_inv_equal_TCC2: OBLIGATION
  FORALL (st: sys_state, app: (st`sp`apps)):
    st`reconf_st(app) /= interrupted IMPLIES
     app`svcs(st`sp`SCRAM_info`configs(st`svclvl)(app));

% Subtype TCC generated (at line 31, column 31) for  n
    % expected type  below[length(app_seq(sp(st)))]
  % proved - complete
inv_not_equal_TCC1: OBLIGATION
  FORALL (n: posnat, st: sys_state, app: (st`sp`apps)):
    NOT n >= card(st`sp`apps) IMPLIES n < st`sp`app_seq`length;

% Subtype TCC generated (at line 33, column 68) for  st`svclvl
    % expected type  (extend[speclvl,
                    %         {sp: speclvl | NOT sp = indeterminate}, bool,
                    %         FALSE]
                    %      (sp(st)`S))
  % proved - complete
inv_not_equal_TCC2: OBLIGATION
  FORALL (n: posnat, st: sys_state, app: (st`sp`apps)):
    NOT n >= card(st`sp`apps) AND finseq_appl(st`sp`app_seq)(n) /= app IMPLIES
     extend[speclvl, {sp: speclvl | NOT sp = indeterminate}, bool, FALSE]
         (st`sp`S)(st`svclvl);

% Subtype TCC generated (at line 33, column 43) for
    % st`sp`SCRAM_info`configs(st`svclvl)(app)
    % expected type  (app`svcs)
  % proved - complete
inv_not_equal_TCC3: OBLIGATION
```

```
  FORALL (n: posnat, st: sys_state, app: (st`sp`apps)):
    NOT n >= card(st`sp`apps) AND finseq_appl(st`sp`app_seq)(n) /= app IMPLIES
     app`svcs(st`sp`SCRAM_info`configs(st`svclvl)(app));

% Subtype TCC generated (at line 34, column 69) for  n - 1
    % expected type  below[length(st`sp`app_seq)]
  % proved - complete
inv_not_equal_TCC4: OBLIGATION
  FORALL (n: posnat, st: sys_state, app: (st`sp`apps)):
    NOT n >= card(st`sp`apps) AND finseq_appl(st`sp`app_seq)(n) /= app IMPLIES
     n - 1 >= 0 AND n - 1 < st`sp`app_seq`length;

% The subtype TCC (at line 36, column 68) in decl inv_not_equal for  st`svclvl
    %expected type  (extend[speclvl,
                    %         {sp: speclvl | NOT sp = indeterminate}, bool,
                    %         FALSE]
                    %      (sp(st)`S))
  % is subsumed by inv_not_equal_TCC2

% The subtype TCC (at line 36, column 43) in decl inv_not_equal for
    %  st`sp`SCRAM_info`configs(st`svclvl)(app)
    %expected type  (app`svcs)
  % is subsumed by inv_not_equal_TCC3

% The subtype TCC (at line 37, column 69) in decl inv_not_equal for  n
    %expected type  below[length(st`sp`app_seq)]
  % is subsumed by inv_not_equal_TCC1

% The subtype TCC (at line 42, column 62) in decl invariant_monitor_middle_app for
m
    %expected type  below[length(app_seq(sp(st)))]
  % is subsumed by change_to_interrupt_rec_app_TCC1

% The subtype TCC (at line 45, column 68) in decl invariant_monitor_middle_app for
st`svclvl
    %expected type  (extend[speclvl,
                    %         {sp: speclvl | NOT sp = indeterminate}, bool,
                    %         FALSE]
                    %      (sp(st)`S))
  % is subsumed by monitor_inv_equal_TCC1

% The subtype TCC (at line 45, column 43) in decl invariant_monitor_middle_app for
    %  st`sp`SCRAM_info`configs(st`svclvl)(app)
    %expected type  (app`svcs)
  % is subsumed by monitor_inv_equal_TCC2

% The subtype TCC (at line 46, column 69) in decl invariant_monitor_middle_app for
n
    %expected type  below[length(st`sp`app_seq)]
  % is subsumed by rm_spec_constant_TCC1

% Subtype TCC generated (at line 54, column 69) for  card(st`sp`apps) - 1
    % expected type  below[length(st`sp`app_seq)]
  % proved - complete
invariant_monitor_rec_app_TCC1: OBLIGATION
  FORALL (st: sys_state, app: (st`sp`apps)):
    (NOT st`reconf_st(app) = interrupted) IMPLIES
     card[app_spec](st`sp`apps) - 1 >= 0 AND
      card[app_spec](st`sp`apps) - 1 < st`sp`app_seq`length;
```

```
% The subtype TCC (at line 53, column 68) in decl invariant_monitor_rec_app for
st`svclvl
    %expected type  (extend[speclvl,
                    %         {sp: speclvl | NOT sp = indeterminate}, bool,
                    %         FALSE]
                    %      (sp(st)`S))
  % is subsumed by monitor_inv_equal_TCC1

% The subtype TCC (at line 53, column 43) in decl invariant_monitor_rec_app for
    %  st`sp`SCRAM_info`configs(st`svclvl)(app)
    %expected type  (app`svcs)
  % is subsumed by monitor_inv_equal_TCC2
```

10. Reconfiguration Properties

```
% Subtype TCC generated (at line 17, column 16) for  s`tr(c)`svclvl
    % expected type  (sp(s)`S)
  % proved - complete
CP2_TCC1: OBLIGATION
  FORALL (s: sys_trace, r: (get_reconfigs(s))):
    NOT (r`end_c - r`start_c = 1 AND
          s`tr(r`end_c)`svclvl = s`tr(r`start_c)`svclvl)
     IMPLIES
     (FORALL (c: cycle):
        r`start_c <= c AND c <= r`end_c IMPLIES s`sp`S(s`tr(c)`svclvl));

% Subtype TCC generated (at line 23, column 10) for  s`tr(r`start_c)`svclvl
    % expected type  (sp(s)`S)
  % proved - complete
CP3_TCC1: OBLIGATION
  FORALL (s: sys_trace, r: (get_reconfigs(s))): s`sp`S(s`tr(r`start_c)`svclvl);

% Subtype TCC generated (at line 23, column 34) for  s`tr(r`end_c)`svclvl
    % expected type  (sp(s)`S)
  % proved - complete
CP3_TCC2: OBLIGATION
  FORALL (s: sys_trace, r: (get_reconfigs(s))): s`sp`S(s`tr(r`end_c)`svclvl);

% Subtype TCC generated (at line 29, column 12) for  s`tr(c)`svclvl
    % expected type  (s`sp`S)
  % proved - complete
CP4_TCC1: OBLIGATION FORALL (c: cycle, s: sys_trace): s`sp`S(s`tr(c)`svclvl);

% Subtype TCC generated (at line 44, column 22) for  app
    % expected type  (tr(s)(c)`sp`apps)
  % proved - complete
CP4_TCC2: OBLIGATION
  FORALL (c: cycle, s: sys_trace):
    NOT inv(s`sp, s`tr(c)`svclvl, s`tr(c)`st) AND c > 0 IMPLIES
     (FORALL (app: (s`sp`apps)): s`tr(c)`sp`apps(app));

% Subtype TCC generated (at line 45, column 25) for  app
    % expected type  (tr(s)(c - 1)`sp`apps)
  % proved - complete
CP4_TCC3: OBLIGATION
```

```
  FORALL (c: cycle, s: sys_trace):
    NOT inv(s`sp, s`tr(c)`svclvl, s`tr(c)`st) AND c > 0 IMPLIES
     (FORALL (app: (s`sp`apps)):
        s`tr(c)`reconf_st(app) = interrupted IMPLIES s`tr(c - 1)`sp`apps(app));

% Subtype TCC generated (at line 45, column 10) for  c - 1
    % expected type  cycle
  % proved - complete
CP4_TCC4: OBLIGATION
  FORALL (c: cycle, s: sys_trace):
    NOT inv(s`sp, s`tr(c)`svclvl, s`tr(c)`st) AND c > 0 IMPLIES
     (FORALL (app: (s`sp`apps)):
        s`tr(c)`reconf_st(app) = interrupted IMPLIES c - 1 >= 0);

% Subtype TCC generated (at line 47, column 29) for  s`tr(c - 1)`svclvl
    % expected type  (extend[speclvl,
                    %         {sp: speclvl | NOT sp = indeterminate}, bool,
                    %         FALSE]
                    %      (sp(s)`S))
  % proved - complete
CP4_TCC5: OBLIGATION
  FORALL (c: cycle, s: sys_trace):
    NOT inv(s`sp, s`tr(c)`svclvl, s`tr(c)`st) AND c > 0 IMPLIES
     (FORALL (app: (s`sp`apps)):
        s`tr(c)`reconf_st(app) = interrupted AND
         (s`tr(c - 1)`reconf_st(app) = halting OR
           s`tr(c - 1)`reconf_st(app) = exec_halting)
         IMPLIES
         extend[speclvl, {sp: speclvl | NOT sp = indeterminate}, bool, FALSE]
             (s`sp`S)(s`tr(c - 1)`svclvl));

% Subtype TCC generated (at line 48, column 29) for  s`tr(c)`svclvl
    % expected type  (extend[speclvl,
                    %         {sp: speclvl | NOT sp = indeterminate}, bool,
                    %         FALSE]
                    %      (sp(s)`S))
  % proved - complete
CP4_TCC6: OBLIGATION
  FORALL (c: cycle, s: sys_trace):
    NOT inv(s`sp, s`tr(c)`svclvl, s`tr(c)`st) AND c > 0 IMPLIES
     (FORALL (app: (s`sp`apps)):
        s`tr(c)`reconf_st(app) = interrupted AND
         (s`tr(c - 1)`reconf_st(app) = halting OR
           s`tr(c - 1)`reconf_st(app) = exec_halting)
         IMPLIES
         extend[speclvl, {sp: speclvl | NOT sp = indeterminate}, bool, FALSE]
             (s`sp`S)(s`tr(c)`svclvl));

% Subtype TCC generated (at line 50, column 40) for  s`tr(c - 1)`last_svc
    % expected type  (extend[speclvl,
                    %         {sp: speclvl | NOT sp = indeterminate}, bool,
                    %         FALSE]
                    %      (sp(s)`S))
  % proved - complete
CP4_TCC7: OBLIGATION
  FORALL (c: cycle, s: sys_trace):
    NOT inv(s`sp, s`tr(c)`svclvl, s`tr(c)`st) AND c > 0 IMPLIES
     (FORALL (app: (s`sp`apps)):
        s`tr(c)`reconf_st(app) = interrupted AND
```

```
          (s`tr(c - 1)`reconf_st(app) = halting OR
            s`tr(c - 1)`reconf_st(app) = exec_halting)
           AND
           (s`sp`SCRAM_info`configs(s`tr(c - 1)`svclvl)(app) /=
             s`sp`SCRAM_info`configs(s`tr(c)`svclvl)(app))
          IMPLIES
          extend[speclvl, {sp: speclvl | NOT sp = indeterminate}, bool, FALSE]
              (s`sp`S)(s`tr(c - 1)`last_svc));

% Subtype TCC generated (at line 50, column 16) for
    % s`sp`SCRAM_info`configs(s`tr(c - 1)`last_svc)(app)
    % expected type  (app`svcs)
  % proved - complete
CP4_TCC8: OBLIGATION
  FORALL (c: cycle, s: sys_trace):
    NOT inv(s`sp, s`tr(c)`svclvl, s`tr(c)`st) AND c > 0 IMPLIES
     (FORALL (app: (s`sp`apps)):
        s`tr(c)`reconf_st(app) = interrupted AND
         (s`tr(c - 1)`reconf_st(app) = halting OR
           s`tr(c - 1)`reconf_st(app) = exec_halting)
          AND
          (s`sp`SCRAM_info`configs(s`tr(c - 1)`svclvl)(app) /=
            s`sp`SCRAM_info`configs(s`tr(c)`svclvl)(app))
         IMPLIES app`svcs(s`sp`SCRAM_info`configs(s`tr(c - 1)`last_svc)(app)));

% Subtype TCC generated (at line 56, column 39) for  s`tr(c)`svclvl
    % expected type  (extend[speclvl,
                    %         {sp: speclvl | NOT sp = indeterminate}, bool,
                    %         FALSE]
                    %      (sp(s)`S))
  % proved - complete
CP4_TCC9: OBLIGATION
  FORALL (c: cycle, s: sys_trace):
    NOT inv(s`sp, s`tr(c)`svclvl, s`tr(c)`st) AND c > 0 IMPLIES
     (FORALL (app: (s`sp`apps)):
        NOT (s`tr(c)`reconf_st(app) = interrupted AND
              (s`tr(c - 1)`reconf_st(app) = halting OR
                s`tr(c - 1)`reconf_st(app) = exec_halting)
               AND
               (s`sp`SCRAM_info`configs(s`tr(c - 1)`svclvl)(app) /=
                 s`sp`SCRAM_info`configs(s`tr(c)`svclvl)(app))
               AND
               inv(app`modules,
                   app`svcmap
                       (s`sp`SCRAM_info`configs(s`tr(c - 1)`last_svc)(app)),
                   s`tr(c)`st))
         IMPLIES
         extend[speclvl, {sp: speclvl | NOT sp = indeterminate}, bool, FALSE]
             (s`sp`S)(s`tr(c)`svclvl));

% Subtype TCC generated (at line 56, column 15) for
    % s`sp`SCRAM_info`configs(s`tr(c)`svclvl)(app)
    % expected type  (app`svcs)
  % proved - complete
CP4_TCC10: OBLIGATION
  FORALL (c: cycle, s: sys_trace):
    NOT inv(s`sp, s`tr(c)`svclvl, s`tr(c)`st) AND c > 0 IMPLIES
     (FORALL (app: (s`sp`apps)):
        NOT (s`tr(c)`reconf_st(app) = interrupted AND
```

```
              (s`tr(c - 1)`reconf_st(app) = halting OR
                s`tr(c - 1)`reconf_st(app) = exec_halting)
               AND
               (s`sp`SCRAM_info`configs(s`tr(c - 1)`svclvl)(app) /=
                 s`sp`SCRAM_info`configs(s`tr(c)`svclvl)(app))
               AND
               inv(app`modules,
                   app`svcmap
                       (s`sp`SCRAM_info`configs(s`tr(c - 1)`last_svc)(app)),
                   s`tr(c)`st))
         IMPLIES app`svcs(s`sp`SCRAM_info`configs(s`tr(c)`svclvl)(app)));

% The subtype TCC (at line 46, column 25) in decl CP4 for  app
    %expected type  (tr(s)(c - 1)`sp`apps)
  % is subsumed by CP4_TCC3

% The subtype TCC (at line 46, column 10) in decl CP4 for  c - 1
    %expected type  cycle
  % is subsumed by CP4_TCC4

% The subtype TCC (at line 47, column 34) in decl CP4 for  c - 1
    %expected type  cycle
  % is subsumed by CP4_TCC4

% The subtype TCC (at line 50, column 45) in decl CP4 for  c - 1
    %expected type  cycle
  % is subsumed by CP4_TCC4

% Subtype TCC generated (at line 65, column 29) for  s`tr(r`start_c)`svclvl
    % expected type  (extend[speclvl,
                    %         {sp: speclvl | NOT sp = indeterminate}, bool,
                    %         FALSE]
                    %      (sp(s)`S))
  % proved - complete
CP5_TCC1: OBLIGATION
  FORALL (s: sys_trace, r: (get_reconfigs(s))):
    r`end_c - r`start_c = 3 IMPLIES
     (FORALL (app: (s`sp`apps)):
        extend[speclvl, {sp: speclvl | NOT sp = indeterminate}, bool, FALSE]
            (s`sp`S)(s`tr(r`start_c)`svclvl));

% Subtype TCC generated (at line 66, column 30) for  s`tr(r`end_c)`svclvl
    % expected type  (extend[speclvl,
                    %         {sp: speclvl | NOT sp = indeterminate}, bool,
                    %         FALSE]
                    %      (sp(s)`S))
  % proved - complete
CP5_TCC2: OBLIGATION
  FORALL (s: sys_trace, r: (get_reconfigs(s))):
    r`end_c - r`start_c = 3 IMPLIES
     (FORALL (app: (s`sp`apps)):
        extend[speclvl, {sp: speclvl | NOT sp = indeterminate}, bool, FALSE]
            (s`sp`S)(s`tr(r`end_c)`svclvl));

% Subtype TCC generated (at line 68, column 17) for
    % s`sp`SCRAM_info`configs(s`tr(r`end_c)`svclvl)(app)
    % expected type  (app`svcs)
  % proved - complete
CP5_TCC3: OBLIGATION
```

```
  FORALL (s: sys_trace, r: (get_reconfigs(s))):
    r`end_c - r`start_c = 3 IMPLIES
     (FORALL (app: (s`sp`apps)):
        s`sp`SCRAM_info`configs(s`tr(r`start_c)`svclvl)(app) /=
         s`sp`SCRAM_info`configs(s`tr(r`end_c)`svclvl)(app)
         IMPLIES app`svcs(s`sp`SCRAM_info`configs(s`tr(r`end_c)`svclvl)(app)));

% Subtype TCC generated (at line 73, column 17) for
    % s`sp`SCRAM_info`configs(s`tr(r`end_c)`svclvl)(app)
    % expected type  (app`svcs)
  % proved - complete
CP5_TCC4: OBLIGATION
  FORALL (s: sys_trace, r: (get_reconfigs(s))):
    r`end_c - r`start_c = 3 IMPLIES
     (FORALL (app: (s`sp`apps)):
        NOT (s`sp`SCRAM_info`configs(s`tr(r`start_c)`svclvl)(app) /=
              s`sp`SCRAM_info`configs(s`tr(r`end_c)`svclvl)(app)
              AND
              pre(app`modules,
                  app`svcmap
                      (s`sp`SCRAM_info`configs(s`tr(r`end_c)`svclvl)(app)),
                  s`tr(r`end_c)`st))
         AND
         s`sp`SCRAM_info`configs(s`tr(r`start_c)`svclvl)(app) =
          s`sp`SCRAM_info`configs(s`tr(r`end_c)`svclvl)(app)
         IMPLIES app`svcs(s`sp`SCRAM_info`configs(s`tr(r`end_c)`svclvl)(app)));

% Subtype TCC generated (at line 79, column 29) for  s`tr(r`start_c)`svclvl
    % expected type  (extend[speclvl,
                    %         {sp: speclvl | NOT sp = indeterminate}, bool,
                    %         FALSE]
                    %      (sp(s)`S))
  % proved - complete
CP5_TCC5: OBLIGATION
  FORALL (s: sys_trace, r: (get_reconfigs(s))):
    NOT (r`end_c - r`start_c = 3 AND
          (FORALL (app: (s`sp`apps)):
             (s`sp`SCRAM_info`configs(s`tr(r`start_c)`svclvl)(app) /=
               s`sp`SCRAM_info`configs(s`tr(r`end_c)`svclvl)(app)
               AND
               pre(app`modules,
                   app`svcmap
                       (s`sp`SCRAM_info`configs(s`tr(r`end_c)`svclvl)(app)),
                   s`tr(r`end_c)`st)
               OR
               (s`sp`SCRAM_info`configs(s`tr(r`start_c)`svclvl)(app) =
                 s`sp`SCRAM_info`configs(s`tr(r`end_c)`svclvl)(app)
                 AND
                 inv(app`modules,
                     app`svcmap
                         (s`sp`SCRAM_info`configs(s`tr(r`end_c)`svclvl)(app)),
                     s`tr(r`end_c)`st)))))
     AND r`end_c - r`start_c = 2
     IMPLIES
     (FORALL (app1: (s`sp`apps)):
        extend[speclvl, {sp: speclvl | NOT sp = indeterminate}, bool, FALSE]
            (s`sp`S)(s`tr(r`start_c)`svclvl));

% Subtype TCC generated (at line 80, column 30) for  s`tr(r`end_c)`svclvl
```

```
    % expected type  (extend[speclvl,
                      %         {sp: speclvl | NOT sp = indeterminate}, bool,
                      %         FALSE]
                      %      (sp(s)`S))
  % proved - complete
CP5_TCC6: OBLIGATION
  FORALL (s: sys_trace, r: (get_reconfigs(s))):
    NOT (r`end_c - r`start_c = 3 AND
          (FORALL (app: (s`sp`apps)):
             (s`sp`SCRAM_info`configs(s`tr(r`start_c)`svclvl)(app) /=
               s`sp`SCRAM_info`configs(s`tr(r`end_c)`svclvl)(app)
               AND
               pre(app`modules,
                   app`svcmap
                       (s`sp`SCRAM_info`configs(s`tr(r`end_c)`svclvl)(app)),
                   s`tr(r`end_c)`st)
               OR
               (s`sp`SCRAM_info`configs(s`tr(r`start_c)`svclvl)(app) =
                 s`sp`SCRAM_info`configs(s`tr(r`end_c)`svclvl)(app)
                 AND
                 inv(app`modules,
                     app`svcmap
                         (s`sp`SCRAM_info`configs(s`tr(r`end_c)`svclvl)(app)),
                     s`tr(r`end_c)`st)))))
     AND r`end_c - r`start_c = 2
     IMPLIES
     (FORALL (app1: (s`sp`apps)):
        extend[speclvl, {sp: speclvl | NOT sp = indeterminate}, bool, FALSE]
            (s`sp`S)(s`tr(r`end_c)`svclvl));

% Subtype TCC generated (at line 82, column 17) for
    % s`sp`SCRAM_info`configs(s`tr(r`end_c)`svclvl)(app)
    % expected type  (app`svcs)
  % proved - complete
CP5_TCC7: OBLIGATION
  FORALL (s: sys_trace, r: (get_reconfigs(s))):
    NOT (r`end_c - r`start_c = 3 AND
          (FORALL (app: (s`sp`apps)):
             (s`sp`SCRAM_info`configs(s`tr(r`start_c)`svclvl)(app) /=
               s`sp`SCRAM_info`configs(s`tr(r`end_c)`svclvl)(app)
               AND
               pre(app`modules,
                   app`svcmap
                       (s`sp`SCRAM_info`configs(s`tr(r`end_c)`svclvl)(app)),
                   s`tr(r`end_c)`st)
               OR
               (s`sp`SCRAM_info`configs(s`tr(r`start_c)`svclvl)(app) =
                 s`sp`SCRAM_info`configs(s`tr(r`end_c)`svclvl)(app)
                 AND
                 inv(app`modules,
                     app`svcmap
                         (s`sp`SCRAM_info`configs(s`tr(r`end_c)`svclvl)(app)),
                     s`tr(r`end_c)`st)))))
     AND r`end_c - r`start_c = 2
     IMPLIES
     (FORALL (app1: (s`sp`apps)):
        s`sp`SCRAM_info`configs(s`tr(r`start_c)`svclvl)(app1) =
         s`sp`SCRAM_info`configs(s`tr(r`end_c)`svclvl)(app1)
         IMPLIES
```

```
        app1`svcs(s`sp`SCRAM_info`configs(s`tr(r`end_c)`svclvl)(app1)));

% Subtype TCC generated (at line 86, column 54) for  app
    % expected type  (tr(s)(end_c(r))`sp`apps)
  % proved - complete
CP5_TCC8: OBLIGATION
  FORALL (s: sys_trace, r: (get_reconfigs(s))):
    NOT (r`end_c - r`start_c = 3 AND
          (FORALL (app: (s`sp`apps)):
             (s`sp`SCRAM_info`configs(s`tr(r`start_c)`svclvl)(app) /=
               s`sp`SCRAM_info`configs(s`tr(r`end_c)`svclvl)(app)
               AND
               pre(app`modules,
                   app`svcmap
                       (s`sp`SCRAM_info`configs(s`tr(r`end_c)`svclvl)(app)),
                   s`tr(r`end_c)`st)
               OR
               (s`sp`SCRAM_info`configs(s`tr(r`start_c)`svclvl)(app) =
                 s`sp`SCRAM_info`configs(s`tr(r`end_c)`svclvl)(app)
                 AND
                 inv(app`modules,
                     app`svcmap
                         (s`sp`SCRAM_info`configs(s`tr(r`end_c)`svclvl)(app)),
                     s`tr(r`end_c)`st)))))
     AND
     NOT (r`end_c - r`start_c = 2 AND
           (FORALL (app: (s`sp`apps)):
              (s`sp`SCRAM_info`configs(s`tr(r`start_c)`svclvl)(app) =
                s`sp`SCRAM_info`configs(s`tr(r`end_c)`svclvl)(app)
                AND
                inv(app`modules,
                    app`svcmap
                        (s`sp`SCRAM_info`configs(s`tr(r`end_c)`svclvl)(app)),
                    s`tr(r`end_c)`st))))
     IMPLIES (FORALL (app1: (s`sp`apps)): s`tr(r`end_c)`sp`apps(app1));

% The subtype TCC (at line 68, column 41) in decl CP5 for  s`tr(r`end_c)`svclvl
    %expected type  (extend[speclvl,
                    %         {sp: speclvl | NOT sp = indeterminate}, bool,
                    %         FALSE]
                    %      (sp(s)`S))
  % is subsumed by CP5_TCC2

% The subtype TCC (at line 70, column 29) in decl CP5 for  s`tr(r`start_c)`svclvl
    %expected type  (extend[speclvl,
                    %         {sp: speclvl | NOT sp = indeterminate}, bool,
                    %         FALSE]
                    %      (sp(s)`S))
  % is subsumed by CP5_TCC1

% The subtype TCC (at line 71, column 30) in decl CP5 for  s`tr(r`end_c)`svclvl
    %expected type  (extend[speclvl,
                    %         {sp: speclvl | NOT sp = indeterminate}, bool,
                    %         FALSE]
                    %      (sp(s)`S))
  % is subsumed by CP5_TCC2

% The subtype TCC (at line 73, column 41) in decl CP5 for  s`tr(r`end_c)`svclvl
    %expected type  (extend[speclvl,
```

```
                  %         {sp: speclvl | NOT sp = indeterminate}, bool,
                  %         FALSE]
                  %      (sp(s)`S))
  % is subsumed by CP5_TCC2

% The subtype TCC (at line 82, column 41) in decl CP5 for  s`tr(r`end_c)`svclvl
    %expected type  (extend[speclvl,
                  %         {sp: speclvl | NOT sp = indeterminate}, bool,
                  %         FALSE]
                  %      (sp(s)`S))
  % is subsumed by CP5_TCC6
```

Part III

Architecture Proofs

```
Proof summary for theory lemmas_inv
   rm_spec_constant_TCC1.............proved - complete   [shostak](   5.59 s)
   rm_spec_constant..................proved - complete   [shostak]( 903.55 s)
   change_to_interrupt_rec_app_TCC1...proved - complete   [shostak](   5.64 s)
   change_to_interrupt_rec_app_TCC2...proved - complete   [shostak]( 365.93 s)
   change_to_interrupt_rec_app........proved - complete   [shostak](1003.11 s)
   monitor_not_equal_nc..............proved - complete   [shostak]( 872.74 s)
   monitor_inv_equal_TCC1............proved - complete   [shostak](   4.99 s)
   monitor_inv_equal_TCC2............proved - complete   [shostak](  67.12 s)
   monitor_inv_equal.................proved - complete   [shostak](1155.83 s)
   inv_not_equal_TCC1................proved - complete   [shostak](   5.36 s)
   inv_not_equal_TCC2................proved - complete   [shostak](   5.50 s)
   inv_not_equal_TCC3................proved - complete   [shostak](  58.64 s)
   inv_not_equal_TCC4................proved - complete   [shostak](   5.81 s)
   inv_not_equal.....................proved - complete   [shostak](6645.93 s)
   invariant_monitor_middle_app.......proved - complete   [shostak](2990.12 s)
   invariant_monitor_rec_app_TCC1.....proved - complete   [shostak]( 362.13 s)
   invariant_monitor_rec_app..........proved - complete   [shostak]( 425.49 s)
   Theory totals: 17 formulas, 17 attempted, 17 succeeded (14883.48 s)

Proof summary for theory lemmas
   reconfig_start?_TCC1..............proved - complete   [shostak](    5.47 s)
   reconfig_end?_TCC1................proved - complete   [shostak](   10.37 s)
   change_to_interrupt_TCC1..........proved - complete   [shostak](   58.82 s)
   change_to_interrupt...............proved - complete   [shostak](15011.61 s)
   change_to_interrupt_rec_TCC1......proved - complete   [shostak](   17.93 s)
   change_to_interrupt_rec_TCC2......proved - complete   [shostak](   15.62 s)
   change_to_interrupt_rec...........proved - complete   [shostak]( 3480.43 s)
   m_speclvl_const...................proved - complete   [shostak](   26.16 s)
   rm_speclvl_const_TCC1.............proved - complete   [shostak](   15.17 s)
   rm_speclvl_const..................proved - complete   [shostak]( 2628.11 s)
   interrupt_st_TCC1.................proved - complete   [shostak](   36.66 s)
   interrupt_st......................proved - complete   [shostak](  121.48 s)
   reconf_halt_TCC1..................proved - complete   [shostak](   18.63 s)
   reconf_halt.......................proved - complete   [shostak](   22.03 s)
   int_halt_st_TCC1..................proved - complete   [shostak](   11.21 s)
   int_halt_st_TCC2..................proved - complete   [shostak](   30.08 s)
   int_halt_st.......................proved - complete   [shostak](  101.15 s)
   int_halt_len......................proved - complete   [shostak](   12.66 s)
   halt_st_TCC1......................proved - complete   [shostak](    4.48 s)
   halt_st_TCC2......................proved - complete   [shostak](   11.84 s)
   halt_st...........................proved - complete   [shostak](  282.12 s)
   reconf_prep_TCC1..................proved - complete   [shostak](   24.49 s)
   reconf_prep_TCC2..................proved - complete   [shostak](   19.50 s)
   reconf_prep_TCC3..................proved - complete   [shostak](   44.51 s)
   reconf_prep_TCC4..................proved - complete   [shostak](   45.27 s)
   reconf_prep.......................proved - complete   [shostak]( 5658.24 s)
   prep_st_TCC1......................proved - complete   [shostak](   18.76 s)
   prep_st_TCC2......................proved - complete   [shostak](   25.67 s)
   prep_st...........................proved - complete   [shostak](  633.30 s)
   int_prep_len_TCC1.................proved - complete   [shostak](   22.77 s)
   int_prep_len......................proved - complete   [shostak](   32.46 s)
   reconf_train_TCC1.................proved - complete   [shostak](   26.83 s)
   reconf_train_TCC2.................proved - complete   [shostak](   29.13 s)
   reconf_train......................proved - complete   [shostak]( 1318.67 s)
   train_st_TCC1.....................proved - complete   [shostak](   22.67 s)
   train_st..........................proved - complete   [shostak](  219.00 s)
   reconf_length.....................proved - complete   [shostak](   42.65 s)
   invariant_monitor_TCC1............proved - complete   [shostak](   55.03 s)
```

```
   invariant_monitor_TCC2.............proved - complete   [shostak](   47.87 s)
   invariant_monitor..................proved - complete   [shostak]( 3856.24 s)
   invariant_monitor_rec..............proved - complete   [shostak]( 1702.00 s)
   cycle_time.........................proved - complete   [shostak](    2.07 s)
   same_conf_or_pre_TCC1..............proved - complete   [shostak](   27.70 s)
   same_conf_or_pre_TCC2..............proved - complete   [shostak](  143.99 s)
   same_conf_or_pre_TCC3..............proved - complete   [shostak](   40.21 s)
   same_conf_or_pre_TCC4..............proved - complete   [shostak](   36.45 s)
   same_conf_or_pre...................proved - complete   [shostak](  254.44 s)
   Theory totals: 47 formulas, 47 attempted, 47 succeeded (36271.95 s)

Proof summary for theory trace
   recursive_monitor_apps_TCC1.........proved - complete   [shostak](  0.63 s)
   recursive_monitor_apps..............proved - complete   [shostak](230.79 s)
   recursive_monitor_svclvl............proved - complete   [shostak](424.09 s)
   system_monitor_TCC1.................proved - complete   [shostak](  0.22 s)
   system_monitor_TCC2.................proved - complete   [shostak](  9.90 s)
   system_monitor_TCC3.................proved - complete   [shostak](460.63 s)
   system_monitor_TCC4.................proved - complete   [shostak]( 17.88 s)
   system_monitor_TCC5.................proved - complete   [shostak](123.99 s)
   sys_trace_TCC1......................proved - complete   [shostak](  0.09 s)
   sys_trace_TCC2......................proved - complete   [shostak](  1.15 s)
   sys_trace_TCC3......................proved - complete   [shostak]( 28.23 s)
   sys_trace_TCC4......................proved - complete   [shostak](  1.10 s)
   sys_trace_TCC5......................proved - complete   [shostak](  0.70 s)
   train_time_TCC1.....................proved - complete   [shostak]( 18.93 s)
   train_time_TCC2.....................proved - complete   [shostak](  9.47 s)
   train_time_TCC3.....................proved - complete   [shostak]( 34.44 s)
   Theory totals: 16 formulas, 16 attempted, 16 succeeded (1362.24 s)

Proof summary for theory monitor
   nonempty_apps.......................proved - complete   [shostak]( 8.42 s)
   success_exec?_TCC1..................proved - complete   [shostak]( 0.02 s)
   success_halt?_TCC1..................proved - complete   [shostak]( 0.02 s)
   success_eh?_TCC1....................proved - complete   [shostak]( 0.01 s)
   success_prep?_TCC1..................proved - complete   [shostak]( 0.02 s)
   success_train?_TCC1.................proved - complete   [shostak]( 0.01 s)
   execute_TCC1........................proved - complete   [shostak]( 2.09 s)
   execute_TCC2........................proved - complete   [shostak](15.20 s)
   monitor_apps_constant...............proved - complete   [shostak](19.35 s)
   monitor_svcs_constant...............proved - complete   [shostak](95.33 s)
   rm_apps_const_TCC1..................proved - complete   [shostak]( 6.11 s)
   rm_apps_const_TCC2..................proved - complete   [shostak]( 4.10 s)
   recursive_monitor_TCC1..............proved - complete   [shostak]( 1.70 s)
   recursive_monitor_TCC2..............proved - complete   [shostak]( 2.19 s)
   recursive_monitor_TCC3..............proved - complete   [shostak]( 1.71 s)
   recursive_monitor_TCC4..............proved - complete   [shostak]( 3.65 s)
   Theory totals: 16 formulas, 16 attempted, 16 succeeded (159.93 s)

Proof summary for theory reconf_spec
   reconf_spec_TCC1....................proved - complete   [shostak](1.99 s)
   post_TCC1...........................proved - complete   [shostak](1.77 s)
   post_TCC2...........................proved - complete   [shostak](8.12 s)
   Theory totals: 3 formulas, 3 attempted, 3 succeeded (11.88 s)

Proof summary for theory SCRAM
   Theory totals: 0 formulas, 0 attempted, 0 succeeded (0.00 s)

Proof summary for theory application
```

```
   post_TCC1.............................proved - complete   [shostak](0.63 s)
   Theory totals: 1 formulas, 1 attempted, 1 succeeded (0.63 s)

Proof summary for theory module
   Theory totals: 0 formulas, 0 attempted, 0 succeeded (0.00 s)

Proof summary for theory state
   Theory totals: 0 formulas, 0 attempted, 0 succeeded (0.00 s)

Proof summary for theory assured_reconfig
   CP1...............................proved - complete   [shostak](   36.36 s)
   CP2_TCC1..........................proved - complete   [shostak](   10.35 s)
   CP2...............................proved - complete   [shostak]( 6641.14 s)
   CP3_TCC1..........................proved - complete   [shostak](   16.64 s)
   CP3_TCC2..........................proved - complete   [shostak](   28.54 s)
   CP3...............................proved - complete   [shostak](  778.75 s)
   CP4_TCC1..........................proved - complete   [shostak](    1.87 s)
   CP4_TCC2..........................proved - complete   [shostak](   10.99 s)
   CP4_TCC3..........................proved - complete   [shostak](   13.84 s)
   CP4_TCC4..........................proved - complete   [shostak](    8.59 s)
   CP4_TCC5..........................proved - complete   [shostak](   16.42 s)
   CP4_TCC6..........................proved - complete   [shostak](   13.83 s)
   CP4_TCC7..........................proved - complete   [shostak](   23.05 s)
   CP4_TCC8..........................proved - complete   [shostak](   45.50 s)
   CP4_TCC9..........................proved - complete   [shostak](   24.49 s)
   CP4_TCC10.........................proved - complete   [shostak](   43.44 s)
   CP4...............................proved - complete   [shostak]( 5922.29 s)
   CP5_TCC1..........................proved - complete   [shostak](   12.09 s)
   CP5_TCC2..........................proved - complete   [shostak](   11.47 s)
   CP5_TCC3..........................proved - complete   [shostak](   28.49 s)
   CP5_TCC4..........................proved - complete   [shostak](   37.42 s)
   CP5_TCC5..........................proved - complete   [shostak](   25.57 s)
   CP5_TCC6..........................proved - complete   [shostak](   25.69 s)
   CP5_TCC7..........................proved - complete   [shostak](   49.90 s)
   CP5_TCC8..........................proved - complete   [shostak](   33.05 s)
   CP5...............................proved - complete   [shostak](13048.43 s)
   Theory totals: 26 formulas, 26 attempted, 26 succeeded (26908.20 s)

Grand Totals: 126 proofs, 126 attempted, 126 succeeded (79598.30 s)
```

1. State

The State theory has no proofs.

2. Module

The Module theory has no proofs.

3. Application

```
Proof scripts for file application-fm.pvs:


application.post_TCC1: proved - complete [shostak](0.63 s)

("" (grind))
```

4. SCRAM

The SCRAM theory has no proofs.

5. Reconfiguration Specification

```
Proof scripts for file reconf_spec-fm.pvs:

reconf_spec.reconf_spec_TCC1: proved - complete [shostak](1.99 s)

("" (skosimp) (grind))


reconf_spec.post_TCC1: proved - complete [shostak](1.77 s)

("" (skosimp) (grind))


reconf_spec.post_TCC2: proved - complete [shostak](8.12 s)

(""
 (skosimp)
 (typepred "r!1`SCRAM_info`configs(svc!1)")
 (("1" (inst -1 "app!1")) ("2" (grind))))
```

6. Monitor

```
Proof scripts for file monitor-fm.pvs:


monitor.nonempty_apps: proved - complete [shostak](8.42 s)

(""
 (skosimp)
 (lemma "card_empty?[app_spec]")
 (typepred "st!1`sp`apps")
 (grind))


monitor.success_exec?_TCC1: proved - complete [shostak](0.02 s)

("" (inst 1 "(lambda(x: sys_state):normal)"))


monitor.success_halt?_TCC1: proved - complete [shostak](0.02 s)

("" (inst 1 "(lambda(x: sys_state):interrupted)"))


monitor.success_eh?_TCC1: proved - complete [shostak](0.01 s)

("" (inst 1 "(lambda(x: sys_state):interrupted)"))


monitor.success_prep?_TCC1: proved - complete [shostak](0.02 s)

("" (inst 1 "(lambda(x: sys_state):interrupted)"))


monitor.success_train?_TCC1: proved - complete [shostak](0.01 s)

("" (inst 1 "(lambda(x: sys_state):interrupted)"))


monitor.execute_TCC1: proved - complete [shostak](2.09 s)

("" (grind))


monitor.execute_TCC2: proved - complete [shostak](15.20 s)

(""
 (skosimp)
 (typepred "st!1`sp`SCRAM_info`configs(st!1`svclvl)")
 (("1" (inst -1 "app!1")) ("2" (grind))))


monitor.monitor_apps_constant: proved - complete [shostak](19.35 s)

("" (grind))


monitor.monitor_svcs_constant: proved - complete [shostak](95.33 s)
```

```
("" (grind))
```

```
monitor.rm_apps_const_TCC1: proved - complete [shostak](6.11 s)
```

```
("" (skosimp) (typepred "n!1") (grind))
```

```
monitor.rm_apps_const_TCC2: proved - complete [shostak](4.10 s)
```

```
("" (skosimp) (grind))
```

```
monitor.recursive_monitor_TCC1: proved - complete [shostak](1.70 s)
```

```
("" (grind))
```

```
monitor.recursive_monitor_TCC2: proved - complete [shostak](2.19 s)
```

```
("" (grind))
```

```
monitor.recursive_monitor_TCC3: proved - complete [shostak](1.71 s)
```

```
("" (grind))
```

```
monitor.recursive_monitor_TCC4: proved - complete [shostak](3.65 s)
```

```
(""
 (lemma "rm_apps_const")
 (skosimp)
 (inst -1 "st!1" "apps!1" "n!1" "v!1")
 (split)
 (("1" (typepred "apps!1(n!1)") (grind)) ("2" (propax))))
```

7. Trace

```
Proof scripts for file trace-fm.pvs:
```

```
trace.recursive_monitor_apps_TCC1: proved - complete [shostak](0.63 s)
```

```
("" (grind))
```

```
trace.recursive_monitor_apps: proved - complete [shostak](230.79 s)
```

```
(""
 (induct "n")
 (("1" (grind))
  ("2"
   (skosimp)
   (expand "recursive_monitor" 1)
```

```
   (skosimp*)
   (inst -1 "st!1")
   (split -1)
   (("1" (assert)) ("2" (expand "monitor") (lift-if) (grind))))
  ("3" (hide 2) (skosimp*) (grind))))
```

trace.recursive_monitor_svclvl: proved - complete [shostak](424.09 s)

```
(""
 (induct "n")
 (("1" (grind))
  ("2"
   (skosimp)
   (expand "recursive_monitor" 1)
   (skosimp*)
   (inst -1 "st!1")
   (split -1)
   (("1" (assert)) ("2" (expand "monitor") (lift-if) (grind))))
  ("3" (hide 2) (skosimp) (grind))))
```

trace.system_monitor_TCC1: proved - complete [shostak](0.22 s)

```
(""
 (skosimp)
 (split)
 (("1"
   (grind)
   (typepred "st!1`sp`apps")
   (expand "nonempty?")
   (expand "empty?")
   (skosimp)
   (expand "is_finite")
   (hide (-1 -3))
   (grind)
   (case "card(st!1`sp`apps) = 0")
   (("1"
     (grind)
     (lemma "empty_card[app_spec]")
     (inst -1 "st!1`sp`apps")
     (grind))
    ("2" (grind))))
  ("2"
   (typepred "st!1`sp`app_seq")
   (typepred "st!1`sp`apps")
   (lemma "card_empty?[app_spec]")
   (inst -1 "st!1`sp`apps")
   (grind))))
```

trace.system_monitor_TCC2: proved - complete [shostak](9.90 s)

```
(""
 (skosimp)
 (skosimp*)
 (lemma "nonempty_apps")
 (inst -1 "st!1")
 (typepred "st!1`sp`app_seq")
```

```
 (hide (-2 -4 -5 1))
 (grind))


trace.system_monitor_TCC3: proved - complete [shostak](460.63 s)

(""
 (skosimp)
 (skosimp)
 (skosimp)
 (hide -1 -2 1)
 (lemma "nonempty_apps")
 (inst -1 "st!1")
 (grind))


trace.system_monitor_TCC4: proved - complete [shostak](17.88 s)

("" (skosimp) (hide 1) (skosimp) (grind))


trace.system_monitor_TCC5: proved - complete [shostak](123.99 s)

("" (skosimp) (hide 1) (lemma "nonempty_apps") (inst -1 "st!1") (grind))


trace.sys_trace_TCC1: proved - complete [shostak](0.09 s)

("" (skosimp) (grind))


trace.sys_trace_TCC2: proved - complete [shostak](1.15 s)

(""
 (skosimp)
 (skosimp)
 (expand "member")
 (typepred "env!1(c1!1 * cycle_time)")
 (inst -1 "e!1")
 (grind))


trace.sys_trace_TCC3: proved - complete [shostak](28.23 s)

("" (skosimp) (hide -1) (skosimp) (typepred "c!1(0)") (grind))


trace.sys_trace_TCC4: proved - complete [shostak](1.10 s)

("" (subtype-tcc))


trace.sys_trace_TCC5: proved - complete [shostak](0.70 s)

(""
 (skosimp)
 (skosimp)
 (skosimp)
 (typepred "app1!1")
```

```
 (typepred "c!1(cyc2!1)")
 (hide -3 -4 -5 -6)
 (grind))


trace.train_time_TCC1: proved - complete [shostak](18.93 s)

("" (skosimp) (typepred "tr(s!1)(c!1)") (grind))


trace.train_time_TCC2: proved - complete [shostak](9.47 s)

("" (grind))


trace.train_time_TCC3: proved - complete [shostak](34.44 s)

(""
 (skosimp)
 (typepred "s!1`sp`SCRAM_info`configs(s!1`tr(c!1)`svclvl)")
 (("1" (inst -1 "app!1")) ("2" (grind))))
```

8. Lemmas

```
Proof scripts for file lemmas-fm.pvs:


lemmas.reconfig_start?_TCC1: proved - complete [shostak](5.47 s)

("" (grind))


lemmas.reconfig_end?_TCC1: proved - complete [shostak](10.37 s)

("" (grind))


lemmas.change_to_interrupt_TCC1: proved - complete [shostak](58.82 s)

("" (grind))


lemmas.change_to_interrupt: proved - complete [shostak](15011.61 s)

(""
 (skosimp)
 (typepred "success_exec?(st!1)")
 (typepred "success_eh?(st!1)")
 (typepred "success_halt?(st!1)")
 (typepred "success_train?(st!1)")
 (typepred "success_prep?(st!1)")
 (skosimp)
 (expand "monitor")
 (expand "execute")
 (expand "halt")
 (expand "exec_halt")
```

```
(expand "prep")
(lift-if)
(split)
(("1"
  (split)
  (("1"
    (split)
    (("1"
      (split)
      (("1"
        (split)
        (("1"
          (split)
          (("1"
            (split)
            (("1"
              (flatten)
              (split)
              (("1"
                (flatten)
                (split 2)
                (("1" (flatten) (grind)) ("2" (flatten))))
               ("2"
                (flatten)
                (split)
                (("1" (split) (("1" (grind)) ("2" (grind))))
                 ("2"
                  (split)
                  (("1" (grind))
                   ("2"
                    (flatten)
                    (split)
                    (("1" (grind)) ("2" (flatten) (grind))))))))))
             ("2" (flatten))))
           ("2"
            (split)
            (("1" (flatten))
             ("2"
              (flatten)
              (split)
              (("1"
                (flatten)
                (split)
                (("1" (split) (("1" (propax)) ("2" (flatten))))
                 ("2" (split) (("1" (propax)) ("2" (flatten))))))
               ("2"
                (split)
                (("1" (propax))
                 ("2"
                  (flatten)
                  (split)
                  (("1"
                    (flatten)
                    (split)
                    (("1"
                      (flatten)
                      (split)
                      (("1"
                        (flatten)
```

```
                    (split)
                    (("1" (flatten) (grind)) ("2" (flatten))))
                   ("2" (flatten))))
                  ("2"
                   (split)
                   (("1"
                     (flatten)
                     (split)
                     (("1" (flatten)) ("2" (grind))))
                    ("2" (flatten))))))
                 ("2"
                  (split)
                  (("1" (flatten))
                   ("2"
                    (flatten)
                    (split)
                    (("1"
                      (flatten)
                      (split)
                      (("1"
                        (flatten)
                        (split)
                        (("1"
                          (split)
                          (("1"
                            (flatten)
                            (split)
                            (("1" (flatten) (grind)) ("2" (grind))))
                           ("2" (flatten))))
                         ("2"
                          (split)
                          (("1"
                            (flatten)
                            (split)
                            (("1" (flatten) (grind)) ("2" (grind))))
                           ("2" (flatten))))))
                       ("2"
                        (split)
                        (("1"
                          (flatten)
                          (split)
                          (("1" (flatten)) ("2" (flatten) (grind))))
                         ("2" (flatten))))))
                     ("2"
                      (split)
                      (("1" (flatten))
                       ("2"
                        (flatten)
                        (split)
                        (("1"
                          (flatten)
                          (split)
                          (("1" (flatten) (grind)) ("2" (grind))))
                         ("2" (flatten) (grind))))))))))))))))))
("2"
 (split)
 (("1"
   (flatten)
   (split)
```

```
 (("1"
   (flatten)
   (split)
   (("1" (flatten) (grind)) ("2" (flatten))))
  ("2"
   (split)
   (("1"
     (flatten)
     (split)
     (("1" (flatten)) ("2" (flatten) (grind))))
    ("2" (flatten))))))
("2"
 (split)
 (("1" (flatten))
  ("2"
   (flatten)
   (split)
   (("1"
     (flatten)
     (split)
     (("1" (split) (("1" (propax)) ("2" (flatten))))
      ("2" (split) (("1" (propax)) ("2" (flatten))))))
    ("2"
     (split)
     (("1" (propax))
      ("2"
       (flatten)
       (split)
       (("1"
         (flatten)
         (split)
         (("1"
           (flatten)
           (split)
           (("1" (flatten) (grind)) ("2" (grind))))
          ("2"
           (flatten)
           (split)
           (("1" (grind)) ("2" (grind))))))
        ("2"
         (flatten)
         (split)
         (("1"
           (flatten)
           (split)
           (("1"
             (flatten)
             (split)
             (("1" (grind)) ("2" (grind))))
            ("2"
             (split)
             (("1" (grind)) ("2" (flatten) (grind))))))
          ("2"
           (split)
           (("1" (flatten))
            ("2"
             (flatten)
             (split)
             (("1" (flatten))
```

```
                    ("2"
                     (split)
                     (("1"
                       (flatten)
                       (split)
                       (("1"
                         (flatten)
                         (split)
                         (("1" (grind)) ("2" (flatten))))
                        ("2"
                         (split)
                         (("1" (flatten) (grind))
                          ("2" (flatten))))))
                      ("2"
                       (flatten)
                       (split)
                       (("1"
                         (flatten)
                         (split)
                         (("1" (grind)) ("2" (grind))))
                        ("2"
                         (split)
                         (("1" (grind))
                          ("2"
                           (flatten)
                           (split)
                           (("1" (grind))
                            ("2"
                             (flatten)
                             (split)
                             (("1" (grind))
                              ("2"
                               (grind)))))))))))))))))))))))))))))))
("2"
 (split)
 (("1"
   (split)
   (("1"
     (split)
     (("1"
       (flatten)
       (split)
       (("1" (grind)) ("2" (flatten) (grind))))
      ("2" (flatten))))
    ("2"
     (split)
     (("1" (flatten))
      ("2"
       (flatten)
       (split)
       (("1" (split) (("1" (propax)) ("2" (flatten))))
        ("2"
         (split)
         (("1" (propax))
          ("2"
           (flatten)
           (hide 4)
           (split)
           (("1" (grind))
```

```
          ("2"
           (flatten)
           (split)
           (("1" (grind) (reveal 1) (grind))
            ("2" (grind))))))))))))))
("2"
 (split)
 (("1"
   (split)
   (("1"
     (flatten)
     (split)
     (("1" (flatten) (grind))
      ("2" (flatten) (split) (("1" (grind)) ("2" (grind))))))
    ("2" (flatten))))
  ("2"
   (split)
   (("1" (flatten))
    ("2"
     (flatten)
     (split)
     (("1"
       (flatten)
       (split)
       (("1" (split) (("1" (propax)) ("2" (flatten))))
        ("2" (split) (("1" (propax)) ("2" (flatten))))))
      ("2"
       (split)
       (("1" (propax))
        ("2"
         (flatten)
         (split)
         (("1"
           (flatten)
           (split)
           (("1"
             (flatten)
             (split)
             (("1" (grind)) ("2" (grind))))
            ("2" (split) (("1" (grind)) ("2" (flatten))))))
          ("2"
           (split)
           (("1" (flatten))
            ("2"
             (flatten)
             (split)
             (("1"
               (flatten)
               (split)
               (("1"
                 (flatten)
                 (split)
                 (("1"
                   (split)
                   (("1" (grind)) ("2" (flatten))))
                  ("2"
                   (split)
                   (("1" (grind)) ("2" (grind))))))
                ("2"
```

```
                          (flatten)
                          (split)
                          (("1"
                            (split)
                            (("1" (grind)) ("2" (flatten))))
                           ("2"
                            (split)
                            (("1" (grind)) ("2" (grind))))))))
                        ("2"
                         (split)
                         (("1" (flatten))
                          ("2"
                           (flatten)
                           (split)
                           (("1" (flatten) (grind))
                            ("2" (grind))))))))))))))))))))))))
    ("2"
     (hide 2)
     (grind)
     (("1" (reveal 1) (grind)) ("2" (reveal 1) (grind))
      ("3" (reveal 1) (grind)) ("4" (reveal 1) (grind))
      ("5" (reveal 1) (grind)) ("6" (reveal 1) (grind))
      ("7" (reveal 1) (grind)) ("8" (reveal 1) (grind))))))
  ("2"
   (hide 2)
   (grind)
   (("1" (reveal 1) (grind)) ("2" (reveal 1) (grind))
    ("3" (reveal 1) (grind)) ("4" (reveal 1) (grind))
    ("5" (reveal 1) (grind)) ("6" (reveal 1) (grind))
    ("7" (reveal 1) (grind)) ("8" (reveal 1) (grind))
    ("9" (reveal 1) (grind)) ("10" (reveal 1) (grind))
    ("11" (reveal 1) (grind)) ("12" (reveal 1) (grind))
    ("13" (reveal 1) (grind)) ("14" (reveal 1) (grind))
    ("15" (reveal 1) (grind)) ("16" (reveal 1) (grind))
    ("17" (reveal 1) (grind)) ("18" (reveal 1) (grind))
    ("19" (reveal 1) (grind)) ("20" (reveal 1) (grind))))))
("2"
 (hide 2)
 (split 1)
 (("1"
   (flatten)
   (grind)
   (("1" (reveal 1) (grind)) ("2" (reveal 1) (grind))
    ("3" (reveal 1) (grind)) ("4" (reveal 1) (grind))
    ("5" (reveal 1) (grind)) ("6" (reveal 1) (grind))
    ("7" (reveal 1) (grind)) ("8" (reveal 1) (grind))))
  ("2"
   (grind)
   (("1" (reveal 1) (grind)) ("2" (reveal 1) (grind))
    ("3" (reveal 1) (grind)) ("4" (reveal 1) (grind))
    ("5" (reveal 1) (grind)) ("6" (reveal 1) (grind))
    ("7" (reveal 1) (grind)) ("8" (reveal 1) (grind))
    ("9" (reveal 1) (grind)) ("10" (reveal 1) (grind))
    ("11" (reveal 1) (grind)) ("12" (reveal 1) (grind))
    ("13" (reveal 1) (grind)) ("14" (reveal 1) (grind))
    ("15" (reveal 1) (grind)) ("16" (reveal 1) (grind))
    ("17" (reveal 1) (grind)) ("18" (reveal 1) (grind))
    ("19" (reveal 1) (grind)) ("20" (reveal 1) (grind))
    ("21" (reveal 1) (grind)) ("22" (reveal 1) (grind))
```

```
      ("23" (reveal 1) (grind)) ("24" (reveal 1) (grind))
      ("25" (reveal 1) (grind)) ("26" (reveal 1) (grind))
      ("27" (reveal 1) (grind)) ("28" (reveal 1) (grind))
      ("29" (reveal 1) (grind)) ("30" (reveal 1) (grind))
      ("31" (reveal 1) (grind)) ("32" (reveal 1) (grind))
      ("33" (reveal 1) (grind)) ("34" (reveal 1) (grind))
      ("35" (reveal 1) (grind)) ("36" (reveal 1) (grind))
      ("37" (reveal 1) (grind)) ("38" (reveal 1) (grind))
      ("39" (reveal 1) (grind)) ("40" (reveal 1) (grind)))))))))
```

lemmas.change_to_interrupt_rec_TCC1: proved - complete [shostak](17.93 s)

```
(""
 (skosimp)
 (lemma "recursive_monitor_apps")
 (inst -1 "st!1" "n!1")
 (grind))
```

lemmas.change_to_interrupt_rec_TCC2: proved - complete [shostak](15.62 s)

```
("" (skosimp) (typepred "st!1`sp`app_seq") (grind))
```

lemmas.change_to_interrupt_rec: proved - complete [shostak](3480.43 s)

```
(""
 (lemma "change_to_interrupt")
 (induct "n")
 (("1"
   (skosimp)
   (expand "recursive_monitor")
   (inst -1 "st!1" "st!1`sp`app_seq(0)" "app!1")
   (("1" (split) (("1" (propax)) ("2" (flatten))))
    ("2" (hide 3 4) (grind))))
  ("2"
   (skosimp)
   (skosimp)
   (expand "recursive_monitor" (2 3))
   (inst -1 "st!1" "app!1")
   (inst -2 "recursive_monitor(st!1, st!1`sp`app_seq, j!1)"
    "st!1`sp`app_seq(1 + j!1)" "app!1")
   (("1" (grind))
    ("2"
     (hide -1 2 3 4)
     (lemma "recursive_monitor_apps")
     (inst -1 "st!1" "j!1")
     (typepred "app!1")
     (reveal 1)
     (grind))
    ("3"
     (hide -1 2 3 4)
     (lemma "recursive_monitor_apps")
     (inst -1 "st!1" "j!1")
     (reveal 1)
     (grind))
    ("4" (hide -1 3 4) (grind)) ("5" (hide -1 3 4) (grind))))
  ("3" (hide -1 2) (skosimp*) (hide 2) (grind))
```

```
  ("4"
   (hide -1 2)
   (skosimp*)
   (hide 2)
   (lemma "recursive_monitor_apps")
   (inst -1 "st!1" "n!2")
   (grind))
  ("5" (hide -1 2) (skosimp*) (grind))
  ("6"
   (hide -1 2)
   (skosimp*)
   (lemma "recursive_monitor_apps")
   (inst -1 "st!1" "n!2")
   (grind))))
```

lemmas.m_speclvl_const: proved - complete [shostak](26.16 s)

```
("" (skosimp) (grind))
```

lemmas.rm_speclvl_const_TCC1: proved - complete [shostak](15.17 s)

```
("" (skosimp) (typepred "st!1`sp`app_seq") (grind))
```

lemmas.rm_speclvl_const: proved - complete [shostak](2628.11 s)

```
(""
 (induct "n")
 (("1" (skosimp) (grind))
  ("2"
   (skosimp)
   (skosimp 1)
   (lemma "m_speclvl_const")
   (expand "recursive_monitor" 2)
   (inst -2 "st!1")
   (inst -1 "recursive_monitor(st!1, st!1`sp`app_seq, j!1)"
    "st!1`sp`app_seq(1 + j!1)")
   (("1" (split -2) (("1" (hide -2 2) (grind)) ("2" (grind))))
    ("2"
     (hide -1 3)
     (lemma "recursive_monitor_apps")
     (inst -1 "st!1" "j!1")
     (grind))
    ("3" (hide 3) (grind)) ("4" (hide 3) (grind))))
  ("3" (hide 2) (grind))))
```

lemmas.interrupt_st_TCC1: proved - complete [shostak](36.66 s)

```
(""
 (grind)
 (lemma "recursive_monitor_apps")
 (inst -1 "s!1`tr(c!1)" "card[app_spec](apps(sp(s!1))) - 1")
 (grind))
```

lemmas.interrupt_st: proved - complete [shostak](121.48 s)

```
("" 
 (skosimp)
 (typepred "s!1`tr")
 (hide (-2 -3 -4))
 (skosimp 1)
 (inst -1 "c!1")
 (skosimp -2)
 (expand "system_monitor")
 (lift-if)
 (ground)
 (("1" (hide (-1 -2 2 3 4)) (inst 1 "app!2"))
  ("2"
   (expand "next_config")
   (lift-if)
   (ground)
   (("1"
     (lemma "change_to_interrupt_rec")
     (inst -1
      "s!1`tr(c!1)
                                  WITH [`reconf_st
                                          := LAMBDA
                                             (app: (s!1`tr(c!1)`sp`apps)):
                                             IF s!1`tr(c!1)`reconf_st(app)
                                                /=
                                                normal
                                             THEN halting
                                             ELSE exec_halting
                                             ENDIF]"
      "app!1" "card(s!1`tr(c!1)`sp`apps) - 1")
     (("1" (grind))
      ("2"
       (hide (-1 -2 -3 2 3 4 5))
       (typepred "s!1`tr(c!1)")
       (lemma "nonempty_apps")
       (inst -1 "s!1`tr(c!1)"))))
    ("2" (hide (-1 -2 1 3 4 5 6)) (inst 1 "app!2"))
    ("3" (hide (-1 1 2 4 5 6 7)) (inst 1 "app!2"))))
  ("3"
   (expand "next_config")
   (lift-if)
   (ground)
   (hide (-1 -3 -4))
   (lemma "change_to_interrupt_rec")
   (inst -1
    "s!1`tr(c!1)
                                WITH [`reconf_st
                                        := LAMBDA
                                           (app: (s!1`tr(c!1)`sp`apps)):
                                           IF s!1`tr(c!1)`reconf_st(app)
                                              /=
                                              normal
                                           THEN halting
                                           ELSE exec_halting
                                           ENDIF]"
    "app!1" "card(s!1`tr(c!1)`sp`apps) - 1")
   (("1" (ground) (lift-if) (ground))
    ("2"
     (hide (-1 2 3 4))
```

```
     (lemma "nonempty_apps")
     (typepred "s!1`tr(c!1)")
     (inst -2 "s!1`tr(c!1)")))))
```

```
lemmas.reconf_halt_TCC1: proved - complete [shostak](18.63 s)

("" (grind))
```

```
lemmas.reconf_halt: proved - complete [shostak](22.03 s)

(""
 (skosimp)
 (typepred "r!1")
 (expand "get_reconfigs")
 (flatten)
 (typepred "s!1`tr")
 (hide (-2 -3 -4))
 (lemma "interrupt_st")
 (inst -1 "s!1" "r!1`start_c")
 (skosimp)
 (ground)
 (("1" (hide (-2 -3 -4 -5 -6)) (inst -1 "app!1") (ground))
  ("2"
   (inst -1 "r!1`start_c")
   (hide (-1 -2 -4 -5 2 3 4))
   (expand "reconfig_start?")
   (propax))))
```

```
lemmas.int_halt_st_TCC1: proved - complete [shostak](11.21 s)

("" (skosimp) (grind))
```

```
lemmas.int_halt_st_TCC2: proved - complete [shostak](30.08 s)

("" (grind))
```

```
lemmas.int_halt_st: proved - complete [shostak](101.15 s)

(""
 (skosimp)
 (skosimp)
 (lemma "rm_speclvl_const")
 (typepred "s!1`tr")
 (inst -1 "r!1`start_c")
 (hide (-2 -3 -4))
 (expand "system_monitor")
 (lift-if)
 (ground)
 (("1"
   (hide (-1 -2 -3 2))
   (typepred "r!1")
   (expand "get_reconfigs")
   (flatten)
   (hide (-1 -3 4))
```

```
   (hide (-2))
   (expand "reconfig_start?")
   (skosimp)
   (inst 1 "app!2"))
  ("2"
   (inst -2 "next_config(s!1`tr(r!1`start_c))"
    "card(s!1`tr(r!1`start_c)`sp`apps) - 1")
   (("1"
     (split)
     (("1" (hide (-2 1 2)) (grind)) ("2" (hide 1) (grind))))
    ("2"
     (hide (-1 2 3))
     (lemma "nonempty_apps")
     (inst -1 "s!1`tr(r!1`start_c)"))))
  ("3"
   (inst -3 "next_config(s!1`tr(r!1`start_c))"
    "card(s!1`tr(r!1`start_c)`sp`apps) - 1")
   (("1" (split) (("1" (hide (-2 -3 1)) (grind)) ("2" (grind))))
    ("2"
     (hide (-1 -2 2))
     (lemma "nonempty_apps")
     (inst -1 "s!1`tr(r!1`start_c)"))))))
```

lemmas.int_halt_len: proved - complete [shostak](12.66 s)

```
("" 
 (skosimp)
 (typepred "r!1")
 (expand "get_reconfigs")
 (flatten)
 (case "r!1`end_c-r!1`start_c > 1")
 (("1"
   (inst -5 "r!1`start_c+1")
   (skosimp -6)
   (hide -3 -4)
   (split)
   (("1" (flatten) (inst -2 "app!1") (grind)) ("2" (grind))
    ("3" (grind))))
  ("2" (hide -2 -3 -4 -5) (grind))))
```

lemmas.halt_st_TCC1: proved - complete [shostak](4.48 s)

```
("" (grind))
```

lemmas.halt_st_TCC2: proved - complete [shostak](11.84 s)

```
("" (grind))
```

lemmas.halt_st: proved - complete [shostak](282.12 s)

```
("" 
 (skosimp)
 (skosimp)
 (typepred "s!1`tr")
 (skosimp)
```

```
(inst -1 "c!1")
(hide (-2 -3 -4))
(expand "system_monitor")
(lift-if)
(split)
(("1"
  (hide -2)
  (flatten)
  (lemma "change_to_interrupt_rec")
  (lemma "nonempty_apps")
  (inst -1 "s!1`tr(c!1)")
  (inst -2
   "s!1`tr(c!1)
                              WITH [`reconf_st
                                      := LAMBDA
                                         (app: (s!1`tr(c!1)`sp`apps)):
                                         IF s!1`tr(c!1)`sp`SCRAM_info`configs
                                            (s!1`tr(c!1)`sp`choose
                                             (s!1`tr(c!1)`svclvl,
                                              s!1`env(c!1 * cycle_time)))
                                            (app)
                                            /=
                                            s!1`tr(c!1)`sp`SCRAM_info`configs
                                            (s!1`tr(c!1)`svclvl)(app)
                                         THEN prepping
                                         ELSE normal
                                         ENDIF,
                                    `app_svclvls
                                      := LAMBDA
                                         (app: (s!1`tr(c!1)`sp`apps)):
                                         s!1`tr(c!1)`sp`SCRAM_info`configs
                                         (s!1`tr(c!1)`sp`choose
                                          (s!1`tr(c!1)`svclvl,
                                           s!1`env(c!1 * cycle_time)))
                                         (app),
                                    `app_last_svcs
                                      := LAMBDA
                                         (app: (s!1`tr(c!1)`sp`apps)):
                                         s!1`tr(c!1)`app_svclvls(app),
                                    `last_svc := s!1`tr(c!1)`svclvl,
                                    `svclvl
                                      := s!1`tr(c!1)`sp`choose
                                         (s!1`tr(c!1)`svclvl,
                                          s!1`env(c!1 * cycle_time))]"
   "app!2" "card(s!1`tr(c!1)`sp`apps) - 1")
  (("1" (ground) (lift-if) (ground))
   ("2"
    (hide (-1 -2 -3 2 3 4 5 6))
    (typepred "app!2")
    (typepred "s!1`tr(c!1)")
    (grind))
   ("3"
    (hide (-1 -2 -3 2 3 4 5 6))
    (skosimp)
    (typepred "s!1`env(c!1*cycle_time)")
    (inst -1 "e!1")
    (grind))
   ("4" (hide (-1 -2 -3 2 3 4 5 6)) (grind))
   ("5" (hide (-1 -2 -3 2 3 4 5 6)) (grind))
```

```
    ("6"
     (hide (-1 -2 -3 2 3 4 5 6))
     (skosimp)
     (skosimp)
     (typepred "s!1`env(c!1*cycle_time)")
     (inst -1 "e!1")
     (grind))))
  ("2"
   (flatten)
   (split)
   (("1"
     (hide (-1 2 3 4 5))
     (inst 1 "app!1")
     (typepred "app!1")
     (typepred "s!1`tr(c!1)")
     (grind))
    ("2" (hide (-2 -3 1 2 3)) (grind))))))
```

lemmas.reconf_prep_TCC1: proved - complete [shostak](24.49 s)

```
("" (grind))
```

lemmas.reconf_prep_TCC2: proved - complete [shostak](19.50 s)

```
("" (skosimp) (grind))
```

lemmas.reconf_prep_TCC3: proved - complete [shostak](44.51 s)

```
("" (grind))
```

lemmas.reconf_prep_TCC4: proved - complete [shostak](45.27 s)

```
("" (grind))
```

lemmas.reconf_prep: proved - complete [shostak](5658.24 s)

```
(""
 (skosimp)
 (skosimp)
 (typepred "r!1")
 (expand "get_reconfigs")
 (lemma "change_to_interrupt_rec")
 (lemma "rm_speclvl_const")
 (typepred "s!1`tr")
 (lemma "nonempty_apps")
 (flatten)
 (hide (-3 -4 -5 -11))
 (expand "reconfig_start?")
 (hide -1)
 (typepred "s!1`env(r!1`start_c+1)")
 (lemma "int_halt_st")
 (lemma "halt_st")
 (expand "system_monitor")
 (hide (-1 -3))
```

```
(hide (-3 -6))
(hide (-3 -4))
(inst -2 "r!1`start_c+1")
(lift-if)
(split)
(("1"
  (flatten)
  (lemma "reconf_halt")
  (inst -1 "s!1" "r!1" "app!1")
  (inst -4 "s!1" "r!1" "app!1")
  (lemma "change_to_interrupt_rec")
  (inst -1
   "s!1`tr(r!1`start_c + 1)
                                WITH [`reconf_st
                                        := LAMBDA
                                           (app:
                                            (s!1`tr(r!1`start_c + 1)`sp`apps)):
                                           IF s!1`tr
                                              (r!1`start_c
                                               +
                                               1)`sp`SCRAM_info`configs
                                              (s!1`tr(r!1`start_c + 1)`sp`choose
                                               (s!1`tr(r!1`start_c + 1)`svclvl,
                                                s!1`env
                                                ((r!1`start_c + 1)
                                                 *
                                                 cycle_time)))
                                              (app)
                                              /=
                                              s!1`tr
                                              (r!1`start_c
                                               +
                                               1)`sp`SCRAM_info`configs
                                              (s!1`tr(r!1`start_c + 1)`svclvl)
                                              (app)
                                           THEN prepping
                                           ELSE normal
                                           ENDIF,
                                       `app_svclvls
                                         := LAMBDA
                                           (app:
                                            (s!1`tr(r!1`start_c + 1)`sp`apps)):
                                           s!1`tr
                                           (r!1`start_c
                                            +
                                            1)`sp`SCRAM_info`configs
                                           (s!1`tr(r!1`start_c + 1)`sp`choose
                                          (s!1`tr(r!1`start_c + 1)`svclvl, s!1`env
                                             ((r!1`start_c + 1) * cycle_time)))
                                           (app),
                                       `app_last_svcs
                                         := LAMBDA
                                           (app:
                                            (s!1`tr(r!1`start_c + 1)`sp`apps)):
                                           s!1`tr(r!1`start_c + 1)`app_svclvls
                                           (app),
                                       `last_svc
                                         := s!1`tr(r!1`start_c + 1)`svclvl,
                                       `svclvl
```

```
                                        := s!1`tr(r!1`start_c + 1)`sp`choose
                                           (s!1`tr(r!1`start_c + 1)`svclvl,
                                            s!1`env
                                            ((r!1`start_c + 1) * cycle_time))]"
 "app!1" "card(s!1`tr(r!1`start_c + 1)`sp`apps) - 1")
(("1"
  (split -1)
  (("1" (hide -2 -3 -4 -5 -6 -7 1 2 3 4) (grind))
   ("2"
    (hide -2 -3 -6 -7 1)
    (reveal -10)
    (inst -1
     "s!1`tr(r!1`start_c + 1)
                                  WITH [`reconf_st
                                          := LAMBDA
                                             (app:
                                            (s!1`tr(r!1`start_c + 1)`sp`apps)):
                                             IF s!1`tr
                                                (r!1`start_c
                                                 +
                                                 1)`sp`SCRAM_info`configs
                                              (s!1`tr(r!1`start_c + 1)`sp`choose
                                                (s!1`tr(r!1`start_c + 1)`svclvl,
                                                   s!1`env
                                                   ((r!1`start_c + 1)
                                                    *
                                                    cycle_time)))
                                                (app)
                                                /=
                                                s!1`tr
                                                (r!1`start_c
                                                 +
                                                 1)`sp`SCRAM_info`configs
                                                (s!1`tr(r!1`start_c + 1)`svclvl)
                                                 (app)
                                             THEN prepping
                                             ELSE normal
                                             ENDIF,
                                        `app_svclvls
                                          := LAMBDA
                                             (app:
                                            (s!1`tr(r!1`start_c + 1)`sp`apps)):
                                             s!1`tr
                                             (r!1`start_c
                                              + 1)`sp`SCRAM_info`configs
                                             (s!1`tr(r!1`start_c + 1)`sp`choose
                                              (s!1`tr(r!1`start_c + 1)`svclvl,
                                               s!1`env
                                              ((r!1`start_c + 1) * cycle_time)))
                                             (app),
                                        `app_last_svcs
                                          := LAMBDA
                                             (app:
                                            (s!1`tr(r!1`start_c + 1)`sp`apps)):
                                            s!1`tr(r!1`start_c + 1)`app_svclvls
                                             (app),
                                        `last_svc
                                          := s!1`tr(r!1`start_c + 1)`svclvl,
                                        `svclvl
```

```
                                                := s!1`tr(r!1`start_c + 1)`sp`choose
                                                   (s!1`tr(r!1`start_c + 1)`svclvl,
                                                    s!1`env
                                                   ((r!1`start_c + 1) * cycle_time))]"
      "card(s!1`tr(r!1`start_c + 1)`sp`apps) - 1")
     (("1"
       (split -1)
       (("1" (hide -2 -3 -4 1 2 3) (grind))
        ("2"
         (typepred "s!1`tr(r!1`start_c+1)")
         (typepred "s!1`tr(r!1`start_c+2)")
         (typepred "s!1`tr(r!1`start_c)")
         (grind))))
      ("2"
       (hide -1 -2 -3 2 3 4)
       (lemma "nonempty_apps")
       (inst -1 "s!1`tr(1 + r!1`start_c)"))
      ("3"
       (hide -1 -2 -3 2 3 4)
       (skosimp)
       (typepred "s!1`env(r!1`start_c * cycle_time + cycle_time)")
       (inst -1 "e!1")
       (grind))
      ("4" (hide -1 -2 -3 2 3 4) (grind))
      ("5" (hide -1 -2 -3 2 3 4) (grind))
      ("6"
       (hide -1 -2 -3 2 3 4)
       (typepred "s!1`env(r!1`start_c * cycle_time + cycle_time)")
       (skosimp*)
       (inst -1 "e!1")
       (grind))))
    ("3" (hide -2 -3 -5 -6 -7 1 2 3) (grind))))
  ("2"
   (hide -1 -2 -3 -4 -5 -6 2 3 4 5)
   (lemma "nonempty_apps")
   (inst -1 "s!1`tr(1 + r!1`start_c)"))
  ("3" (hide -1 -2 -3 -4 -5 -6 2 3 4 5) (grind))
  ("4"
   (hide -1 -2 -3 -4 -5 -6 2 3 4 5)
   (skosimp)
   (typepred "s!1`env(r!1`start_c * cycle_time + cycle_time)")
   (inst -1 "e!1")
   (grind))
  ("5" (hide -1 -2 -3 -4 -5 -6 2 3 4 5) (grind))
  ("6" (hide -1 -2 -3 -4 -5 -6 2 3 4 5) (grind))
  ("7"
   (hide -1 -2 -3 -4 -5 -6 2 3 4 5)
   (skosimp)
   (skosimp)
   (typepred "s!1`env(r!1`start_c * cycle_time + cycle_time)")
   (inst -1 "e!1")
   (grind))))
("2"
 (flatten)
 (hide -1 -2 2 3 4)
 (split)
 (("1"
   (reveal -7 -14)
   (skosimp -1)
```

```
 (reveal -3)
 (inst -1 "r!1`start_c")
 (lift-if)
 (split)
 (("1"
   (flatten)
   (hide -2)
   (skosimp)
   (inst 1 "app!2")
   (hide -3 2)
   (grind))
  ("2"
   (flatten)
   (hide 1)
   (expand "next_config")
   (lift-if)
   (split)
   (("1"
     (flatten)
     (hide -1 -4 -5 -6)
     (lemma "change_to_interrupt_rec")
     (reveal -3)
     (inst -2
      "s!1`tr(r!1`start_c)
                                  WITH [`reconf_st
                                          := LAMBDA
                                        (app: (s!1`tr(r!1`start_c)`sp`apps)):
                                        IF s!1`tr(r!1`start_c)`reconf_st(app)
                                                /=
                                                normal
                                             THEN halting
                                             ELSE exec_halting
                                             ENDIF]"
      "app!2" "card(s!1`tr(r!1`start_c)`sp`apps) - 1")
     (("1"
       (split)
       (("1" (hide -2 -3 -4 1) (grind))
        ("2"
         (inst 1 "app!2")
         (("1" (grind)) ("2" (hide -1 -2 -3 -4) (grind))))
        ("3"
         (hide 1)
         (lemma "int_halt_len")
         (inst -1 "s!1" "r!1")
         (split)
         (("1" (hide -2 -4 -5) (grind)) ("2" (grind))))))
      ("2"
       (hide -1 -2 -3 2)
       (lemma "nonempty_apps")
       (inst -1 "s!1`tr(r!1`start_c)"))
      ("3" (hide -1 -2 -3 2) (grind))))
    ("2"
     (flatten)
     (hide -1 -3 -4 -5 2)
     (inst 1 "app!2")
     (grind))))))
("2"
 (lemma "int_halt_len")
 (inst -1 "s!1" "r!1")
```

```
     (split)
     (("1" (hide -2) (grind))
      ("2" (skosimp) (inst 1 "app!2") (grind))))))))


lemmas.prep_st_TCC1: proved - complete [shostak](18.76 s)

("" (grind))


lemmas.prep_st_TCC2: proved - complete [shostak](25.67 s)

("" (grind))


lemmas.prep_st: proved - complete [shostak](633.30 s)

(""
 (skosimp)
 (skosimp)
 (typepred "s!1`tr")
 (hide -2 -3 -4)
 (inst -1 "r!1`start_c+2")
 (lemma "reconf_prep")
 (typepred "r!1")
 (expand "get_reconfigs")
 (flatten)
 (inst -4 "r!1`start_c+2")
 (hide -1 -2 -3)
 (split -1)
 (("1"
   (hide -1)
   (lemma "change_to_interrupt_rec")
   (expand "system_monitor")
   (lift-if)
   (hide -2)
   (hide -2 -4 -5 -6 1 2 3)
   (reveal -1 2 3)
   (reveal -2)
   (split -1)
   (("1"
     (flatten)
     (lemma "reconf_prep")
     (hide -3)
     (inst -1 "s!1" "r!1")
     (split)
     (("1"
       (skosimp -2)
       (inst -1 "app!2")
       (("1" (hide -4) (hide 1) (grind))
        ("2"
         (hide -3 2)
         (typepred "app!2")
         (typepred "s!1`tr(2+r!1`start_c)")
         (grind))))
      ("2" (hide -1 -3 -4 2 3 4) (grind))))
    ("2"
     (flatten)
     (hide 1)
```

```
(expand "next_config")
(lift-if)
(split)
(("1"
  (flatten)
  (hide -2 -4)
  (reveal -9)
  (inst -1 "r!1`start_c+2")
  (split)
  (("1"
    (expand "reconfig_end?")
    (flatten)
    (skosimp -1)
    (inst 2 "app!2")
    (hide 2)
    (typepred "app!2")
    (typepred "s!1`tr(2 + r!1`start_c)")
    (grind))
   ("2" (assert)) ("3" (assert))))
 ("2"
  (flatten)
  (split)
  (("1"
    (flatten)
    (hide -2 -4 1)
    (skosimp)
    (lemma "reconf_prep")
    (inst -1 "s!1" "r!1")
    (split)
    (("1"
      (inst -1 "app!2")
      (("1" (grind))
       ("2"
        (typepred "app!2")
        (typepred "s!1`tr(2 + r!1`start_c)")
        (grind))))
     ("2" (assert))))
   ("2"
    (flatten)
    (split)
    (("1"
      (flatten)
      (inst -4
       "s!1`tr(2 + r!1`start_c)
                                    WITH [`reconf_st
                                            := LAMBDA
                                               (app:
                                              (s!1`tr(2 + r!1`start_c)`sp`apps)):
                                            IF (s!1`tr(2 + r!1`start_c)`reconf_st
                                                      (app)
                                                      =
                                                      prepping)
                                               THEN training
                                               ELSE normal
                                               ENDIF]"
       "app!1" "card(s!1`tr(2 + r!1`start_c)`sp`apps) - 1")
      (("1"
        (split)
        (("1" (hide -2 -3 -4 -5 1 2 3 4) (grind))
```

```
              ("2" (hide -2 1 2) (grind)) ("3" (hide -2 1 2) (grind))))
            ("2"
             (hide -1 -2 -3 -4 2 3 4 5)
             (lemma "nonempty_apps")
             (inst -1 "s!1`tr(2 + r!1`start_c)"))
            ("3" (hide -1 -2 -3 -4 2 3 4 5) (grind))))
          ("2"
           (flatten)
           (inst 1 "app!1")
           (("1" (hide -1 -3 2 3) (grind))
            ("2" (hide -1 -2 -3 -4 2 3) (grind))))))))))))
  ("2" (hide -1 -2 -4 2 3) (grind)) ("3" (hide -1 -2 -4 2 3) (grind))))
```

lemmas.int_prep_len_TCC1: proved - complete [shostak](22.77 s)

```
("" (grind))
```

lemmas.int_prep_len: proved - complete [shostak](32.46 s)

```
(""
 (skosimp)
 (typepred "r!1")
 (expand "get_reconfigs")
 (flatten)
 (inst -4 "r!1`start_c+2")
 (case "r!1`end_c - r!1`start_c > 2")
 (("1"
   (split)
   (("1"
     (hide (-1 -3 -4 -5 -6 -7 1))
     (grind)
     (reveal -1)
     (grind)
     (reveal -4)
     (reveal -4 -5)
     (reveal -6)
     (hide -2 -5)
     (skosimp -2)
     (inst -1 "r!1`start_c+2")
     (reveal -1)
     (grind))
    ("2" (grind)) ("3" (grind))))
  ("2" (hide (-1 -2 -3 -4 -6)) (grind))))
```

lemmas.reconf_train_TCC1: proved - complete [shostak](26.83 s)

```
("" (grind))
```

lemmas.reconf_train_TCC2: proved - complete [shostak](29.13 s)

```
("" (grind))
```

lemmas.reconf_train: proved - complete [shostak](1318.67 s)

```
("" 
 (skosimp)
 (lemma "prep_st")
 (lemma "reconf_prep")
 (skosimp)
 (inst -1 "s!1" "r!1")
 (inst -2 "s!1" "r!1")
 (typepred "s!1`tr")
 (inst -1 "r!1`start_c+2")
 (hide (-2 -3 -4))
 (split -2)
 (("1"
   (hide -3)
   (expand "system_monitor")
   (lift-if)
   (split -2)
   (("1"
     (flatten)
     (hide -2)
     (hide 1 2)
     (skosimp -1)
     (inst -2 "app!2")
     (("1"
       (split)
       (("1" (flatten) (hide 1) (grind))
        ("2" (flatten) (hide -1) (grind)) ("3" (grind))))
      ("2" (grind))))
    ("2"
     (flatten)
     (lemma "rm_speclvl_const")
     (inst -1 "next_config(s!1`tr(2 + r!1`start_c))"
      "card(s!1`tr(2 + r!1`start_c)`sp`apps) - 1")
     (("1"
       (split -1)
       (("1" (hide -2 -3 -4 1 2) (grind))
        ("2"
         (hide -4 1)
         (inst -3 "app!1")
         (split -3)
         (("1" (flatten) (grind)) ("2" (flatten) (grind))
          ("3" (reveal -2) (typepred "r!1") (grind))))))
      ("2"
       (hide -1 -2 -3 2 3)
       (lemma "nonempty_apps")
       (inst -1 "s!1`tr(2 + r!1`start_c)")))))))
  ("2" (hide (-1 -2 2)) (assert))))


lemmas.train_st_TCC1: proved - complete [shostak](22.67 s)

("" (grind))


lemmas.train_st: proved - complete [shostak](219.00 s)

("" 
 (skosimp)
 (skosimp)
 (lemma "reconf_prep")
```

```
(inst -1 "s!1" "r!1")
(split)
(("1"
  (typepred "s!1`tr")
  (inst -1 "r!1`start_c+2")
  (hide -2 -3 -4)
  (expand "system_monitor")
  (lift-if)
  (split -1)
  (("1"
    (flatten)
    (lemma "int_prep_len")
    (hide -3)
    (inst -1 "s!1" "r!1")
    (split -1)
    (("1" (hide -2 -3 1 2 3 4) (grind))
     ("2"
      (skosimp -1)
      (inst -2 "app!2")
      (("1" (hide -3 1 2 3 4 5) (grind))
       ("2"
        (hide -1 -2 2 3 4 5 6)
        (grind)
        (typepred "app!2")
        (typepred "s!1`tr(2 + r!1`start_c)")
        (grind))))
     ("3" (hide -1 -2 2 3 4 5) (grind))))
   ("2"
    (flatten)
    (hide 1)
    (lemma "change_to_interrupt_rec")
    (inst -1 "next_config(s!1`tr(2 + r!1`start_c))" "app!1"
     "card(s!1`tr(2 + r!1`start_c)`sp`apps) - 1")
    (("1"
      (split -1)
      (("1" (hide -2 -3 -4 1 2 3) (grind))
       ("2"
        (hide -1)
        (lemma "change_to_interrupt_rec")
        (expand "next_config")
        (lift-if)
        (split -2)
        (("1"
          (flatten)
          (skosimp -1)
          (hide -2 -3 -5 1 2 3)
          (lemma "int_prep_len")
          (inst -1 "s!1" "r!1")
          (reveal -3)
          (hide -4)
          (grind))
         ("2"
          (flatten)
          (split -1)
          (("1"
            (flatten)
            (hide -2 -3 -5 1 2 3 4)
            (skosimp)
            (inst -2 "app!2")
```

```
            (("1" (grind)) ("2" (grind))))
           ("2"
            (flatten)
            (hide 1 2)
            (split -1)
            (("1"
              (flatten)
              (inst -3
               "s!1`tr(2 + r!1`start_c)
                                        WITH [`reconf_st
                                                := LAMBDA
                                                   (app:
                                              (s!1`tr(2 + r!1`start_c)`sp`apps)):
                                               IF (s!1`tr(2 + r!1`start_c)`reconf_st
                                                             (app)
                                                             =
                                                             prepping)
                                                          THEN training
                                                          ELSE normal
                                                          ENDIF]"
               "app!1" "card(s!1`tr(2 + r!1`start_c)`sp`apps) - 1")
              (("1"
                (split -3)
                (("1" (hide -2 -3 -4 -5 1 2 3) (grind))
                 ("2" (hide -2 -4) (grind))
                 ("3" (hide -2 -4 -5) (grind))))
               ("2"
                (hide -1 -2 -3 -4 2 3 4)
                (lemma "nonempty_apps")
                (inst -1 "s!1`tr(2 + r!1`start_c)"))
               ("3" (hide -1 -2 -3 -4 2 3 4) (grind))))
             ("2"
              (flatten)
              (hide 1 -3 -4)
              (inst -2
               "s!1`tr(2 + r!1`start_c)
                                        WITH [`reconf_st
                                                := LAMBDA
                                                   (app:
                                              (s!1`tr(2 + r!1`start_c)`sp`apps)):
                                                          normal]"
               "app!1" "card(s!1`tr(2 + r!1`start_c)`sp`apps) - 1")
              (("1"
                (split -2)
                (("1" (hide -2 1 2 3) (grind)) ("2" (grind))
                 ("3" (grind))))
               ("2"
                (hide -1 2 3 4)
                (lemma "nonempty_apps")
                (inst -1 "s!1`tr(2 + r!1`start_c)"))
               ("3" (hide -1 2 3 4) (grind)))))))))))
      ("3" (hide -3 -4 1 2) (grind))))
     ("2"
      (hide -1 -2 -3 2 3 4)
      (lemma "nonempty_apps")
      (inst -1 "s!1`tr(2 + r!1`start_c)"))
     ("3" (hide -1 -2 -3 2 3 4) (grind))))))
("2" (hide 2 3 4) (grind))))
```

```
lemmas.reconf_length: proved - complete [shostak](42.65 s)

(""
 (skosimp)
 (typepred "r!1")
 (expand "get_reconfigs")
 (flatten)
 (lemma "prep_st")
 (inst -1 "s!1" "r!1")
 (split -1)
 (("1"
   (case "r!1`end_c - r!1`start_c > 3")
   (("1"
     (typepred "s!1`tr")
     (hide -2 -3 -4)
     (lemma "train_st")
     (lemma "train_time")
     (expand "reconfig_end?")
     (hide -5)
     (hide -5 1)
     (split)
     (("1"
       (inst -7 "r!1`start_c+3")
       (split)
       (("1"
         (flatten)
         (skosimp 1)
         (inst -3 "s!1" "r!1")
         (hide -1 -2 -4 -6)
         (split -1)
         (("1" (inst 3 "app!1") (inst -1 "app!1") (grind))
          ("2" (grind))))
        ("2" (hide -1 -2 -3 -4 -6) (grind))
        ("3" (hide -1 -2 -3 -4 -6) (grind))))
      ("2"
       (inst -7 "r!1`start_c+3")
       (split -7)
       (("1"
         (flatten)
         (skosimp 1)
         (inst 3 "app!1")
         (hide -1 -2 -4 -6)
         (inst -1 "s!1" "r!1")
         (split -1)
         (("1" (inst -1 "app!1") (grind)) ("2" (grind))))
        ("2" (hide -1 -2 -3 -4 -6) (grind))
        ("3" (hide -1 -2 -3 -4 -6) (grind))))))
    ("2" (hide -1 -2 -3 -4 -5) (grind))))
  ("2" (hide -1 -2 -3 -4) (grind))))


lemmas.invariant_monitor_TCC1: proved - complete [shostak](55.03 s)

("" (skosimp) (hide -1) (grind))


lemmas.invariant_monitor_TCC2: proved - complete [shostak](47.87 s)
```

```
("" 
 (skosimp)
 (hide -1)
 (typepred "st!1`sp`SCRAM_info`configs(st!1`svclvl)")
 (("1" (grind)) ("2" (grind))))


lemmas.invariant_monitor: proved - complete [shostak](3856.24 s)

("" 
 (skosimp)
 (split)
 (("1"
   (typepred "st!1`sp`SCRAM_info`configs(st!1`svclvl)")
   (("1"
     (inst -1 "app!1")
     (expand "inv")
     (skosimp*)
     (typepred "st!1`sp`SCRAM_info`configs(st!1`svclvl)")
     (inst -1 "app!2")
     (inst -3 "app!2")
     (expand "inv")
     (skosimp)
     (inst -3 "m!1")
     (typepred "m!1")
     (typepred "st!1`sp`apps")
     (inst -3 "app!1" "app!2")
     (typepred
      "m!1`inv(app!2`svcmap
                          (st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!2))(m!1))")
     (("1"
       (inst -1 "st!1`st")
       (expand "monitor")
       (lift-if)
       (ground)
       (("1"
         (expand "execute")
         (typepred
          "app!1`execute

(st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1))`f")
         (inst -3
          "app!1`execute
                               (st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1))`f
                                         (st!1`st)")
         (("1"
           (ground)
           (("1"
             (skosimp)
             (inst -2 "st!1`st" "id!1")
             (inst 3 "id!1")
             (grind))
            ("2"
             (skosimp 1)
             (inst -1 "st!1`st" "id!1")
             (inst 4 "id!1")
             (grind))))
          ("2" (hide (-1 -2 -3 -4 -5 -6 -7 -8 2 3)) (grind))))
        ("2"
```

```
          (hide (-1 -3 2 3))
          (expand "halt")
          (typepred
           "app!1`halt(st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1))`f")
          (typepred "app!1`halt")
          (expand "post")
          (inst -1 "st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1)"
           "st!1`st")
          (("1"
            (case "app!2 = app!1")
            (("1"
              (inst -2 "m!1")
              (("1"
                (typepred "m!1`post")
                (ground)
                (("1" (grind))
                 ("2"
                  (hide -4 -5 -6 1 3)
                  (expand "implies")
                  (hide -7)
                  (inst -1
                   "(app!2`svcmap

(st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!2))(m!1))"
                   "app!1`halt(st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1))`f
                                                    (st!1`st)")
                  (("1" (grind))
                   ("2"
                    (hide -2 2)
                    (typepred
                     "app!2`svcmap

(st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!2))")
                    (inst -1 "m!1"))
                   ("3" (hide -1 -2 -3 -4 -5 2) (grind))))))
              ("2" (hide (-2 -3 -4 -6 -7 -8 2)) (grind))))
            ("2"
             (inst -3
              "app!1`halt(st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1))`f
                                          (st!1`st)")
             (("1"
               (ground)
               (("1"
                 (skosimp 1)
                 (inst 4 "id!1")
                 (hide -1)
                 (typepred "id!1")
                 (expand "app_scope")
                 (ground)
                 (("1"
                   (inst 1 "m!1")
                   (hide -2 -8 3)
                   (inst -2 "st!1`st" "id!1")
                   (split -2)
                   (("1" (propax))
                    ("2"
                     (skosimp -1)
                     (hide 1 2 3)
                     (typepred "m!2")
```

```
                    (typepred "st!1`sp`apps")
                    (hide -1 -2)
                    (typepred "app!1")
                    (typepred "app!2")
                    (inst -3 "app!1" "app!2")
                    (split)
                    (("1"
                      (inst 1 "id!1")
                      (expand "app_scope")
                      (split)
                      (("1" (inst 1 "m!2")) ("2" (inst 1 "m!1"))))
                     ("2" (reveal 3) (assert))))))
                   ("2" (inst 1 "m!1"))))
                  ("2"
                   (skosimp 1)
                   (hide -1)
                   (inst -1 "st!1`st" "id!1")
                   (split)
                   (("1" (propax))
                    ("2"
                     (typepred "id!1")
                     (hide 1 2 3 -7)
                     (inst 2 "id!1")
                     (split)
                     (("1" (propax)) ("2" (grind)))))))))
              ("2" (hide (-1 -2 -3 -4 -5 -6 -7 2 3 4)) (grind))))))
          ("2" (hide (-1 -2 -3 -4 -5 -6 -7 2 3)) (grind))))
         ("3"
          (hide (-1 -3 -4 2 3 4))
          (expand "exec_halt")
          (typepred "app!1`exec_halt")
          (typepred
           "app!1`exec_halt

(st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1))`f")
          (inst -3
           "app!1`exec_halt
                                (st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1))`f
                                       (st!1`st)")
          (("1"
            (inst -2 "st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1)"
             "st!1`st")
            (("1"
              (case "app!1 = app!2")
              (("1" (grind))
               ("2"
                (hide -2)
                (ground)
                (("1"
                  (skosimp 1)
                  (inst -2 "st!1`st" "id!1")
                  (inst 4 "id!1")
                  (typepred "id!1")
                  (grind))
                 ("2"
                  (skosimp 1)
                  (inst 5 "id!1")
                  (typepred "id!1")
                  (inst -2 "st!1`st" "id!1")
```

```
          (grind))))))
     ("2" (grind))))
   ("2" (grind))))
("4"
 (hide (-1 -3 -4 2 3 4 5))
 (expand "prep")
 (inst -1
  "app!1`prep(st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1))`f
                                (st!1`st)")
 (("1"
   (typepred "app!1`prep")
   (typepred
    "app!1`prep(st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1))`f")
   (inst -2 "st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1)"
    "st!1`st")
   (("1"
     (expand "trans")
     (flatten)
     (case "app!1=app!2")
     (("1"
       (ground)
       (("1"
         (skosimp 1)
         (inst 3 "id!1")
         (inst -3 "st!1`st" "id!1")
         (grind))
        ("2"
         (skosimp 1)
         (inst 4 "id!1")
         (typepred "id!1")
         (grind))))
      ("2"
       (ground)
       (("1"
         (skosimp 1)
         (inst -2 "st!1`st" "id!1")
         (typepred "id!1")
         (inst 4 "id!1")
         (grind))
        ("2"
         (skosimp 1)
         (inst 5 "id!1")
         (typepred "id!1")
         (grind))))))
    ("2" (grind))))
  ("2" (grind))))
("5"
 (hide (-2 -3 1 3 4 5 6))
 (expand "execute")
 (inst -1
  "app!1`execute
                      (st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1))`f
                              (st!1`st)")
 (("1"
   (ground)
   (("1"
     (skosimp 1)
     (inst 3 "id!1")
     (typepred "id!1")
```

```
              (case "app!1=app!2")
              (("1" (grind))
               ("2"
                (typepred
               "app!1`execute(st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1))`f")
                (inst -1 "st!1`st" "id!1")
                (grind))))
             ("2"
              (skosimp 1)
              (inst 4 "id!1")
              (typepred "id!1")
              (case "app!1=app!2")
              (("1" (grind))
               ("2"
                (typepred
               "app!1`execute(st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1))`f")
                (inst -1 "st!1`st" "id!1")
                (grind))))))
           ("2" (grind))))
         ("6"
          (expand "execute")
          (typepred "app!1`execute")
          (inst -1 "st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1)"
           "st!1`st")
          (("1" (expand "inv") (inst -1 "m!1") (grind)) ("2" (grind))))
         ("7"
          (expand "halt")
          (typepred "app!1`halt")
          (expand "post")
          (inst -1 "st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1)"
           "st!1`st")
          (("1"
            (inst -1 "m!1")
            (typepred "m!1`post")
            (typepred "m!1")
            (hide (-4 -7 -8 2 3))
            (expand "IMPLIES")
            (hide -1)
            (inst -1
             "app!2`svcmap

(st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!2))(m!1)"
             "app!1`halt(st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1))`f
                                    (st!1`st)")
            (("1" (grind))
             ("2"
              (typepred
               "app!2`svcmap

(st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!2))")
              (inst -1 "m!1"))
             ("3" (grind))))
           ("2" (grind))))
         ("8"
          (hide (-1 -4 -5 2 3 4))
          (expand "exec_halt")
          (typepred "app!1`exec_halt")
          (inst -1 "st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1)"
           "st!1`st")
```

```
          (("1"
            (grind)
            (typepred "m!1`post")
            (expand "implies")
            (inst -1
             "app!2`svcmap
                          (st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!2))(m!1)"
             "app!2`exec_halt(st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!2))`f
                           (st!1`st)")
            (("1" (grind))
             ("2"
              (hide -1 -2 -3 -4 -5 -6 2 3)
              (typepred
               "app!2`svcmap
                             (st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!2))")
              (grind))))
           ("2" (grind))))
         ("9"
          (hide (-1 -4 -5 2 3 4 5))
          (expand "prep")
          (typepred "app!1`prep")
          (expand "post")
          (inst -1 "st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1)"
           "st!1`st")
          (("1"
            (flatten)
            (expand "trans")
            (inst -2 "m!1")
            (typepred "m!1`trans")
            (expand "IMPLIES")
            (inst -1
             "app!2`svcmap

(st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!2))(m!1)"
             "app!1`prep(st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1))`f
                                          (st!1`st)")
            (("1" (grind))
             ("2"
              (typepred
               "app!2`svcmap

(st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!2))")
              (inst -1 "m!1"))
             ("3" (grind))))
           ("2" (grind))))
         ("10"
          (hide (-3 -4 1 3 4 5 6))
          (expand "execute")
          (typepred "app!1`execute")
          (expand "inv")
          (inst -1 "st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1)"
           "st!1`st")
          (("1" (inst -1 "m!1") (grind)) ("2" (grind))))))
       ("2"
        (typepred
         "app!2`svcmap
                           (st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!2))")
        (("1" (inst -1 "m!1")) ("2" (propax))))
       ("3" (propax))))
```

```
  ("2" (hide (-1 2)) (grind))))
("2"
 (flatten)
 (expand "inv")
 (skosimp)
 (expand "monitor")
 (lift-if)
 (split)
 (("1"
   (flatten)
   (expand "execute")
   (split)
   (("1"
     (flatten)
     (typepred "app!1`execute")
     (inst -1 "st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1)"
      "st!1`st")
     (("1" (expand "inv") (inst -1 "m!1"))
      ("2"
       (hide -1 -2 2 3)
       (typepred "st!1`sp`SCRAM_info`configs(st!1`svclvl)")
       (("1" (inst -1 "app!1")) ("2" (grind))))
      ("3" (hide -1 -2 2 3) (grind))))
    ("2"
     (flatten)
     (typepred "app!1`execute")
     (inst -1 "st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1)"
      "st!1`st")
     (("1"
       (expand "inv")
       (inst -1 "m!1")
       (split)
       (("1" (flatten)) ("2" (flatten))))
      ("2"
       (hide -1 2 3 4)
       (typepred "st!1`sp`SCRAM_info`configs(st!1`svclvl)")
       (("1" (inst -1 "app!1")) ("2" (grind))))
      ("3" (hide -1 2 3 4) (grind))))))
  ("2"
   (flatten)
   (typepred "st!1`sp`SCRAM_info`configs(st!1`svclvl)")
   (("1"
     (inst -1 "app!1")
     (split)
     (("1" (flatten) (hide -2 1 2) (grind))
      ("2"
       (flatten)
       (split)
       (("1"
         (flatten)
         (expand "halt")
         (typepred "app!1`halt")
         (expand "post")
         (inst -1 "st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1)"
          "st!1`st")
         (("1"
           (inst -1 "m!1")
           (typepred "m!1`post")
           (expand "implies")
```

```
              (inst -1
               "app!1`svcmap
                           (st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1))(m!1)"
               "app!1`halt(st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1))`f
                              (st!1`st)")
              (("1"
                (split -1)
                (("1" (split 1) (("1" (flatten)) ("2" (grind))))
                 ("2" (propax))))
               ("2"
                (hide -1 -2 2 3 4 5)
                (grind)
                (typepred
                 "app!1`svcmap
                              (st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1))")
                (inst -1 "m!1"))
               ("3" (hide -1 -2 -3 2 3 4 5) (grind))))
             ("2" (hide -1 -2 2 3 4 5) (grind))))
            ("2"
             (flatten)
             (typepred
              "app!1`svcmap
                              (st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1))")
             (("1"
               (inst -1 "m!1")
               (split)
               (("1"
                 (flatten)
                 (expand "exec_halt")
                 (typepred "app!1`exec_halt")
                 (inst -1
                  "st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1)"
                  "st!1`st")
                 (("1"
                   (split)
                   (("1"
                     (flatten)
                     (expand "post")
                     (inst -2 "m!1")
                     (typepred "m!1`post")
                     (expand "implies")
                     (inst -1
                      "app!1`svcmap

(st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1))(m!1)"

"app!1`exec_halt(st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1))`f
                                    (st!1`st)")
                     (("1" (grind))
                      ("2" (hide -1 -2 -3 -4 -5 2 3 4 5 6) (grind))))
                    ("2"
                     (flatten)
                     (expand "post")
                     (inst -1 "m!1")
                     (typepred "m!1`post")
                     (expand "implies")
                     (inst -1
                      "app!1`svcmap
```

```
(st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1))(m!1)"

"app!1`exec_halt(st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1))`f
                                  (st!1`st)")
                   (("1" (grind))
                    ("2" (hide -1 -2 -3 -4 2 3 4 5 6 7) (grind))))))
                 ("2" (hide -1 -2 -3 2 3 4 5 6) (grind))))
               ("2"
                (flatten)
                (split)
                (("1"
                  (flatten)
                  (expand "prep")
                  (typepred "app!1`prep")
                  (inst -1
                   "st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1)"
                   "st!1`st")
                  (("1"
                    (flatten)
                    (expand "trans")
                    (inst -2 "m!1")
                    (typepred "m!1`trans")
                    (expand "IMPLIES")
                    (inst -1
                     "app!1`svcmap

(st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1))(m!1)"
                     "app!1`prep(st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1))`f
                                  (st!1`st)")
                     (("1"
                       (split -1)
                       (("1" (split) (("1" (flatten)) ("2" (flatten))))
                        ("2" (propax))))
                      ("2" (hide -1 -2 -3 -4 -5 2 3 4 5 6 7) (grind))))
                    ("2" (hide -1 -2 -3 2 3 4 5 6 7) (grind))))
                  ("2"
                   (flatten)
                   (expand "execute")
                   (typepred "app!1`execute")
                   (inst -1
                    "st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1)"
                    "st!1`st")
                   (("1"
                     (expand "inv")
                     (inst -1 "m!1")
                     (split)
                     (("1" (flatten))
                      ("2"
                       (flatten)
                       (split)
                       (("1" (flatten)) ("2" (flatten))))))
                    ("2" (hide -1 -2 2 3 4 5 6 7 8) (grind)))))))))
             ("2" (propax)))))))))
      ("2" (hide 2 3 4) (grind)))))))))


lemmas.invariant_monitor_rec: proved - complete [shostak](1702.00 s)
```

```
("" 
 (lemma "invariant_monitor")
 (induct "n")
 (("1"
   (skosimp)
   (inst -2 "st!1" "st!1`sp`app_seq(0)")
   (("1" (grind))
    ("2"
     (lemma "nonempty_apps")
     (inst -1 "st!1")
     (hide (-2 3))
     (typepred "st!1`sp`app_seq")
     (hide -2)
     (grind))))
  ("2"
   (skosimp)
   (skosimp)
   (expand "recursive_monitor" 2)
   (inst -1 "st!1")
   (split -1)
   (("1" (assert))
    ("2"
     (hide -2 -3)
     (expand "inv")
     (skosimp)
     (inst -1 "app!1")
     (reveal -3)
     (case "FORALL (st: sys_state, n: nat) : n >= card(st`sp`apps) OR
recursive_monitor(st, st`sp`app_seq, n)`sp = st`sp")
     (("1"
       (inst -1 "st!1" "j!1")
       (split -1)
       (("1" (assert))
        ("2"
         (case "FORALL (st: sys_state, n: nat) : n >= card(st`sp`apps) OR
recursive_monitor(st, st`sp`app_seq, n)`svclvl = st`svclvl")
         (("1"
           (inst -1 "st!1" "j!1")
           (split -1)
           (("1" (assert))
            ("2"
             (inst -3 "recursive_monitor(st!1, st!1`sp`app_seq, j!1)"
              "st!1`sp`app_seq(1 + j!1)")
             (("1"
               (flatten)
               (hide -4)
               (split -3)
               (("1" (hide -4) (grind))
                ("2"
                 (reveal -4)
                 (hide -4 3)
                 (expand "inv")
                 (skosimp)
                 (inst -1 "app!2")
                 (("1"
                   (expand "inv")
                   (skosimp)
                   (inst -1 "m!1")
                   (assert))
```

```
                ("2" (hide 2) (assert))))))
             ("2" (hide -3 3) (assert)) ("3" (hide -3 3) (assert))
             ("4" (hide -3 3) (assert))))))
          ("2"
           (hide -2 -3 3)
           (induct "n")
           (("1" (grind))
            ("2"
             (skosimp)
             (skosimp)
             (expand "recursive_monitor" 2)
             (inst -1 "st!2")
             (grind))
            ("3" (hide 2) (grind))))))))
      ("2"
       (hide -1 -2 3)
       (induct "n")
       (("1" (grind))
        ("2"
         (skosimp)
         (skosimp)
         (expand "recursive_monitor" 2)
         (inst -1 "st!2")
         (grind))
        ("3" (hide 2) (skosimp) (skosimp) (grind))
        ("4" (hide 2) (skosimp) (grind))))
      ("3" (hide -1 -2 3) (grind))))
    ("3" (propax))))
  ("3"
   (hide -1 2)
   (skosimp)
   (typepred "st!1`sp`app_seq")
   (hide -2 -3)
   (grind))))
```

lemmas.cycle_time: proved - complete [shostak](2.07 s)

```
("" 
 (skosimp)
 (typepred "cycle_time")
 (case "cycle_time = 0")
 (("1" (grind))
  ("2"
   (case "c!1 > 1")
   (("1"
     (hide -3)
     (case "c!1*cycle_time > cycle_time")
     (("1" (grind))
      ("2"
       (lemma "lt_times_lt_pos1")
       (inst -1 "1" "cycle_time" "c!1" "cycle_time")
       (("1" (grind)) ("2" (grind))))))
    ("2" (grind))))))
```

lemmas.same_conf_or_pre_TCC1: proved - complete [shostak](27.70 s)

```
("" (grind))
```

```
lemmas.same_conf_or_pre_TCC2: proved - complete [shostak](143.99 s)

("" 
 (grind)
 (("1"
   (typepred "s!1`sp`SCRAM_info`configs(s!1`tr(r!1`end_c)`svclvl)")
   (hide -2 -3 -4 -5 -6 1)
   (grind))
  ("2"
   (typepred "s!1`sp`SCRAM_info`configs(s!1`tr(r!1`end_c)`svclvl)")
   (grind))
  ("3"
   (typepred "s!1`sp`SCRAM_info`configs(s!1`tr(r!1`end_c)`svclvl)")
   (grind))))


lemmas.same_conf_or_pre_TCC3: proved - complete [shostak](40.21 s)

("" 
 (skosimp)
 (hide -1 1 2)
 (skosimp)
 (typepred "app1!1")
 (typepred "s!1`tr(r!1`end_c)")
 (grind))


lemmas.same_conf_or_pre_TCC4: proved - complete [shostak](36.45 s)

("" 
 (skosimp)
 (hide 1 2 3)
 (typepred "app!1")
 (typepred "s!1`tr(r!1`end_c)")
 (grind))


lemmas.same_conf_or_pre: proved - complete [shostak](254.44 s)

("" 
 (skosimp)
 (lemma "reconf_length")
 (inst -1 "s!1" "r!1")
 (typepred "r!1")
 (expand "get_reconfigs")
 (flatten)
 (expand "reconfig_end?")
 (case "r!1`end_c-r!1`start_c=1")
 (("1"
   (lemma "int_halt_st")
   (hide -3 -4 -5 -6 -7 2 3 4)
   (inst -1 "s!1" "r!1" "app!1")
   (grind))
  ("2"
   (case "r!1`end_c-r!1`start_c=2")
   (("1"
     (lemma "reconf_prep")
```

```
 (inst -1 "s!1" "r!1")
 (split -1)
 (("1"
   (inst -1 "app!1")
   (hide -3 -4 -6 -7 1 3)
   (split -3)
   (("1"
     (inst -1 "app!1")
     (split 2)
     (("1" (hide -1 -2 2 3) (grind))
      ("2"
       (split -2)
       (("1" (flatten) (hide 1 2 3 4) (grind))
        ("2" (flatten) (hide -2 -3 1 3) (grind))
        ("3" (hide -2 1 2) (grind))))))
    ("2"
     (skosimp -1)
     (hide -2 1 3)
     (split)
     (("1" (grind)) ("2" (inst 1 "app!2"))))))
  ("2" (hide -1 -3 -4 -5 -6 3 4 5 6) (grind))))
("2"
 (case "r!1`end_c-r!1`start_c=3")
 (("1"
   (hide 1 2)
   (lemma "reconf_prep")
   (lemma "reconf_train")
   (lemma "train_time")
   (inst -2 "s!1" "r!1")
   (inst -3 "s!1" "r!1")
   (split -2)
   (("1"
     (split -3)
     (("1"
       (inst -1 "app!1")
       (inst -2 "app!1")
       (inst -3 "s!1" "r!1`start_c+3" "app!1")
       (split -1)
       (("1"
         (flatten)
         (typepred "s!1`tr")
         (hide -2 -3 -4)
         (hide -6 -10)
         (hide -6)
         (inst -1 "r!1`start_c+2")
         (expand "system_monitor")
         (lift-if)
         (split -1)
         (("1"
           (flatten)
           (hide -2)
           (skosimp -1)
           (reveal -11)
           (inst -1 "app!2")
           (("1" (hide -3 -4 -5 -6 -7 -8 1 2 3 4 5 6) (grind))
            ("2"
             (hide -1 -2 -3 -4 -5 -6 -7 2 3 4 5 6 7)
             (grind)
             (typepred "app!2")
```

```
                  (grind))))
                ("2"
                 (reveal -10)
                 (flatten)
                 (hide 1)
                 (expand "next_config")
                 (lift-if)
                 (split)
                 (("1"
                   (flatten)
                   (hide -2 -3 -4 -5 -6 -8 1 2 3 4 5)
                   (skosimp)
                   (inst -3 "r!1`start_c+2")
                   (split)
                   (("1"
                     (flatten)
                     (inst 2 "app!2")
                     (("1" (assert))
                      ("2"
                       (hide -1 -2 2)
                       (typepred "app!2")
                       (typepred "s!1`tr(2 + r!1`start_c)")
                       (grind))))
                    ("2" (assert)) ("3" (assert))))
                  ("2"
                   (lift-if)
                   (split)
                   (("1"
                     (flatten)
                     (hide 1 5)
                     (hide -2 -4 -5 -6 -7 -8 -9 1 2 3 4)
                     (skosimp)
                     (inst -2 "app!2")
                     (("1" (grind))
                      ("2"
                       (typepred "app!2")
                       (typepred "s!1`tr(2 + r!1`start_c)")
                       (grind))))
                    ("2"
                     (lift-if)
                     (split)
                     (("1"
                       (flatten)
                       (hide -1 6)
                       (hide 1 2)
                       (hide -2 -8)
                       (lemma "change_to_interrupt_rec")
                       (inst -1
                        "s!1`tr(2 + r!1`start_c)
                                                WITH [`reconf_st
                                                        := LAMBDA
                                                           (app:
                                                            (s!1`tr(2 +
r!1`start_c)`sp`apps)):
                                                           IF (s!1`tr(2 +
r!1`start_c)`reconf_st
                                                               (app)
                                                               =
                                                               prepping)
```

```
                                                          THEN training
                                                          ELSE normal
                                                          ENDIF]"
                     "app!1"
                     "card(s!1`tr(2 + r!1`start_c)`sp`apps) - 1")
                    (("1"
                      (split -1)
                      (("1" (hide -2 -3 -4 -5 -6 -7 1 2 3 4) (grind))
                       ("2"
                        (hide -4 -7 1 2 4)
                        (split -4)
                        (("1" (hide -2 -3 -4) (grind))
                         ("2" (hide 2) (reveal 4) (grind))))
                       ("3" (hide -3 -4 -5 -7 1 2 3) (grind))))
                     ("2"
                      (hide -1 -2 -3 -4 -5 -6 2 3 4 5)
                      (lemma "nonempty_apps")
                      (inst -1 "s!1`tr(2 + r!1`start_c)"))
                     ("3" (hide -1 -2 -3 -4 -5 -6 2 3 4 5) (grind))))
                   ("2"
                    (flatten)
                    (inst 1 "app!1")
                    (("1"
                      (hide -1 -2 -4 -5 -6 -7 -8 2 3 4 5 6 7 8)
                      (grind))
                     ("2"
                      (hide -1 -2 -3 -4 -5 -6 -7 -8 2 3 4 5 6 7 8)
                      (grind))))))))))))
          ("2" (hide -3 -5 -6 -7 -8 -9 2 3 4) (grind))
          ("3"
           (hide -2 -3 -5 -6 -7 -9 1 2 3 4)
           (inst -3 "r!1`start_c+2")
           (split)
           (("1" (flatten) (inst 2 "app!1")) ("2" (assert))
            ("3" (assert))))))
        ("2" (assert))))
      ("2" (assert))))
    ("2" (hide -1 -2 -3 -4 4 5 6 7) (grind) (reveal -1) (grind))))))))
```

9. Invariant Lemmas

Proof scripts for file lemmas_inv-fm.pvs:

lemmas_inv.rm_spec_constant_TCC1: proved - complete [shostak](5.59 s)

```
("" (grind))
```

lemmas_inv.rm_spec_constant: proved - complete [shostak](903.55 s)

```
(""
 (induct "n")
 (("1" (grind))
  ("2"
   (skosimp)
```

```
   (skosimp)
   (inst -1 "st!1")
   (split -1)
   (("1" (assert))
    ("2"
     (expand "recursive_monitor" 2)
     (expand "monitor")
     (lift-if)
     (split)
     (("1" (flatten) (grind))
      ("2"
       (flatten)
       (split)
       (("1" (grind))
        ("2"
         (flatten)
         (split)
         (("1" (grind))
          ("2"
           (flatten)
           (split)
           (("1" (grind))
            ("2"
             (flatten)
             (split)
             (("1" (flatten) (grind))
              ("2" (hide 2 3 4 5 6) (flatten) (grind)))))))))))))
  ("3" (hide 2) (skosimp) (grind)) ("4" (hide 2) (grind))))
```

lemmas_inv.change_to_interrupt_rec_app_TCC1: proved - complete [shostak](5.64 s)

```
("" (grind))
```

lemmas_inv.change_to_interrupt_rec_app_TCC2: proved - complete [shostak](365.93 s)

```
(""
 (grind)
 (lemma "recursive_monitor_apps")
 (inst -1 "st!1" "n!1")
 (grind))
```

lemmas_inv.change_to_interrupt_rec_app: proved - complete [shostak](1003.11 s)

```
(""
 (induct "n")
 (("1" (skosimp) (inst 2 "0") (expand "recursive_monitor") (grind))
  ("2"
   (skosimp)
   (skosimp)
   (expand "recursive_monitor" 3)
   (inst -1 "st!1" "app!1")
   (split -1)
   (("1" (assert))
    ("2" (inst 2 "1+j!1") (split 2) (("1" (assert)) ("2" (grind))))
    ("3" (hide 3) (skosimp) (inst 2 "m!1") (assert)))))
```

```
  ("3" (hide 2) (skosimp*) (grind))
  ("4"
   (hide 2)
   (skosimp)
   (lemma "recursive_monitor_apps")
   (inst -1 "st!1" "n!2")
   (assert))
  ("5"
   (hide 2)
   (skosimp*)
   (grind)
   (typepred "st!1`sp`app_seq")
   (grind))))
```

lemmas_inv.monitor_not_equal_nc: proved - complete [shostak](872.74 s)

```
(""
 (skosimp)
 (expand "monitor")
 (lift-if)
 (split 2)
 (("1"
   (flatten)
   (expand "execute")
   (typepred
    "app!1`execute(st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1))`f")
   (("1"
     (inst -1 "st!1`st" "id!1")
     (split -1)
     (("1" (grind)) ("2" (propax))))
    ("2"
     (hide 2)
     (typepred "st!1`sp`SCRAM_info`configs(st!1`svclvl)")
     (("1" (grind)) ("2" (grind))))
    ("3" (hide 2) (grind))))
  ("2"
   (flatten)
   (split 2)
   (("1" (propax))
    ("2"
     (flatten)
     (split 2)
     (("1"
       (flatten)
       (expand "halt")
       (typepred
        "app!1`halt(st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1))`f")
       (("1"
         (inst -1 "st!1`st" "id!1")
         (split -1)
         (("1" (grind)) ("2" (propax))))
        ("2"
         (hide 2)
         (typepred "st!1`sp`SCRAM_info`configs(st!1`svclvl)")
         (("1" (grind)) ("2" (grind))))
        ("3" (grind))))
      ("2"
       (flatten)
```

```
       (split)
       (("1"
         (flatten)
         (expand "exec_halt")
         (typepred
          "app!1`exec_halt
                           (st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1))`f")
         (("1"
           (inst -1 "st!1`st" "id!1")
           (split -1)
           (("1" (grind)) ("2" (propax))))
          ("2"
           (hide 2)
           (typepred "st!1`sp`SCRAM_info`configs(st!1`svclvl)")
           (("1" (grind)) ("2" (grind))))
          ("3" (grind))))
        ("2"
         (flatten)
         (split)
         (("1"
           (flatten)
           (expand "prep")
           (typepred
            "app!1`prep(st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1))`f")
           (("1"
             (inst -1 "st!1`st" "id!1")
             (split -1)
             (("1" (grind)) ("2" (propax))))
            ("2"
             (hide 2)
             (typepred "st!1`sp`SCRAM_info`configs(st!1`svclvl)")
             (("1" (grind)) ("2" (grind))))
            ("3" (grind))))
          ("2"
           (flatten)
           (expand "execute")
           (typepred
            "app!1`execute(st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1))`f")
           (("1"
             (inst -1 "st!1`st" "id!1")
             (split -1)
             (("1" (grind)) ("2" (propax))))
            ("2"
             (hide 3)
             (typepred "st!1`sp`SCRAM_info`configs(st!1`svclvl)")
             (("1" (grind)) ("2" (grind))))
            ("3" (grind)))))))))))))))


lemmas_inv.monitor_inv_equal_TCC1: proved - complete [shostak](4.99 s)

("" (grind))


lemmas_inv.monitor_inv_equal_TCC2: proved - complete [shostak](67.12 s)

(""
 (skosimp*)
 (typepred "st!1`sp`SCRAM_info`configs(st!1`svclvl)")
```

```
 (("1" (grind)) ("2" (grind))))


lemmas_inv.monitor_inv_equal: proved - complete [shostak](1155.83 s)

(""
 (skosimp)
 (expand "inv")
 (skosimp)
 (expand "monitor")
 (lift-if)
 (split)
 (("1"
   (flatten)
   (expand "execute")
   (typepred "app!1`execute")
   (inst -1 "st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1)" "st!1`st")
   (("1" (expand "inv") (inst -1 "m!1") (grind))
    ("2"
     (hide 2)
     (typepred "st!1`sp`SCRAM_info`configs(st!1`svclvl)")
     (("1" (grind)) ("2" (grind))))
    ("3" (hide 2) (grind))))
  ("2"
   (flatten)
   (split)
   (("1" (flatten) (hide 1) (grind))
    ("2"
     (flatten)
     (split)
     (("1"
       (flatten)
       (expand "halt")
       (typepred "app!1`halt")
       (inst -1 "st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1)"
        "st!1`st")
       (("1"
         (expand "post")
         (inst -1 "m!1")
         (typepred "m!1`post")
         (expand "implies")
         (inst -1
          "app!1`svcmap

(st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1))(m!1)"
          "app!1`halt(st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1))`f
                                (st!1`st)")
         (("1" (split -1) (("1" (hide -2) (grind)) ("2" (propax))))
          ("2"
           (hide -1 2)
           (typepred
            "app!1`svcmap
                                (st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1))")
           (("1" (grind))
            ("2"
             (typepred "st!1`sp`SCRAM_info`configs(st!1`svclvl)")
             (("1" (grind)) ("2" (grind))))
            ("3" (grind))))
          ("3"
```

```
            (hide -1 2)
            (typepred "st!1`sp`SCRAM_info`configs(st!1`svclvl)")
            (("1" (grind)) ("2" (grind))))
           ("4" (hide 2) (grind))))
         ("2"
          (hide 2)
          (typepred "st!1`sp`SCRAM_info`configs(st!1`svclvl)")
          (("1" (grind)) ("2" (grind))))
         ("3" (hide 2) (grind))))
       ("2"
        (typepred "st!1`sp`SCRAM_info`configs(st!1`svclvl)")
        (("1"
          (inst -1 "app!1")
          (typepred
           "app!1`svcmap

(st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1))")
          (("1"
            (inst -1 "m!1")
            (flatten)
            (split 2)
            (("1"
              (flatten)
              (expand "exec_halt")
              (typepred "app!1`exec_halt")
              (inst -1 "st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1)"
               "st!1`st")
              (("1"
                (expand "post")
                (inst -1 "m!1")
                (typepred "m!1`post")
                (expand "implies")
                (inst -1
                 "app!1`svcmap

(st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1))(m!1)"
                 "app!1`exec_halt(st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1))`f
                                                  (st!1`st)")
                (("1"
                  (split -1)
                  (("1" (hide -2) (grind)) ("2" (propax))))
                 ("2" (hide -1 2) (grind))))
                ("2" (hide 2) (grind))))
               ("2"
                (flatten)
                (split 2)
                (("1"
                  (expand "prep")
                  (flatten)
                  (typepred "app!1`prep")
                  (inst -1
                   "st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1)"
                   "st!1`st")
                  (("1"
                    (flatten)
                    (hide -1)
                    (expand "trans")
                    (inst -1 "m!1")
                    (typepred "m!1`trans")
```

```
                (expand "implies")
                (inst -1
                 "app!1`svcmap

(st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1))(m!1)"
                 "app!1`prep

(st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1))`f
                                                              (st!1`st)")
                (("1"
                  (split -1)
                  (("1" (hide -2) (grind)) ("2" (propax))))
                 ("2" (hide -1 2) (grind))))
               ("2" (hide 2) (grind))))
             ("2"
              (flatten)
              (expand "execute")
              (typepred "app!1`execute")
              (inst -1
               "st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1)"
               "st!1`st")
              (("1" (expand "inv") (inst -1 "m!1") (grind))
               ("2" (hide 3) (grind))))))))
          ("2" (propax))))
        ("2" (hide 2) (grind)))))))))))


lemmas_inv.inv_not_equal_TCC1: proved - complete [shostak](5.36 s)

("" (grind))


lemmas_inv.inv_not_equal_TCC2: proved - complete [shostak](5.50 s)

("" (grind))


lemmas_inv.inv_not_equal_TCC3: proved - complete [shostak](58.64 s)

(""
 (skosimp*)
 (typepred "st!1`sp`SCRAM_info`configs(st!1`svclvl)")
 (("1" (grind)) ("2" (grind))))


lemmas_inv.inv_not_equal_TCC4: proved - complete [shostak](5.81 s)

("" (grind))


lemmas_inv.inv_not_equal: proved - complete [shostak](6645.93 s)

(""
 (skosimp)
 (expand "recursive_monitor" 3)
 (expand "inv")
 (skosimp)
 (inst -1 "m!1")
 (typepred
```

```
 "m!1`inv
           (app!1`svcmap
                (st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1))(m!1))")
(("1"
  (inst -1 "recursive_monitor(st!1, st!1`sp`app_seq, n!1 - 1)`st")
  (("1"
    (split -1)
    (("1"
      (inst -1
       "monitor(recursive_monitor(st!1, st!1`sp`app_seq, n!1 - 1),
                           st!1`sp`app_seq(n!1))`st")
      (("1"
        (split -1)
        (("1" (propax))
         ("2"
          (hide -1 4)
          (lemma "monitor_not_equal_nc")
          (skosimp)
          (inst -1 "recursive_monitor(st!1, st!1`sp`app_seq, n!1 - 1)"
           "finseq_appl(st!1`sp`app_seq)(n!1)" "id!1")
          (("1"
            (split -1)
            (("1" (propax))
             ("2"
              (typepred "id!1")
              (typepred "m!1")
              (hide 1)
              (typepred "st!1`sp`apps")
              (hide -1 -2)
              (inst -1 "app!1" "st!1`sp`app_seq(n!1)")
              (("1"
                (split -1)
                (("1" (inst 1 "id!1") (grind)) ("2" (grind))))
               ("2" (grind))))))
           ("2"
            (hide 2)
            (lemma "recursive_monitor_apps")
            (inst -1 "st!1" "n!1-1")
            (grind))
           ("3" (hide 2) (grind)) ("4" (hide 2) (grind))))))
       ("2"
        (hide -1 4)
        (lemma "recursive_monitor_apps")
        (inst -1 "st!1" "n!1-1")
        (grind))
       ("3" (hide -1 4) (grind)) ("4" (hide -1 4) (grind))))
     ("2" (propax))))
   ("2" (hide -1 4) (grind))))
 ("2"
  (hide -1 4)
  (typepred
   "app!1`svcmap
                 (st!1`sp`SCRAM_info`configs(st!1`svclvl)(app!1))")
  (("1" (grind))
   ("2"
    (typepred "st!1`sp`SCRAM_info`configs(st!1`svclvl)")
    (("1" (grind)) ("2" (grind))))
   ("3" (grind))))
 ("3"
```

```
   (hide -1 4)
   (typepred "st!1`sp`SCRAM_info`configs(st!1`svclvl)")
   (("1" (grind)) ("2" (grind))))
  ("4" (hide -1 4) (grind))))


lemmas_inv.invariant_monitor_middle_app: proved - complete [shostak](2990.12 s)

(""
 (induct "n")
 (("1"
   (skosimp)
   (skosimp)
   (expand "recursive_monitor")
   (lemma "invariant_monitor")
   (inst -1 "st!1" "app!1")
   (flatten)
   (hide -1)
   (split -1)
   (("1" (typepred "m!1") (assert)) ("2" (propax))))
  ("2"
   (skosimp)
   (skosimp)
   (skosimp -2)
   (case "m!1 = j!1+1")
   (("1"
     (hide -2)
     (lemma "monitor_inv_equal")
     (expand "recursive_monitor")
     (inst -1 "recursive_monitor(st!1, st!1`sp`app_seq, j!1)" "app!1")
     (("1"
       (split -1)
       (("1"
         (lemma "rm_speclvl_const")
         (inst -1 "st!1" "j!1")
         (split -1)
         (("1" (hide -2 3) (grind))
          ("2"
           (lemma "rm_spec_constant")
           (inst -1 "st!1" "j!1")
           (split -1)
           (("1" (assert)) ("2" (grind))))))
        ("2"
         (lemma "change_to_interrupt_rec_app")
         (inst -1 "st!1" "app!1" "j!1")
         (split -1)
         (("1" (assert)) ("2" (hide 4) (assert))
          ("3"
           (skosimp -1)
           (hide 3)
           (typepred "st!1`sp`app_seq")
           (expand "bijective?")
           (flatten)
           (expand "injective?")
           (inst -2 "m!1" "m!2")
           (("1" (grind)) ("2" (grind)) ("3" (grind))))))))
      ("2"
       (hide 4)
       (lemma "recursive_monitor_apps")
```

```
       (typepred "app!1")
       (inst -2 "st!1" "j!1")
       (split -2)
       (("1" (assert)) ("2" (assert))))
      ("3" (hide 4) (assert))))
    ("2"
     (inst -1 "st!1" "app!1")
     (split -1)
     (("1" (hide 4) (grind))
      ("2"
       (lemma "inv_not_equal")
       (inst -1 "st!1" "app!1" "j!1+1")
       (split -1)
       (("1" (propax)) ("2" (propax)) ("3" (hide 5) (assert))
        ("4"
         (hide -1 5)
         (typepred "st!1`sp`app_seq")
         (expand "bijective?")
         (flatten)
         (expand "injective?")
         (inst -2 "m!1" "j!1+1")
         (("1" (grind)) ("2" (grind)) ("3" (grind))))))
      ("3" (hide 5) (inst 1 "m!1") (grind)) ("4" (hide 5) (grind))))))
  ("3" (hide 2) (skosimp*) (grind))
  ("4"
   (hide 2)
   (skosimp*)
   (typepred "st!1`sp`SCRAM_info`configs(st!1`svclvl)")
   (("1" (grind)) ("2" (grind))))
  ("5" (hide 2) (skosimp) (grind))
  ("6"
   (hide 2)
   (skosimp)
   (grind)
   (typepred "st!1`sp`app_seq")
   (grind))))
```

lemmas_inv.invariant_monitor_rec_app_TCC1: proved - complete [shostak](362.13 s)

```
("" (skosimp*) (lemma "nonempty_apps") (inst -1 "st!1") (grind))
```

lemmas_inv.invariant_monitor_rec_app: proved - complete [shostak](425.49 s)

```
(""
 (lemma "invariant_monitor_middle_app")
 (skosimp 1)
 (inst -1 "st!1" "app!1" "card(st!1`sp`apps) - 1")
 (("1"
   (split -1)
   (("1" (hide 2) (grind)) ("2" (propax))
    ("3"
     (typepred "st!1`sp`app_seq")
     (expand "bijective?")
     (flatten)
     (expand "surjective?")
     (inst -3 "app!1")
     (skosimp)
```

```
     (inst 1 "x!1")
     (hide 3)
     (grind))
    ("4" (hide 3) (grind))))
  ("2" (hide 3) (lemma "nonempty_apps") (inst -1 "st!1"))))
```

10. Reconfiguration Properties

Proof scripts for file properties-fm.pvs:

assured_reconfig.CP1: proved - complete [shostak](36.36 s)

```
(""
 (skosimp)
 (typepred "r!1")
 (expand "get_reconfigs")
 (flatten)
 (split)
 (("1" (propax)) ("2" (grind)) ("3" (grind)) ("4" (grind))))
```

assured_reconfig.CP2_TCC1: proved - complete [shostak](10.35 s)

```
("" (grind))
```

assured_reconfig.CP2: proved - complete [shostak](6641.14 s)

```
(""
 (skosimp)
 (inst 2 "r!1`start_c+1")
 (typepred "r!1")
 (expand "get_reconfigs")
 (flatten)
 (typepred "s!1`tr")
 (case "r!1`end_c-r!1`start_c=1")
 (("1"
   (inst -2 "r!1`start_c")
   (hide -3 -4 -5)
   (hide 2)
   (expand "system_monitor")
   (lift-if)
   (split -2)
   (("1"
     (flatten)
     (hide -2)
     (expand "reconfig_start?")
     (skosimp -4)
     (inst 1 "app!1")
     (grind))
    ("2"
     (flatten)
     (hide 1)
     (lemma "reconf_halt")
     (expand "reconfig_end?")
```

```
   (lemma "rm_speclvl_const")
   (inst -1 "next_config(s!1`tr(r!1`start_c))"
    "card(s!1`tr(r!1`start_c)`sp`apps) - 1")
   (("1"
     (split -1)
     (("1" (hide -2 -3 -4 -5 -6 -7 -8 1) (grind))
      ("2"
       (split 1)
       (("1" (propax)) ("2" (hide -2 -5 -6 -7 -8) (grind))))))
    ("2"
     (hide -1 -2 -3 -4 -5 -6 -7 2)
     (lemma "nonempty_apps")
     (inst -1 "s!1`tr(r!1`start_c)")))))
("2"
 (hide -2 -3 -4 2)
 (case "s!1`tr(r!1`start_c+2)`svclvl = s!1`sp`choose
                   (s!1`tr(r!1`start_c + 1)`svclvl,
                    s!1`env((r!1`start_c + 1) * cycle_time))")
 (("1"
   (hide -2)
   (case "r!1`end_c-r!1`start_c=2")
   (("1"
     (split 2)
     (("1" (hide -1 -2 -3 -4 -5 -6) (grind))
      ("2" (hide -2 -3 -4 -5 -6) (grind))
      ("3" (hide -3 -4 -5 -6) (assert))))
    ("2"
     (case "r!1`end_c - r!1`start_c = 3")
     (("1"
       (hide -3 -4 1 2)
       (split 1)
       (("1" (hide -2) (assert)) ("2" (hide -2) (assert))
        ("3"
         (reveal -3)
         (inst -1 "r!1`start_c+2")
         (expand "system_monitor")
         (lift-if)
         (split)
         (("1"
           (expand "reconfig_end?")
           (flatten)
           (hide -2)
           (skosimp -1)
           (split -4)
           (("1"
             (inst -1 "app!1")
             (("1"
               (lemma "reconf_prep")
               (inst -1 "s!1" "r!1")
               (split -1)
               (("1"
                 (inst -1 "app!1")
                 (("1"
                   (split -1)
                   (("1" (hide -5 -6 -7 1 2 -2) (grind))
                    ("2"
                     (hide -4 -5 -6 1 2)
                     (flatten)
                     (hide -1 -3)
```

```
                    (grind))
                   ("3"
                    (inst 1 "app!1")
                    (hide -2 -3 -4 -5 -6 2)
                    (grind))))
                 ("2"
                  (hide -1 -2 -3 -4 -5 2 3)
                  (typepred "app!1")
                  (typepred "tr(s!1)(2 + start_c(r!1))")
                  (grind))))
               ("2" (hide -1 -2 -4 -5 2 3) (assert))))
              ("2"
               (hide -1 -2 -3 -4 2 3)
               (typepred "app!1")
               (typepred "tr(s!1)(2 + start_c(r!1))")
               (grind))))
             ("2"
              (lemma "reconf_prep")
              (inst -1 "s!1" "r!1")
              (split -1)
              (("1"
                (hide -2 -4 -5 -6 2)
                (inst -1 "app!1")
                (("1" (grind))
                 ("2"
                  (typepred "app!1")
                  (typepred "tr(s!1)(2 + start_c(r!1))")
                  (grind))))
               ("2" (hide -1 -2 -4 -5 2 3) (assert))))))
           ("2"
            (flatten)
            (hide 1)
            (hide -4 -5)
            (lemma "rm_speclvl_const")
            (inst -1 "next_config(s!1`tr(2 + r!1`start_c))"
             "card(s!1`tr(2 + r!1`start_c)`sp`apps) - 1")
            (("1"
              (split -1)
              (("1" (hide -2 -3 -4 1) (grind)) ("2" (grind))))
             ("2"
              (hide -1 -2 -3 2)
              (lemma "nonempty_apps")
              (inst -1 "s!1`tr(2 + r!1`start_c)")))))))))
       ("2"
        (hide -1 -3 -4 -5 4)
        (lemma "reconf_length")
        (inst -1 "s!1" "r!1")
        (grind))))))
("2"
 (inst -1 "r!1`start_c+1")
 (hide -2 -3 -4 2 3)
 (lemma "rm_speclvl_const")
 (expand "system_monitor")
 (lift-if)
 (split -2)
 (("1"
   (hide -3)
   (inst -2
    "s!1`tr(1 + r!1`start_c)
```

```
                                    WITH [`reconf_st
                                            := LAMBDA
                                               (app:
                                              (s!1`tr(r!1`start_c + 1)`sp`apps)):
                                               IF NOT
                                                  s!1`tr
                                                  (1
                                                   +
                                               r!1`start_c)`sp`SCRAM_info`configs
                                               (s!1`tr(1 + r!1`start_c)`sp`choose
                                                 (s!1`tr(1 + r!1`start_c)`svclvl,
                                                    s!1`env
                                                    (r!1`start_c * cycle_time
                                                     +
                                                     cycle_time)))
                                                  (app)
                                                  =
                                                  s!1`tr
                                                  (1
                                                   +
                                               r!1`start_c)`sp`SCRAM_info`configs
                                                 (s!1`tr(1 + r!1`start_c)`svclvl)
                                                  (app)
                                               THEN prepping
                                               ELSE normal
                                               ENDIF,
                                         `app_svclvls
                                            := LAMBDA
                                               (app:
                                              (s!1`tr(r!1`start_c + 1)`sp`apps)):
                                               s!1`tr
                                               (1
                                                +
                                               r!1`start_c)`sp`SCRAM_info`configs
                                               (s!1`tr(1 + r!1`start_c)`sp`choose
                                                (s!1`tr(1 + r!1`start_c)`svclvl,
                                                 s!1`env
                                                 (r!1`start_c * cycle_time
                                                  +
                                                  cycle_time)))
                                               (app),
                                         `app_last_svcs
                                            := LAMBDA
                                               (app:
                                              (s!1`tr(r!1`start_c + 1)`sp`apps)):
                                              s!1`tr(1 + r!1`start_c)`app_svclvls
                                               (app),
                                      `last_svc := s!1`tr(1 + r!1`start_c)`svclvl,
                                         `svclvl
                                            := s!1`tr(1 + r!1`start_c)`sp`choose
                                               (s!1`tr(1 + r!1`start_c)`svclvl,
                                                s!1`env
                                                (r!1`start_c * cycle_time
                                                 +
                                                 cycle_time))]"
 "card(s!1`tr(1 + r!1`start_c)`sp`apps) - 1")
(("1"
  (split -2)
  (("1" (hide -2) (hide 1) (grind)) ("2" (flatten) (grind))))
```

```
  ("2"
   (hide -1 -2 2)
   (lemma "nonempty_apps")
   (inst -1 "s!1`tr(1 + r!1`start_c)"))
  ("3"
   (hide -1 -2 2)
   (typepred "s!1`env(r!1`start_c * cycle_time + cycle_time)")
   (grind))
  ("4" (hide -1 -2 2) (grind)) ("5" (hide -1 -2 2) (grind))
  ("6"
   (hide -1 -2 2)
   (skosimp*)
   (typepred "s!1`env(r!1`start_c * cycle_time + cycle_time)")
   (grind))))
("2"
 (flatten)
 (lemma "reconf_halt")
 (split 1)
 (("1"
   (reveal -2)
   (expand "reconfig_start?")
   (reveal -3)
   (skosimp -2)
   (hide -1 -3 -5 -6 2)
   (inst 1 "app!1")
   (("1"
     (hide -2)
     (reveal -8)
     (inst -1 "r!1`start_c")
     (expand "system_monitor")
     (lift-if)
     (split -1)
     (("1"
       (flatten)
       (inst 1 "app!1")
       (hide -1 -2 -3 2)
       (typepred "s!1`tr(r!1`start_c)")
       (grind))
      ("2"
       (flatten)
       (hide 1)
       (reveal 6)
       (case "r!1`end_c-r!1`start_c = 1")
       (("1"
         (split 1)
         (("1" (propax))
          ("2"
           (lemma "rm_speclvl_const")
           (inst -1 "next_config(s!1`tr(r!1`start_c))"
            "card(s!1`tr(r!1`start_c)`sp`apps) - 1")
           (("1"
             (split -1)
             (("1" (hide -2 -3 -4 1 2) (grind)) ("2" (grind))))
            ("2"
             (hide -1 -2 -3 2 3)
             (lemma "nonempty_apps")
             (inst -1 "s!1`tr(r!1`start_c)"))))))
        ("2"
         (lemma "change_to_interrupt_rec")
```

```
               (inst -1 "next_config(s!1`tr(r!1`start_c))" "app!1"
                "card(s!1`tr(r!1`start_c)`sp`apps) - 1")
               (("1"
                 (split -1)
                 (("1" (hide -2 -3 1 2 3) (grind)) ("2" (grind))
                  ("3"
                   (lemma "int_halt_len")
                   (inst -1 "s!1" "r!1")
                   (split -1)
                   (("1" (propax))
                    ("2" (inst 1 "app!1") (assert) (grind))))))
                ("2"
                 (hide -1 -2 2 3 4)
                 (lemma "nonempty_apps")
                 (inst -1 "s!1`tr(r!1`start_c)"))
                ("3" (hide -1 -2 2 3 4) (grind))))))))
          ("2"
           (hide -1 -2)
           (typepred "tr(s!1)(1 + start_c(r!1))")
           (grind))))
         ("2"
          (lemma "int_halt_len")
          (reveal 1)
          (inst -1 "s!1" "r!1")
          (split -1)
          (("1" (propax))
           ("2"
            (skosimp -1)
            (inst 1 "app!1")
            (("1" (assert))
             ("2"
              (hide -1 -2 -3 -4 -5 2 3)
              (typepred "app!1")
              (typepred "tr(s!1)(1 + start_c(r!1))")
              (grind)))))))))
    ("3" (hide -1 -2 -3 -4 -5 2 3) (grind))))))
```

```
assured_reconfig.CP3_TCC1: proved - complete [shostak](16.64 s)

("" (grind))
```

```
assured_reconfig.CP3_TCC2: proved - complete [shostak](28.54 s)

("" (skosimp) (typepred "s!1`tr(r!1`end_c)") (grind))
```

```
assured_reconfig.CP3: proved - complete [shostak](778.75 s)

(""
 (skosimp)
 (lemma "reconf_length")
 (inst -1 "s!1" "r!1")
 (typepred
  "s!1`sp`T(s!1`tr(r!1`start_c)`svclvl, s!1`tr(r!1`end_c)`svclvl)")
 (("1"
   (case "r!1`end_c-r!1`start_c+1 <= 4")
   (("1"
```

```
     (lemma "both_sides_times_pos_le1")
     (case "cycle_time > 0")
     (("1"
       (inst -2 "cycle_time" "r!1`end_c - r!1`start_c + 1" "4")
       (grind))
      ("2"
       (case "cycle_time = 0")
       (("1" (hide -2) (grind))
        ("2" (typepred "cycle_time") (assert))))))
    ("2" (grind))))
  ("2" (grind)) ("3" (grind))))


assured_reconfig.CP4_TCC1: proved - complete [shostak](1.87 s)

("" (skosimp) (typepred "s!1`tr(c!1)") (grind))


assured_reconfig.CP4_TCC2: proved - complete [shostak](10.99 s)

("" (skosimp) (hide 1) (skosimp) (typepred "tr(s!1)(c!1)") (grind))


assured_reconfig.CP4_TCC3: proved - complete [shostak](13.84 s)

(""
 (skosimp)
 (hide 1)
 (skosimp)
 (typepred "tr(s!1)(c!1 - 1)")
 (("1" (grind)) ("2" (assert))))


assured_reconfig.CP4_TCC4: proved - complete [shostak](8.59 s)

("" (skosimp) (hide 1) (skosimp) (assert))


assured_reconfig.CP4_TCC5: proved - complete [shostak](16.42 s)

(""
 (skosimp)
 (hide 1)
 (skosimp)
 (typepred "s!1`tr(c!1 - 1)")
 (("1" (grind)) ("2" (assert))))


assured_reconfig.CP4_TCC6: proved - complete [shostak](13.83 s)

("" (skosimp) (hide 1) (skosimp) (typepred "s!1`tr(c!1)") (grind))


assured_reconfig.CP4_TCC7: proved - complete [shostak](23.05 s)

(""
 (skosimp)
 (hide 1)
 (skosimp)
```

```
 (hide -3 1)
 (typepred "s!1`tr(c!1 - 1)")
 (("1" (grind)) ("2" (assert))))


assured_reconfig.CP4_TCC8: proved - complete [shostak](45.50 s)

(""
 (skosimp)
 (hide 1)
 (skosimp)
 (hide -3 1)
 (typepred "s!1`tr(c!1 - 1)")
 (("1"
   (grind)
   (typepred "s!1`sp`SCRAM_info`configs(s!1`tr(c!1 - 1)`last_svc)")
   (grind))
  ("2" (assert))))


assured_reconfig.CP4_TCC9: proved - complete [shostak](24.49 s)

(""
 (skosimp)
 (hide 1)
 (skosimp)
 (hide 1)
 (typepred "s!1`tr(c!1)")
 (grind))


assured_reconfig.CP4_TCC10: proved - complete [shostak](43.44 s)

(""
 (skosimp)
 (hide 1)
 (skosimp)
 (hide 1)
 (typepred "s!1`sp`SCRAM_info`configs(s!1`tr(c!1)`svclvl)")
 (("1" (grind)) ("2" (typepred "s!1`tr(c!1)") (grind))))


assured_reconfig.CP4: proved - complete [shostak](n/a s)

(""
 (induct "c")
 (("1" (skosimp) (typepred "s!1`tr") (propax))
  ("2"
   (skosimp)
   (skosimp)
   (inst -1 "s!1")
   (split -1)
   (("1"
     (typepred "s!1`tr")
     (split 2)
     (("1" (typepred "j!1") (assert))
      ("2"
       (skosimp)
       (hide -2 -4)
```

```
(inst -1 "j!1")
(hide 3)
(lemma "nonempty_apps")
(inst -1 "s!1`tr(j!1)")
(hide 1)
(expand "system_monitor")
(lemma "invariant_monitor_rec_app")
(lemma "rm_speclvl_const")
(lift-if)
(split -4)
(("1"
  (flatten)
  (inst -3
   "s!1`tr(j!1)
                          WITH [`reconf_st
                                  := LAMBDA (app: (s!1`tr(j!1)`sp`apps)):
                                     IF s!1`tr(j!1)`sp`SCRAM_info`configs
                                              (s!1`tr(j!1)`sp`choose
                                                   (s!1`tr(j!1)`svclvl,
                                              s!1`env(j!1 * cycle_time)))
                                              (app)
                                           /=
                                       s!1`tr(j!1)`sp`SCRAM_info`configs
                                              (s!1`tr(j!1)`svclvl)(app)
                                        THEN prepping
                                       ELSE normal
                                       ENDIF,
                                `app_svclvls
                                  := LAMBDA (app: (s!1`tr(j!1)`sp`apps)):
                                       s!1`tr(j!1)`sp`SCRAM_info`configs
                                           (s!1`tr(j!1)`sp`choose
                                                (s!1`tr(j!1)`svclvl,
                                              s!1`env(j!1 * cycle_time)))
                                           (app),
                                `app_last_svcs
                                  := LAMBDA (app: (s!1`tr(j!1)`sp`apps)):
                                       s!1`tr(j!1)`app_svclvls(app),
                                `last_svc := s!1`tr(j!1)`svclvl,
                                `svclvl
                                  := s!1`tr(j!1)`sp`choose
                                         (s!1`tr(j!1)`svclvl,
                                          s!1`env(j!1 * cycle_time))]"
   "card(s!1`tr(j!1)`sp`apps) - 1")
  (("1"
    (split -3)
    (("1" (hide -2 -3 -4 -5 -6 -7 1 2) (grind))
     ("2"
      (inst -4
       "s!1`tr(j!1)
                                    WITH [`reconf_st
                                     := LAMBDA (app: (s!1`tr(j!1)`sp`apps)):
                                        IF s!1`tr(j!1)`sp`SCRAM_info`configs
                                                    (s!1`tr(j!1)`sp`choose
                                                      (s!1`tr(j!1)`svclvl,
                                                 s!1`env(j!1 * cycle_time)))
                                                       (app)
                                                    /=
                                          s!1`tr(j!1)`sp`SCRAM_info`configs
                                                   (s!1`tr(j!1)`svclvl)(app)
```

```
                                                    THEN prepping
                                                  ELSE normal
                                                  ENDIF,
                                          `app_svclvls
                                      := LAMBDA (app: (s!1`tr(j!1)`sp`apps)):
                                           s!1`tr(j!1)`sp`SCRAM_info`configs
                                                      (s!1`tr(j!1)`sp`choose
                                                        (s!1`tr(j!1)`svclvl,
                                                 s!1`env(j!1 * cycle_time)))
                                                      (app),
                                          `app_last_svcs
                                      := LAMBDA (app: (s!1`tr(j!1)`sp`apps)):
                                             s!1`tr(j!1)`app_svclvls(app),
                                          `last_svc := s!1`tr(j!1)`svclvl,
                                          `svclvl
                                            := s!1`tr(j!1)`sp`choose
                                                    (s!1`tr(j!1)`svclvl,
                                               s!1`env(j!1 * cycle_time))]"
      "app!1")
     (("1"
       (split -4)
       (("1" (hide -3 -5 -6 -7 1) (grind))
        ("2" (hide -2 -3 -4 -5 -6 -7 2) (grind))))
      ("2" (hide -1 -2 -3 -4 -5 -6 2 3) (grind))
      ("3"
       (hide -1 -2 -3 -4 -5 -6 2 3)
       (typepred "s!1`env(j!1 * cycle_time)")
       (grind))
      ("4" (hide -1 -2 -3 -4 -5 -6 2 3) (grind))
      ("5"
       (hide -1 -2 -3 -4 -5 -6 2 3)
       (typepred "s!1`env(j!1 * cycle_time)")
       (grind))
      ("6"
       (hide -1 -2 -3 -4 -5 -6 2 3)
       (typepred "s!1`env(j!1 * cycle_time)")
       (grind))))))
  ("2"
   (hide -1 -2 -3 -4 -5 -6 2 3)
   (typepred "s!1`env(j!1 * cycle_time)")
   (grind))
  ("3" (hide -1 -2 -3 -4 -5 -6 2 3) (grind))
  ("4" (hide -1 -2 -3 -4 -5 -6 2 3) (grind))
  ("5"
   (hide -1 -2 -3 -4 -5 -6 2 3)
   (typepred "s!1`env(j!1 * cycle_time)")
   (grind))))
("2"
 (flatten)
 (inst -2 "next_config(s!1`tr(j!1))"
  "card(s!1`tr(j!1)`sp`apps) - 1")
 (split -2)
 (("1" (hide -2 -3 -4 -5 -6 1 2) (grind))
  ("2"
   (inst -3 "next_config(s!1`tr(j!1))" "app!1")
   (("1"
     (split -3)
     (("1"
       (hide -4 -5 1)
```

```
            (expand "inv")
            (inst -4 "app!1")
            (skosimp)
            (inst -1 "m!1")
            (expand "inv")
            (inst -4 "m!1")
            (grind))
           ("2"
            (hide -2 -3 -4 -5 -6 1 2)
            (expand "next_config")
            (lift-if)
            (grind))))
         ("2" (hide -1 -2 -3 -4 -5 2 3) (grind))))))))))
("2"
 (flatten)
 (split 2)
 (("1" (assert))
  ("2"
   (skosimp)
   (inst -2 "app!1")
   (split -2)
   (("1"
     (flatten)
     (hide 4)
     (lemma "nonempty_apps")
     (inst -1 "s!1`tr(j!1)")
     (typepred "s!1`tr")
     (hide -2 -3 -4)
     (inst -1 "j!1")
     (expand "system_monitor")
     (lift-if)
     (split -1)
     (("1"
       (flatten)
       (inst 1 "app!1")
       (hide -1 -2 -3 -4 -5 -6 -7 2 3 4)
       (grind))
      ("2"
       (flatten)
       (expand "next_config")
       (lift-if)
       (split -1)
       (("1"
         (flatten)
         (hide 1 2 3 -4 -5 -1)
         (lemma "invariant_monitor_rec_app")
         (lemma "rm_speclvl_const")
         (inst -1
          "s!1`tr(j!1)
                                                  WITH [`reconf_st
                                                        := LAMBDA
                                                 (app: (s!1`tr(j!1)`sp`apps)):
                                                 IF s!1`tr(j!1)`reconf_st(app)
                                                                  /=
                                                                  normal
                                                             THEN halting
                                                           ELSE exec_halting
                                                             ENDIF]"
          "card(s!1`tr(j!1)`sp`apps) - 1")
```

```
     (inst -2
      "s!1`tr(j!1)
                                                  WITH [`reconf_st
                                                           := LAMBDA
                                                 (app: (s!1`tr(j!1)`sp`apps)):
                                                 IF s!1`tr(j!1)`reconf_st(app)
                                                                  /=
                                                                  normal
                                                             THEN halting
                                                           ELSE exec_halting
                                                             ENDIF]"
      "app!1")
     (("1"
       (split -1)
       (("1" (hide -2 -3 -4 -5 -6 1) (grind))
        ("2"
         (split -2)
         (("1" (grind))
          ("2" (hide -2 -3 -4 -5 -6 1) (grind))))))
      ("2" (hide -1 -2 -3 -4 -5 2) (grind))))
    ("2"
     (flatten)
     (inst 1 "app!1")
     (hide -1 -2 -3 -4 -5 -6 2 3 4 5)
     (grind))))))
("2"
 (lemma "nonempty_apps")
 (inst -1 "s!1`tr(j!1)")
 (typepred "s!1`tr")
 (hide -2 -3 -4)
 (inst -1 "j!1")
 (expand "system_monitor")
 (lift-if)
 (split -1)
 (("1"
   (flatten)
   (hide -1 1 2 4)
   (lemma "invariant_monitor_rec_app")
   (lemma "rm_speclvl_const")
   (inst -1
    "s!1`tr(j!1)
                              WITH [`reconf_st
                                   := LAMBDA (app: (s!1`tr(j!1)`sp`apps)):
                                      IF s!1`tr(j!1)`sp`SCRAM_info`configs
                                                   (s!1`tr(j!1)`sp`choose
                                                      (s!1`tr(j!1)`svclvl,
                                                 s!1`env(j!1 * cycle_time)))
                                                      (app)
                                                    /=
                                                s!1`tr(j!1)`sp`SCRAM_info`configs
                                                        (s!1`tr(j!1)`svclvl)(app)
                                                     THEN prepping
                                                   ELSE normal
                                                   ENDIF,
                                    `app_svclvls
                                   := LAMBDA (app: (s!1`tr(j!1)`sp`apps)):
                                        s!1`tr(j!1)`sp`SCRAM_info`configs
                                               (s!1`tr(j!1)`sp`choose
                                                    (s!1`tr(j!1)`svclvl,
```

```
                                                s!1`env(j!1 * cycle_time)))
                                                 (app),
                                       `app_last_svcs
                                     := LAMBDA (app: (s!1`tr(j!1)`sp`apps)):
                                             s!1`tr(j!1)`app_svclvls(app),
                                       `last_svc := s!1`tr(j!1)`svclvl,
                                       `svclvl
                                         := s!1`tr(j!1)`sp`choose
                                                (s!1`tr(j!1)`svclvl,
                                                s!1`env(j!1 * cycle_time))]"
 "card(s!1`tr(j!1)`sp`apps) - 1")
(("1"
  (split -1)
  (("1" (hide -2 -3 -4 -5 -6 1) (grind))
   ("2"
    (inst -2
     "s!1`tr(j!1)
                                      WITH [`reconf_st
                                    := LAMBDA (app: (s!1`tr(j!1)`sp`apps)):
                                       IF s!1`tr(j!1)`sp`SCRAM_info`configs
                                                    (s!1`tr(j!1)`sp`choose
                                                      (s!1`tr(j!1)`svclvl,
                                                 s!1`env(j!1 * cycle_time)))
                                                             (app)
                                                          /=
                                              s!1`tr(j!1)`sp`SCRAM_info`configs
                                                      (s!1`tr(j!1)`svclvl)(app)
                                                           THEN prepping
                                                         ELSE normal
                                                         ENDIF,
                                                `app_svclvls
                                    := LAMBDA (app: (s!1`tr(j!1)`sp`apps)):
                                         s!1`tr(j!1)`sp`SCRAM_info`configs
                                                    (s!1`tr(j!1)`sp`choose
                                                      (s!1`tr(j!1)`svclvl,
                                                 s!1`env(j!1 * cycle_time)))
                                                           (app),
                                                `app_last_svcs
                                    := LAMBDA (app: (s!1`tr(j!1)`sp`apps)):
                                                  s!1`tr(j!1)`app_svclvls(app),
                                           `last_svc := s!1`tr(j!1)`svclvl,
                                              `svclvl
                                                := s!1`tr(j!1)`sp`choose
                                                       (s!1`tr(j!1)`svclvl,
                                                  s!1`env(j!1 * cycle_time))]"
     "app!1")
    (("1"
      (split -2)
      (("1" (grind)) ("2" (hide -2 -3 -4 -5 -6 1) (grind))))
     ("2" (hide -1 -2 -3 -4 -5 2) (grind))
     ("3"
      (hide -1 -2 -3 -4 -5 2)
      (typepred "s!1`env(j!1 * cycle_time)")
      (grind))
     ("4" (hide -1 -2 -3 -4 -5 2) (grind))
     ("5" (hide -1 -2 -3 -4 -5 2) (grind))
     ("6"
      (hide -1 -2 -3 -4 -5 2)
      (typepred "s!1`env(j!1 * cycle_time)")
```

```
              (grind))))))
           ("2"
            (hide -1 -2 -3 -4 -5 2)
            (typepred "s!1`env(j!1 * cycle_time)")
            (grind))
           ("3" (hide -1 -2 -3 -4 -5 2) (grind))
           ("4" (hide -1 -2 -3 -4 -5 2) (grind))
           ("5"
            (hide -1 -2 -3 -4 -5 2)
            (typepred "s!1`env(j!1 * cycle_time)")
            (grind))))
         ("2"
          (flatten)
          (hide 1 4)
          (hide -4 1)
          (lemma "rm_speclvl_const")
          (lemma "invariant_monitor_rec_app")
          (inst -1 "next_config(s!1`tr(j!1))" "app!1")
          (("1"
            (inst -2 "next_config(s!1`tr(j!1))"
             "card(s!1`tr(j!1)`sp`apps) - 1")
            (split -2)
            (("1" (hide -2 -3 -4 -5 1) (grind))
             ("2"
              (split -2)
              (("1"
                (expand "inv")
                (skosimp)
                (inst -1 "m!1")
                (inst -5 "m!1")
                (grind))
               ("2"
                (hide -2 -3 -4 -5 1)
                (expand "next_config")
                (lift-if)
                (grind))))))
           ("2" (hide -1 -2 -3 -4 2) (grind))))))))))))
("3"
 (skosimp*)
 (hide -1 1 2 4 5)
 (typepred "app!1")
 (typepred "s!1`sp`SCRAM_info`configs(s!1`tr(c!2)`svclvl)")
 (("1" (grind)) ("2" (grind))))
("4" (hide 2) (skosimp) (skosimp) (hide 1 2) (grind))
("5"
 (hide 2)
 (skosimp)
 (skosimp)
 (hide -1 -2 -3 1 2)
 (typepred "s!1`sp`SCRAM_info`configs(s!1`tr(c!2 - 1)`last_svc)")
 (("1"
   (inst -1 "app!1")
   (typepred "s!1`sp`SCRAM_info`configs(s!1`tr(c!2 - 1)`svclvl)")
   (("1" (grind)) ("2" (grind))))
  ("2" (grind)) ("3" (reveal -1) (grind))))
("6" (hide 2) (skosimp) (skosimp) (hide -1 -2 -3 1 2) (grind))
("7" (hide 2) (skosimp*) (hide -2 -3 1 2) (assert))
("8" (hide 2) (skosimp*) (hide -1 -2 -3 1) (grind))
("9" (hide 2) (skosimp*) (hide -1 -2 -3 1) (grind))
```

```
  ("10" (hide 2) (skosimp*) (hide -2 -3 1) (assert))
  ("11" (hide 2) (skosimp*) (hide -2 1 2) (assert))
  ("12"
   (hide 2)
   (skosimp*)
   (hide -1 -2 1 2)
   (typepred "tr(s!1)(c!2 - 1)")
   (("1" (grind)) ("2" (reveal -1) (grind))))
  ("13" (hide 2) (skosimp*) (hide -2 -3 1) (assert))
  ("14"
   (hide 2)
   (skosimp*)
   (typepred "tr(s!1)(c!2 - 1)")
   (("1" (hide 1) (grind)) ("2" (assert))))
  ("15" (hide 2) (skosimp*) (typepred "tr(s!1)(c!2)") (assert))
  ("16" (hide 2) (skosimp) (grind))))


assured_reconfig.CP5_TCC1: proved - complete [shostak](12.09 s)

("" (skosimp) (skosimp) (typepred "s!1`tr(r!1`start_c)") (grind))


assured_reconfig.CP5_TCC2: proved - complete [shostak](11.47 s)

("" (skosimp) (skosimp) (typepred "s!1`tr(r!1`end_c)") (grind))


assured_reconfig.CP5_TCC3: proved - complete [shostak](28.49 s)

(""
 (skosimp)
 (skosimp)
 (typepred "s!1`sp`SCRAM_info`configs(s!1`tr(r!1`end_c)`svclvl)")
 (("1" (grind)) ("2" (grind))))


assured_reconfig.CP5_TCC4: proved - complete [shostak](37.42 s)

(""
 (skosimp)
 (skosimp)
 (hide -1 -2 1)
 (typepred "s!1`sp`SCRAM_info`configs(s!1`tr(r!1`end_c)`svclvl)")
 (("1" (grind)) ("2" (grind))))


assured_reconfig.CP5_TCC5: proved - complete [shostak](25.57 s)

(""
 (skosimp)
 (hide 1)
 (skosimp)
 (typepred "tr(s!1)(end_c(r!1))")
 (grind))


assured_reconfig.CP5_TCC6: proved - complete [shostak](25.69 s)
```

```
("" (skosimp) (hide 1) (skosimp) (typepred "s!1`tr(r!1`end_c)") (grind))
```

assured_reconfig.CP5_TCC7: proved - complete [shostak](49.90 s)

```
(""
 (skosimp)
 (hide 1)
 (skosimp)
 (typepred "s!1`sp`SCRAM_info`configs(s!1`tr(r!1`end_c)`svclvl)")
 (("1" (grind)) ("2" (typepred "s!1`tr(r!1`end_c)") (grind))))
```

assured_reconfig.CP5_TCC8: proved - complete [shostak](33.05 s)

```
(""
 (skosimp)
 (hide 1 2)
 (skosimp)
 (typepred "tr(s!1)(end_c(r!1))")
 (grind))
```

assured_reconfig.CP5: proved - complete [shostak](13048.43 s)

```
(""
 (skosimp)
 (split)
 (("1"
   (lemma "reconf_length")
   (inst -1 "s!1" "r!1")
   (typepred "r!1")
   (typepred "s!1`tr")
   (expand "get_reconfigs")
   (hide -2 -3 -4)
   (flatten)
   (case "r!1`end_c - r!1`start_c = 1")
   (("1"
     (lemma "reconf_halt")
     (expand "reconfig_end?")
     (split -6)
     (("1"
       (expand "reconfig_start?")
       (skosimp)
       (inst -1 "app!1")
       (inst -2 "s!1" "r!1" "app!1")
       (hide -4 -5 -6 -7 -8)
       (grind))
      ("2" (propax))))
    ("2"
     (case "r!1`end_c - r!1`start_c = 2")
     (("1"
       (lemma "reconf_prep")
       (expand "reconfig_end?")
       (inst -1 "s!1" "r!1")
       (split -1)
       (("1"
         (split -6)
         (("1"
```

```
          (split 3)
          (("1" (propax))
           ("2"
            (skosimp 1)
            (case "r!1`end_c - r!1`start_c = 2")
            (("1"
              (inst -2 "app!1")
              (inst -3 "app!1")
              (hide -5 -6 -7 -8 2 3)
              (split -3)
              (("1" (hide 1 2) (grind))
               ("2"
                (lemma "CP4")
                (inst -1 "s!1" "r!1`start_c+2")
                (split -1)
                (("1"
                  (flatten)
                  (expand "inv")
                  (inst -1 "app!1")
                  (expand "inv")
                  (split 1)
                  (("1" (hide -1 -3 -5 -6 -7 2) (grind))
                   ("2"
                    (skosimp)
                    (inst -1 "m!1")
                    (hide -2 -3 -5 -6 -7 2)
                    (grind))))
                 ("2"
                  (flatten)
                  (inst -2 "app!1")
                  (split)
                  (("1"
                    (flatten)
                    (inst 3 "app!1")
                    (hide -2 -3 -4 -5 -6 -8 -9 -10 1 2)
                    (grind))
                   ("2" (hide -2 -6 -7 -8 2) (grind))))))
              ("3" (inst 2 "app!1") (hide -3 -4 -5 1) (grind))))
             ("2" (propax))))))
        ("2" (propax))))
      ("2" (hide -1 -2 -6 -4 -5 4 5) (assert))))
    ("2" (hide -1 -3 -4 -5 4 5) (assert))))))
("2"
 (skosimp)
 (lemma "reconf_prep")
 (inst -1 "s!1" "r!1")
 (split)
 (("1"
   (inst -1 "app!1")
   (split -1)
   (("1"
     (flatten)
     (lemma "reconf_train")
     (inst -1 "s!1" "r!1")
     (split)
     (("1"
       (inst -1 "app!1")
       (hide 3 4)
       (case "r!1`end_c-r!1`start_c < 3")
```

```
(("1"
  (hide -2 -3 1 2)
  (typepred "r!1")
  (expand "get_reconfigs")
  (flatten)
  (case "r!1`end_c - r!1`start_c = 1")
  (("1"
    (expand "reconfig_end?")
    (split -4)
    (("1"
      (lemma "reconf_halt")
      (inst -2 "app!1")
      (inst -1 "s!1" "r!1" "app!1")
      (hide -5 -6 -7 1)
      (grind))
     ("2" (propax))))
   ("2"
    (case "r!1`end_c-r!1`start_c = 2")
    (("1"
      (reveal -2)
      (expand "reconfig_end?")
      (split -5)
      (("1" (inst -1 "app!1") (hide -4 -5 -6 -7 1 2) (grind))
       ("2" (propax))))
     ("2" (hide -2 -3 -4 3) (grind))))))
 ("2"
  (typepred "s!1`tr")
  (inst -1 "r!1`start_c+2")
  (hide -2 -3 -4)
  (expand "system_monitor")
  (lift-if)
  (split)
  (("1"
    (flatten)
    (hide -2)
    (skosimp)
    (reveal -8)
    (inst -1 "app!2")
    (("1" (hide -3 -4 1 2 3 4 5) (grind))
     ("2"
      (hide -1 -2 -3 2 3 4 5 6)
      (typepred "app!2")
      (typepred "s!1`tr(r!1`start_c+2)")
      (grind))))
   ("2"
    (flatten)
    (hide 1)
    (expand "next_config")
    (lift-if)
    (split)
    (("1"
      (flatten)
      (lemma "int_prep_len")
      (inst -1 "s!1" "r!1")
      (split)
      (("1" (hide -2 -3 -4 -5 2 3 4) (assert))
       ("2" (hide -2 -3 -4 2 3 4 5) (grind))
       ("3" (hide -1 -2 -3 -4 3 4 5) (grind))))
     ("2"
```

```
(split)
(("1"
  (flatten)
  (split)
  (("1"
    (flatten)
    (skosimp)
    (reveal -7)
    (inst -1 "app!2")
    (("1"
      (hide -3 -4 -5 -6 2 3 4)
      (split)
      (("1" (grind)) ("2" (grind)) ("3" (grind))))
     ("2"
      (hide -1 -2 -3 -4 -5 2 3 4 5)
      (typepred "app!2")
      (typepred "s!1`tr(r!1`start_c+2)")
      (grind))))
   ("2"
    (flatten)
    (split)
    (("1"
      (flatten)
      (hide -1 -2 -5 1 2 5)
      (lemma "reconf_length")
      (inst -1 "s!1" "r!1")
      (grind))
     ("2"
      (flatten)
      (hide -1 -2 -3 2 3 4 5 6)
      (inst 1 "app!1")
      (("1" (grind))
       ("2"
        (typepred "app!1")
        (typepred "s!1`tr(r!1`start_c+2)")
        (grind))))))))
 ("2"
  (flatten)
  (split)
  (("1"
    (flatten)
    (reveal -7)
    (hide -3 -4 -5 1 2 3 4 5)
    (skosimp)
    (inst -1 "app!2")
    (("1" (grind))
     ("2"
      (typepred "s!1`tr(2 + r!1`start_c)")
      (grind))))
   ("2"
    (flatten)
    (split)
    (("1"
      (flatten)
      (lemma "train_time")
      (inst -1 "s!1" "r!1`start_c+3" "app!1")
      (split)
      (("1"
        (hide -2 -3 -4 1 5)
```

```
                        (lemma "reconf_length")
                        (inst -1 "s!1" "r!1")
                        (grind))
                       ("2"
                        (hide -1 -3 2 3 4 6)
                        (lemma "change_to_interrupt_rec")
                        (inst -1
                         "s!1`tr(2 + r!1`start_c)
                                                WITH [`reconf_st
                                                        := LAMBDA
                                                           (app:
                                                            (s!1`tr(2 +
r!1`start_c)`sp`apps)):
                                                           IF (s!1`tr(2 +
r!1`start_c)`reconf_st
                                                               (app)
                                                               =
                                                               prepping)
                                                           THEN training
                                                           ELSE normal
                                                           ENDIF]"
                         "app!1"
                         "card(s!1`tr(2 + r!1`start_c)`sp`apps) - 1")
                        (("1"
                          (split -1)
                          (("1" (hide -2 -3 1 2 3) (grind))
                           ("2" (grind))
                           ("3"
                            (inst 3 "app!1")
                            (lemma "reconf_length")
                            (inst -1 "s!1" "r!1")
                            (grind))))
                         ("2"
                          (hide -1 -2 2 3 4)
                          (lemma "nonempty_apps")
                          (inst -1 "s!1`tr(2 + r!1`start_c)"))
                         ("3"
                          (hide -1 -2 2 3 4)
                          (typepred "s!1`tr(2 + r!1`start_c)")
                          (grind))))))
                     ("2"
                      (flatten)
                      (inst 1 "app!1")
                      (("1" (hide -1 -2 2 3 4 5 6 7) (grind))
                       ("2"
                        (hide -1 -2 -3 2 3 4 5 6 7)
                        (typepred "s!1`tr(2 + r!1`start_c)")
                        (grind))))))))))))))))
             ("2"
              (hide -1 2 3 4 5)
              (reveal -3)
              (typepred "r!1")
              (expand "get_reconfigs")
              (flatten)
              (reveal -1)
              (case "r!1`end_c - r!1`start_c <= 2")
              (("1"
                (case "r!1`end_c-r!1`start_c=2")
                (("1"
```

```
       (inst -7 "r!1`start_c+2")
       (hide -2 -4 -5)
       (expand "reconfig_end?")
       (split)
       (("1" (inst -1 "app!1") (hide -4 -5 1 2) (grind))
        ("2" (propax))))
      ("2"
       (case "r!1`end_c-r!1`start_c=1")
       (("1"
         (expand "reconfig_end?")
         (lemma "reconf_halt")
         (inst -1 "s!1" "r!1" "app!1")
         (hide -3 -4 -5 -6 -8 -9)
         (split -3)
         (("1" (inst -1 "app!1") (grind)) ("2" (propax))))
        ("2" (hide -2 -4 -5 -6 -7 3 4) (grind))))))
    ("2" (hide -1 -2 -3 -4 -5 -6 3) (grind))))))
("2"
 (hide 1 3)
 (lemma "CP4")
 (inst -1 "s!1" "r!1`start_c+2")
 (case "r!1`end_c-r!1`start_c=2")
 (("1"
   (split -2)
   (("1"
     (expand "inv")
     (inst -1 "app!1")
     (split 1)
     (("1" (hide -1 2) (grind))
      ("2"
       (skosimp)
       (expand "inv")
       (inst -1 "m!1")
       (hide -3 -4 2)
       (grind))))
    ("2"
     (flatten)
     (inst -2 "app!1")
     (split -2)
     (("1" (flatten) (hide -2 -3 -4 -5 -6 1 2 3) (grind))
      ("2" (hide -2 -5 2) (grind))))))
  ("2"
   (reveal -1)
   (hide -2)
   (case "r!1`end_c-r!1`start_c = 1")
   (("1"
     (hide -2)
     (lemma "reconf_halt")
     (inst -1 "s!1" "r!1" "app!1")
     (typepred "r!1")
     (expand "get_reconfigs")
     (flatten)
     (expand "reconfig_end?")
     (split -3)
     (("1" (inst -1 "app!1") (hide -2 -3 -4 -7 -8 1 2 3) (grind))
      ("2" (propax))))
    ("2"
     (case "r!1`end_c-r!1`start_c=3")
     (("1"
```

```
       (inst -2 "s!1" "r!1`start_c+3")
       (lemma "reconf_train")
       (inst -1 "s!1" "r!1")
       (split -1)
       (("1"
         (inst -1 "app!1")
         (split 3)
         (("1" (hide -3 2 3 4) (grind))
          ("2"
           (split -3)
           (("1"
             (expand "inv")
             (skosimp)
             (inst -1 "app!1")
             (expand "inv")
             (inst -1 "m!1")
             (hide -2 -4 -5 2 3 4)
             (grind))
            ("2"
             (flatten)
             (inst -2 "app!1")
             (split -2)
             (("1"
               (flatten)
               (inst 5 "app!1")
               (hide -2 -3 -4 -5 -7 -8 1 2 3 4)
               (grind))
              ("2"
               (expand "inv")
               (skosimp)
               (inst -1 "m!1")
               (hide -2 -3 -5 -6 2 3 4)
               (grind))))))))
        ("2" (hide -2 -3 2 3 4 5) (grind))))
      ("2"
       (lemma "reconf_length")
       (typepred "r!1")
       (expand "get_reconfigs")
       (flatten)
       (hide -2 -3 -4 -6 -7 -8 4 5)
       (inst -2 "s!1" "r!1")
       (grind))))))))
("3"
 (hide 1 2 3)
 (inst 1 "app!1")
 (lemma "int_prep_len")
 (inst -1 "s!1" "r!1")
 (split)
 (("1" (grind)) ("2" (inst 1 "app!1"))
  ("3"
   (case "r!1`end_c-r!1`start_c=1")
   (("1"
     (lemma "reconf_halt")
     (inst -1 "s!1" "r!1" "app!1")
     (typepred "r!1")
     (expand "get_reconfigs")
     (flatten)
     (expand "reconfig_end?")
     (hide -2 -4 -7)
```

```
       (split)
       (("1" (inst -1 "app!1") (grind))
        ("2" (skosimp) (reveal 1) (inst 1 "app!2"))))
      ("2"
       (typepred "r!1")
       (expand "get_reconfigs")
       (flatten)
       (hide -2 -3 -4 -5)
       (grind))))))))
("2"
 (hide 2 3 4)
 (typepred "r!1")
 (expand "get_reconfigs")
 (flatten)
 (case "r!1`end_c-r!1`start_c=1")
 (("1"
   (expand "reconfig_end?")
   (lemma "reconf_halt")
   (inst -1 "s!1" "r!1" "app!1")
   (hide -4 -6)
   (split -4)
   (("1" (inst -1 "app!1") (grind)) ("2" (propax))))
  ("2" (hide -2 -3 -4 3) (grind)))))))
```

Part IV

Example Instantiation Specification

1. Example State

```
ex_state : THEORY
BEGIN

IMPORTING state

% This theory models the entire state of the system. It essentially emulates a
% type system that would normally be created as the base of a system specification,
% but which cannot be the base in this case because the type system is used to model
% the architecture rather than the computation.
% Thus, it is ugly and confusing, but it works.
```

1.1. Data values

```
cur_alt, cur_hdg, calc_left_aileron, calc_right_aileron, calc_rudder,
   calc_elevator, adjust_left_aileron, adjust_right_aileron, adjust_rudder,
   left_aileron, right_aileron, rudder, elevator, alt_hold, hdg_hold,
   tgt_alt, tgt_hdg, ap_fcs_cmd, elec_monitor, rudder_monitor : data_id

disabled, off, engaged, ah_only: data_value
battery, alternator: data_value
a_up, a_mid_up, a_neutral, a_mid_down, a_down: data_value
r_left, r_neutral, r_right, r_ho_left, r_ho_right: data_value
e_up, e_neutral, e_down: data_value
climb, descend, hold_alt, left, right, hold_hdg, nop: data_value

% status for autopilot altitude hold mode and heading hold mode
% off means operational but not engaged
ap_mode_status: set[data_value] =
   {v: data_value | v = disabled OR v = off OR v = engaged OR v = ah_only}

% status for alternator and battery in electrics subsystem
% backup means servicable but now working as backup
elec_status: set[data_value] =
   {v: data_value | v = battery OR v = alternator}

% status for ailerons, rudder, and elevator in flight control system
aileron_status: set[data_value] =
   {v: data_value | v = a_up OR v = a_mid_up OR v = a_neutral OR v = a_mid_down OR
      v = a_down}
rudder_status: set[data_value] =
   {v: data_value | v = r_left OR v = r_neutral OR v = r_right OR v = r_ho_left OR
      v = r_ho_right}
elevator_status: set[data_value] =
   {v: data_value | v = e_up OR v = e_neutral OR v = e_down}

% command type send from autopilot to FCS, nop for not specified command
cmd_type: set[data_value] =
   {v: data_value | v = climb OR v = descend OR v = hold_alt OR v = left OR
      v = right OR v = hold_hdg OR v = nop}

values_unique: AXIOM
battery /= alternator AND
```

```
a_up /= a_mid_up AND a_up /= a_neutral AND a_up /= a_mid_down AND a_up /= a_down
AND a_mid_up /= a_neutral AND a_mid_up /= a_mid_down AND a_mid_up /= a_down AND
a_neutral /= a_mid_down AND a_neutral /= a_down AND a_mid_down /= a_down AND

r_left /= r_neutral AND r_left /= r_right AND r_left /= r_ho_left AND r_left /=
r_ho_right AND r_neutral /= r_right AND r_neutral /= r_ho_left AND r_neutral /=
r_ho_right AND r_right /= r_ho_left AND r_right /= r_ho_right AND r_ho_left /=
r_ho_right AND

e_up /= e_neutral AND e_up /= e_down AND e_neutral /= e_down AND

climb /= descend AND climb /= hold_alt AND climb /= left AND climb /= right AND
climb /= hold_hdg AND climb /= nop AND descend /= hold_alt AND descend /= left AND
descend /= right AND descend /= hold_hdg AND descend /= nop AND hold_alt /= left
AND hold_alt /= right
AND hold_alt /= hold_hdg AND hold_alt /= nop AND left /= right AND left /= hold_hdg
AND left /= nop AND right /= hold_hdg AND right /= nop AND hold_hdg /= nop
```

1.2. Type encodings

```
type_constraints: AXIOM
    FORALL (st: data_state) :
        % for sensors, rely on int as the base type for state
        % int(st(cur_alt)) AND
        % int(st(cur_hdg)) AND

        % enumerated sensors values
        elec_status(st(elec_monitor)) AND
        rudder_status(st(rudder_monitor)) AND

        % for fcs
        aileron_status(st(calc_left_aileron)) AND
        aileron_status(st(calc_right_aileron)) AND
        rudder_status(st(calc_rudder)) AND
        elevator_status(st(calc_elevator)) AND

        aileron_status(st(adjust_left_aileron)) AND
        aileron_status(st(adjust_right_aileron)) AND
        rudder_status(st(adjust_rudder)) AND

        aileron_status(st(left_aileron)) AND
        aileron_status(st(right_aileron)) AND
        rudder_status(st(rudder)) AND
        elevator_status(st(elevator)) AND

        % command send from autopilot to FCS
        cmd_type(st(ap_fcs_cmd)) AND

        % for autopilot
        ap_mode_status(st(alt_hold)) AND
        ap_mode_status(st(hdg_hold)) AND
        %int?(tgt_alt) AND
        st(tgt_hdg) >= 0 AND st(tgt_hdg) < 360

END ex_state
```

2. Environment

```
environment: THEORY
BEGIN

IMPORTING SCRAM

% The nonequal axioms are again because we have simulated refinement rather than
% actually using it.
```

2.1. Environmental State Parameters

Electrical Subsystem State

```
electrics: env_id
alternator, battery: env_param
electrics_params : set[env_param] =
    {e: env_param | e = alternator OR e = battery}
```

Autopilot Subsystem State

```
autopilot: env_id
fullsvc, alt_hold_only, disabled: env_param
autopilot_params : set[env_param] =
    {e: env_param | e = fullsvc OR e = alt_hold_only OR e = disabled}
```

Rudder State

```
rudder: env_id
working, hard_over_left, hard_over_right: env_param
rudder_params : set[env_param] =
    {e: env_param | e = working OR e = hard_over_left OR e = hard_over_right}

different_env_params: AXIOM
alternator /= battery AND alternator /= fullsvc AND alternator /= alt_hold_only
AND alternator /= disabled AND alternator /= working AND alternator /=
hard_over_left AND alternator /= hard_over_right AND battery /= fullsvc AND
battery /= alt_hold_only AND battery /= disabled AND battery /= working AND battery
/= hard_over_left AND battery /= hard_over_right AND fullsvc /= alt_hold_only AND
fullsvc /= disabled AND fullsvc /= working AND fullsvc /= hard_over_left AND
fullsvc /= hard_over_right AND alt_hold_only /= disabled AND alt_hold_only /=
working AND alt_hold_only /= hard_over_left AND alt_hold_only /= hard_over_right
AND disabled /= working AND disabled /= hard_over_left AND disabled /=
hard_over_right AND working /= hard_over_left AND working /= hard_over_right AND
hard_over_left /= hard_over_right
```

2.2. Valid Environmental States

```
proto_env_id: set[env_id] =
    {e: env_id | e = autopilot OR e = electrics OR e = rudder}

proto_valid_env: valid_env = (LAMBDA (id: env_id) :
```

```
    IF id = autopilot THEN
       {e: env_param | e = fullsvc OR e = alt_hold_only OR e = disabled}
    ELSIF id = electrics THEN
       {e: env_param | e = alternator OR e = battery}
    ELSE % rudder
       {e: env_param | e = working OR e = hard_over_left OR e = hard_over_right}
    ENDIF)

proto_envs: set[env(proto_valid_env)] = {e: env(proto_valid_env) | true}
```

2.3. Possible Transitions

```
% only degrade - can't upgrade
degraded(source, target: env(proto_valid_env)) : bool =
    (source(electrics) = alternator AND target(electrics) = battery) OR
    (source(autopilot) = fullsvc AND target(autopilot) = alt_hold_only) OR
    (source(autopilot) = alt_hold_only AND target(autopilot) = disabled) OR
    (source(autopilot) = fullsvc AND target(autopilot) = disabled) OR
    (source(rudder) = working AND target(rudder) = hard_over_left) OR
    (source(rudder) = working AND target(rudder) = hard_over_right)

proto_env_txns: set[env_txn(proto_valid_env, proto_envs)] =
    {et: env_txn(proto_valid_env, proto_envs) | degraded(et`source, et`target)}
```

2.4. Reachable Environmental States

```
proto_reachable_env : reachable_env(proto_valid_env) =
(# `D := proto_envs,
    `txns := proto_env_txns
#)

END environment
```

3. Sensors

```
sensors : THEORY
BEGIN

IMPORTING application
IMPORTING ex_state
IMPORTING environment

% Sensors has one module:
% Sensors_update updates environmental input parameters (altitude, heading).
```

3.1. Sensors_update

Sensors_update Module Types

```
% Instantiations of the abstract system state types.
sensors_update_full : module_svc
sensors_update_svcs: set[module_svc] = {svc: module_svc | svc =
sensors_update_full}

sensors_update_svclvl : data_id
sensors_update_full : data_value

sensors_update_scope : set[data_id] =
    {d: data_id | d = cur_alt OR d = cur_hdg OR d = sensors_update_svclvl}
```

Sensors_update Module Functions

```
% Simple simulation of aircraft response to flight control surface positions.
% Both ailerons, elevator, and rudder centered: no change in altitude or heading
% Elevator up: altitude reduced 5 ft.
% Elevator down: altitude increased 5 ft.
% Ailerons are not currently used to increase aircraft altitude
% Left aileron up, right aileron down: heading change of 6 degrees counterclockwise
% Right aileron up, left aileron down: heading change of 6 degrees clockwise
% Left aileron half-up, right aileron half-down:
%   heading change of 3 degrees counterclockwise
% Right aileron half-up, left aileron half-down:
%   heading change of 3 degrees clockwise
% Rudder right: heading change of 3 degrees counterclockwise
% Rudder left: heading change of 3 degrees clockwise

% change is measured in degrees clockwise
heading_input_sim(st: data_state) : int =
    LET change: int =
        % contribution from ailerons
        IF (st(left_aileron) = a_up AND st(right_aileron) = a_down) THEN -6
        ELSIF (st(left_aileron) = a_down AND st(right_aileron) = a_up) THEN 6
        ELSIF (st(left_aileron) = a_mid_up AND st(right_aileron) = a_mid_down)
            THEN -3
        ELSIF (st(left_aileron) = a_mid_down AND st(right_aileron) = a_mid_up)
            THEN 3
        ELSE 0
        ENDIF
```

```
        +
        % contribution from rudder
        IF st(rudder) = r_left THEN -3
        ELSIF st(rudder) = r_right THEN 3
        ELSE 0
        ENDIF
    IN st(cur_hdg) + change

% change is measured in altitude gain
altitude_input_sim(st: data_state) : int =
    LET change: int =
        % contribution from elevator
        IF st(elevator) = e_up THEN -5
        ELSIF st(elevator) = e_down THEN 5
        ELSE 0
        ENDIF
    IN st(cur_alt) + change

% sets environmental state to new calculated values
sensors_update_execute(sv: (sensors_update_svcs)) : func(sensors_update_scope) =
(# pre := (LAMBDA (st: data_state) : true),
    f := (LAMBDA (st: data_state) : st WITH
           [`cur_hdg := heading_input_sim(st),
            `cur_alt := altitude_input_sim(st)])
#)
```

Sensors_update Module

```
% defined to work accurately when called at any time
sensors_update: module_spec =
(#  `scope := sensors_update_scope,
    `sv := sensors_update_svcs,
    `svclvl_parm := sensors_update_svclvl,
    `pre := (LAMBDA (sv: (sensors_update_svcs)) : (LAMBDA (s: data_state) : true)),
    `trans := (LAMBDA (sv: (sensors_update_svcs)) :
                  (LAMBDA (s: data_state) : true)),
    `post := (LAMBDA (sv: (sensors_update_svcs)) :
                  (LAMBDA (s: data_state) : true)),
    `inv := (LAMBDA (sv: (sensors_update_svcs)) :(LAMBDA (s: data_state) : true))
#)
```

3.2. Sensors Application

Application Structures

```
sensors_full : app_svclvl
sensors_svcs: set[app_svclvl] = {s: app_svclvl | s = sensors_full}
sensors_modules: finite_set[module_spec] = {m: module_spec | m = sensors_update}
sensors_scope: set[data_id] = app_scope(sensors_modules)

sensors_full_svcmap: service_map(sensors_modules) =
    (lambda (m: (sensors_modules)) : sensors_update_full)
sensors_svcmap: [(sensors_svcs) -> service_map(sensors_modules)] =
    (lambda (s: (sensors_svcs)) : sensors_full_svcmap)
```

Application Functions

```
sensors_execute(s: (sensors_svcs)) : func(sensors_scope) =
    (# pre := (LAMBDA (st: data_state) : true),
       f := (LAMBDA (st: data_state) :
           sensors_update_execute
               (sensors_svcmap(sensors_full)(sensors_update))`f(st))
    #)

sensors_exec_halt(s: (sensors_svcs)) : func(sensors_scope) =
    (# pre := (LAMBDA (st: data_state) : true),
       f := (LAMBDA (st: data_state) :
           sensors_update_execute
               (sensors_svcmap(sensors_full)(sensors_update))`f(st))
    #)

sensors_prep(s: (sensors_svcs)) : func(sensors_scope) =
    (# pre := (LAMBDA (st: data_state) : true),
       f := (LAMBDA (st: data_state) : st)
    #)

sensors_halt(s: (sensors_svcs)) : func(sensors_scope) =
    (# pre := (LAMBDA (st: data_state) : true),
       f := (LAMBDA (st: data_state) : st)
    #)
```

Application Type

```
% defined to work accurately at any time
sensors_app: app_spec =
(# `svcs := sensors_svcs,
    `modules := sensors_modules,
    `svcmap := sensors_svcmap,
    `execute := sensors_execute,
    `exec_halt := sensors_exec_halt,
    `prep := sensors_prep,
    `halt := sensors_halt
#)

END sensors
```

4. Pilot Interface

```
pilot_interface : THEORY
BEGIN

IMPORTING application
IMPORTING ex_state

% Pilot_interface has one module:
% Get_input reads any new commands from the pilot and updates state accordingly.
```

4.1. Get_input

Get_input Module Types

```
get_input_standard : module_svc
get_input_svcs: set[module_svc] = {svc: module_svc | svc = get_input_standard}
get_input_svclvl : data_id
get_input_standard : data_value

get_input_scope : set[data_id] =
    {d: data_id | d = alt_hold OR d = hdg_hold OR d = tgt_alt OR d = tgt_hdg OR
                  d = get_input_svclvl}
```

Get_input Module Functions

```
% is altitude hold engaged?
altitude_hold(st: data_state) : bool
% is heading hold engaged?
heading_hold(st: data_state) : bool

% what is the current target altitude?
target_alt(st: data_state) : int
% what is the current target heading?
target_heading(st: data_state) : {i: int | i < 360}

get_input_execute(sv: (get_input_svcs)) : func(get_input_scope) =
(# pre := (LAMBDA (st: data_state) : true),
    f := (LAMBDA (st: data_state) : st WITH
           [`alt_hold := altitude_hold(st),
            `hdg_hold := heading_hold(st),
            `tgt_alt := target_alt(st),
            `tgt_hdg := target_heading(st)])
#)
```

Get_input Module

```
% can apply at any time
get_input: module_spec =
(# `scope := get_input_scope,
    `sv := get_input_svcs,
    `svclvl_parm := get_input_svclvl,
    `pre := (LAMBDA (sv: (get_input_svcs)) : (LAMBDA (s: data_state) : true)),
    `trans := (LAMBDA (sv: (get_input_svcs)) : (LAMBDA (s: data_state) : true)),
    `post := (LAMBDA (sv: (get_input_svcs)) : (LAMBDA (s: data_state) : true)),
```

```
    `inv := (LAMBDA (sv: (get_input_svcs)) :(LAMBDA (s: data_state) : true))
#)
```

4.2. Pilot_interface Application

Application Structures

```
interface_standard : app_svclvl
interface_svcs: set[app_svclvl] = {s: app_svclvl | s = interface_standard}
interface_modules: finite_set[module_spec] = {m: module_spec | m = get_input}
interface_scope: set[data_id] = app_scope(interface_modules)

interface_standard_svcmap: service_map(interface_modules) =
    (LAMBDA (m: (interface_modules)) : get_input_standard)
interface_svcmap: [(interface_svcs) -> service_map(interface_modules)] =
    (LAMBDA (s: (interface_svcs)) : interface_standard_svcmap)
```

Application Functions

```
interface_execute(s: (interface_svcs)) : func(interface_scope) =
    (# pre := (LAMBDA (st: data_state) : true),
       f := (LAMBDA (st: data_state) :
           get_input_execute
               (interface_svcmap(interface_standard)(get_input))`f(st))
    #)

interface_exec_halt(s: (interface_svcs)) : func(interface_scope) =
    (# pre := (LAMBDA (st: data_state) : true),
       f := (LAMBDA (st: data_state) :
           get_input_execute
               (interface_svcmap(interface_standard)(get_input))`f(st))
    #)

interface_prep(s: (interface_svcs)) : func(interface_scope) =
    (# pre := (LAMBDA (st: data_state) : true),
       f := (LAMBDA (st: data_state) : st)
    #)

interface_halt(s: (interface_svcs)) : func(interface_scope) =
    (# pre := (LAMBDA (st: data_state) : true),
       f := (LAMBDA (st: data_state) : st)
    #)
```

Application Type

```
pilot_interface_app: app_spec =
(#  `svcs := interface_svcs,
    `modules := interface_modules,
    `svcmap := interface_svcmap,
    `execute := interface_execute,
    `exec_halt := interface_exec_halt,
    `prep := interface_prep,
    `halt := interface_halt
#)

END pilot_interface
```

5. Flight Control System Functional Specification

```
fcs_functionality : THEORY
BEGIN

IMPORTING ex_state

% FCS has three modules:
% FCS_calc: computes normal commands based on (human or autopilot) inputs
% FCS_adjust: adjusts those commands in a rudder hardover condition
% FCS_output: writes the calculated values to actuator state variables
```

5.1. FCS_calc

```
fcs_calc_svclvl : data_id

fcs_calc_scope : set[data_id] =
    {d: data_id | d = calc_left_aileron OR d = calc_right_aileron OR
       d = calc_rudder OR d = calc_elevator OR d = fcs_calc_svclvl}

% single service
fcs_calc_execute_standard : func(fcs_calc_scope) =
(# pre := (LAMBDA (st: data_state) : true),
    f := (LAMBDA (st: data_state) :
       COND
          st(ap_fcs_cmd) = climb -> st WITH [calc_elevator := e_up],
          st(ap_fcs_cmd) = hold_alt -> st WITH [calc_elevator := e_neutral],
          st(ap_fcs_cmd) = descend -> st WITH [calc_elevator := e_down],
          st(ap_fcs_cmd) = left -> st WITH [calc_left_aileron := a_mid_up,
                                            calc_right_aileron := a_mid_down,
                                            calc_rudder := r_left],
          st(ap_fcs_cmd) = hold_hdg -> st WITH [calc_left_aileron := a_neutral,
                                            calc_right_aileron := a_neutral,
                                            calc_rudder := r_neutral],
          st(ap_fcs_cmd) = right -> st WITH [calc_left_aileron := a_mid_down,
                                            calc_right_aileron := a_mid_up,
                                            calc_rudder := r_right],
          ELSE -> st
       ENDCOND)
#)
```

5.2. FCS_adjust

```
fcs_adjust_svclvl : data_id
fcs_adjust_scope : set[data_id] =
    {d: data_id | d = adjust_left_aileron OR d = adjust_right_aileron OR
       d = adjust_rudder OR d = fcs_adjust_svclvl}

% rudder is working properly, no adjustment needed
fcs_adjust_execute_nc: func(fcs_adjust_scope) =
(# pre := (LAMBDA (st: data_state) : true),
    f := (LAMBDA (st: data_state) : st)
```

```
#)

% rudder hard over left
fcs_adjust_execute_rudder_ho_left : func(fcs_adjust_scope) =
(# pre := (LAMBDA (st: data_state) : true),
   f := (LAMBDA (st: data_state) :
      COND
         (st(calc_left_aileron) = a_mid_up AND
            st(calc_right_aileron) = a_mid_down) ->
                  st WITH [adjust_left_aileron := a_neutral,
                           adjust_right_aileron := a_neutral,
                           adjust_rudder := left],
         (st(calc_left_aileron) = a_neutral AND
            st(calc_right_aileron) = a_neutral) ->
                  st WITH [adjust_left_aileron := a_mid_down,
                           adjust_right_aileron := a_mid_up,
                           adjust_rudder := left],
         (st(calc_left_aileron) = a_mid_down AND
            st(calc_right_aileron) = a_mid_up) ->
                  st WITH [adjust_left_aileron := a_down,
                           adjust_right_aileron := a_up,
                           adjust_rudder := left],
         ELSE -> st
      ENDCOND)
#)

% rudder hard over right
fcs_adjust_execute_rudder_ho_right : func(fcs_adjust_scope) =
(# pre := (LAMBDA (st: data_state) : true),
   f := (LAMBDA (st: data_state) :
      COND
         (st(calc_left_aileron) = a_mid_up AND
            st(calc_right_aileron) = a_mid_down) ->
                  st WITH [adjust_left_aileron := a_up,
                           adjust_right_aileron := a_down,
                           adjust_rudder := right],
         (st(calc_left_aileron) = a_neutral AND
            st(calc_right_aileron) = a_neutral) ->
                  st WITH [adjust_left_aileron := a_mid_up,
                           adjust_right_aileron := a_mid_down,
                           adjust_rudder := right],
         (st(calc_left_aileron) = a_mid_down AND
            st(calc_right_aileron) = a_mid_up) ->
                  st WITH [adjust_left_aileron := a_neutral,
                           adjust_right_aileron := a_neutral,
                           adjust_rudder := right],
         ELSE -> st
      ENDCOND)
#)
```

5.3. FCS_output

```
% writes the appropriate output (normal or adjusted) to the actuators.
fcs_output_svclvl: data_id
fcs_output_scope : set[data_id] =
   {d: data_id | d = left_aileron OR d = right_aileron OR d = rudder OR
```

```
       d = elevator OR d = fcs_output_svclvl}

% not adjusted
fcs_output_standard: func(fcs_output_scope) =
(# pre := (LAMBDA (st: data_state) : true),
   f := (LAMBDA (st: data_state) : st WITH
          [elevator := st(calc_elevator),
          left_aileron := st(calc_left_aileron),
          right_aileron := st(calc_right_aileron),
          rudder := st(calc_rudder)])
#)

% adjusted
fcs_output_rudder_ho: func(fcs_output_scope) =
(# pre := (LAMBDA (st: data_state) : true),
   f := (LAMBDA (st: data_state) : st WITH
          [elevator := st(calc_elevator),
          left_aileron := st(adjust_left_aileron),
          right_aileron := st(adjust_right_aileron),
          rudder := st(adjust_rudder)])
#)

END fcs_functionality
```

6. Flight Control System

```
fcs : THEORY
BEGIN

IMPORTING application
IMPORTING fcs_functionality

% FCS has three modules:
% FCS_calc: computes normal commands based on (human or autopilot) inputs
% FCS_adjust: adjusts those commands in a rudder hardover condition.
% FCS_output: writes the calculated values to actuator state variables
```

6.1. FCS_calc

FCS _calc Module Types

```
% FCS_calc has only a single service, since FCS_adjust will alter values for
% alternative services
fcs_calc_standard: module_svc
fcs_calc_standard: data_value
fcs_calc_svcs: set[module_svc] = {svc: module_svc | svc = fcs_calc_standard}
```

FCS _calc Module Functions

```
% maps to the functionality specification
fcs_calc_execute(sv: (fcs_calc_svcs)) : func(fcs_calc_scope) =
fcs_calc_execute_standard
```

FCS _calc Module

```
% effectively no consistency conditions for the control surfaces
fcs_calc: module_spec =
(# `scope := fcs_calc_scope,
   `sv := fcs_calc_svcs,
   `svclvl_parm := fcs_calc_svclvl,
   `pre := (LAMBDA (sv: (fcs_calc_svcs)) : (LAMBDA (s: data_state) : true)),
   `trans := (LAMBDA (sv: (fcs_calc_svcs)) : (LAMBDA (s: data_state) : true)),
   `post := (LAMBDA (sv: (fcs_calc_svcs)) : (LAMBDA (s: data_state) : true)),
   `inv := (LAMBDA (sv: (fcs_calc_svcs)) : (LAMBDA (s: data_state) : true))
#)
```

6.2. FCS_adjust

FCS_adjust Module Types

```
fcs_adjust_nc, fcs_adjust_ho_left, fcs_adjust_ho_right: module_svc
fcs_adjust_svcs: set[module_svc] =
   {svc: module_svc | svc = fcs_adjust_nc OR svc = fcs_adjust_ho_left OR
      svc = fcs_adjust_ho_right}
```

```
fcs_adjust_svclvl: data_id
fcs_adjust_nc, fcs_adjust_ho_left, fcs_adjust_ho_right: data_value
```

FCS_adjust Module Functions

```
fcs_adjust_execute(sv: (fcs_adjust_svcs)) : func(fcs_adjust_scope) =
   IF sv = fcs_adjust_ho_left THEN
      fcs_adjust_execute_rudder_ho_left
   ELSIF sv = fcs_adjust_ho_right THEN
      fcs_adjust_execute_rudder_ho_right
   ELSE % sv = fcs_adjust_nc
      fcs_adjust_execute_nc
   ENDIF
```

FCS_adjust Module

```
% effectively no consistency conditions for the control surfaces
fcs_adjust: module_spec =
(# `scope := fcs_adjust_scope,
   `sv := fcs_adjust_svcs,
   `svclvl_parm := fcs_adjust_svclvl,
   `pre := (LAMBDA (sv: (fcs_adjust_svcs)) : (LAMBDA (s: data_state) : true)),
   `trans := (LAMBDA (sv: (fcs_adjust_svcs)) : (LAMBDA (s: data_state) : true)),
   `post := (LAMBDA (sv: (fcs_adjust_svcs)) : (LAMBDA (s: data_state) : true)),
   `inv := (LAMBDA (sv: (fcs_adjust_svcs)) : (LAMBDA (s: data_state) : true))
#)
```

6.3. FCS_output

FCS _output Module Types

```
% FCS_output has two services:
% FCS_output_standard: applies only FCS_calc's computations
% FCS_output_ho: adjusts the calculated outputs for the hard-over condition
fcs_output_standard, fcs_output_ho: module_svc
fcs_output_standard, fcs_output_ho: data_value
fcs_output_svcs: set[module_svc] =
   {svc: module_svc | svc = fcs_output_standard OR svc = fcs_output_ho}
```

FCS _output Module Functions

```
% maps to the appropriate functionality specification
fcs_output_execute(sv: (fcs_output_svcs)) : func(fcs_output_scope) =
   IF sv = fcs_output_ho THEN fcs_output_rudder_ho
   ELSE fcs_output_standard
   ENDIF
```

FCS _output Module

```
% effectively no consistency conditions for the control surfaces
fcs_output: module_spec =
(# `scope := fcs_output_scope,
   `sv := fcs_output_svcs,
   `svclvl_parm := fcs_output_svclvl,
   `pre := (LAMBDA (sv: (fcs_output_svcs)) : (LAMBDA (s: data_state) : true)),
   `trans := (LAMBDA (sv: (fcs_output_svcs)) : (LAMBDA (s: data_state) : true)),
```

```
    `post := (LAMBDA (sv: (fcs_output_svcs)) : (LAMBDA (s: data_state) : true)),
    `inv := (LAMBDA (sv: (fcs_output_svcs)) : (LAMBDA (s: data_state) : true))
#)
```

6.4. FCS Application

Application Structures

```
fcs_standard, fcs_rudder_ho_left, fcs_rudder_ho_right : app_svclvl
fcs_svcs: set[app_svclvl] =
    {s: app_svclvl | s = fcs_standard OR s = fcs_rudder_ho_left OR
       s = fcs_rudder_ho_right}
fcs_modules: finite_set[module_spec] =
    {m: module_spec | m = fcs_calc OR m = fcs_adjust OR m = fcs_output}
fcs_scope: set[data_id] = app_scope(fcs_modules)

fcs_standard_svcmap: service_map(fcs_modules) =
    (LAMBDA (m: (fcs_modules)) :
       IF (m = fcs_calc) THEN fcs_calc_standard
       ELSIF (m = fcs_adjust) THEN fcs_adjust_nc
       ELSE fcs_output_standard
       ENDIF)
fcs_ho_left_svcmap: service_map(fcs_modules) =
    (LAMBDA (m: (fcs_modules)) :
       IF (m = fcs_calc) THEN fcs_calc_standard
       ELSIF (m = fcs_adjust) THEN fcs_adjust_ho_left
       ELSE fcs_output_ho
       ENDIF)
fcs_ho_right_svcmap: service_map(fcs_modules) =
    (LAMBDA (m: (fcs_modules)) :
       IF (m = fcs_calc) THEN fcs_calc_standard
       ELSIF (m = fcs_adjust) THEN fcs_adjust_ho_right
       ELSE fcs_output_ho
       ENDIF)
fcs_svcmap: [(fcs_svcs) -> service_map(fcs_modules)] =
    (LAMBDA (sv: (fcs_svcs)) :
       IF (sv = fcs_rudder_ho_left) THEN fcs_ho_left_svcmap
       ELSIF (sv = fcs_rudder_ho_right) THEN fcs_ho_right_svcmap
       ELSE fcs_standard_svcmap
       ENDIF)
```

Application Functions

```
% normal function
fcs_execute(s: (fcs_svcs)) : func(fcs_scope) =
    (# pre := (LAMBDA (st: data_state) :
                  fcs_calc_execute(fcs_svcmap(s)(fcs_calc))`pre(st) AND
                  fcs_adjust_execute(fcs_svcmap(s)(fcs_adjust))`pre(st) AND
                  fcs_output_execute(fcs_svcmap(s)(fcs_output))`pre(st)),
       f :=  (LAMBDA (st: data_state) :
                  fcs_output_execute(fcs_svcmap(s)(fcs_output))`f
                  (fcs_adjust_execute(fcs_svcmap(s)(fcs_adjust))`f
                  (fcs_calc_execute(fcs_svcmap(s)(fcs_calc))`f(st))))
    #)

% same as execute
```

```
fcs_exec_halt(s: (fcs_svcs)) : func(fcs_scope) =
   (# pre := (LAMBDA (st: data_state) :
                fcs_calc_execute(fcs_svcmap(s)(fcs_calc))`pre(st) AND
                fcs_adjust_execute(fcs_svcmap(s)(fcs_adjust))`pre(st) AND
                fcs_output_execute(fcs_svcmap(s)(fcs_output))`pre(st)),
      f :=  (LAMBDA (st: data_state) :
                fcs_output_execute(fcs_svcmap(s)(fcs_output))`f
                (fcs_adjust_execute(fcs_svcmap(s)(fcs_adjust))`f
                (fcs_calc_execute(fcs_svcmap(s)(fcs_calc))`f(st))))
   #)

% change service level parameters to specify new function, otherwise no change
fcs_prep(s: (fcs_svcs)) : func(fcs_scope) =
   % rudder hard over right
   IF s = fcs_rudder_ho_right THEN
      (# pre := (LAMBDA (st: data_state) : true),
         f := (LAMBDA (st: data_state) : st WITH
                [ fcs_calc_svclvl := fcs_calc_standard,
                  fcs_adjust_svclvl := fcs_adjust_ho_right,
                  fcs_output_svclvl := fcs_output_ho])
      #)
   % rudder hard over left
   ELSIF s = fcs_rudder_ho_left THEN
      (# pre := (LAMBDA (st: data_state) : true),
         f := (LAMBDA (st: data_state) : st WITH
                [ fcs_calc_svclvl := fcs_calc_standard,
                  fcs_adjust_svclvl := fcs_adjust_ho_left,
                  fcs_output_svclvl := fcs_output_ho])
      #)
   % standard
   ELSE
      (# pre := (LAMBDA (st: data_state) : true),
         f := (LAMBDA (st: data_state) : st WITH
                [ fcs_calc_svclvl := fcs_calc_standard,
                  fcs_adjust_svclvl := fcs_adjust_nc,
                  fcs_output_svclvl := fcs_output_standard])
      #)
   ENDIF

% no execution and no other change
fcs_halt(s: (fcs_svcs)) : func(fcs_scope) =
   (# pre := (LAMBDA (st: data_state) : true),
      f := (LAMBDA (st: data_state) : st)
   #)
```

Application Type

```
fcs_app: app_spec =
(#  `svcs := fcs_svcs,
   `modules := fcs_modules,
   `svcmap := fcs_svcmap,
   `execute := fcs_execute,
   `exec_halt := fcs_exec_halt,
   `prep := fcs_prep,
   `halt := fcs_halt
#)

END fcs
```

7. Autopilot Functional Specification

```
autopilot_functionality : THEORY
BEGIN

IMPORTING application
IMPORTING ex_state

% The autopilot has one module:
% AP_compute: computes command to send to the FCS
% based on user input and sensor data
```

7.1. AP_compute

```
ap_compute_svclvl: data_id
ap_compute_standard, ap_compute_ah_only, ap_compute_off: data_value

ap_compute_scope: set[data_id] =
    {d: data_id | d = ap_fcs_cmd OR d = ap_compute_svclvl}

% give altitude changes precedence over heading changes
% since the FCS cannot effect both changes simultaneously

% 5 ft. difference in altitude or 6 degree difference in heading required to make
% adjustments so that implementation will not flutter around a value
ap_compute_execute: func(ap_compute_scope) =
(# pre := (LAMBDA (st: data_state) : true),
    f := (LAMBDA (st: data_state) :
           IF st(ap_compute_svclvl) = ap_compute_standard THEN
               % full service
               st WITH
                   [`ap_fcs_cmd :=
                       IF (st(alt_hold) = engaged AND
                              st(tgt_alt) > st(cur_alt) + 5) THEN climb
                       ELSIF (st(alt_hold) = engaged AND
                              st(tgt_alt) < st(cur_alt) - 5) THEN descend
                       ELSIF (st(hdg_hold) = engaged AND
                              ((st(tgt_hdg) < st(cur_hdg) - 6 AND
                              st(cur_hdg) - st(tgt_hdg) < 180)
                                     OR
                              (st(tgt_hdg) > st(cur_hdg) + 6 AND
                              st(tgt_hdg) - st(cur_hdg) > 180))) THEN left
                       ELSIF (st(hdg_hold) = engaged AND
                              ((st(tgt_hdg) > st(cur_hdg) + 6 AND
                              st(tgt_hdg) - st(cur_hdg) < 180)
                                     OR
                              (st(tgt_hdg) < st(cur_hdg) - 6 AND
                              st(cur_hdg) - st(tgt_hdg) > 180))) THEN right
                       ELSIF st(alt_hold) = engaged THEN hold_alt
                       ELSIF st(hdg_hold) = engaged THEN hold_hdg
                       ELSE nop
                       ENDIF]

           ELSIF st(ap_compute_svclvl) = ap_compute_ah_only THEN
```

```
            % altitude hold only
            st WITH
                [`ap_fcs_cmd :=
                   IF (st(alt_hold) = engaged AND
                          st(tgt_alt) > st(cur_alt) + 5) THEN climb
                   ELSIF (st(alt_hold) = engaged AND
                          st(tgt_alt) < st(cur_alt) - 5) THEN descend
                   ELSIF st(alt_hold) = engaged THEN hold_alt
                   ELSE nop
                   ENDIF]

         ELSE
            % off
            st WITH [`ap_fcs_cmd := nop]

         ENDIF
         )
#)

% give altitude changes precedence over heading changes
% since the FCS cannot effect both changes simultaneously
ap_compute_exec_halt: func(ap_compute_scope) =
(# pre := (LAMBDA (st: data_state) : true),
   f := (LAMBDA (st: data_state) :
         IF st(ap_compute_svclvl) = ap_compute_standard THEN
            % full service
            st WITH
                [`ap_fcs_cmd :=
                   IF (st(alt_hold) = engaged AND
                          st(tgt_alt) > st(cur_alt) + 5) THEN climb
                   ELSIF (st(alt_hold) = engaged AND
                          st(tgt_alt) < st(cur_alt) - 5) THEN descend
                   ELSIF (st(hdg_hold) = engaged AND
                          ((st(tgt_hdg) < st(cur_hdg) - 6 AND
                          st(cur_hdg) - st(tgt_hdg) < 180)
                                 OR
                          (st(tgt_hdg) > st(cur_hdg) + 6 AND
                          st(tgt_hdg) - st(cur_hdg) > 180))) THEN left
                   ELSIF (st(hdg_hold) = engaged AND
                          ((st(tgt_hdg) > st(cur_hdg) + 6 AND
                          st(tgt_hdg) - st(cur_hdg) < 180)
                                 OR
                          (st(tgt_hdg) < st(cur_hdg) - 6 AND
                          st(cur_hdg) - st(tgt_hdg) > 180))) THEN right
                   ELSIF st(alt_hold) = engaged THEN hold_alt
                   ELSIF st(hdg_hold) = engaged THEN hold_hdg
                   ELSE nop
                   ENDIF]

         ELSIF st(ap_compute_svclvl) = ap_compute_ah_only THEN
            % altitude hold only
            st WITH
                [`ap_fcs_cmd :=
                   IF (st(alt_hold) = engaged AND
                          st(tgt_alt) > st(cur_alt) + 5) THEN climb
                   ELSIF (st(alt_hold) = engaged AND
                          st(tgt_alt) < st(cur_alt) - 5) THEN descend
                   ELSIF st(alt_hold) = engaged THEN hold_alt
                   ELSE nop
```

```
                    ENDIF]

          ELSE
              % off
              st WITH [`ap_fcs_cmd := nop]

          ENDIF)

#)

END autopilot_functionality
```

8. Autopilot

```
autopilot : THEORY
BEGIN

IMPORTING autopilot_functionality
% The autopilot has one module:
% AP_compute: computes command to send to the FCS
% based on user input and sensor data
```

8.1. AP_compute Module

```
ap_compute_standard, ap_compute_ah_only, ap_compute_off: module_svc
ap_compute_svcs: set[module_svc] =
    {m: module_svc | m = ap_compute_standard OR m = ap_compute_ah_only OR
                     m = ap_compute_off}

% the autopilot is turned off when a transition occurs
% any function that the autopilot can no longer maintain is disabled
ap_compute_module: module_spec =
(#  scope := ap_compute_scope,
    sv := ap_compute_svcs,
    svclvl_parm := ap_compute_svclvl,

    % precondition must hold after first operation cycle: at this point, the
    % autopilot may be sending values
    pre := (LAMBDA (sv: (ap_compute_svcs)) :
        IF sv = ap_compute_standard THEN (LAMBDA (s: data_state) : true)
        ELSIF sv = ap_compute_ah_only THEN
            (LAMBDA (s: data_state) :
               s(ap_fcs_cmd) = climb OR s(ap_fcs_cmd) = descend OR
               s(ap_fcs_cmd) = nop)
        % ap_none_svc
        ELSE (LAMBDA (s: data_state) : s(ap_fcs_cmd) = nop)
        ENDIF),

    % autopilot does not send commands to fcs during transition
    trans := (LAMBDA (sv: (ap_compute_svcs)) :
        (LAMBDA (s: data_state) : s(ap_fcs_cmd) = nop)),

    post := (LAMBDA (sv: (ap_compute_svcs)) :
        (LAMBDA (s: data_state) : s(ap_fcs_cmd) = nop)),

    % disallows unachievable commands
    inv := (LAMBDA (sv: (ap_compute_svcs)) :
        IF sv = ap_compute_standard THEN (LAMBDA (s: data_state) : true)
        ELSIF sv = ap_compute_ah_only THEN
            (LAMBDA (s: data_state) :
               s(ap_fcs_cmd) = climb OR s(ap_fcs_cmd) = descend OR
               s(ap_fcs_cmd) = nop)
        % ap_none_svc
        ELSE (LAMBDA (s: data_state) : s(ap_fcs_cmd) = nop)
        ENDIF)
#)
```

8.2. Autopilot Application

Application Structures

```
ap_standard, ap_ah_only, ap_off: app_svclvl
ap_svcs: set[app_svclvl] =
   {s: app_svclvl | s = ap_standard OR s = ap_ah_only OR s = ap_off}
ap_modules: finite_set[module_spec] = {m: module_spec | m = ap_compute_module}
ap_scope: set[data_id] = app_scope(ap_modules)

ap_standard_svcmap: service_map(ap_modules) =
   (lambda (m: (ap_modules)) : ap_compute_standard)
ap_ah_only_svcmap: service_map(ap_modules) =
   (lambda (m: (ap_modules)) : ap_compute_ah_only)
ap_off_svcmap: service_map(ap_modules) =
   (lambda (m: (ap_modules)) : ap_compute_off)
ap_svcmap: [(ap_svcs) -> service_map(ap_modules)] =
   (lambda (s: (ap_svcs)) :
      IF s = ap_standard THEN ap_standard_svcmap
      ELSIF s = ap_ah_only THEN ap_ah_only_svcmap
      ELSE ap_off_svcmap
      ENDIF)
```

Application functions

```
% sends no commands to FCS
```

0.0.1. Prep

```
ap_prep_full: func(ap_scope) =
(# pre := (LAMBDA (st: data_state) : true),
   f := (LAMBDA (st: data_state) : st WITH
          [   `ap_fcs_cmd := nop,
              `ap_compute_svclvl := ap_compute_standard])
#)

ap_prep_ah_only: func(ap_scope) =
(# pre := (LAMBDA (st: data_state) : true),
   f := (LAMBDA (st: data_state) : st WITH
          [   `ap_fcs_cmd := nop,
              `ap_compute_svclvl := ap_compute_ah_only])
#)

ap_prep_none: func(ap_scope) =
(# pre := (LAMBDA (st: data_state) : true),
   f := (LAMBDA (st: data_state) : st WITH
          [   `ap_fcs_cmd := nop,
              `ap_compute_svclvl := ap_compute_off])
#)
```

0.0.2. Halt

```
ap_halt: func(ap_scope) =
(# pre := (LAMBDA (st: data_state) : true),
   f := (LAMBDA (st: data_state) : st WITH [`ap_fcs_cmd := nop])
#)
```

Application Type

```
autopilot_app: app_spec =
(# `svcs := ap_svcs,
   `modules := ap_modules,
   `svcmap := ap_svcmap,
   `execute := (lambda (sv: (ap_svcs)) : ap_compute_execute),
   `exec_halt := (lambda (sv: (ap_svcs)) : ap_compute_exec_halt),
   `prep := (lambda (sv: (ap_svcs)) :
      IF sv = ap_standard THEN ap_prep_full
      ELSIF sv = ap_ah_only THEN ap_prep_ah_only
      % ap_none_svc
      ELSE ap_prep_none
      ENDIF),
   `halt := (lambda (sv: (ap_svcs)) : ap_halt)
#)

END autopilot
```

9. Example System Reconfiguration Specification

```
prototype_reconf_spec: THEORY
BEGIN

IMPORTING reconf_spec
IMPORTING sensors, pilot_interface, fcs, autopilot
IMPORTING environment

% applications in the prototype system
proto_apps: set[app_spec] =
    {app: app_spec | app = sensors_app OR app = pilot_interface_app OR
        app = autopilot_app OR app = fcs_app}

% to make things easier since PVS compares equality by structure rather than name
different_apps: AXIOM
sensors_app /= pilot_interface_app AND sensors_app /= autopilot_app AND
sensors_app /= fcs_app AND pilot_interface_app /= autopilot_app AND
pilot_interface_app /= fcs_app AND autopilot_app /= fcs_app
```

9.1. System Configurations

```
% primary specification
full_service: speclvl

% partial autopilot malfunction, sufficient power, rudder OK
alt_hold_only: speclvl

% complete autopilot malfunction or battery power, rudder OK
fcs_only: speclvl

% working autopilot, sufficient power, rudder hard-over left
rudder_ho_left : speclvl

% working autopilot, sufficient power, rudder hard-over right
rudder_ho_right : speclvl

% partially malfunctioning autopilot, sufficient power, rudder hard-over left
rudder_ho_left_ah_only : speclvl

% partially malfunctioning autopilot, sufficient power, rudder hard-over right
rudder_ho_right_ah_only : speclvl

% broken autopilot, sufficient power, rudder hard-over left
rudder_ho_left_fcs_only : speclvl

% broken autopilot, sufficient power, rudder hard-over right
rudder_ho_right_fcs_only : speclvl

% yet another hack
different_lvls: AXIOM
    full_service /= alt_hold_only AND full_service /= fcs_only AND
        full_service /= rudder_ho_left AND full_service /= rudder_ho_right AND
        full_service /= rudder_ho_left_ah_only AND
```

```
        full_service /= rudder_ho_right_ah_only AND
        full_service /= rudder_ho_left_fcs_only AND
        full_service /= rudder_ho_right_fcs_only AND
        full_service /= indeterminate AND
    alt_hold_only /= fcs_only AND alt_hold_only /= rudder_ho_left AND
        alt_hold_only /= rudder_ho_right AND
        alt_hold_only /= rudder_ho_left_ah_only AND
        alt_hold_only /= rudder_ho_right_ah_only AND
        alt_hold_only /= rudder_ho_left_fcs_only AND
        alt_hold_only /= rudder_ho_right_fcs_only AND
        alt_hold_only /= indeterminate AND
    fcs_only /= rudder_ho_left AND fcs_only /= rudder_ho_right AND
        fcs_only /= rudder_ho_left_ah_only AND
        fcs_only /= rudder_ho_right_ah_only AND
        fcs_only /= rudder_ho_left_fcs_only AND
        fcs_only /= rudder_ho_right_fcs_only AND
        fcs_only /= indeterminate AND
    rudder_ho_left /= rudder_ho_right AND
        rudder_ho_left /= rudder_ho_left_ah_only AND
        rudder_ho_left /= rudder_ho_right_ah_only AND
        rudder_ho_left /= rudder_ho_left_fcs_only AND
        rudder_ho_left /= rudder_ho_right_fcs_only AND
        rudder_ho_left /= indeterminate AND
    rudder_ho_right /= rudder_ho_left_ah_only AND
        rudder_ho_right /= rudder_ho_right_ah_only AND
        rudder_ho_right /= rudder_ho_left_fcs_only AND
        rudder_ho_right /= rudder_ho_right_fcs_only AND
        rudder_ho_right /= indeterminate AND
    rudder_ho_left_ah_only /= rudder_ho_right_ah_only AND
        rudder_ho_left_ah_only /= rudder_ho_left_fcs_only AND
        rudder_ho_left_ah_only /= rudder_ho_right_fcs_only AND
        rudder_ho_left_ah_only /= indeterminate AND
    rudder_ho_right_ah_only /= rudder_ho_left_fcs_only AND
        rudder_ho_right_ah_only /= rudder_ho_right_fcs_only AND
        rudder_ho_right_ah_only /= indeterminate AND
    rudder_ho_left_fcs_only /= rudder_ho_right_fcs_only AND
        rudder_ho_left_fcs_only /= indeterminate AND
    rudder_ho_right_fcs_only /= indeterminate

% set of specification levels
proto_speclvl: set[speclvl] =
    {s: speclvl | s = full_service OR s = alt_hold_only OR s = fcs_only OR
        s = rudder_ho_left OR s = rudder_ho_right ORs = rudder_ho_left_ah_only OR
        s = rudder_ho_right_ah_only OR s = rudder_ho_left_fcs_only OR
        s = rudder_ho_right_fcs_only}

% application configurations in the system configurations
proto_sys_configs: sys_configs(proto_speclvl, proto_apps) =
    (LAMBDA (sv: (proto_speclvl)) :
        IF sv = full_service THEN
            (LAMBDA (app: app_spec) :
                IF app = sensors_app THEN sensors_full
                ELSIF app = pilot_interface_app THEN interface_standard
                ELSIF app = autopilot_app THEN ap_standard
                ELSE fcs_standard
                ENDIF)
        ELSIF sv = alt_hold_only THEN
            (LAMBDA (app: app_spec) :
                IF app = sensors_app THEN sensors_full
```

```
       ELSIF app = pilot_interface_app THEN interface_standard
       ELSIF app = autopilot_app THEN ap_ah_only
       ELSE fcs_standard
       ENDIF)
ELSIF sv = rudder_ho_left THEN
   (LAMBDA (app: app_spec) :
       IF app = sensors_app THEN sensors_full
       ELSIF app = pilot_interface_app THEN interface_standard
       ELSIF app = autopilot_app THEN ap_standard
       ELSE fcs_rudder_ho_left
       ENDIF)
ELSIF sv = rudder_ho_right THEN
   (LAMBDA (app: app_spec) :
       IF app = sensors_app THEN sensors_full
       ELSIF app = pilot_interface_app THEN interface_standard
       ELSIF app = autopilot_app THEN ap_standard
       ELSE fcs_rudder_ho_right
       ENDIF)
ELSIF sv = rudder_ho_left_ah_only THEN
   (LAMBDA (app: app_spec) :
       IF app = sensors_app THEN sensors_full
       ELSIF app = pilot_interface_app THEN interface_standard
       ELSIF app = autopilot_app THEN ap_ah_only
       ELSE fcs_rudder_ho_left
       ENDIF)
ELSIF sv = rudder_ho_right_ah_only THEN
   (LAMBDA (app: app_spec) :
       IF app = sensors_app THEN sensors_full
       ELSIF app = pilot_interface_app THEN interface_standard
       ELSIF app = autopilot_app THEN ap_ah_only
       ELSE fcs_rudder_ho_right
       ENDIF)
ELSIF sv = rudder_ho_left_fcs_only THEN
   (LAMBDA (app: app_spec) :
       IF app = sensors_app THEN sensors_full
       ELSIF app = pilot_interface_app THEN interface_standard
       ELSIF app = autopilot_app THEN ap_off
       ELSE fcs_rudder_ho_left
       ENDIF)
ELSIF sv = rudder_ho_right_fcs_only THEN
   (LAMBDA (app: app_spec) :
       IF app = sensors_app THEN sensors_full
       ELSIF app = pilot_interface_app THEN interface_standard
       ELSIF app = autopilot_app THEN ap_off
       ELSE fcs_rudder_ho_right
       ENDIF)
ELSE %fcs_only
   (LAMBDA (app: app_spec) :
       IF app = sensors_app THEN sensors_full
       ELSIF app = pilot_interface_app THEN interface_standard
       ELSIF app = autopilot_app THEN ap_off
       ELSE fcs_standard
       ENDIF)
ENDIF)
```

9.2. System Transitions

```
% only allow degradation, no repair
degraded(source, target: (proto_speclvl)) : bool =
    IF source = full_service THEN source /= target
    ELSIF source = alt_hold_only THEN
        (target /= full_service AND
            (target = rudder_ho_left_ah_only OR target = rudder_ho_right_ah_only OR
             target = rudder_ho_left_fcs_only OR target =
rudder_ho_right_fcs_only))
    ELSIF source = fcs_only THEN
           (target = rudder_ho_left_fcs_only OR target = rudder_ho_right_fcs_only)
    ELSIF source = rudder_ho_left THEN
           (target = rudder_ho_left_ah_only OR target = rudder_ho_left_fcs_only)
    ELSIF source = rudder_ho_right THEN
           (target = rudder_ho_right_ah_only OR target = rudder_ho_right_fcs_only)
    ELSIF source = rudder_ho_left_ah_only THEN target = rudder_ho_left_fcs_only
    ELSIF source = rudder_ho_right_ah_only THEN target = rudder_ho_right_fcs_only
    ELSE false
    ENDIF

% matches the operating environment at the time
appropriate(target: (proto_speclvl), env: (proto_envs)) : bool =
    IF env(rudder) = hard_over_left THEN
        (target = rudder_ho_left OR target = rudder_ho_left_ah_only OR
         target = rudder_ho_left_fcs_only)
    ELSIF env(rudder) = hard_over_right THEN
        (target = rudder_ho_right OR target = rudder_ho_right_ah_only OR
           target = rudder_ho_right_fcs_only)
    ELSE %rudder working
        (target /= rudder_ho_left AND target /= rudder_ho_left_ah_only AND
         target /= rudder_ho_left_fcs_onlyAND target /= rudder_ho_right AND
         target /= rudder_ho_right_ah_only AND target /= rudder_ho_right_fcs_only)
    ENDIF
    AND
    IF env(autopilot) = alt_hold_only THEN
        (target = alt_hold_only OR target = fcs_only OR
         target = rudder_ho_left_ah_only OR target = rudder_ho_left_fcs_onlyOR
         target = rudder_ho_right_ah_only OR target = rudder_ho_right_fcs_only)
    ELSIF env(autopilot) = disabled THEN
        (target = fcs_only OR target = rudder_ho_left_fcs_onlyOR
         target = rudder_ho_right_fcs_only)
    ELSE % autopilot is working
       true
    ENDIF
    AND
    IF env(electrics) = battery THEN
        (target = fcs_only OR target = rudder_ho_left_fcs_only OR
         target = rudder_ho_right_fcs_only)
    ELSE % running on alternator
       true
    ENDIF
    AND
    IF (env(electrics) = alternator AND env(autopilot) = fullsvc) THEN
        (target /= alt_hold_only AND target /= fcs_only AND
         target /= rudder_ho_left_ah_only AND target /= rudder_ho_left_fcs_only AND
         target /= rudder_ho_right_ah_only AND target /= rudder_ho_right_fcs_only)
    ELSIF (env(electrics) = alternator AND env(autopilot) = alt_hold_only) THEN
```

```
       (target /= fcs_only AND target /= rudder_ho_left_fcs_onlyAND
        target /= rudder_ho_right_fcs_only)
   ELSE true
   ENDIF


% set of transitions that satisfy the above predicates
proto_trans: set[transition(proto_apps, proto_speclvl, proto_valid_env,
   proto_reachable_env)] =
   {t: transition(proto_apps, proto_speclvl, proto_valid_env,
          proto_reachable_env) |
       (degraded(t`source, t`target) OR t`source = t`target) AND
       appropriate(t`target, t`trigger)}

% axiom to create a mapping that ignores nonexistant env_ids
% when testing for equality
equal_trans: AXIOM
   FORALL (v: valid_env, f1, f2: env(v)) :
       f1(electrics) = f2(electrics) AND f1(autopilot) = f2(autopilot) AND
       f1(rudder) = f2(rudder) => f1 = f2

% axiom to define equality for specification transitions
equal_spec_trans: AXIOM
   FORALL (t1, t2: (proto_trans)) :
       (t1`source = t2`source AND t1`target = t2`target AND
        t1`trigger = t2`trigger) => t1 = t2

% hack
different_env_ids: AXIOM
   electrics /= autopilot AND autopilot /= rudder AND electrics /= rudder

% lemmas to help with TCCs
cov_txn_1: LEMMA
   % Have to cover all transitions out of any start state
   FORALL (e: {e: env(proto_valid_env) |
                   e(electrics) = alternator AND e(autopilot) = fullsvc AND
                   e(rudder) = working},
           t: (proto_reachable_env`txns)) :
       t`source = e =>
          EXISTS (txn: (proto_trans)) :
             txn`source = full_service AND txn`trigger = t`target

cov_txn_2: LEMMA
   % include start state here so that there can be self-transitions from it
   FORALL (source: (proto_speclvl), t_s: (proto_trans),
          d: (proto_reachable_env`D)) :
       t_s`target = source AND t_s`trigger = d =>
          (FORALL (t_e: (proto_reachable_env`txns)) : t_e`source = d =>
             EXISTS (t_t: (proto_trans)) : t_t`trigger = t_e`target)

cov_txn_3: LEMMA
   % transitions have to be deterministic
   FORALL (source: (proto_speclvl), trigger: (proto_reachable_env`D)) :
       NOT EXISTS (t1, t2: (proto_trans)) :
          t1 /= t2 AND t1`source = source AND t1`trigger = trigger AND
          t2`source = source AND t2`trigger = trigger
```

9.3. SCRAM table

```
% System configuration information
proto_SCRAM_table: SCRAM_table(proto_apps, proto_speclvl,
                               proto_valid_env, proto_reachable_env) =
(# `configs := proto_sys_configs,
   `primary := full_service,
   `safe := {s: (proto_speclvl) | s = fcs_only},
   `start_env := {e: env(proto_valid_env) |
                  e(electrics) = alternator AND e(autopilot) = fullsvc AND
                  e(rudder) = working},
   `txns := proto_trans
#)
```

9.4. Reconfiguration Specification

```
% Function to choose the target specification of a transition
prototype_choose(sv: (proto_speclvl),
          env: env(proto_valid_env)) : (proto_speclvl) =
   IF (EXISTS (target: (proto_speclvl)) :
          EXISTS (t: (proto_SCRAM_table`txns)) :
             t`source = sv AND t`trigger = env AND t`target = target)
   THEN choose({target: (proto_speclvl) |
      EXISTS (t: (proto_SCRAM_table`txns)) :
          t`source = sv AND t`trigger = env AND t`target = target})
   ELSE sv
   ENDIF

% Time allowed for a prototype reconfiguration
proto_t: real_time = 4*cycle_time

% Complete system specification structure
prototype_reconf_spec: reconf_spec =
(# `apps := proto_apps,
   `app_seq :=
      list2finseq((: sensors_app, pilot_interface_app,
          autopilot_app, fcs_app :)),
   `S := proto_speclvl,
   `E := proto_valid_env,
   `R := proto_reachable_env,
   `SCRAM_info := proto_SCRAM_table,
   `choose := prototype_choose,
   `T := (LAMBDA (s1, s2: (proto_speclvl)) : proto_t)
#)

END prototype_reconf_spec
```

Part V

Instantiation TCCs

1. Example State

The ex_state theory has no TCCs.

2. Environment

The environment theory has no TCCs.

3. Sensors

```
% Subtype TCC generated (at line 74, column 48) for
    % (LAMBDA (s: data_state): TRUE)
    % expected type  predicate(sensors_update_scope)
  % proved - complete
sensors_update_TCC1: OBLIGATION
  FORALL (sv: (sensors_update_svcs)):
    FORALL (st: data_state): FORALL (st2: data_state): TRUE;

% The subtype TCC (at line 73, column 49) in decl sensors_update for
    %  (LAMBDA (s: data_state): TRUE)
    %expected type  predicate(sensors_update_scope)
  % is subsumed by sensors_update_TCC1

% The subtype TCC (at line 71, column 48) in decl sensors_update for
    %  (LAMBDA (s: data_state): TRUE)
    %expected type  predicate(sensors_update_scope)
  % is subsumed by sensors_update_TCC1

% The subtype TCC (at line 72, column 50) in decl sensors_update for
    %  (LAMBDA (s: data_state): TRUE)
    %expected type  predicate(sensors_update_scope)
  % is subsumed by sensors_update_TCC1

% Subtype TCC generated (at line 78, column 43) for
    % {m: module_spec | m = sensors_update}
    % expected type  finite_set[module_spec]
  % proved - complete
sensors_modules_TCC1: OBLIGATION
  is_finite[module_spec]({m: module_spec | m = sensors_update});

% Subtype TCC generated (at line 82, column 2) for
    % (LAMBDA (m: (sensors_modules)): sensors_update_full)
    % expected type  service_map(sensors_modules)
  % proved - complete
sensors_full_svcmap_TCC1: OBLIGATION
  FORALL (mod: (sensors_modules)): mod`sv(sensors_update_full);

% Subtype TCC generated (at line 88, column 55) for  sensors_update
    % expected type  (sensors_modules)
  % proved - complete
sensors_execute_TCC1: OBLIGATION
  FORALL (s: (sensors_svcs)): sensors_modules(sensors_update);
```

```
% Subtype TCC generated (at line 88, column 41) for  sensors_full
    % expected type  (sensors_svcs)
  % proved - complete
sensors_execute_TCC2: OBLIGATION
  FORALL (s: (sensors_svcs)): sensors_svcs(sensors_full);

% Subtype TCC generated (at line 88, column 26) for
    % sensors_svcmap(sensors_full)(sensors_update)
    % expected type  (sensors_update_svcs)
  % proved - complete
sensors_execute_TCC3: OBLIGATION
  FORALL (s: (sensors_svcs)):
    sensors_update_svcs(sensors_svcmap(sensors_full)(sensors_update));

% The subtype TCC (at line 94, column 55) in decl sensors_exec_halt for
sensors_update
    %expected type  (sensors_modules)
  % is subsumed by sensors_execute_TCC1

% The subtype TCC (at line 94, column 41) in decl sensors_exec_halt for
sensors_full
    %expected type  (sensors_svcs)
  % is subsumed by sensors_execute_TCC2

% The subtype TCC (at line 94, column 26) in decl sensors_exec_halt for
    %  sensors_svcmap(sensors_full)(sensors_update)
    %expected type  (sensors_update_svcs)
  % is subsumed by sensors_execute_TCC3

% Subtype TCC generated (at line 111, column 13) for  sensors_execute
    % expected type  [(sensors_svcs) -> func(app_scope(sensors_modules))]
  % proved - complete
sensors_app_TCC1: OBLIGATION
  FORALL (x1: (sensors_svcs)):
    (FORALL (st: data_state):
       (sensors_execute(x1)`pre(st) =>
         (FORALL (st2: data_state):
            (FORALL (id: (app_scope(sensors_modules))): st2(id) = st(id)) =>
             sensors_execute(x1)`pre(st2))))
     AND
     (FORALL (d: data_state, id: data_id):
        NOT app_scope(sensors_modules)(id) =>
         sensors_execute(x1)`f(d)(id) = d(id));

% Subtype TCC generated (at line 112, column 15) for  sensors_exec_halt
    % expected type  [(sensors_svcs) -> func(app_scope(sensors_modules))]
  % proved - complete
sensors_app_TCC2: OBLIGATION
  FORALL (x1: (sensors_svcs)):
    (FORALL (st: data_state):
       (sensors_exec_halt(x1)`pre(st) =>
         (FORALL (st2: data_state):
            (FORALL (id: (app_scope(sensors_modules))): st2(id) = st(id)) =>
             sensors_exec_halt(x1)`pre(st2))))
     AND
     (FORALL (d: data_state, id: data_id):
        NOT app_scope(sensors_modules)(id) =>
         sensors_exec_halt(x1)`f(d)(id) = d(id));
```

```
% Subtype TCC generated (at line 114, column 10) for  sensors_halt
    % expected type  [(sensors_svcs) -> func(app_scope(sensors_modules))]
  % proved - complete
sensors_app_TCC3: OBLIGATION
  FORALL (x1: (sensors_svcs)):
    (FORALL (st: data_state):
       (sensors_halt(x1)`pre(st) =>
         (FORALL (st2: data_state):
            (FORALL (id: (app_scope(sensors_modules))): st2(id) = st(id)) =>
             sensors_halt(x1)`pre(st2))))
     AND
     (FORALL (d: data_state, id: data_id):
        NOT app_scope(sensors_modules)(id) =>
         sensors_halt(x1)`f(d)(id) = d(id));

% Subtype TCC generated (at line 113, column 10) for  sensors_prep
    % expected type  [(sensors_svcs) -> func(app_scope(sensors_modules))]
  % proved - complete
sensors_app_TCC4: OBLIGATION
  FORALL (x1: (sensors_svcs)):
    (FORALL (st: data_state):
       (sensors_prep(x1)`pre(st) =>
         (FORALL (st2: data_state):
            (FORALL (id: (app_scope(sensors_modules))): st2(id) = st(id)) =>
             sensors_prep(x1)`pre(st2))))
     AND
     (FORALL (d: data_state, id: data_id):
        NOT app_scope(sensors_modules)(id) =>
         sensors_prep(x1)`f(d)(id) = d(id));
```

4. Pilot interface

The pilot interface theory has no TCCs.

5. FCS Functionality

```
% Disjointness TCC generated (at line 20, column 2) for
    % COND st(ap_fcs_cmd) = climb -> st WITH [calc_elevator := e_up],
    %      st(ap_fcs_cmd) = hold_alt ->
    %        st WITH [calc_elevator := e_neutral],
    %      st(ap_fcs_cmd) = descend -> st WITH [calc_elevator := e_down],
    %      st(ap_fcs_cmd) = left ->
    %        st
    %          WITH [calc_left_aileron := a_mid_up,
    %                calc_right_aileron := a_mid_down,
    %                calc_rudder := r_left],
    %      st(ap_fcs_cmd) = hold_hdg ->
    %        st
    %          WITH [calc_left_aileron := a_neutral,
    %                calc_right_aileron := a_neutral,
    %                calc_rudder := r_neutral],
```

```
    %      st(ap_fcs_cmd) = right ->
    %        st
    %          WITH [calc_left_aileron := a_mid_down,
    %                calc_right_aileron := a_mid_up,
    %                calc_rudder := r_right],
    %      ELSE -> st
    % ENDCOND
  % proved - complete
fcs_calc_execute_standard_TCC1: OBLIGATION
  FORALL (st: data_state):
         NOT (st(ap_fcs_cmd) = climb AND st(ap_fcs_cmd) = hold_alt)
     AND NOT (st(ap_fcs_cmd) = climb AND st(ap_fcs_cmd) = descend)
     AND NOT (st(ap_fcs_cmd) = climb AND st(ap_fcs_cmd) = left)
     AND NOT (st(ap_fcs_cmd) = climb AND st(ap_fcs_cmd) = hold_hdg)
     AND NOT (st(ap_fcs_cmd) = climb AND st(ap_fcs_cmd) = right)
     AND NOT (st(ap_fcs_cmd) = hold_alt AND st(ap_fcs_cmd) = descend)
     AND NOT (st(ap_fcs_cmd) = hold_alt AND st(ap_fcs_cmd) = left)
     AND NOT (st(ap_fcs_cmd) = hold_alt AND st(ap_fcs_cmd) = hold_hdg)
     AND NOT (st(ap_fcs_cmd) = hold_alt AND st(ap_fcs_cmd) = right)
     AND NOT (st(ap_fcs_cmd) = descend AND st(ap_fcs_cmd) = left)
     AND NOT (st(ap_fcs_cmd) = descend AND st(ap_fcs_cmd) = hold_hdg)
     AND NOT (st(ap_fcs_cmd) = descend AND st(ap_fcs_cmd) = right)
     AND NOT (st(ap_fcs_cmd) = left AND st(ap_fcs_cmd) = hold_hdg)
     AND NOT (st(ap_fcs_cmd) = left AND st(ap_fcs_cmd) = right)
     AND NOT (st(ap_fcs_cmd) = hold_hdg AND st(ap_fcs_cmd) = right);

% The disjointness TCC (at line 20, column 2) in decl fcs_calc_execute_standard for
    %  COND st(ap_fcs_cmd) = climb -> st WITH [calc_elevator := e_up],
    %      st(ap_fcs_cmd) = hold_alt ->
    %        st WITH [calc_elevator := e_neutral],
    %      st(ap_fcs_cmd) = descend -> st WITH [calc_elevator := e_down],
    %      st(ap_fcs_cmd) = left ->
    %        st
    %          WITH [calc_left_aileron := a_mid_up,
    %                calc_right_aileron := a_mid_down,
    %                calc_rudder := r_left],
    %      st(ap_fcs_cmd) = hold_hdg ->
    %        st
    %          WITH [calc_left_aileron := a_neutral,
    %                calc_right_aileron := a_neutral,
    %                calc_rudder := r_neutral],
    %      st(ap_fcs_cmd) = right ->
    %        st
    %          WITH [calc_left_aileron := a_mid_down,
    %                calc_right_aileron := a_mid_up,
    %                calc_rudder := r_right],
    %      ELSE -> st
    % ENDCOND
  % was not generated because it simplifies to TRUE.

% Disjointness TCC generated (at line 51, column 2) for
    % COND (st(calc_left_aileron) = a_mid_up AND
    %        st(calc_right_aileron) = a_mid_down)
    %        ->
    %        st
    %          WITH [adjust_left_aileron := a_neutral,
    %                adjust_right_aileron := a_neutral,
    %                adjust_rudder := left],
    %      (st(calc_left_aileron) = a_neutral AND
```

```
    %        st(calc_right_aileron) = a_neutral)
    %        ->
    %        st
    %          WITH [adjust_left_aileron := a_mid_down,
    %                adjust_right_aileron := a_mid_up,
    %                adjust_rudder := left],
    %      (st(calc_left_aileron) = a_mid_down AND
    %        st(calc_right_aileron) = a_mid_up)
    %        ->
    %        st
    %          WITH [adjust_left_aileron := a_down,
    %                adjust_right_aileron := a_up,
    %                adjust_rudder := left],
    %      ELSE -> st
    % ENDCOND
  % proved - complete
fcs_adjust_execute_rudder_ho_left_TCC1: OBLIGATION
  FORALL (st: data_state):
    NOT ((st(calc_left_aileron) = a_mid_up AND
           st(calc_right_aileron) = a_mid_down)
          AND
          (st(calc_left_aileron) = a_neutral AND
            st(calc_right_aileron) = a_neutral))
     AND
     NOT ((st(calc_left_aileron) = a_mid_up AND
            st(calc_right_aileron) = a_mid_down)
           AND
           (st(calc_left_aileron) = a_mid_down AND
             st(calc_right_aileron) = a_mid_up))
      AND
      NOT ((st(calc_left_aileron) = a_neutral AND
             st(calc_right_aileron) = a_neutral)
            AND
            (st(calc_left_aileron) = a_mid_down AND
              st(calc_right_aileron) = a_mid_up));

% The disjointness TCC (at line 51, column 2) in decl
fcs_adjust_execute_rudder_ho_left for
    %  COND (st(calc_left_aileron) = a_mid_up AND
    %        st(calc_right_aileron) = a_mid_down)
    %        ->
    %        st
    %          WITH [adjust_left_aileron := a_neutral,
    %                adjust_right_aileron := a_neutral,
    %                adjust_rudder := left],
    %      (st(calc_left_aileron) = a_neutral AND
    %        st(calc_right_aileron) = a_neutral)
    %        ->
    %        st
    %          WITH [adjust_left_aileron := a_mid_down,
    %                adjust_right_aileron := a_mid_up,
    %                adjust_rudder := left],
    %      (st(calc_left_aileron) = a_mid_down AND
    %        st(calc_right_aileron) = a_mid_up)
    %        ->
    %        st
    %          WITH [adjust_left_aileron := a_down,
    %                adjust_right_aileron := a_up,
    %                adjust_rudder := left],
```

```
    %      ELSE -> st
    % ENDCOND
  % was not generated because it simplifies to TRUE.

% The disjointness TCC (at line 75, column 2) in decl
fcs_adjust_execute_rudder_ho_right for
    %  COND (st(calc_left_aileron) = a_mid_up AND
    %        st(calc_right_aileron) = a_mid_down)
    %        ->
    %        st
    %          WITH [adjust_left_aileron := a_up,
    %                adjust_right_aileron := a_down,
    %                adjust_rudder := right],
    %      (st(calc_left_aileron) = a_neutral AND
    %        st(calc_right_aileron) = a_neutral)
    %        ->
    %        st
    %          WITH [adjust_left_aileron := a_mid_up,
    %                adjust_right_aileron := a_mid_down,
    %                adjust_rudder := right],
    %      (st(calc_left_aileron) = a_mid_down AND
    %        st(calc_right_aileron) = a_mid_up)
    %        ->
    %        st
    %          WITH [adjust_left_aileron := a_neutral,
    %                adjust_right_aileron := a_neutral,
    %                adjust_rudder := right],
    %      ELSE -> st
    % ENDCOND
  % is subsumed by fcs_adjust_execute_rudder_ho_left_TCC1

% The disjointness TCC (at line 75, column 2) in decl
fcs_adjust_execute_rudder_ho_right for
    %  COND (st(calc_left_aileron) = a_mid_up AND
    %        st(calc_right_aileron) = a_mid_down)
    %        ->
    %        st
    %          WITH [adjust_left_aileron := a_up,
    %                adjust_right_aileron := a_down,
    %                adjust_rudder := right],
    %      (st(calc_left_aileron) = a_neutral AND
    %        st(calc_right_aileron) = a_neutral)
    %        ->
    %        st
    %          WITH [adjust_left_aileron := a_mid_up,
    %                adjust_right_aileron := a_mid_down,
    %                adjust_rudder := right],
    %      (st(calc_left_aileron) = a_mid_down AND
    %        st(calc_right_aileron) = a_mid_up)
    %        ->
    %        st
    %          WITH [adjust_left_aileron := a_neutral,
    %                adjust_right_aileron := a_neutral,
    %                adjust_rudder := right],
    %      ELSE -> st
    % ENDCOND
  % was not generated because it simplifies to TRUE.
```

6. FCS Application

```
% Subtype TCC generated (at line 26, column 42) for
    % (LAMBDA (s: data_state): TRUE)
    % expected type  predicate(fcs_calc_scope)
  % proved - complete
fcs_calc_TCC1: OBLIGATION
  FORALL (sv: (fcs_calc_svcs)):
    FORALL (st: data_state): FORALL (st2: data_state): TRUE;

% The subtype TCC (at line 25, column 43) in decl fcs_calc for
    %  (LAMBDA (s: data_state): TRUE)
    %expected type  predicate(fcs_calc_scope)
  % is subsumed by fcs_calc_TCC1

% The subtype TCC (at line 23, column 42) in decl fcs_calc for
    %  (LAMBDA (s: data_state): TRUE)
    %expected type  predicate(fcs_calc_scope)
  % is subsumed by fcs_calc_TCC1

% The subtype TCC (at line 24, column 44) in decl fcs_calc for
    %  (LAMBDA (s: data_state): TRUE)
    %expected type  predicate(fcs_calc_scope)
  % is subsumed by fcs_calc_TCC1

% Subtype TCC generated (at line 50, column 44) for
    % (LAMBDA (s: data_state): TRUE)
    % expected type  predicate(fcs_adjust_scope)
  % proved - complete
fcs_adjust_TCC1: OBLIGATION
  FORALL (sv: (fcs_adjust_svcs)):
    FORALL (st: data_state): FORALL (st2: data_state): TRUE;

% The subtype TCC (at line 49, column 45) in decl fcs_adjust for
    %  (LAMBDA (s: data_state): TRUE)
    %expected type  predicate(fcs_adjust_scope)
  % is subsumed by fcs_adjust_TCC1

% The subtype TCC (at line 47, column 44) in decl fcs_adjust for
    %  (LAMBDA (s: data_state): TRUE)
    %expected type  predicate(fcs_adjust_scope)
  % is subsumed by fcs_adjust_TCC1

% The subtype TCC (at line 48, column 46) in decl fcs_adjust for
    %  (LAMBDA (s: data_state): TRUE)
    %expected type  predicate(fcs_adjust_scope)
  % is subsumed by fcs_adjust_TCC1

% Subtype TCC generated (at line 72, column 44) for
    % (LAMBDA (s: data_state): TRUE)
    % expected type  predicate(fcs_output_scope)
  % proved - complete
fcs_output_TCC1: OBLIGATION
  FORALL (sv: (fcs_output_svcs)):
    FORALL (st: data_state): FORALL (st2: data_state): TRUE;

% The subtype TCC (at line 71, column 45) in decl fcs_output for
    %  (LAMBDA (s: data_state): TRUE)
```

```
    %expected type  predicate(fcs_output_scope)
  % is subsumed by fcs_output_TCC1

% The subtype TCC (at line 69, column 44) in decl fcs_output for
    %  (LAMBDA (s: data_state): TRUE)
    %expected type  predicate(fcs_output_scope)
  % is subsumed by fcs_output_TCC1

% The subtype TCC (at line 70, column 46) in decl fcs_output for
    %  (LAMBDA (s: data_state): TRUE)
    %expected type  predicate(fcs_output_scope)
  % is subsumed by fcs_output_TCC1

% Subtype TCC generated (at line 79, column 1) for
    % {m: module_spec | m = fcs_calc OR m = fcs_adjust OR m = fcs_output}
    % expected type  finite_set[module_spec]
  % proved - complete
fcs_modules_TCC1: OBLIGATION
  is_finite[module_spec]
      ({m: module_spec | m = fcs_calc OR m = fcs_adjust OR m = fcs_output});

% Subtype TCC generated (at line 83, column 2) for
    % (LAMBDA (m: (fcs_modules)):
    %    IF (m = fcs_calc) THEN fcs_calc_standard
    %    ELSIF (m = fcs_adjust) THEN fcs_adjust_nc
    %    ELSE fcs_output_standard
    %    ENDIF)
    % expected type  service_map(fcs_modules)
  % proved - complete
fcs_standard_svcmap_TCC1: OBLIGATION
  FORALL (mod: (fcs_modules)):
    mod`sv
        (IF (mod = fcs_calc) THEN fcs_calc_standard
         ELSIF (mod = fcs_adjust) THEN fcs_adjust_nc
         ELSE fcs_output_standard
         ENDIF);

% Subtype TCC generated (at line 89, column 2) for
    % (LAMBDA (m: (fcs_modules)):
    %    IF (m = fcs_calc) THEN fcs_calc_standard
    %    ELSIF (m = fcs_adjust) THEN fcs_adjust_ho_left
    %    ELSE fcs_output_ho
    %    ENDIF)
    % expected type  service_map(fcs_modules)
  % proved - complete
fcs_ho_left_svcmap_TCC1: OBLIGATION
  FORALL (mod: (fcs_modules)):
    mod`sv
        (IF (mod = fcs_calc) THEN fcs_calc_standard
         ELSIF (mod = fcs_adjust) THEN fcs_adjust_ho_left
         ELSE fcs_output_ho
         ENDIF);

% Subtype TCC generated (at line 95, column 2) for
    % (LAMBDA (m: (fcs_modules)):
    %    IF (m = fcs_calc) THEN fcs_calc_standard
    %    ELSIF (m = fcs_adjust) THEN fcs_adjust_ho_right
    %    ELSE fcs_output_ho
    %    ENDIF)
```

```
    % expected type  service_map(fcs_modules)
  % proved - complete
fcs_ho_right_svcmap_TCC1: OBLIGATION
  FORALL (mod: (fcs_modules)):
    mod`sv
        (IF (mod = fcs_calc) THEN fcs_calc_standard
         ELSIF (mod = fcs_adjust) THEN fcs_adjust_ho_right
         ELSE fcs_output_ho
         ENDIF);

% Subtype TCC generated (at line 115, column 37) for  fcs_calc
    % expected type  (fcs_modules)
  % proved - complete
fcs_execute_TCC1: OBLIGATION FORALL (s: (fcs_svcs)): fcs_modules(fcs_calc);

% Subtype TCC generated (at line 115, column 23) for
    % fcs_svcmap(s)(fcs_calc)
    % expected type  (fcs_calc_svcs)
  % proved - complete
fcs_execute_TCC2: OBLIGATION
  FORALL (s: (fcs_svcs)): fcs_calc_svcs(fcs_svcmap(s)(fcs_calc));

% Subtype TCC generated (at line 114, column 39) for  fcs_adjust
    % expected type  (fcs_modules)
  % proved - complete
fcs_execute_TCC3: OBLIGATION FORALL (s: (fcs_svcs)): fcs_modules(fcs_adjust);

% Subtype TCC generated (at line 114, column 25) for
    % fcs_svcmap(s)(fcs_adjust)
    % expected type  (fcs_adjust_svcs)
  % proved - complete
fcs_execute_TCC4: OBLIGATION
  FORALL (s: (fcs_svcs)): fcs_adjust_svcs(fcs_svcmap(s)(fcs_adjust));

% Subtype TCC generated (at line 113, column 38) for  fcs_output
    % expected type  (fcs_modules)
  % proved - complete
fcs_execute_TCC5: OBLIGATION FORALL (s: (fcs_svcs)): fcs_modules(fcs_output);

% Subtype TCC generated (at line 113, column 24) for
    % fcs_svcmap(s)(fcs_output)
    % expected type  (fcs_output_svcs)
  % proved - complete
fcs_execute_TCC6: OBLIGATION
  FORALL (s: (fcs_svcs)): fcs_output_svcs(fcs_svcmap(s)(fcs_output));

% The subtype TCC (at line 109, column 36) in decl fcs_execute for  fcs_calc
    %expected type  (fcs_modules)
  % is subsumed by fcs_execute_TCC1

% The subtype TCC (at line 109, column 22) in decl fcs_execute for
fcs_svcmap(s)(fcs_calc)
    %expected type  (fcs_calc_svcs)
  % is subsumed by fcs_execute_TCC2

% The subtype TCC (at line 110, column 38) in decl fcs_execute for  fcs_adjust
    %expected type  (fcs_modules)
  % is subsumed by fcs_execute_TCC3
```

```
% The subtype TCC (at line 110, column 24) in decl fcs_execute for
    %  fcs_svcmap(s)(fcs_adjust)
    %expected type  (fcs_adjust_svcs)
  % is subsumed by fcs_execute_TCC4

% The subtype TCC (at line 111, column 38) in decl fcs_execute for  fcs_output
    %expected type  (fcs_modules)
  % is subsumed by fcs_execute_TCC5

% The subtype TCC (at line 111, column 24) in decl fcs_execute for
    %  fcs_svcmap(s)(fcs_output)
    %expected type  (fcs_output_svcs)
  % is subsumed by fcs_execute_TCC6

% The subtype TCC (at line 127, column 37) in decl fcs_exec_halt for  fcs_calc
    %expected type  (fcs_modules)
  % is subsumed by fcs_execute_TCC1

% The subtype TCC (at line 127, column 23) in decl fcs_exec_halt for
fcs_svcmap(s)(fcs_calc)
    %expected type  (fcs_calc_svcs)
  % is subsumed by fcs_execute_TCC2

% The subtype TCC (at line 126, column 39) in decl fcs_exec_halt for  fcs_adjust
    %expected type  (fcs_modules)
  % is subsumed by fcs_execute_TCC3

% The subtype TCC (at line 126, column 25) in decl fcs_exec_halt for
    %  fcs_svcmap(s)(fcs_adjust)
    %expected type  (fcs_adjust_svcs)
  % is subsumed by fcs_execute_TCC4

% The subtype TCC (at line 125, column 38) in decl fcs_exec_halt for  fcs_output
    %expected type  (fcs_modules)
  % is subsumed by fcs_execute_TCC5

% The subtype TCC (at line 125, column 24) in decl fcs_exec_halt for
    %  fcs_svcmap(s)(fcs_output)
    %expected type  (fcs_output_svcs)
  % is subsumed by fcs_execute_TCC6

% The subtype TCC (at line 121, column 36) in decl fcs_exec_halt for  fcs_calc
    %expected type  (fcs_modules)
  % is subsumed by fcs_execute_TCC1

% The subtype TCC (at line 121, column 22) in decl fcs_exec_halt for
fcs_svcmap(s)(fcs_calc)
    %expected type  (fcs_calc_svcs)
  % is subsumed by fcs_execute_TCC2

% The subtype TCC (at line 122, column 38) in decl fcs_exec_halt for  fcs_adjust
    %expected type  (fcs_modules)
  % is subsumed by fcs_execute_TCC3

% The subtype TCC (at line 122, column 24) in decl fcs_exec_halt for
    %  fcs_svcmap(s)(fcs_adjust)
    %expected type  (fcs_adjust_svcs)
  % is subsumed by fcs_execute_TCC4
```

```
% The subtype TCC (at line 123, column 38) in decl fcs_exec_halt for  fcs_output
    %expected type  (fcs_modules)
  % is subsumed by fcs_execute_TCC5

% The subtype TCC (at line 123, column 24) in decl fcs_exec_halt for
    %  fcs_svcmap(s)(fcs_output)
    %expected type  (fcs_output_svcs)
  % is subsumed by fcs_execute_TCC6

% Subtype TCC generated (at line 167, column 13) for  fcs_execute
    % expected type  [(fcs_svcs) -> func(app_scope(fcs_modules))]
  % proved - complete
fcs_app_TCC1: OBLIGATION
  FORALL (x1: (fcs_svcs)):
    (FORALL (st: data_state):
       (fcs_execute(x1)`pre(st) =>
         (FORALL (st2: data_state):
            (FORALL (id: (app_scope(fcs_modules))): st2(id) = st(id)) =>
             fcs_execute(x1)`pre(st2))))
     AND
     (FORALL (d: data_state, id: data_id):
        NOT app_scope(fcs_modules)(id) => fcs_execute(x1)`f(d)(id) = d(id));

% Subtype TCC generated (at line 168, column 15) for  fcs_exec_halt
    % expected type  [(fcs_svcs) -> func(app_scope(fcs_modules))]
  % proved - complete
fcs_app_TCC2: OBLIGATION
  FORALL (x1: (fcs_svcs)):
    (FORALL (st: data_state):
       (fcs_exec_halt(x1)`pre(st) =>
         (FORALL (st2: data_state):
            (FORALL (id: (app_scope(fcs_modules))): st2(id) = st(id)) =>
             fcs_exec_halt(x1)`pre(st2))))
     AND
     (FORALL (d: data_state, id: data_id):
        NOT app_scope(fcs_modules)(id) => fcs_exec_halt(x1)`f(d)(id) = d(id));

% Subtype TCC generated (at line 170, column 10) for  fcs_halt
    % expected type  [(fcs_svcs) -> func(app_scope(fcs_modules))]
  % proved - complete
fcs_app_TCC3: OBLIGATION
  FORALL (x1: (fcs_svcs)):
    (FORALL (st: data_state):
       (fcs_halt(x1)`pre(st) =>
         (FORALL (st2: data_state):
            (FORALL (id: (app_scope(fcs_modules))): st2(id) = st(id)) =>
             fcs_halt(x1)`pre(st2))))
     AND
     (FORALL (d: data_state, id: data_id):
        NOT app_scope(fcs_modules)(id) => fcs_halt(x1)`f(d)(id) = d(id));

% Subtype TCC generated (at line 169, column 10) for  fcs_prep
    % expected type  [(fcs_svcs) -> func(app_scope(fcs_modules))]
  % proved - complete
fcs_app_TCC4: OBLIGATION
  FORALL (x1: (fcs_svcs)):
    (FORALL (st: data_state):
       (fcs_prep(x1)`pre(st) =>
         (FORALL (st2: data_state):
```

```
            (FORALL (id: (app_scope(fcs_modules))): st2(id) = st(id)) =>
             fcs_prep(x1)`pre(st2))))
     AND
     (FORALL (d: data_state, id: data_id):
        NOT app_scope(fcs_modules)(id) => fcs_prep(x1)`f(d)(id) = d(id));
```

7. Autopilot Functionality

The autopilot functionality theory has no TCCs.

8. Autopilot

```
% Subtype TCC generated (at line 38, column 36) for
    % (LAMBDA (s: data_state): TRUE)
    % expected type  predicate(ap_compute_scope)
  % proved - complete
ap_compute_module_TCC1: OBLIGATION
  FORALL (sv: (ap_compute_svcs)):
    sv = ap_compute_standard IMPLIES
     (FORALL (st: data_state): FORALL (st2: data_state): TRUE);

% Subtype TCC generated (at line 40, column 4) for
    % (LAMBDA (s: data_state):
    %    s(ap_fcs_cmd) = climb OR
    %     s(ap_fcs_cmd) = descend OR s(ap_fcs_cmd) = nop)
    % expected type  predicate(ap_compute_scope)
  % proved - complete
ap_compute_module_TCC2: OBLIGATION
  FORALL (sv: (ap_compute_svcs)):
    NOT sv = ap_compute_standard AND sv = ap_compute_ah_only IMPLIES
     (FORALL (st: data_state):
        (st(ap_fcs_cmd) = climb OR
          st(ap_fcs_cmd) = descend OR st(ap_fcs_cmd) = nop
          =>
          (FORALL (st2: data_state):
             (FORALL (id: (ap_compute_scope)): st2(id) = st(id)) =>
              st2(ap_fcs_cmd) = climb OR
               st2(ap_fcs_cmd) = descend OR st2(ap_fcs_cmd) = nop)));

% Subtype TCC generated (at line 42, column 9) for
    % (LAMBDA (s: data_state): s(ap_fcs_cmd) = nop)
    % expected type  predicate(ap_compute_scope)
  % proved - complete
ap_compute_module_TCC3: OBLIGATION
  FORALL (sv: (ap_compute_svcs)):
    NOT sv = ap_compute_standard AND NOT sv = ap_compute_ah_only IMPLIES
     (FORALL (st: data_state):
        (st(ap_fcs_cmd) = nop =>
          (FORALL (st2: data_state):
             (FORALL (id: (ap_compute_scope)): st2(id) = st(id)) =>
              st2(ap_fcs_cmd) = nop)));

% Subtype TCC generated (at line 34, column 3) for
```

```
    % (LAMBDA (s: data_state): s(ap_fcs_cmd) = nop)
    % expected type  predicate(ap_compute_scope)
  % proved - complete
ap_compute_module_TCC4: OBLIGATION
  FORALL (sv: (ap_compute_svcs)):
    FORALL (st: data_state):
      (st(ap_fcs_cmd) = nop =>
        (FORALL (st2: data_state):
           (FORALL (id: (ap_compute_scope)): st2(id) = st(id)) =>
            st2(ap_fcs_cmd) = nop));

% The subtype TCC (at line 22, column 36) in decl ap_compute_module for
    %  (LAMBDA (s: data_state): TRUE)
    %expected type  predicate(ap_compute_scope)
  % is subsumed by ap_compute_module_TCC1

% The subtype TCC (at line 24, column 4) in decl ap_compute_module for
    %  (LAMBDA (s: data_state):
    %    s(ap_fcs_cmd) = climb OR
    %     s(ap_fcs_cmd) = descend OR s(ap_fcs_cmd) = nop)
    %expected type  predicate(ap_compute_scope)
  % is subsumed by ap_compute_module_TCC2

% The subtype TCC (at line 26, column 9) in decl ap_compute_module for
    %  (LAMBDA (s: data_state): s(ap_fcs_cmd) = nop)
    %expected type  predicate(ap_compute_scope)
  % is subsumed by ap_compute_module_TCC4

% The subtype TCC (at line 31, column 3) in decl ap_compute_module for
    %  (LAMBDA (s: data_state): s(ap_fcs_cmd) = nop)
    %expected type  predicate(ap_compute_scope)
  % is subsumed by ap_compute_module_TCC4

% Subtype TCC generated (at line 49, column 38) for
    % {m: module_spec | m = ap_compute_module}
    % expected type  finite_set[module_spec]
  % proved - complete
ap_modules_TCC1: OBLIGATION
  is_finite[module_spec]({m: module_spec | m = ap_compute_module});

% Subtype TCC generated (at line 53, column 2) for
    % (LAMBDA (m: (ap_modules)): ap_compute_standard)
    % expected type  service_map(ap_modules)
  % proved - complete
ap_standard_svcmap_TCC1: OBLIGATION
  FORALL (mod: (ap_modules)): mod`sv(ap_compute_standard);

% Subtype TCC generated (at line 55, column 2) for
    % (LAMBDA (m: (ap_modules)): ap_compute_ah_only)
    % expected type  service_map(ap_modules)
  % proved - complete
ap_ah_only_svcmap_TCC1: OBLIGATION
  FORALL (mod: (ap_modules)): mod`sv(ap_compute_ah_only);

% Subtype TCC generated (at line 57, column 2) for
    % (LAMBDA (m: (ap_modules)): ap_compute_off)
    % expected type  service_map(ap_modules)
  % proved - complete
ap_off_svcmap_TCC1: OBLIGATION
```

```
  FORALL (mod: (ap_modules)): mod`sv(ap_compute_off);

% Subtype TCC generated (at line 93, column 39) for  ap_compute_execute
    % expected type  func(app_scope(ap_modules))
  % proved - complete
autopilot_app_TCC1: OBLIGATION
  FORALL (sv: (ap_svcs)):
    (FORALL (st: data_state):
       (ap_compute_execute`pre(st) =>
         (FORALL (st2: data_state):
            (FORALL (id: (app_scope(ap_modules))): st2(id) = st(id)) =>
             ap_compute_execute`pre(st2))))
     AND
     (FORALL (d: data_state, id: data_id):
        NOT app_scope(ap_modules)(id) => ap_compute_execute`f(d)(id) = d(id));

% Subtype TCC generated (at line 94, column 41) for  ap_compute_exec_halt
    % expected type  func(app_scope(ap_modules))
  % proved - complete
autopilot_app_TCC2: OBLIGATION
  FORALL (sv: (ap_svcs)):
    (FORALL (st: data_state):
       (ap_compute_exec_halt`pre(st) =>
         (FORALL (st2: data_state):
            (FORALL (id: (app_scope(ap_modules))): st2(id) = st(id)) =>
             ap_compute_exec_halt`pre(st2))))
     AND
     (FORALL (d: data_state, id: data_id):
        NOT app_scope(ap_modules)(id) => ap_compute_exec_halt`f(d)(id) = d(id));

% Subtype TCC generated (at line 101, column 36) for  ap_halt
    % expected type  func(app_scope(ap_modules))
  % proved - complete
autopilot_app_TCC3: OBLIGATION
  FORALL (sv: (ap_svcs)):
    (FORALL (st: data_state):
       (ap_halt`pre(st) =>
         (FORALL (st2: data_state):
            (FORALL (id: (app_scope(ap_modules))): st2(id) = st(id)) =>
             ap_halt`pre(st2))))
     AND
     (FORALL (d: data_state, id: data_id):
        NOT app_scope(ap_modules)(id) => ap_halt`f(d)(id) = d(id));

% Subtype TCC generated (at line 96, column 28) for  ap_prep_full
    % expected type  func(app_scope(ap_modules))
  % proved - complete
autopilot_app_TCC4: OBLIGATION
  FORALL (sv: (ap_svcs)):
    sv = ap_standard IMPLIES
     (FORALL (st: data_state):
        (ap_prep_full`pre(st) =>
          (FORALL (st2: data_state):
             (FORALL (id: (app_scope(ap_modules))): st2(id) = st(id)) =>
              ap_prep_full`pre(st2))))
      AND
      (FORALL (d: data_state, id: data_id):
         NOT app_scope(ap_modules)(id) => ap_prep_full`f(d)(id) = d(id));
```

```
% Subtype TCC generated (at line 97, column 31) for  ap_prep_ah_only
    % expected type  func(app_scope(ap_modules))
  % proved - complete
autopilot_app_TCC5: OBLIGATION
  FORALL (sv: (ap_svcs)):
    NOT sv = ap_standard AND sv = ap_ah_only IMPLIES
     (FORALL (st: data_state):
        (ap_prep_ah_only`pre(st) =>
          (FORALL (st2: data_state):
             (FORALL (id: (app_scope(ap_modules))): st2(id) = st(id)) =>
              ap_prep_ah_only`pre(st2))))
      AND
      (FORALL (d: data_state, id: data_id):
         NOT app_scope(ap_modules)(id) => ap_prep_ah_only`f(d)(id) = d(id));

% Subtype TCC generated (at line 99, column 8) for  ap_prep_none
    % expected type  func(app_scope(ap_modules))
  % proved - complete
autopilot_app_TCC6: OBLIGATION
  FORALL (sv: (ap_svcs)):
    NOT sv = ap_standard AND NOT sv = ap_ah_only IMPLIES
     (FORALL (st: data_state):
        (ap_prep_none`pre(st) =>
          (FORALL (st2: data_state):
             (FORALL (id: (app_scope(ap_modules))): st2(id) = st(id)) =>
              ap_prep_none`pre(st2))))
      AND
      (FORALL (d: data_state, id: data_id):
         NOT app_scope(ap_modules)(id) => ap_prep_none`f(d)(id) = d(id));
```

9. Example System Reconfiguration Specification

```
% Subtype TCC generated (at line 93, column 4) for
    % (restrict[app_spec, (proto_apps), app_svclvl]
    %      ((LAMBDA (app: app_spec):
    %          IF app = sensors_app THEN sensors_full
    %          ELSIF app = pilot_interface_app THEN interface_standard
    %          ELSIF app = autopilot_app THEN ap_standard
    %          ELSE fcs_standard
    %          ENDIF)))
    % expected type  valid_app_svcs(proto_apps)
  % proved - complete
proto_sys_configs_TCC1: OBLIGATION
  FORALL (sv: (proto_speclvl)):
    sv = full_service IMPLIES
     (FORALL (app_1: (proto_apps)):
        app_1`svcs
            (restrict[app_spec, (proto_apps), app_svclvl]
                 ((LAMBDA (app: app_spec):
                     IF app = sensors_app THEN sensors_full
                     ELSIF app = pilot_interface_app THEN interface_standard
                     ELSIF app = autopilot_app THEN ap_standard
                     ELSE fcs_standard
                     ENDIF))
                 (app_1)));
```

```
% Subtype TCC generated (at line 100, column 4) for
    % (restrict[app_spec, (proto_apps), app_svclvl]
    %      ((LAMBDA (app: app_spec):
    %          IF app = sensors_app THEN sensors_full
    %          ELSIF app = pilot_interface_app THEN interface_standard
    %          ELSIF app = autopilot_app THEN ap_ah_only
    %          ELSE fcs_standard
    %          ENDIF)))
    % expected type  valid_app_svcs(proto_apps)
  % proved - complete
proto_sys_configs_TCC2: OBLIGATION
  FORALL (sv: (proto_speclvl)):
    NOT sv = full_service AND sv = alt_hold_only IMPLIES
     (FORALL (app_1: (proto_apps)):
        app_1`svcs
            (restrict[app_spec, (proto_apps), app_svclvl]
                 ((LAMBDA (app: app_spec):
                     IF app = sensors_app THEN sensors_full
                     ELSIF app = pilot_interface_app THEN interface_standard
                     ELSIF app = autopilot_app THEN ap_ah_only
                     ELSE fcs_standard
                     ENDIF))
                 (app_1)));

% Subtype TCC generated (at line 107, column 4) for
    % (restrict[app_spec, (proto_apps), app_svclvl]
    %      ((LAMBDA (app: app_spec):
    %          IF app = sensors_app THEN sensors_full
    %          ELSIF app = pilot_interface_app THEN interface_standard
    %          ELSIF app = autopilot_app THEN ap_standard
    %          ELSE fcs_rudder_ho_left
    %          ENDIF)))
    % expected type  valid_app_svcs(proto_apps)
  % proved - complete
proto_sys_configs_TCC3: OBLIGATION
  FORALL (sv: (proto_speclvl)):
    NOT sv = full_service AND NOT sv = alt_hold_only AND sv = rudder_ho_left
     IMPLIES
     (FORALL (app_1: (proto_apps)):
        app_1`svcs
            (restrict[app_spec, (proto_apps), app_svclvl]
                 ((LAMBDA (app: app_spec):
                     IF app = sensors_app THEN sensors_full
                     ELSIF app = pilot_interface_app THEN interface_standard
                     ELSIF app = autopilot_app THEN ap_standard
                     ELSE fcs_rudder_ho_left
                     ENDIF))
                 (app_1)));

% Subtype TCC generated (at line 114, column 4) for
    % (restrict[app_spec, (proto_apps), app_svclvl]
    %      ((LAMBDA (app: app_spec):
    %          IF app = sensors_app THEN sensors_full
    %          ELSIF app = pilot_interface_app THEN interface_standard
    %          ELSIF app = autopilot_app THEN ap_standard
    %          ELSE fcs_rudder_ho_right
    %          ENDIF)))
    % expected type  valid_app_svcs(proto_apps)
  % proved - complete
```

```
proto_sys_configs_TCC4: OBLIGATION
  FORALL (sv: (proto_speclvl)):
    NOT sv = full_service AND
     NOT sv = alt_hold_only AND
      NOT sv = rudder_ho_left AND sv = rudder_ho_right
     IMPLIES
     (FORALL (app_1: (proto_apps)):
        app_1`svcs
            (restrict[app_spec, (proto_apps), app_svclvl]
                 ((LAMBDA (app: app_spec):
                     IF app = sensors_app THEN sensors_full
                     ELSIF app = pilot_interface_app THEN interface_standard
                     ELSIF app = autopilot_app THEN ap_standard
                     ELSE fcs_rudder_ho_right
                     ENDIF))
                 (app_1)));

% Subtype TCC generated (at line 121, column 4) for
    % (restrict[app_spec, (proto_apps), app_svclvl]
    %       ((LAMBDA (app: app_spec):
    %           IF app = sensors_app THEN sensors_full
    %           ELSIF app = pilot_interface_app THEN interface_standard
    %           ELSIF app = autopilot_app THEN ap_ah_only
    %           ELSE fcs_rudder_ho_left
    %           ENDIF)))
    % expected type  valid_app_svcs(proto_apps)
  % proved - complete
proto_sys_configs_TCC5: OBLIGATION
  FORALL (sv: (proto_speclvl)):
         NOT sv = full_service AND NOT sv = alt_hold_only
     AND NOT sv = rudder_ho_left AND NOT sv = rudder_ho_right
     AND sv = rudder_ho_left_ah_only
     IMPLIES
     (FORALL (app_1: (proto_apps)):
        app_1`svcs
            (restrict[app_spec, (proto_apps), app_svclvl]
                 ((LAMBDA (app: app_spec):
                     IF app = sensors_app THEN sensors_full
                     ELSIF app = pilot_interface_app THEN interface_standard
                     ELSIF app = autopilot_app THEN ap_ah_only
                     ELSE fcs_rudder_ho_left
                     ENDIF))
                 (app_1)));

% Subtype TCC generated (at line 128, column 4) for
    % (restrict[app_spec, (proto_apps), app_svclvl]
    %       ((LAMBDA (app: app_spec):
    %           IF app = sensors_app THEN sensors_full
    %           ELSIF app = pilot_interface_app THEN interface_standard
    %           ELSIF app = autopilot_app THEN ap_ah_only
    %           ELSE fcs_rudder_ho_right
    %           ENDIF)))
    % expected type  valid_app_svcs(proto_apps)
  % proved - complete
proto_sys_configs_TCC6: OBLIGATION
  FORALL (sv: (proto_speclvl)):
         NOT sv = full_service AND NOT sv = alt_hold_only
     AND NOT sv = rudder_ho_left AND NOT sv = rudder_ho_right
     AND NOT sv = rudder_ho_left_ah_only AND sv = rudder_ho_right_ah_only
```

```
     IMPLIES
     (FORALL (app_1: (proto_apps)):
        app_1`svcs
            (restrict[app_spec, (proto_apps), app_svclvl]
                 ((LAMBDA (app: app_spec):
                     IF app = sensors_app THEN sensors_full
                     ELSIF app = pilot_interface_app THEN interface_standard
                     ELSIF app = autopilot_app THEN ap_ah_only
                     ELSE fcs_rudder_ho_right
                     ENDIF))
                 (app_1)));

% Subtype TCC generated (at line 135, column 4) for
    % (restrict[app_spec, (proto_apps), app_svclvl]
    %      ((LAMBDA (app: app_spec):
    %          IF app = sensors_app THEN sensors_full
    %          ELSIF app = pilot_interface_app THEN interface_standard
    %          ELSIF app = autopilot_app THEN ap_off
    %          ELSE fcs_rudder_ho_left
    %          ENDIF)))
    % expected type  valid_app_svcs(proto_apps)
  % proved - complete
proto_sys_configs_TCC7: OBLIGATION
  FORALL (sv: (proto_speclvl)):
         NOT sv = full_service AND NOT sv = alt_hold_only
     AND NOT sv = rudder_ho_left AND NOT sv = rudder_ho_right
     AND NOT sv = rudder_ho_left_ah_only AND NOT sv = rudder_ho_right_ah_only
     AND sv = rudder_ho_left_fcs_only
     IMPLIES
     (FORALL (app_1: (proto_apps)):
        app_1`svcs
            (restrict[app_spec, (proto_apps), app_svclvl]
                 ((LAMBDA (app: app_spec):
                     IF app = sensors_app THEN sensors_full
                     ELSIF app = pilot_interface_app THEN interface_standard
                     ELSIF app = autopilot_app THEN ap_off
                     ELSE fcs_rudder_ho_left
                     ENDIF))
                 (app_1)));

% Subtype TCC generated (at line 142, column 4) for
    % (restrict[app_spec, (proto_apps), app_svclvl]
    %      ((LAMBDA (app: app_spec):
    %          IF app = sensors_app THEN sensors_full
    %          ELSIF app = pilot_interface_app THEN interface_standard
    %          ELSIF app = autopilot_app THEN ap_off
    %          ELSE fcs_rudder_ho_right
    %          ENDIF)))
    % expected type  valid_app_svcs(proto_apps)
  % proved - complete
proto_sys_configs_TCC8: OBLIGATION
  FORALL (sv: (proto_speclvl)):
         NOT sv = full_service AND NOT sv = alt_hold_only
     AND NOT sv = rudder_ho_left AND NOT sv = rudder_ho_right
     AND NOT sv = rudder_ho_left_ah_only AND NOT sv = rudder_ho_right_ah_only
     AND NOT sv = rudder_ho_left_fcs_only AND sv = rudder_ho_right_fcs_only
     IMPLIES
     (FORALL (app_1: (proto_apps)):
        app_1`svcs
```

```
            (restrict[app_spec, (proto_apps), app_svclvl]
                 ((LAMBDA (app: app_spec):
                     IF app = sensors_app THEN sensors_full
                     ELSIF app = pilot_interface_app THEN interface_standard
                     ELSIF app = autopilot_app THEN ap_off
                     ELSE fcs_rudder_ho_right
                     ENDIF))
                 (app_1)));

% Subtype TCC generated (at line 149, column 4) for
    % (restrict[app_spec, (proto_apps), app_svclvl]
    %      ((LAMBDA (app: app_spec):
    %          IF app = sensors_app THEN sensors_full
    %          ELSIF app = pilot_interface_app THEN interface_standard
    %          ELSIF app = autopilot_app THEN ap_off
    %          ELSE fcs_standard
    %          ENDIF)))
    % expected type  valid_app_svcs(proto_apps)
  % proved - complete
proto_sys_configs_TCC9: OBLIGATION
  FORALL (sv: (proto_speclvl)):
         NOT sv = full_service AND NOT sv = alt_hold_only
     AND NOT sv = rudder_ho_left AND NOT sv = rudder_ho_right
     AND NOT sv = rudder_ho_left_ah_only AND NOT sv = rudder_ho_right_ah_only
     AND NOT sv = rudder_ho_left_fcs_only AND NOT sv = rudder_ho_right_fcs_only
     IMPLIES
     (FORALL (app_1: (proto_apps)):
        app_1`svcs
            (restrict[app_spec, (proto_apps), app_svclvl]
                 ((LAMBDA (app: app_spec):
                     IF app = sensors_app THEN sensors_full
                     ELSIF app = pilot_interface_app THEN interface_standard
                     ELSIF app = autopilot_app THEN ap_off
                     ELSE fcs_standard
                     ENDIF))
                 (app_1)));

% The subtype TCC (at line 93, column 4) in decl proto_sys_configs for
    %  (restrict[app_spec, (proto_apps), app_svclvl]
    %      ((LAMBDA (app: app_spec):
    %          IF app = sensors_app THEN sensors_full
    %          ELSIF app = pilot_interface_app THEN interface_standard
    %          ELSIF app = autopilot_app THEN ap_standard
    %          ELSE fcs_standard
    %          ENDIF)))
    %expected type  valid_app_svcs(proto_apps)
  % is subsumed by proto_sys_configs_TCC1

% The subtype TCC (at line 100, column 4) in decl proto_sys_configs for
    %  (restrict[app_spec, (proto_apps), app_svclvl]
    %      ((LAMBDA (app: app_spec):
    %          IF app = sensors_app THEN sensors_full
    %          ELSIF app = pilot_interface_app THEN interface_standard
    %          ELSIF app = autopilot_app THEN ap_ah_only
    %          ELSE fcs_standard
    %          ENDIF)))
    %expected type  valid_app_svcs(proto_apps)
  % is subsumed by proto_sys_configs_TCC2
```

```
% The subtype TCC (at line 107, column 4) in decl proto_sys_configs for
    %  (restrict[app_spec, (proto_apps), app_svclvl]
    %      ((LAMBDA (app: app_spec):
    %          IF app = sensors_app THEN sensors_full
    %          ELSIF app = pilot_interface_app THEN interface_standard
    %          ELSIF app = autopilot_app THEN ap_standard
    %          ELSE fcs_rudder_ho_left
    %          ENDIF)))
    %expected type  valid_app_svcs(proto_apps)
  % is subsumed by proto_sys_configs_TCC3

% The subtype TCC (at line 114, column 4) in decl proto_sys_configs for
    %  (restrict[app_spec, (proto_apps), app_svclvl]
    %      ((LAMBDA (app: app_spec):
    %          IF app = sensors_app THEN sensors_full
    %          ELSIF app = pilot_interface_app THEN interface_standard
    %          ELSIF app = autopilot_app THEN ap_standard
    %          ELSE fcs_rudder_ho_right
    %          ENDIF)))
    %expected type  valid_app_svcs(proto_apps)
  % is subsumed by proto_sys_configs_TCC4

% The subtype TCC (at line 121, column 4) in decl proto_sys_configs for
    %  (restrict[app_spec, (proto_apps), app_svclvl]
    %      ((LAMBDA (app: app_spec):
    %          IF app = sensors_app THEN sensors_full
    %          ELSIF app = pilot_interface_app THEN interface_standard
    %          ELSIF app = autopilot_app THEN ap_ah_only
    %          ELSE fcs_rudder_ho_left
    %          ENDIF)))
    %expected type  valid_app_svcs(proto_apps)
  % is subsumed by proto_sys_configs_TCC5

% The subtype TCC (at line 128, column 4) in decl proto_sys_configs for
    %  (restrict[app_spec, (proto_apps), app_svclvl]
    %      ((LAMBDA (app: app_spec):
    %          IF app = sensors_app THEN sensors_full
    %          ELSIF app = pilot_interface_app THEN interface_standard
    %          ELSIF app = autopilot_app THEN ap_ah_only
    %          ELSE fcs_rudder_ho_right
    %          ENDIF)))
    %expected type  valid_app_svcs(proto_apps)
  % is subsumed by proto_sys_configs_TCC6

% The subtype TCC (at line 135, column 4) in decl proto_sys_configs for
    %  (restrict[app_spec, (proto_apps), app_svclvl]
    %      ((LAMBDA (app: app_spec):
    %          IF app = sensors_app THEN sensors_full
    %          ELSIF app = pilot_interface_app THEN interface_standard
    %          ELSIF app = autopilot_app THEN ap_off
    %          ELSE fcs_rudder_ho_left
    %          ENDIF)))
    %expected type  valid_app_svcs(proto_apps)
  % is subsumed by proto_sys_configs_TCC7

% The subtype TCC (at line 142, column 4) in decl proto_sys_configs for
    %  (restrict[app_spec, (proto_apps), app_svclvl]
    %      ((LAMBDA (app: app_spec):
    %          IF app = sensors_app THEN sensors_full
```

```
    %          ELSIF app = pilot_interface_app THEN interface_standard
    %          ELSIF app = autopilot_app THEN ap_off
    %          ELSE fcs_rudder_ho_right
    %          ENDIF)))
    %expected type  valid_app_svcs(proto_apps)
  % is subsumed by proto_sys_configs_TCC8

% The subtype TCC (at line 149, column 4) in decl proto_sys_configs for
    %  (restrict[app_spec, (proto_apps), app_svclvl]
    %      ((LAMBDA (app: app_spec):
    %          IF app = sensors_app THEN sensors_full
    %          ELSIF app = pilot_interface_app THEN interface_standard
    %          ELSIF app = autopilot_app THEN ap_off
    %          ELSE fcs_standard
    %          ENDIF)))
    %expected type  valid_app_svcs(proto_apps)
  % is subsumed by proto_sys_configs_TCC9

% Subtype TCC generated (at line 222, column 24) for  t`trigger
    % expected type  (proto_envs)
  % proved - complete
proto_trans_TCC1: OBLIGATION
  FORALL (t:
            transition(proto_apps, proto_speclvl, proto_valid_env,
                       proto_reachable_env)):
    (degraded(t`source, t`target) OR t`source = t`target) IMPLIES
     proto_envs(t`trigger);

% Subtype TCC generated (at line 268, column 13) for  full_service
    % expected type  (proto_speclvl)
  % proved - complete
proto_SCRAM_table_TCC1: OBLIGATION proto_speclvl(full_service);

% Subtype TCC generated (at line 280, column 20) for
    % {target: (proto_speclvl) |
    %    EXISTS (t: (proto_SCRAM_table`txns)):
    %      t`source = sv AND t`trigger = env AND t`target = target}
    % expected type  p: (nonempty?[(proto_speclvl)])
  % proved - complete
prototype_choose_TCC1: OBLIGATION
  FORALL (env: env(proto_valid_env), sv: (proto_speclvl)):
    (EXISTS (target: (proto_speclvl)):
       EXISTS (t: (proto_SCRAM_table`txns)):
         t`source = sv AND t`trigger = env AND t`target = target)
     IMPLIES
     nonempty?[(proto_speclvl)]
         ({target: (proto_speclvl) |
             EXISTS (t: (proto_SCRAM_table`txns)):
               t`source = sv AND t`trigger = env AND t`target = target});

% Subtype TCC generated (at line 293, column 2) for
    % list2finseq((: sensors_app, pilot_interface_app, autopilot_app,
    %                fcs_app :))
    % expected type  finseq[(proto_apps)]
  % proved - complete
prototype_reconf_spec_TCC1: OBLIGATION
  FORALL (x1:
            below[list2finseq[app_spec]
                      ((: sensors_app, pilot_interface_app, autopilot_app,
```

```
                          fcs_app :))`length]):
    proto_apps(list2finseq[app_spec]
                  ((: sensors_app, pilot_interface_app, autopilot_app,
                      fcs_app :))`seq
                  (x1));

% Subtype TCC generated (at line 297, column 16) for  proto_SCRAM_table
    % expected type  SCRAM_table(proto_apps,
                  %             extend[speclvl,
                  %                    {sp: speclvl |
                  %                     NOT sp = indeterminate},
                  %                    bool,
                  %                    FALSE]
                  %                 (restrict[speclvl,
                  %                           {sp: speclvl |
                  %                            NOT sp = indeterminate},
                  %                           boolean]
                  %                  (proto_speclvl)),
                  %             proto_valid_env, proto_reachable_env)
  % proved - complete
prototype_reconf_spec_TCC2: OBLIGATION
       FORALL (x: speclvl):
         proto_speclvl(x) IFF
          extend[speclvl, {sp: speclvl | NOT sp = indeterminate}, bool, FALSE]
              (restrict[speclvl, {sp: speclvl | NOT sp = indeterminate},
                        boolean]
                   (proto_speclvl))
              (x)
   AND FORALL (x: speclvl):
         proto_speclvl(x) IFF
          extend[speclvl, {sp: speclvl | NOT sp = indeterminate}, bool, FALSE]
              (restrict[speclvl, {sp: speclvl | NOT sp = indeterminate},
                        boolean]
                   (proto_speclvl))
              (x)
   AND covering_txns(proto_apps,
                     extend[speclvl, {sp: speclvl | NOT sp = indeterminate},
                            bool, FALSE]
                         (restrict[speclvl,
                                   {sp: speclvl | NOT sp = indeterminate},
                                   boolean]
                              (proto_speclvl)),
                     proto_valid_env, proto_reachable_env,
                     proto_SCRAM_table`txns, proto_SCRAM_table`primary,
                     proto_SCRAM_table`start_env)
   AND FORALL (x: speclvl):
         proto_speclvl(x) IFF
          extend[speclvl, {sp: speclvl | NOT sp = indeterminate}, bool, FALSE]
              (restrict[speclvl, {sp: speclvl | NOT sp = indeterminate},
                        boolean]
                   (proto_speclvl))
              (x)
   AND extend[speclvl, {sp: speclvl | NOT sp = indeterminate}, bool, FALSE]
           (restrict[speclvl, {sp: speclvl | NOT sp = indeterminate}, boolean]
                (proto_speclvl))
           (proto_SCRAM_table`primary)
   AND FORALL (x: speclvl):
         proto_speclvl(x) IFF
          extend[speclvl, {sp: speclvl | NOT sp = indeterminate}, bool, FALSE]
```

```
              (restrict[speclvl, {sp: speclvl | NOT sp = indeterminate},
                        boolean]
                   (proto_speclvl))
              (x);

% Subtype TCC generated (at line 298, column 12) for  prototype_choose
    % expected type  [[(restrict[speclvl,
                    %             {sp: speclvl | NOT sp = indeterminate},
                    %             boolean]
                    %        (proto_speclvl)),
                    %   env(proto_valid_env)] ->
                    %    (restrict[speclvl,
                    %              {sp: speclvl | NOT sp = indeterminate},
                    %              boolean]
                    %         (proto_speclvl))]
  % proved - complete
prototype_reconf_spec_TCC3: OBLIGATION
  (FORALL (x: speclvl):
     proto_speclvl(x) IFF
      NOT x = indeterminate AND
       restrict[speclvl, {sp: speclvl | NOT sp = indeterminate}, boolean]
           (proto_speclvl)(x))
   AND
   (FORALL (x1: [(proto_speclvl), env(proto_valid_env)]):
      NOT prototype_choose(x1) = indeterminate AND
       restrict[speclvl, {sp: speclvl | NOT sp = indeterminate}, boolean]
           (proto_speclvl)(prototype_choose(x1)));

% Subtype TCC generated (at line 299, column 43) for  proto_t
    % expected type  {t: real_time | t >= 4 * cycle_time}
  % proved - complete
prototype_reconf_spec_TCC4: OBLIGATION
  FORALL (s1, s2: (proto_speclvl)): proto_t >= 4 * cycle_time;

% Subtype TCC generated (at line 299, column 8) for
    % (LAMBDA (s1, s2: (proto_speclvl)): proto_t)
    % expected type  [[(restrict[speclvl,
                    %             {sp: speclvl | NOT sp = indeterminate},
                    %             boolean]
                    %        (proto_speclvl)),
                    %   (restrict[speclvl,
                    %             {sp: speclvl | NOT sp = indeterminate},
                    %             boolean]
                    %        (proto_speclvl))] ->
                    %    {t: real_time | t >= 4 * cycle_time}]
  % proved - complete
prototype_reconf_spec_TCC5: OBLIGATION
  (FORALL (x: speclvl):
     proto_speclvl(x) IFF
      NOT x = indeterminate AND
       restrict[speclvl, {sp: speclvl | NOT sp = indeterminate}, boolean]
           (proto_speclvl)(x))
   AND
   (FORALL (x: speclvl):
      proto_speclvl(x) IFF
       NOT x = indeterminate AND
        restrict[speclvl, {sp: speclvl | NOT sp = indeterminate}, boolean]
            (proto_speclvl)(x));
```

Part VI

Instantiation Proofs

```
Proof summary for theory reconf_spec
   reconf_spec_TCC1.....................proved - complete   [shostak](1.99 s)
   post_TCC1............................proved - complete   [shostak](1.77 s)
   post_TCC2............................proved - complete   [shostak](8.12 s)
   Theory totals: 3 formulas, 3 attempted, 3 succeeded (11.88 s)

Proof summary for theory sensors
   sensors_update_TCC1..................proved - complete   [shostak](0.02 s)
   sensors_modules_TCC1.................proved - complete   [shostak](0.12 s)
   sensors_full_svcmap_TCC1.............proved - complete   [shostak](0.08 s)
   sensors_execute_TCC1.................proved - complete   [shostak](0.04 s)
   sensors_execute_TCC2.................proved - complete   [shostak](0.02 s)
   sensors_execute_TCC3.................proved - complete   [shostak](0.03 s)
   sensors_app_TCC1.....................proved - complete   [shostak](0.71 s)
   sensors_app_TCC2.....................proved - complete   [shostak](0.92 s)
   sensors_app_TCC3.....................proved - complete   [shostak](0.10 s)
   sensors_app_TCC4.....................proved - complete   [shostak](0.09 s)
   Theory totals: 10 formulas, 10 attempted, 10 succeeded (2.13 s)

Proof summary for theory SCRAM
   Theory totals: 0 formulas, 0 attempted, 0 succeeded (0.00 s)

Proof summary for theory environment
   Theory totals: 0 formulas, 0 attempted, 0 succeeded (0.00 s)

Proof summary for theory pilot_interface
   target_heading_TCC1..................proved - complete   [shostak](0.02 s)
   get_input_TCC1.......................proved - complete   [shostak](0.02 s)
   interface_modules_TCC1...............proved - complete   [shostak](0.11 s)
   interface_standard_svcmap_TCC1.......proved - complete   [shostak](0.09 s)
   interface_execute_TCC1...............proved - complete   [shostak](0.04 s)
   interface_execute_TCC2...............proved - complete   [shostak](0.02 s)
   interface_execute_TCC3...............proved - complete   [shostak](0.03 s)
   pilot_interface_app_TCC1.............proved - complete   [shostak](0.34 s)
   pilot_interface_app_TCC2.............proved - complete   [shostak](0.32 s)
   pilot_interface_app_TCC3.............proved - complete   [shostak](0.10 s)
   pilot_interface_app_TCC4.............proved - complete   [shostak](0.09 s)
   Theory totals: 11 formulas, 11 attempted, 11 succeeded (1.18 s)

Proof summary for theory fcs
   fcs_calc_TCC1........................proved - complete   [shostak](0.01 s)
   fcs_adjust_TCC1......................proved - complete   [shostak](0.02 s)
   fcs_output_TCC1......................proved - complete   [shostak](0.01 s)
   fcs_modules_TCC1.....................proved - complete   [shostak](0.99 s)
   fcs_standard_svcmap_TCC1.............proved - complete   [shostak](0.31 s)
   fcs_ho_left_svcmap_TCC1..............proved - complete   [shostak](0.31 s)
   fcs_ho_right_svcmap_TCC1.............proved - complete   [shostak](0.30 s)
   fcs_execute_TCC1.....................proved - complete   [shostak](0.09 s)
   fcs_execute_TCC2.....................proved - complete   [shostak](0.07 s)
   fcs_execute_TCC3.....................proved - complete   [shostak](0.10 s)
   fcs_execute_TCC4.....................proved - complete   [shostak](0.60 s)
   fcs_execute_TCC5.....................proved - complete   [shostak](0.10 s)
   fcs_execute_TCC6.....................proved - complete   [shostak](0.32 s)
   fcs_app_TCC1.........................proved - complete   [shostak](0.25 s)
   fcs_app_TCC2.........................proved - complete   [shostak](0.20 s)
   fcs_app_TCC3.........................proved - complete   [shostak](0.17 s)
   fcs_app_TCC4.........................proved - complete   [shostak](0.16 s)
   Theory totals: 17 formulas, 17 attempted, 17 succeeded (4.01 s)
```

```
Proof summary for theory fcs_functionality
   fcs_calc_execute_standard_TCC1........proved - complete   [shostak](6.16 s)
   fcs_adjust_execute_rudder_ho_left_TCC1proved - complete   [shostak](2.88 s)
   Theory totals: 2 formulas, 2 attempted, 2 succeeded (9.04 s)

Proof summary for theory autopilot
   ap_compute_module_TCC1................proved - complete   [shostak](0.01 s)
   ap_compute_module_TCC2................proved - complete   [shostak](0.11 s)
   ap_compute_module_TCC3................proved - complete   [shostak](0.07 s)
   ap_compute_module_TCC4................proved - complete   [shostak](0.06 s)
   ap_modules_TCC1.......................proved - complete   [shostak](0.10 s)
   ap_standard_svcmap_TCC1...............proved - complete   [shostak](0.13 s)
   ap_ah_only_svcmap_TCC1................proved - complete   [shostak](0.13 s)
   ap_off_svcmap_TCC1....................proved - complete   [shostak](0.12 s)
   autopilot_app_TCC1....................proved - complete   [shostak](0.65 s)
   autopilot_app_TCC2....................proved - complete   [shostak](0.69 s)
   autopilot_app_TCC3....................proved - complete   [shostak](0.20 s)
   autopilot_app_TCC4....................proved - complete   [shostak](0.16 s)
   autopilot_app_TCC5....................proved - complete   [shostak](0.17 s)
   autopilot_app_TCC6....................proved - complete   [shostak](0.17 s)
   Theory totals: 14 formulas, 14 attempted, 14 succeeded (2.77 s)

Proof summary for theory application
   post_TCC1.............................proved - complete   [shostak](0.63 s)
   Theory totals: 1 formulas, 1 attempted, 1 succeeded (0.63 s)

Proof summary for theory module
   Theory totals: 0 formulas, 0 attempted, 0 succeeded (0.00 s)

Proof summary for theory state
   Theory totals: 0 formulas, 0 attempted, 0 succeeded (0.00 s)

Proof summary for theory ex_state
   Theory totals: 0 formulas, 0 attempted, 0 succeeded (0.00 s)

Proof summary for theory autopilot_functionality
   Theory totals: 0 formulas, 0 attempted, 0 succeeded (0.00 s)

Proof summary for theory prototype_reconf_spec
   proto_sys_configs_TCC1.............proved - complete   [shostak](   4.63 s)
   proto_sys_configs_TCC2.............proved - complete   [shostak](   2.78 s)
   proto_sys_configs_TCC3.............proved - complete   [shostak](   2.44 s)
   proto_sys_configs_TCC4.............proved - complete   [shostak](   2.40 s)
   proto_sys_configs_TCC5.............proved - complete   [shostak](   3.16 s)
   proto_sys_configs_TCC6.............proved - complete   [shostak](   3.12 s)
   proto_sys_configs_TCC7.............proved - complete   [shostak](   3.08 s)
   proto_sys_configs_TCC8.............proved - complete   [shostak](   3.09 s)
   proto_sys_configs_TCC9.............proved - complete   [shostak](   3.08 s)
   proto_trans_TCC1...................proved - complete   [shostak](   0.07 s)
   cov_txn_1..........................proved - complete   [shostak](  84.49 s)
   cov_txn_2..........................proved - complete   [shostak](  86.63 s)
   cov_txn_3..........................proved - complete   [shostak](2908.07 s)
   proto_SCRAM_table_TCC1.............proved - complete   [shostak](   0.02 s)
   prototype_choose_TCC1..............proved - complete   [shostak](   0.07 s)
   prototype_reconf_spec_TCC1.........proved - complete   [shostak](   2.10 s)
   prototype_reconf_spec_TCC2.........proved - complete   [shostak](  10.22 s)
   prototype_reconf_spec_TCC3.........proved - complete   [shostak](   3.29 s)
   prototype_reconf_spec_TCC4.........proved - complete   [shostak](   0.44 s)
   prototype_reconf_spec_TCC5.........proved - complete   [shostak](   4.74 s)
```

```
    Theory totals: 20 formulas, 20 attempted, 20 succeeded (3127.92 s)
Grand Totals: 78 proofs, 78 attempted, 78 succeeded (3159.56 s)
```

1. Example State

The example state theory has no proofs.

2. Environment

The environment theory has no proofs.

3. Sensors

```
Proof scripts for file sensors-fm.pvs:


sensors.sensors_update_TCC1: proved - complete [shostak](0.80 s)

("" (grind))


sensors.sensors_modules_TCC1: proved - complete [shostak](13.06 s)

(""
 (expand "is_finite")
 (inst 1 "1" "(lambda (m: {m: module_spec | m = sensors_update}) : 0)")
 (grind))


sensors.sensors_full_svcmap_TCC1: proved - complete [shostak](0.86 s)

("" (grind))


sensors.sensors_execute_TCC1: proved - complete [shostak](0.74 s)

("" (grind))


sensors.sensors_execute_TCC2: proved - complete [shostak](0.66 s)

("" (grind))


sensors.sensors_execute_TCC3: proved - complete [shostak](0.72 s)

("" (grind))


sensors.sensors_app_TCC1: proved - complete [shostak](22.49 s)

(""
 (skosimp*)
 (ground)
 (("1" (skosimp*) (grind))
```

```
  ("2"
   (skosimp*)
   (typepred "sensors_execute(x1!1)`f")
   (inst -1 "d!1" "id!1")
   (grind))))


sensors.sensors_app_TCC2: proved - complete [shostak](7.63 s)

(""
 (ground)
 (skosimp)
 (ground)
 (("1" (grind))
  ("2" (skosimp) (typepred "sensors_exec_halt(x1!1)`f") (grind))))


sensors.sensors_app_TCC3: proved - complete [shostak](4.19 s)

(""
 (skosimp)
 (ground)
 (("1" (grind))
  ("2" (skosimp) (typepred "sensors_halt(x1!1)`f") (grind))))


sensors.sensors_app_TCC4: proved - complete [shostak](3.49 s)

(""
 (skosimp)
 (ground)
 (("1" (grind)) ("2" (typepred "sensors_prep(x1!1)`f") (grind))))
```

4. Pilot Interface

The pilot interface theory has no proofs.

5. FCS Functionality

```
Proof scripts for file fcs_functionality-fm.pvs:


fcs_functionality.fcs_calc_execute_standard_TCC1: proved - complete
[shostak](57.29 s)

("" (lemma "values_unique") (grind) (postpone))


fcs_functionality.fcs_adjust_execute_rudder_ho_left_TCC1: proved - complete
[shostak](8.41 s)

("" (lemma "values_unique") (grind))
```

6. FCS

```
Proof scripts for file fcs-fm.pvs:


fcs.fcs_calc_TCC1: proved - complete [shostak](0.01 s)

("" (grind))


fcs.fcs_adjust_TCC1: proved - complete [shostak](0.01 s)

("" (grind))


fcs.fcs_output_TCC1: proved - complete [shostak](0.01 s)

("" (grind))


fcs.fcs_modules_TCC1: proved - complete [shostak](1.17 s)

(""
 (expand "is_finite")
 (inst 1 "3"
  "(lambda (m: {m: module_spec |
                      m = fcs_calc OR m = fcs_adjust OR m = fcs_output}) : IF m
= fcs_calc THEN 0 ELSIF m = fcs_adjust THEN 1 ELSE 2 ENDIF)")
 (grind))


fcs.fcs_standard_svcmap_TCC1: proved - complete [shostak](0.33 s)

("" (grind))


fcs.fcs_ho_left_svcmap_TCC1: proved - complete [shostak](0.32 s)

("" (grind))


fcs.fcs_ho_right_svcmap_TCC1: proved - complete [shostak](0.32 s)

("" (grind))


fcs.fcs_execute_TCC1: proved - complete [shostak](0.10 s)

("" (grind))


fcs.fcs_execute_TCC2: proved - complete [shostak](1.22 s)

("" (grind))


fcs.fcs_execute_TCC3: proved - complete [shostak](0.90 s)
```

```
("" (grind))
```

fcs.fcs_execute_TCC4: proved - complete [shostak](7.86 s)

```
(""
 (skosimp)
 (typepred "fcs_svcmap(s!1)")
 (inst -1 "fcs_adjust")
 (("1" (expand "fcs_adjust_svcs") (grind)) ("2" (grind))))
```

fcs.fcs_execute_TCC5: proved - complete [shostak](0.90 s)

```
("" (grind))
```

fcs.fcs_execute_TCC6: proved - complete [shostak](10.91 s)

```
(""
 (skosimp)
 (typepred "fcs_svcmap(s!1)")
 (inst -1 "fcs_output")
 (("1" (grind)) ("2" (grind))))
```

fcs.fcs_app_TCC1: proved - complete [shostak](20.27 s)

```
(""
 (skosimp)
 (ground)
 (("1"
   (skosimp)
   (skosimp)
   (typepred "fcs_execute(x1!1)`pre")
   (inst -1 "st!1")
   (split)
   (("1"
     (inst -1 "st2!1")
     (split)
     (("1" (propax)) ("2" (hide (-1 2)) (grind))))
    ("2" (propax))))
  ("2"
   (typepred "fcs_execute(x1!1)`f")
   (skosimp)
   (inst -1 "d!1" "id!1")
   (split)
   (("1" (propax)) ("2" (hide 2) (grind))))))
```

fcs.fcs_app_TCC2: proved - complete [shostak](17.46 s)

```
(""
 (skosimp)
 (split)
 (("1"
   (skosimp)
   (skosimp*)
   (typepred "fcs_exec_halt(x1!1)`pre")
```

```
   (inst -1 "st!1")
   (split)
   (("1"
     (inst -1 "st2!1")
     (split)
     (("1" (propax))
      ("2"
       (hide (-1 2))
       (skosimp)
       (inst -1 "id!1")
       (typepred "id!1")
       (hide 2)
       (grind))))
    ("2" (propax))))
  ("2"
   (skosimp)
   (typepred "fcs_exec_halt(x1!1)`f")
   (inst -1 "d!1" "id!1")
   (split)
   (("1" (propax)) ("2" (hide 2) (grind))))))


fcs.fcs_app_TCC3: proved - complete [shostak](17.73 s)

(""
 (skosimp)
 (split)
 (("1"
   (skosimp*)
   (typepred "fcs_halt(x1!1)`pre")
   (inst -1 "st!1")
   (split)
   (("1"
     (inst -1 "st2!1")
     (split)
     (("1" (propax))
      ("2"
       (hide (-1 2))
       (skosimp)
       (inst -1 "id!1")
       (hide 2)
       (typepred "id!1")
       (grind))))
    ("2" (propax))))
  ("2"
   (skosimp)
   (typepred "fcs_halt(x1!1)`f")
   (inst -1 "d!1" "id!1")
   (split)
   (("1" (propax)) ("2" (hide 2) (grind))))))


fcs.fcs_app_TCC4: proved - complete [shostak](15.09 s)

(""
 (skosimp)
 (split)
 (("1"
   (skosimp)
```

```
   (skosimp)
   (typepred "fcs_prep(x1!1)`pre")
   (inst -1 "st!1")
   (split)
   (("1"
     (inst -1 "st2!1")
     (split)
     (("1" (propax))
      ("2"
       (hide (-1 2))
       (skosimp)
       (inst -1 "id!1")
       (typepred "id!1")
       (hide 2)
       (grind))))
    ("2" (propax))))
  ("2"
   (skosimp)
   (typepred "fcs_prep(x1!1)`f")
   (inst -1 "d!1" "id!1")
   (split)
   (("1" (propax)) ("2" (hide 2) (grind))))))
```

7. Autopilot Functionality

The autopilot functionality theory has no proofs.

8. Autopilot

```
Proof scripts for file autopilot-fm.pvs:


autopilot.ap_compute_module_TCC1: proved - complete [shostak](0.29 s)

("" (grind))


autopilot.ap_compute_module_TCC2: proved - complete [shostak](11.87 s)

(""
 (skosimp)
 (skosimp)
 (skosimp*)
 (hide (-1 1))
 (inst -2 "ap_fcs_cmd")
 (("1" (split) (("1" (grind)) ("2" (grind)) ("3" (grind))))
  ("2" (hide (-1 2 3 4)) (grind))))


autopilot.ap_compute_module_TCC3: proved - complete [shostak](4.84 s)

(""
 (skosimp)
```

```
 (skosimp)
 (skosimp*)
 (hide (1 2))
 (inst -2 "ap_fcs_cmd")
 (("1" (grind)) ("2" (grind))))
```

autopilot.ap_compute_module_TCC4: proved - complete [shostak](2.75 s)

```
("" (skosimp*) (inst -2 "ap_fcs_cmd") (("1" (grind)) ("2" (grind))))
```

autopilot.ap_modules_TCC1: proved - complete [shostak](0.38 s)

```
(""
 (expand "is_finite")
 (expand "injective?")
 (inst 1 "1"
  "(lambda (m: {m: module_spec | m = ap_compute_module}) : 0)"))
```

autopilot.ap_standard_svcmap_TCC1: proved - complete [shostak](0.32 s)

```
("" (grind))
```

autopilot.ap_ah_only_svcmap_TCC1: proved - complete [shostak](0.27 s)

```
("" (grind))
```

autopilot.ap_off_svcmap_TCC1: proved - complete [shostak](0.27 s)

```
("" (grind))
```

autopilot.autopilot_app_TCC1: proved - complete [shostak](25.03 s)

```
(""
 (skosimp)
 (split)
 (("1"
   (skosimp)
   (skosimp*)
   (typepred "ap_compute_execute`pre")
   (inst -1 "st!1")
   (split)
   (("1"
     (inst -1 "st2!1")
     (split)
     (("1" (propax))
      ("2"
       (hide (-1 2))
       (skosimp)
       (inst -1 "id!1")
       (typepred "id!1")
       (hide 2)
       (grind)
       (("1" (inst 1 "ap_compute_module") (grind))
```

```
        ("2" (inst 1 "ap_compute_module") (grind))))))
    ("2" (propax))))
  ("2"
   (skosimp)
   (typepred "ap_compute_execute`f")
   (inst -1 "d!1" "id!1")
   (split)
   (("1" (propax))
    ("2"
     (hide 2)
     (grind)
     (("1" (inst 1 "ap_compute_module") (grind))
      ("2" (inst 1 "ap_compute_module") (grind)))))))
```

autopilot.autopilot_app_TCC2: proved - complete [shostak](24.25 s)

```
(""
 (skosimp)
 (split)
 (("1"
   (skosimp*)
   (typepred "ap_compute_exec_halt`pre")
   (inst -1 "st!1")
   (split)
   (("1"
     (inst -1 "st2!1")
     (split)
     (("1" (propax))
      ("2"
       (hide (-1 2))
       (skosimp)
       (inst -1 "id!1")
       (typepred "id!1")
       (grind)
       (("1" (inst 1 "ap_compute_module") (grind))
        ("2" (inst 1 "ap_compute_module") (grind))))))
    ("2" (propax))))
  ("2"
   (skosimp)
   (typepred "ap_compute_exec_halt`f")
   (inst -1 "d!1" "id!1")
   (split)
   (("1" (propax))
    ("2"
     (hide 2)
     (grind)
     (("1" (inst 1 "ap_compute_module") (grind))
      ("2" (inst 1 "ap_compute_module") (grind)))))))
```

autopilot.autopilot_app_TCC3: proved - complete [shostak](12.80 s)

```
(""
 (skosimp)
 (split)
 (("1"
   (skosimp*)
   (typepred "ap_halt`pre")
```

```
   (inst -1 "st!1")
   (split)
   (("1"
     (inst -1 "st2!1")
     (split)
     (("1" (propax))
      ("2"
       (hide -1 2)
       (skosimp)
       (inst -1 "id!1")
       (grind)
       (typepred "id!1")
       (grind))))
    ("2" (propax))))
  ("2"
   (skosimp)
   (typepred "ap_halt`f")
   (inst -1 "d!1" "id!1")
   (grind))))
```

autopilot.autopilot_app_TCC4: proved - complete [shostak](12.01 s)

```
(""
 (skosimp)
 (split)
 (("1"
   (skosimp*)
   (typepred "ap_prep_full`pre")
   (inst -1 "st!1")
   (split)
   (("1"
     (inst -1 "st2!1")
     (split)
     (("1" (propax))
      ("2"
       (hide (-1 3 2))
       (skosimp)
       (inst -1 "id!1")
       (typepred "id!1")
       (grind))))
    ("2" (propax))))
  ("2"
   (skosimp)
   (typepred "ap_prep_full`f")
   (inst -1 "d!1" "id!1")
   (grind))))
```

autopilot.autopilot_app_TCC5: proved - complete [shostak](12.04 s)

```
(""
 (skosimp)
 (split)
 (("1"
   (skosimp*)
   (typepred "ap_prep_ah_only`pre")
   (inst -1 "st!1")
   (split)
```

```
   (("1"
     (inst -1 "st2!1")
     (split)
     (("1" (propax))
      ("2"
       (hide (-1 -3 2 3))
       (skosimp)
       (inst -1 "id!1")
       (typepred "id!1")
       (grind))))
    ("2" (propax))))
  ("2"
   (skosimp)
   (typepred "ap_prep_ah_only`f")
   (inst -1 "d!1" "id!1")
   (grind))))
```

```
autopilot.autopilot_app_TCC6: proved - complete [shostak](11.95 s)
```

```
(""
 (skosimp)
 (split)
 (("1"
   (skosimp*)
   (typepred "ap_prep_none`pre")
   (inst -1 "st!1")
   (split)
   (("1"
     (inst -1 "st2!1")
     (split)
     (("1" (propax))
      ("2"
       (hide (-1 2 3 4))
       (skosimp)
       (inst -1 "id!1")
       (typepred "id!1")
       (grind))))
    ("2" (propax))))
  ("2"
   (skosimp)
   (typepred "ap_prep_none`f")
   (inst -1 "d!1" "id!1")
   (grind))))
```

9. Example System Reconfiguration Specification

```
Proof scripts for file prototype_surv_spec-fm.pvs:
```

```
prototype_reconf_spec.proto_sys_configs_TCC1: proved - complete [shostak](10.89
s)
```

```
(""
 (skosimp)
 (skosimp)
```

```
 (typepred "app!1")
 (expand "proto_apps")
 (lemma "different_apps")
 (flatten)
 (expand "restrict")
 (split 7)
 (("1" (hide (-1 -2 2 3 4 5 6 7)) (grind))
  ("2"
   (flatten)
   (split)
   (("1" (hide (-1 -2 2 3 4 5 6 7 8)) (grind))
    ("2"
     (flatten)
     (split)
     (("1" (hide (-1 -2 2 3 4 5 6 7 8 9)) (grind)) ("2" (grind))))))))
```

prototype_reconf_spec.proto_sys_configs_TCC2: proved - complete [shostak](6.30 s)

```
(""
 (skosimp)
 (skosimp)
 (expand "restrict")
 (split)
 (("1" (grind))
  ("2"
   (flatten)
   (split)
   (("1" (grind))
    ("2"
     (flatten)
     (split)
     (("1" (grind))
      ("2" (typepred "app!1") (expand "proto_apps") (grind)))))))))
```

prototype_reconf_spec.proto_sys_configs_TCC3: proved - complete [shostak](5.22 s)

```
(""
 (skosimp*)
 (expand "restrict")
 (split)
 (("1" (grind))
  ("2"
   (flatten)
   (split)
   (("1" (grind))
    ("2"
     (flatten)
     (split)
     (("1" (hide (-1 2 3 4 5)) (grind))
      ("2"
       (typepred "app!1")
       (expand "proto_apps")
       (hide (4 5))
       (grind)))))))))
```

prototype_reconf_spec.proto_sys_configs_TCC4: proved - complete [shostak](5.08 s)

```
("" 
 (skosimp*)
 (hide (-1 1 2 3))
 (expand "restrict")
 (split)
 (("1" (grind))
  ("2"
   (flatten)
   (split)
   (("1" (hide 2) (grind))
    ("2"
     (flatten)
     (split)
     (("1" (hide 2 3) (grind))
      ("2" (typepred "app!1") (expand "proto_apps") (grind))))))))
```

prototype_reconf_spec.proto_sys_configs_TCC5: proved - complete [shostak](6.86 s)

```
("" 
 (skosimp*)
 (hide (-1 1 2 3 4))
 (expand "restrict")
 (split)
 (("1" (grind))
  ("2"
   (flatten)
   (split)
   (("1" (hide 2) (grind))
    ("2"
     (flatten)
     (split)
     (("1" (grind)) ("2" (typepred "app!1") (grind))))))))
```

prototype_reconf_spec.proto_sys_configs_TCC6: proved - complete [shostak](6.81 s)

```
("" 
 (skosimp*)
 (hide (-1 1 2 3 4 5))
 (expand "restrict")
 (split)
 (("1" (grind))
  ("2"
   (flatten)
   (split)
   (("1" (grind))
    ("2"
     (flatten)
     (split)
     (("1" (grind)) ("2" (typepred "app!1") (grind))))))))
```

prototype_reconf_spec.proto_sys_configs_TCC7: proved - complete [shostak](6.84 s)

```
("" 
 (skosimp*)
 (hide -1 1 2 3 4 5 6)
```

```
 (expand "restrict")
 (split)
 (("1" (grind))
  ("2"
   (flatten)
   (split)
   (("1" (grind))
    ("2"
     (flatten)
     (split)
     (("1" (grind)) ("2" (typepred "app!1") (grind)))))))))
```

prototype_reconf_spec.proto_sys_configs_TCC8: proved - complete [shostak](6.84 s)

```
(""
 (skosimp*)
 (hide -1 1 2 3 4 5 6 7)
 (expand "restrict")
 (split)
 (("1" (grind))
  ("2"
   (flatten)
   (split)
   (("1" (grind))
    ("2"
     (flatten)
     (split)
     (("1" (grind)) ("2" (typepred "app!1") (grind)))))))))
```

prototype_reconf_spec.proto_sys_configs_TCC9: proved - complete [shostak](6.84 s)

```
(""
 (skosimp*)
 (hide 1 2 3 4 5 6 7 8)
 (expand "restrict")
 (split)
 (("1" (grind))
  ("2"
   (flatten)
   (split)
   (("1" (grind))
    ("2"
     (flatten)
     (split)
     (("1" (grind)) ("2" (typepred "app!1") (grind)))))))))
```

prototype_reconf_spec.proto_trans_TCC1: proved - complete [shostak](0.16 s)

```
("" (grind))
```

prototype_reconf_spec.cov_txn_1: proved - complete [shostak](277.21 s)

```
(""
 (skosimp)
 (typepred "t!1")
```

```
 (expand "proto_reachable_env")
 (expand "proto_env_txns")
 (case "forall (t: (proto_trans)) : (degraded(t`source, t`target) OR t`source =
t`target) AND
                              appropriate(t`target, t`trigger)")
 (("1"
   (hide -1)
   (case "t!1`target(electrics) = alternator")
   (("1"
     (case "t!1`target(autopilot) = fullsvc")
     (("1"
       (case "t!1`target(rudder) = working")
       (("1" (lemma "different_env_params") (grind))
        ("2"
         (case "t!1`target(rudder) = hard_over_left")
         (("1"
           (inst 2
          "(# source := full_service, target := rudder_ho_left, trigger := (LAMBDA
(id: env_id) : (IF id = autopilot THEN fullsvc ELSIF id = electrics THEN alternator
                           ELSE hard_over_left ENDIF))#)")
           (("1"
             (lemma "different_env_params")
             (lemma "equal_trans")
             (inst -1 "proto_valid_env" "t!1`target"
              "(LAMBDA (id: env_id):
                                     (IF id = autopilot THEN fullsvc
                                      ELSIF id = electrics THEN alternator
                                      ELSE hard_over_left
                                      ENDIF))")
             (("1" (grind)) ("2" (grind))))
            ("2"
             (expand "proto_trans")
             (split)
             (("1" (grind))
              ("2"
               (lemma "different_env_params")
               (grind)
               (("1" (lemma "different_lvls") (grind))
                ("2" (lemma "different_lvls") (grind))
                ("3" (lemma "different_lvls") (grind))
                ("4" (lemma "different_lvls") (grind))
                ("5" (lemma "different_lvls") (grind))
                ("6" (lemma "different_lvls") (grind))))))
             ("3" (grind)) ("4" (grind))))
           ("2"
            (case "t!1`target(rudder) = hard_over_right")
            (("1"
              (lemma "different_env_params")
              (lemma "equal_trans")
              (inst 3
               "(# source := full_service, target := rudder_ho_right, trigger :=
(LAMBDA (id: env_id) : (IF id = autopilot THEN fullsvc ELSIF id = electrics THEN
alternator
                               ELSE hard_over_right ENDIF))#)")
              (("1"
                (inst -1 "proto_valid_env" "t!1`target"
                 "(LAMBDA (id: env_id):
                                          (IF id = autopilot THEN fullsvc
                                           ELSIF id = electrics THEN alternator
```

```
                                          ELSE hard_over_right
                                          ENDIF))")
              (("1" (grind)) ("2" (grind))))
             ("2"
              (grind)
              (("1" (lemma "different_lvls") (grind))
               ("2" (lemma "different_lvls") (grind))
               ("3" (lemma "different_lvls") (grind))
               ("4" (lemma "different_lvls") (grind))
               ("5" (lemma "different_lvls") (grind))
               ("6" (lemma "different_lvls") (grind))))
              ("3" (grind)) ("4" (grind))))
            ("2"
             (hide -6 4)
             (lemma "different_env_params")
             (typepred "t!1`target")
             (inst -1 "rudder")
             (grind))))))))
      ("2"
       (lemma "different_env_params")
       (lemma "equal_trans")
       (case "t!1`target(autopilot) = alt_hold_only")
       (("1"
         (case "t!1`target(rudder) = working")
         (("1"
           (inst 2
            "(# source := full_service, target := alt_hold_only, trigger := (LAMBDA
(id: env_id) : (IF id = autopilot THEN alt_hold_only ELSIF id = electrics THEN
alternator ELSE working ENDIF))#)")
           (("1"
             (inst -3 "proto_valid_env" "t!1`target"
              "(LAMBDA (id: env_id):
                                    (IF id = autopilot THEN alt_hold_only
                                     ELSIF id = electrics THEN alternator
                                     ELSE working
                                     ENDIF))")
             (("1" (grind)) ("2" (grind))))
            ("2"
             (expand "proto_trans")
             (lemma "different_lvls")
             (grind))
            ("3" (grind)) ("4" (grind))))
           ("2"
            (case "t!1`target(rudder) = hard_over_left")
            (("1"
              (lemma "different_lvls")
              (inst 3
               "(# source := full_service, target := rudder_ho_left_ah_only, trigger
:= (LAMBDA (id: env_id) : (IF id = autopilot THEN alt_hold_only ELSIF id =
electrics THEN alternator ELSE hard_over_left ENDIF))#)")
              (("1"
                (inst -4 "proto_valid_env" "t!1`target"
                 "(LAMBDA (id: env_id):
                                       (IF id = autopilot THEN alt_hold_only
                                        ELSIF id = electrics THEN alternator
                                        ELSE hard_over_left
                                        ENDIF))")
                (("1" (grind)) ("2" (grind))))
               ("2" (grind)) ("3" (grind)) ("4" (grind))))
```

```
            ("2"
             (lemma "different_lvls")
             (case "t!1`target(rudder) = hard_over_right")
             (("1"
               (inst 4
                "(# source := full_service, target := rudder_ho_right_ah_only,
trigger := (LAMBDA (id: env_id) : (IF id = autopilot THEN alt_hold_only ELSIF id
= electrics THEN alternator ELSE hard_over_right ENDIF))#)")
               (("1"
                 (inst -4 "proto_valid_env" "t!1`target"
                  "(LAMBDA (id: env_id):
                                      (IF id = autopilot THEN alt_hold_only
                                       ELSIF id = electrics THEN alternator
                                           ELSE hard_over_right
                                           ENDIF))")
                 (("1" (grind)) ("2" (grind))))
                ("2" (grind)) ("3" (grind)) ("4" (grind))))
              ("2"
               (hide 5)
               (hide -3)
               (typepred "t!1`target")
               (inst -1 "rudder")
               (grind))))))))
        ("2"
         (case "t!1`target(autopilot) = disabled")
         (("1"
           (case "t!1`target(rudder) = working")
           (("1"
             (lemma "different_lvls")
             (inst 3
              "(# source := full_service, target := fcs_only, trigger := (LAMBDA
(id: env_id) : (IF id = autopilot THEN disabled ELSIF id = electrics THEN
alternator ELSE working ENDIF))#)")
             (("1"
               (inst -4 "proto_valid_env" "t!1`target"
                "(LAMBDA (id: env_id):
                                     (IF id = autopilot THEN disabled
                                      ELSIF id = electrics THEN alternator
                                      ELSE working
                                      ENDIF))")
               (("1" (grind)) ("2" (grind))))
              ("2" (grind)) ("3" (grind)) ("4" (grind))))
            ("2"
             (lemma "different_lvls")
             (case "t!1`target(rudder) = hard_over_left")
             (("1"
               (inst 4
                "(# source := full_service, target := rudder_ho_left_fcs_only,
trigger := (LAMBDA (id: env_id) : (IF id = autopilot THEN disabled ELSIF id =
electrics THEN alternator ELSE hard_over_left ENDIF))#)")
               (("1"
                 (inst -4 "proto_valid_env" "t!1`target"
                  "(LAMBDA (id: env_id):
                                        (IF id = autopilot THEN disabled
                                      ELSIF id = electrics THEN alternator
                                          ELSE hard_over_left
                                          ENDIF))")
                 (("1" (grind)) ("2" (grind))))
                ("2" (grind)) ("3" (grind)) ("4" (grind))))
```

```
              ("2"
               (case "t!1`target(rudder) = hard_over_right")
               (("1"
                 (inst 5
                  "(# source := full_service, target := rudder_ho_right_fcs_only,
trigger := (LAMBDA (id: env_id) : (IF id = autopilot THEN disabled ELSIF id =
electrics THEN alternator ELSE hard_over_right ENDIF))#)")
                 (("1"
                   (inst -4 "proto_valid_env" "t!1`target"
                    "(LAMBDA (id: env_id):
                                              (IF id = autopilot THEN disabled
                                            ELSIF id = electrics THEN alternator
                                                   ELSE hard_over_right
                                                   ENDIF))")
                   (("1" (grind)) ("2" (grind))))
                  ("2" (grind)) ("3" (grind)) ("4" (grind))))
                ("2"
                 (hide -9 -3)
                 (typepred "t!1`target")
                 (inst -1 "rudder")
                 (grind))))))))
          ("2"
           (typepred "t!1`target")
           (inst -1 "autopilot")
           (grind))))))))
    ("2"
     (lemma "equal_trans")
     (lemma "different_env_params")
     (lemma "different_lvls")
     (case "t!1`target(electrics) = battery")
     (("1"
       (case "t!1`target(rudder) = working")
       (("1"
         (case "t!1`target(autopilot) = fullsvc")
         (("1"
           (inst 2
            "(# source := full_service, target := fcs_only, trigger := (LAMBDA
(id: env_id) : (IF id = autopilot THEN fullsvc ELSIF id = electrics THEN battery
ELSE working ENDIF))#)")
           (("1"
             (inst -6 "proto_valid_env" "t!1`target"
              "(LAMBDA (id: env_id):
                                      (IF id = autopilot THEN fullsvc
                                       ELSIF id = electrics THEN battery
                                       ELSE working
                                       ENDIF))")
             (("1" (grind)) ("2" (grind))))
            ("2" (grind)) ("3" (grind)) ("4" (grind))))
           ("2"
            (case "t!1`target(autopilot) = alt_hold_only")
            (("1"
              (inst 3
               "(# source := full_service, target := fcs_only, trigger := (LAMBDA
(id: env_id) : (IF id = autopilot THEN alt_hold_only ELSIF id = electrics THEN
battery ELSE working ENDIF))#)")
              (("1"
                (inst -6 "proto_valid_env" "t!1`target"
                 "(LAMBDA (id: env_id):
                                           (IF id = autopilot THEN alt_hold_only
```

```
                                          ELSIF id = electrics THEN battery
                                          ELSE working
                                          ENDIF))")
               (("1" (grind)) ("2" (grind))))
              ("2" (grind)) ("3" (grind)) ("4" (grind))))
             ("2"
              (case "t!1`target(autopilot) = disabled")
              (("1"
                (inst 4
                "(# source := full_service, target := fcs_only, trigger := (LAMBDA
(id: env_id) : (IF id = autopilot THEN disabled ELSIF id = electrics THEN battery
ELSE working ENDIF))#)")
                (("1"
                  (inst -6 "proto_valid_env" "t!1`target"
                   "(LAMBDA (id: env_id):
                                             (IF id = autopilot THEN disabled
                                             ELSIF id = electrics THEN battery
                                              ELSE working
                                              ENDIF))")
                  (("1" (grind)) ("2" (grind))))
                 ("2" (grind)) ("3" (grind)) ("4" (grind))))
                ("2"
                 (typepred "t!1`target")
                 (inst -1 "autopilot")
                 (grind))))))))
         ("2"
          (case "t!1`target(rudder) = hard_over_left")
          (("1"
            (case "t!1`target(autopilot) = fullsvc")
            (("1"
              (inst 3
              "(# source := full_service, target := rudder_ho_left_fcs_only, trigger
:= (LAMBDA (id: env_id) : (IF id = autopilot THEN fullsvc ELSIF id = electrics
THEN battery ELSE hard_over_left ENDIF))#)")
              (("1"
                (inst -6 "proto_valid_env" "t!1`target"
                 "(LAMBDA (id: env_id):
                                           (IF id = autopilot THEN fullsvc
                                            ELSIF id = electrics THEN battery
                                            ELSE hard_over_left
                                            ENDIF))")
                (("1" (grind)) ("2" (grind))))
               ("2" (grind)) ("3" (grind)) ("4" (grind))))
              ("2"
               (case "t!1`target(autopilot) = alt_hold_only")
               (("1"
                 (inst 4
                  "(# source := full_service, target := rudder_ho_left_fcs_only,
trigger := (LAMBDA (id: env_id) : (IF id = autopilot THEN alt_hold_only ELSIF id
= electrics THEN battery ELSE hard_over_left ENDIF))#)")
                 (("1"
                   (inst -6 "proto_valid_env" "t!1`target"
                    "(LAMBDA (id: env_id):
                                           (IF id = autopilot THEN alt_hold_only
                                               ELSIF id = electrics THEN battery
                                                ELSE hard_over_left
                                                ENDIF))")
                   (("1" (grind)) ("2" (grind))))
                  ("2" (grind)) ("3" (grind)) ("4" (grind))))
```

```
              ("2"
               (case "t!1`target(autopilot) = disabled")
               (("1"
                 (inst 5
                  "(# source := full_service, target := rudder_ho_left_fcs_only,
trigger := (LAMBDA (id: env_id) : (IF id = autopilot THEN disabled ELSIF id =
electrics THEN battery ELSE hard_over_left ENDIF))#)")
                 (("1"
                   (inst -6 "proto_valid_env" "t!1`target"
                    "(LAMBDA (id: env_id):
                                          (IF id = autopilot THEN disabled
                                         ELSIF id = electrics THEN battery
                                             ELSE hard_over_left
                                             ENDIF))")
                   (("1" (grind)) ("2" (grind))))
                  ("2" (grind)) ("3" (grind)) ("4" (grind))))
                ("2"
                 (typepred "t!1`target")
                 (inst -1 "autopilot")
                 (grind))))))))
          ("2"
           (case "t!1`target(rudder) = hard_over_right")
           (("1"
             (case "t!1`target(autopilot) = disabled")
             (("1"
               (inst 4
                "(# source := full_service, target := rudder_ho_right_fcs_only,
trigger := (LAMBDA (id: env_id) : (IF id = autopilot THEN disabled ELSIF id =
electrics THEN battery ELSE hard_over_right ENDIF))#)")
               (("1"
                 (inst -6 "proto_valid_env" "t!1`target"
                  "(LAMBDA (id: env_id):
                                         (IF id = autopilot THEN disabled
                                         ELSIF id = electrics THEN battery
                                          ELSE hard_over_right
                                          ENDIF))")
                 (("1" (grind)) ("2" (grind))))
                ("2" (grind)) ("3" (grind)) ("4" (grind))))
              ("2"
               (case "t!1`target(autopilot) = alt_hold_only")
               (("1"
                 (inst 5
                  "(# source := full_service, target := rudder_ho_right_fcs_only,
trigger := (LAMBDA (id: env_id) : (IF id = autopilot THEN alt_hold_only ELSIF id
= electrics THEN battery ELSE hard_over_right ENDIF))#)")
                 (("1"
                   (inst -6 "proto_valid_env" "t!1`target"
                    "(LAMBDA (id: env_id):
                                      (IF id = autopilot THEN alt_hold_only
                                          ELSIF id = electrics THEN battery
                                               ELSE hard_over_right
                                               ENDIF))")
                   (("1" (grind)) ("2" (grind))))
                  ("2" (grind)) ("3" (grind)) ("4" (grind))))
                ("2"
                 (case "t!1`target(autopilot) = fullsvc")
                 (("1"
                   (inst 6
```

```
                "(# source := full_service, target := rudder_ho_right_fcs_only,
trigger := (LAMBDA (id: env_id) : (IF id = autopilot THEN fullsvc ELSIF id =
electrics THEN battery ELSE hard_over_right ENDIF))#)")
                  (("1"
                    (inst -6 "proto_valid_env" "t!1`target"
                     "(LAMBDA (id: env_id):
                                              (IF id = autopilot THEN fullsvc
                                            ELSIF id = electrics THEN battery
                                                   ELSE hard_over_right
                                                   ENDIF))")
                    (("1" (grind)) ("2" (grind))))
                   ("2" (grind)) ("3" (grind)) ("4" (grind))))
                 ("2"
                  (typepred "t!1`target")
                  (inst -1 "autopilot")
                  (grind))))))))
           ("2"
            (typepred "t!1`target")
            (inst -1 "rudder")
            (grind))))))))
     ("2" (typepred "t!1`target") (inst -1 "electrics") (grind))))))
  ("2" (skosimp 1) (typepred "t!2") (hide -2 -3 2) (grind))
  ("3" (hide -1 -2 2) (grind))))


prototype_reconf_spec.cov_txn_2: proved - complete [shostak](559.00 s)

(""
 (skosimp)
 (skosimp)
 (typepred "t_e!1")
 (expand "proto_reachable_env")
 (expand "proto_env_txns")
 (case "t_e!1`target(electrics) = alternator")
 (("1"
   (case "t_e!1`target(rudder) = working")
   (("1"
     (case "t_e!1`target(autopilot) = fullsvc")
     (("1"
       (hide 5- 6- -7 1)
       (expand "degraded")
       (lemma "different_env_params")
       (split)
       (("1" (flatten) (grind)) ("2" (grind)) ("3" (grind))
        ("4" (grind)) ("5" (grind)) ("6" (grind))))
      ("2"
       (case "t_e!1`target(autopilot) = alt_hold_only")
       (("1"
         (hide -4)
         (lemma "different_env_ids")
         (lemma "equal_trans")
         (inst 2
          "(# source := full_service, target := alt_hold_only, trigger :=
(lambda(id: env_id) : IF id = autopilot THEN
        alt_hold_only ELSIF id = electrics THEN alternator ELSE working ENDIF)#)")
         (("1"
           (inst -1 "proto_valid_env" "t_e!1`target"
            "(LAMBDA (id: env_id):
                     IF id = autopilot THEN alt_hold_only
```

```
                    ELSIF id = electrics THEN alternator
                    ELSE working
                    ENDIF)")
          (("1" (grind)) ("2" (grind))))
         ("2"
          (lemma "different_env_params")
          (lemma "different_lvls")
          (grind))
         ("3" (grind)) ("4" (grind))))
        ("2"
         (lemma "different_env_params")
         (lemma "different_env_ids")
         (lemma "different_lvls")
         (inst 3
          "(# source := full_service, target := fcs_only, trigger := (lambda(id:
env_id) : IF id = autopilot THEN
           disabled ELSIF id = electrics THEN alternator ELSE working ENDIF)#)")
         (("1"
           (lemma "equal_trans")
           (inst -1 "proto_valid_env" "t_e!1`target"
            "(LAMBDA (id: env_id):
                    IF id = autopilot THEN disabled
                    ELSIF id = electrics THEN alternator
                    ELSE working
                    ENDIF)")
           (("1" (grind)) ("2" (hide -6) (grind))))
          ("2" (hide -6) (grind)) ("3" (grind)) ("4" (grind))))))))
    ("2"
     (hide -2)
     (lemma "equal_trans")
     (lemma "different_env_ids")
     (lemma "different_env_params")
     (lemma "different_lvls")
     (case "t_e!1`target(rudder) = hard_over_left")
     (("1"
       (case "t_e!1`target(autopilot) = fullsvc")
       (("1"
         (inst 2
          "(# source := full_service, target := rudder_ho_left, trigger :=
(lambda(id: env_id) : IF id = autopilot THEN
           fullsvc ELSIF id = electrics THEN alternator ELSE hard_over_left
ENDIF)#)")
         (("1"
           (inst -6 "proto_valid_env" "t_e!1`target"
            "(LAMBDA (id: env_id):
                    IF id = autopilot THEN fullsvc
                    ELSIF id = electrics THEN alternator
                    ELSE hard_over_left
                    ENDIF)")
           (("1" (grind)) ("2" (grind))))
          ("2" (grind)) ("3" (grind)) ("4" (grind))))
        ("2"
         (case "t_e!1`target(autopilot) = alt_hold_only")
         (("1"
           (inst 3
           "(# source := full_service, target := rudder_ho_left_ah_only, trigger
:= (lambda(id: env_id) : IF id = autopilot THEN
           alt_hold_only ELSIF id = electrics THEN alternator ELSE hard_over_left
ENDIF)#)")
```

```
           (("1"
             (inst -6 "proto_valid_env" "t_e!1`target"
              "(LAMBDA (id: env_id):
                       IF id = autopilot THEN alt_hold_only
                       ELSIF id = electrics THEN alternator
                       ELSE hard_over_left
                       ENDIF)")
             (("1" (grind)) ("2" (grind))))
            ("2" (grind)) ("3" (grind)) ("4" (grind))))
          ("2"
           (case "t_e!1`target(autopilot) = disabled")
           (("1"
             (inst 4
           "(# source := full_service, target := rudder_ho_left_fcs_only, trigger
:= (lambda(id: env_id) : IF id = autopilot THEN
               disabled ELSIF id = electrics THEN alternator ELSE hard_over_left
ENDIF)#)")
             (("1"
               (inst -6 "proto_valid_env" "t_e!1`target"
                "(LAMBDA (id: env_id):
                         IF id = autopilot THEN disabled
                         ELSIF id = electrics THEN alternator
                         ELSE hard_over_left
                         ENDIF)")
               (("1" (grind)) ("2" (grind))))
              ("2" (grind)) ("3" (grind)) ("4" (grind))))
            ("2"
             (typepred "t_e!1`target")
             (inst -1 "autopilot")
             (hide -7)
             (grind)))))))
      ("2"
       (case "t_e!1`target(rudder) = hard_over_right")
       (("1"
         (case "t_e!1`target(autopilot) = disabled")
         (("1"
           (inst 3
          "(# source := full_service, target := rudder_ho_right_fcs_only, trigger
:= (lambda(id: env_id) : IF id = autopilot THEN disabled ELSIF id = electrics THEN
alternator ELSE hard_over_right ENDIF)#)")
           (("1"
             (inst -6 "proto_valid_env" "t_e!1`target"
              "(LAMBDA (id: env_id):
                       IF id = autopilot THEN disabled
                       ELSIF id = electrics THEN alternator
                       ELSE hard_over_right
                       ENDIF)")
             (("1" (grind)) ("2" (grind))))
            ("2" (grind)) ("3" (grind)) ("4" (grind))))
          ("2"
           (case "t_e!1`target(autopilot) = alt_hold_only")
           (("1"
             (inst 4
           "(# source := full_service, target := rudder_ho_right_ah_only, trigger
:= (lambda(id: env_id) : IF id = autopilot THEN alt_hold_only ELSIF id = electrics
THEN alternator ELSE hard_over_right ENDIF)#)")
             (("1"
               (inst -6 "proto_valid_env" "t_e!1`target"
                "(LAMBDA (id: env_id):
```

```
                        IF id = autopilot THEN alt_hold_only
                        ELSIF id = electrics THEN alternator
                        ELSE hard_over_right
                        ENDIF)")
                 (("1" (grind)) ("2" (grind))))
                ("2" (grind)) ("3" (grind)) ("4" (grind))))
              ("2"
               (case "t_e!1`target(autopilot) = fullsvc")
               (("1"
                 (inst 5
                  "(# source := full_service, target := rudder_ho_right, trigger :=
(lambda(id: env_id) : IF id = autopilot THEN fullsvc ELSIF id = electrics THEN
alternator ELSE hard_over_right ENDIF)#)")
                 (("1"
                   (inst -6 "proto_valid_env" "t_e!1`target"
                    "(LAMBDA (id: env_id):
                             IF id = autopilot THEN fullsvc
                             ELSIF id = electrics THEN alternator
                             ELSE hard_over_right
                             ENDIF)")
                   (("1" (grind)) ("2" (grind))))
                  ("2" (grind)) ("3" (grind)) ("4" (grind))))
                ("2"
                 (typepred "t_e!1`target")
                 (inst -1 "autopilot")
                 (grind))))))))
         ("2"
          (typepred "t_e!1`target")
          (inst -1 "rudder")
          (grind))))))))
  ("2"
   (hide -1)
   (lemma "equal_trans")
   (lemma "different_env_ids")
   (lemma "different_env_params")
   (lemma "different_lvls")
   (case "t_e!1`target(electrics) = battery")
   (("1"
     (case "t_e!1`target(rudder) = working")
     (("1"
       (case "t_e!1`target(autopilot) = fullsvc")
       (("1"
         (inst 2
          "(# source := full_service, target := fcs_only, trigger := (lambda(id:
env_id) : IF id = autopilot THEN fullsvc ELSIF id = electrics THEN battery ELSE
working ENDIF)#)")
          (("1"
            (inst -7 "proto_valid_env" "t_e!1`target"
             "(LAMBDA (id: env_id):
                      IF id = autopilot THEN fullsvc
                      ELSIF id = electrics THEN battery
                      ELSE working
                      ENDIF)")
            (("1" (grind)) ("2" (grind))))
           ("2" (grind)) ("3" (grind)) ("4" (grind))))
         ("2"
          (case "t_e!1`target(autopilot) = alt_hold_only")
          (("1"
            (inst 3
```

```
          "(# source := full_service, target := fcs_only, trigger := (lambda(id:
env_id) : IF id = autopilot THEN alt_hold_only ELSIF id = electrics THEN battery
ELSE working ENDIF)#)")
           (("1"
             (inst -7 "proto_valid_env" "t_e!1`target"
              "(LAMBDA (id: env_id):
                       IF id = autopilot THEN alt_hold_only
                       ELSIF id = electrics THEN battery
                       ELSE working
                       ENDIF)")
             (("1" (grind)) ("2" (grind))))
            ("2" (grind)) ("3" (grind)) ("4" (grind))))
           ("2"
            (case "t_e!1`target(autopilot) = disabled")
            (("1"
              (inst 4
            "(# source := full_service, target := fcs_only, trigger := (lambda(id:
env_id) : IF id = autopilot THEN disabled ELSIF id = electrics THEN battery ELSE
working ENDIF)#)")
              (("1"
                (inst -7 "proto_valid_env" "t_e!1`target"
                 "(LAMBDA (id: env_id):
                          IF id = autopilot THEN disabled
                          ELSIF id = electrics THEN battery
                          ELSE working
                          ENDIF)")
                (("1" (grind)) ("2" (grind))))
               ("2" (grind)) ("3" (grind)) ("4" (grind))))
              ("2"
               (typepred "t_e!1`target")
               (inst -1 "autopilot")
               (grind))))))))
      ("2"
       (case "t_e!1`target(rudder) = hard_over_left")
       (("1"
         (case "t_e!1`target(autopilot) = disabled")
         (("1"
           (inst 3
          "(# source := full_service, target := rudder_ho_left_fcs_only, trigger
:= (lambda(id: env_id) : IF id = autopilot THEN disabled ELSIF id = electrics THEN
battery ELSE hard_over_left ENDIF)#)")
           (("1"
             (inst -7 "proto_valid_env" "t_e!1`target"
              "(LAMBDA (id: env_id):
                       IF id = autopilot THEN disabled
                       ELSIF id = electrics THEN battery
                       ELSE hard_over_left
                       ENDIF)")
             (("1" (grind)) ("2" (grind))))
            ("2" (grind)) ("3" (grind)) ("4" (grind))))
           ("2"
            (case "t_e!1`target(autopilot) = alt_hold_only")
            (("1"
              (inst 4
            "(# source := full_service, target := rudder_ho_left_fcs_only, trigger
:= (lambda(id: env_id) : IF id = autopilot THEN alt_hold_only ELSIF id = electrics
THEN battery ELSE hard_over_left ENDIF)#)")
              (("1"
                (inst -7 "proto_valid_env" "t_e!1`target"
```

```
                "(LAMBDA (id: env_id):
                         IF id = autopilot THEN alt_hold_only
                         ELSIF id = electrics THEN battery
                         ELSE hard_over_left
                         ENDIF)")
               (("1" (grind)) ("2" (grind))))
              ("2" (grind)) ("3" (grind)) ("4" (grind))))
             ("2"
              (case "t_e!1`target(autopilot) = fullsvc")
              (("1"
                (inst 5
                 "(# source := full_service, target := rudder_ho_left_fcs_only,
trigger := (lambda(id: env_id) : IF id = autopilot THEN fullsvc ELSIF id =
electrics THEN battery ELSE hard_over_left ENDIF)#)")
                (("1"
                  (inst -7 "proto_valid_env" "t_e!1`target"
                   "(LAMBDA (id: env_id):
                            IF id = autopilot THEN fullsvc
                            ELSIF id = electrics THEN battery
                            ELSE hard_over_left
                            ENDIF)")
                   (("1" (grind)) ("2" (grind))))
                  ("2" (grind)) ("3" (grind)) ("4" (grind))))
                 ("2"
                  (typepred "t_e!1`target")
                  (inst -1 "autopilot")
                  (grind)))))))))
          ("2"
           (case "t_e!1`target(rudder) = hard_over_right")
           (("1"
             (case "t_e!1`target(autopilot) = fullsvc")
             (("1"
               (inst 4
                "(# source := full_service, target := rudder_ho_right_fcs_only,
trigger := (lambda(id: env_id) : IF id = autopilot THEN fullsvc ELSIF id =
electrics THEN battery ELSE hard_over_right ENDIF)#)")
               (("1"
                 (inst -7 "proto_valid_env" "t_e!1`target"
                  "(LAMBDA (id: env_id):
                           IF id = autopilot THEN fullsvc
                           ELSIF id = electrics THEN battery
                           ELSE hard_over_right
                           ENDIF)")
                 (("1" (grind)) ("2" (grind))))
                ("2" (grind)) ("3" (grind)) ("4" (grind))))
               ("2"
                (case "t_e!1`target(autopilot) = alt_hold_only")
                (("1"
                  (inst 5
                   "(# source := full_service, target := rudder_ho_right_fcs_only,
trigger := (lambda(id: env_id) : IF id = autopilot THEN alt_hold_only ELSIF id =
electrics THEN battery ELSE hard_over_right ENDIF)#)")
                  (("1"
                    (inst -7 "proto_valid_env" "t_e!1`target"
                     "(LAMBDA (id: env_id):
                              IF id = autopilot THEN alt_hold_only
                              ELSIF id = electrics THEN battery
                              ELSE hard_over_right
                              ENDIF)")
```

```
                 (("1" (grind)) ("2" (grind))))
                ("2" (grind)) ("3" (grind)) ("4" (grind))))
               ("2"
                (case "t_e!1`target(autopilot) = disabled")
                (("1"
                  (inst 6
                   "(# source := full_service, target := rudder_ho_right_fcs_only,
trigger := (lambda(id: env_id) : IF id = autopilot THEN disabled ELSIF id =
electrics THEN battery ELSE hard_over_right ENDIF)#)")
                  (("1"
                    (inst -7 "proto_valid_env" "t_e!1`target"
                     "(LAMBDA (id: env_id):
                              IF id = autopilot THEN disabled
                              ELSIF id = electrics THEN battery
                              ELSE hard_over_right
                              ENDIF)")
                    (("1" (grind)) ("2" (grind))))
                   ("2" (grind)) ("3" (grind)) ("4" (grind))))
                  ("2"
                   (typepred "t_e!1`target")
                   (inst -1 "autopilot")
                   (grind))))))))
           ("2"
            (typepred "t_e!1`target")
            (inst -1 "rudder")
            (grind))))))))
    ("2" (typepred "t_e!1`target") (inst -1 "electrics") (grind))))))
```

prototype_reconf_spec.cov_txn_3: proved - complete [shostak](6064.18 s)

```
(""
 (lemma "equal_spec_trans")
 (skosimp)
 (skosimp)
 (inst -1 "t1!1" "t2!1")
 (typepred "t1!1")
 (expand "proto_trans")
 (flatten)
 (typepred "t2!1")
 (expand "proto_trans")
 (flatten)
 (lemma "different_lvls")
 (lemma "different_env_params")
 (lemma "different_env_ids")
 (typepred "trigger!1")
 (inst -1 "autopilot")
 (typepred "trigger!1")
 (inst -1 "electrics")
 (typepred "trigger!1")
 (inst -1 "rudder")
 (lemma "equal_spec_trans")
 (grind)
 (("1" (typepred "t1!1`target") (expand "proto_speclvl") (propax))
  ("2" (typepred "t2!1`target") (expand "proto_speclvl") (propax))
  ("3" (typepred "t1!1`target") (expand "proto_speclvl") (propax))))
```

prototype_reconf_spec.proto_SCRAM_table_TCC1: proved - complete [shostak](0.04 s)

```
("" (grind))


prototype_reconf_spec.prototype_choose_TCC1: proved - complete [shostak](0.13 s)

(""
 (skosimp)
 (expand "nonempty?")
 (expand "empty?")
 (skosimp -1)
 (expand "member")
 (inst -2 "target!1"))


prototype_reconf_spec.prototype_reconf_spec_TCC1: proved - complete
[shostak](4.96 s)

(""
 (skosimp)
 (typepred "x1!1")
 (typepred
  "list2finseq((: sensors_app, pilot_interface_app, autopilot_app,
                       fcs_app :))`length")
 (expand "proto_apps")
 (flatten)
 (expand "list2finseq")
 (expand "length")
 (expand "length")
 (expand "length")
 (expand "length")
 (expand "length")
 (expand "nth")
 (expand "nth")
 (expand "nth")
 (expand "nth")
 (ground)
 (lift-if)
 (ground))


prototype_reconf_spec.prototype_reconf_spec_TCC2: proved - complete
[shostak](24.59 s)

(""
 (lemma "different_lvls")
 (expand "extend")
 (expand "restrict")
 (split)
 (("1" (grind)) ("2" (grind))
  ("3"
   (expand "covering_txns")
   (lemma "cov_txn_1")
   (lemma "cov_txn_2")
   (lemma "cov_txn_3")
   (skosimp 1)
   (split 1)
   (("1"
     (skosimp 1)
```

```
 (expand "proto_SCRAM_table")
 (expand "restrict")
 (hide -2 -3)
 (inst -2 "t!1`source" "t!1")
 (skosimp)
 (inst 1 "txn!1")
 (("1" (grind)) ("2" (grind))))
("2"
 (hide -1 -3)
 (skosimp)
 (typepred "t_s!1")
 (expand "proto_SCRAM_table")
 (expand "proto_trans")
 (inst -2 "t_s!1`source" "t_s!1" "t_s!1`trigger")
 (("1"
   (split -2)
   (("1"
     (skosimp)
     (inst -1 "t_e!1")
     (split -1)
     (("1"
       (skosimp)
       (inst 1 "t_t!1")
       (typepred "t_t!1")
       (expand "proto_SCRAM_table")
       (propax))
      ("2" (propax))))
    ("2"
     (skosimp)
     (typepred "t_e!1")
     (expand "proto_reachable_env")
     (expand "proto_env_txns")
     (split -2)
     (("1"
       (typepred "proto_SCRAM_table`txns")
       (expand "covering_txns")
       (flatten)
       (hide -1 -3)
       (inst -1 "t_s!1")
       (inst -1 "t_e!1")
       (grind))
      ("2" (grind))))))
  ("2" (hide 2) (grind))))
("3"
 (hide -3)
 (hide -2)
 (skosimp)
 (inst -2 "source!1" "trigger!1")
 (("1"
   (skosimp)
   (inst 2 "t1!1" "t2!1")
   (("1" (grind))
    ("2"
     (expand "proto_trans")
     (typepred "t2!1")
     (expand "proto_SCRAM_table")
     (expand "proto_trans")
     (propax))
    ("3" (typepred "t1!1") (expand "proto_SCRAM_table") (propax))))
```

```
      ("2" (typepred "source!1") (hide -2) (assert))))))
  ("4" (grind)) ("5" (propax)) ("6" (grind)) ("7" (grind))))
```

prototype_reconf_spec.prototype_reconf_spec_TCC3: proved - complete
[shostak](7.91 s)

```
(""
 (split)
 (("1" (expand "restrict") (skosimp) (lemma "different_lvls") (grind))
  ("2"
   (skosimp)
   (split)
   (("1"
     (lemma "different_lvls")
     (expand "prototype_choose")
     (lift-if)
     (split)
     (("1"
       (flatten)
       (typepred
        "choose({target: (proto_speclvl) |
                       EXISTS (t: (proto_SCRAM_table`txns)):
                         t`source = x1!1`1 AND
                          t`trigger = x1!1`2 AND t`target = target})")
       (("1" (hide -2 -3) (grind))
        ("2"
         (hide -2)
         (skosimp -1)
         (skosimp)
         (expand "nonempty?")
         (expand "empty?")
         (inst -1 "target!1")
         (grind))))
      ("2" (typepred "x1!1`1") (flatten -2) (hide 1) (grind))))
    ("2" (grind))))))
```

prototype_reconf_spec.prototype_reconf_spec_TCC4: proved - complete
[shostak](4.12 s)

```
("" (skosimp) (typepred "proto_t") (grind))
```

prototype_reconf_spec.prototype_reconf_spec_TCC5: proved - complete
[shostak](18.62 s)

```
(""
 (split)
 (("1" (lemma "different_lvls") (grind))
  ("2" (lemma "different_lvls") (grind))))
```