A Market Compliant with COVID-19 Regulations

Manuel Carro
manuel.carro@upm.es

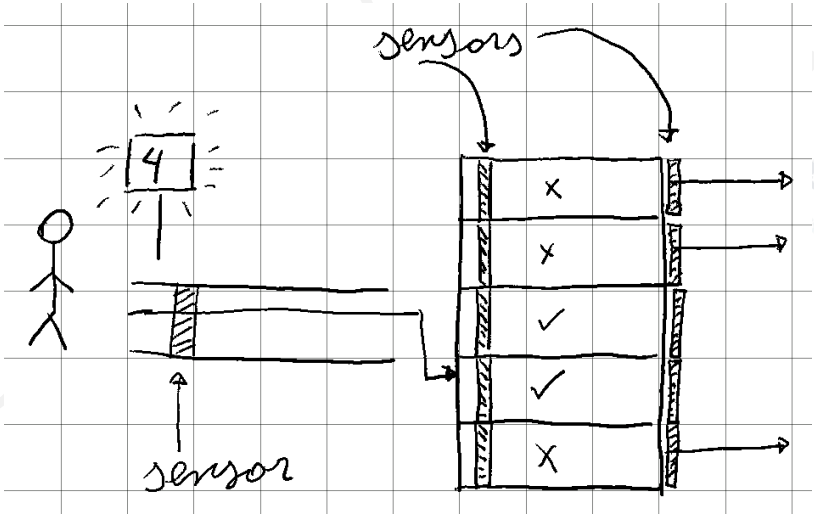
Universidad Politécnica de Madrid &
IMDEA Software Institute

Scenario

- We have to automate the checkout desk of a market.
- We have to control when clients enter the checkout area.
- Expected behavior:
  - Clients wait in front of a screen displaying a number or “WAIT”.
  - When a number appears, client walks to the corresponding counter.
  - As soon as it passes by the screen, “WAIT” is displayed.
  - When the client reaches the counter, either a new number is displayed (if there are free counters) or “WAIT” (otherwise).
  - When a client leaves, a counter number is displayed.
- Sensors register people movements.
- People behave (no need for physical barriers).

- **Note:** non-complete model.
- Focus on showing use of sets and giving a taste of model checking.

What we see



(Sizes not necessarily proportional)

Requirements

REQ 1	The market exit is divided in three areas: the *waiting area*, the *checkout counters* and a *checkout corridor* that connects them.

REQ 2	At most one client can be in the corridor at any time.

REQ 3	At most one client can be in a checkout counter at any time.

REQ 4	A screen at the entrance of the tells clients to either wait for the corridor to be clear or a counter to be free, or displays the identifier of an available counter.

Requirements

REQ 5	When the corridor is not empty, the screen displays "WAIT".

REQ 6	When no counter is free, the screen displays "WAIT".

REQ 7	When access to the corridor is possible, the screen displays the identifier of one of the available counters.

REQ 8	There are sensors that register people passing at the entrance of the corredor and at the entrance and exit of every counter.

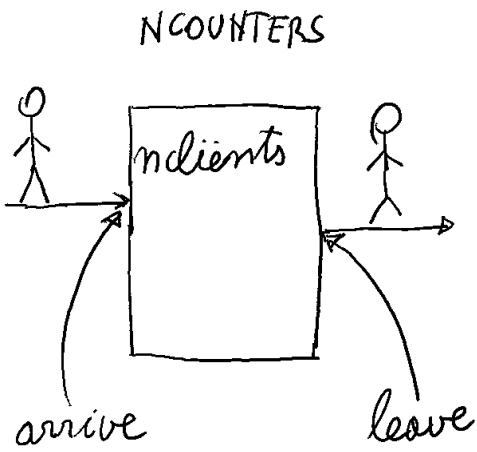
Modeling approach

- As usual: bird's-eye view.
- Include more requirements, details as we "get closer".
- Do not to overspecify early: refinement may become impossible.

Stages

1. Initial model: just number of clients
2. First refinement: distinguish checkout desks
3. Second refinement: entrance corridor and screen
4. Third refinement: sensors
5. Variant: sets instead of indicator functions

High-level view, visible events



- Clients arrive at the checkout desks.
- Clients leave the checkout desks.
- We only check that we do not have more clients than counters.
- Partial fullfillment of

REQ 9	At most one client can be in a checkout counter at any time.

Model

Context c0

CONSTANTS NCOUNTERS
AXIOMS NCOUNTERS $\in$??

Machine m0

VARIABLES nclients
INVARIANTS nclients $\in$ 0..NCOUNTERS

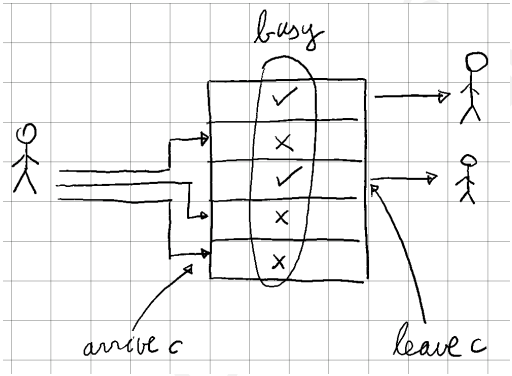
Event arrive
when $nclients < NCOUNTERS$
then
$nclients := nclients + 1$
end

Event leave
when $nclients > 0$
then
$nclients := nclients - 1$
end

Stages

1. Initial model: just number of clients
2. First refinement: distinguish checkout desks
3. Second refinement: entrance corridor and screen
4. Third refinement: sensors
5. Variant: sets instead of indicator functions

High-level view



- Keep track of (non) available counters.
- Fullfill

REQ 10	At most one client can be in a checkout counter at any time.

- Do not *follow* people.

Model state

- Need to model which counter is available.
- Possibility?

$$available \in 1..NCOUNTERS \to \mathrm{BOOL}$$

- But a function $A \to \mathrm{BOOL}$ denotes a set $S \subseteq A$.
  (it is the *characteristic* or *indicator* function of the set)
- Why not using directly a set?
- The set of busy counters is more useful than the set of available counters (will see later why).
- Do we need it to be $1..NCOUNTERS$?
  - Actually no. We are not going to compare counters.
  - An abstract set will do.

Model state: context and invariants

Context c1

EXTENDS c0
SETS COUNTERS
AXIOMS $card(COUNTERS) = NCOUNTERS$

Create it!

- WD PO not discharged!
- *card* requires the set to be finite.

AXIOMS
$finite(COUNTERS)$
$card(COUNTERS) = NCOUNTERS$

(in that order)

Machine m1

- Refine m0 to track busy counters, create m1.
- SEES c1

VARIABLES busy
INVARIANTS ???
$busy \subseteq COUNTERS$
$card(busy) = nclients$

Events

- Initially, $busy = \varnothing$
- We see event arrive when some client goes to a free counter and the counter becomes busy.
- An event parameter is the easiest way to model this.

Event arrive
refines arrive
any c
where
$c \in COUNTERS$
$c \notin busy$
then
$busy := busy \cup \{c\}$

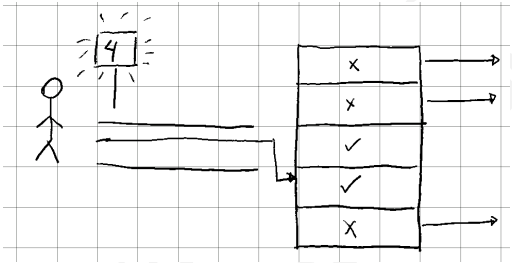
Event leave
refines leave
any c
where
$c \in busy$
then
$busy := busy \setminus \{c\}$

Fill in the Rodin model. POs should become green (otherwise, lasso + P0/ML)

Stages

1. Initial model: just number of clients
2. First refinement: distinguish checkout desks
3. Second refinement: entrance corridor and screen
4. Third refinement: sensors
5. Variant: sets instead of indicator functions

High-level view

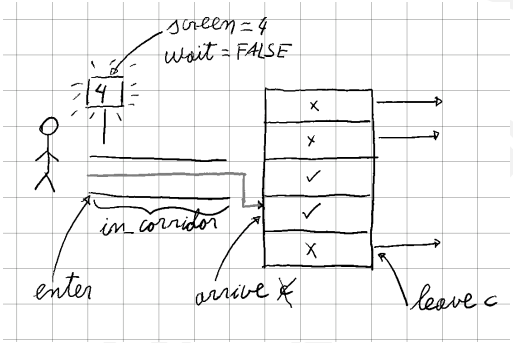


- Will introduce several components.
- Screen: tells clients what to do (controls entrance to corridor).

- One-person, one-way corridor: changes contents of screen.
- Selection of available counter via screen.

Difference with car semaphores: screen goes “red” even if there are free counters (when people in corridor), then may go “green” again.

Initial model considerations



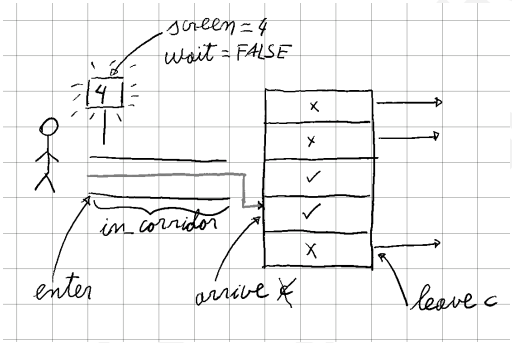
Two variables for display, one for corridor:

- *wait* $\in$ BOOL: clients need to wait?
- *next_counter* $\in$ *COUNTERS*: show free counter / register client destination. (can be used to open physical barrier?).
- *in_corridor* $\in$ BOOL

Relationship below.
Will be captured via invariants.

in_corridor	***wait***	*meaning of* ***next_counter***
FALSE	FALSE	Destination of client (displayed)
FALSE	TRUE	Meaningless (all counters busy, not displayed)
TRUE	FALSE	IMPOSSIBLE
TRUE	TRUE	Destination of client (not displayed)

Initial model considerations



- Introducing event enter.
- Refining events arrive, leave.
- Events & variables model both people, controller.
  - Will be split in next refinement.

Handling the screen

- Could be checked after every state-changing event.
  - Repeated reasoning, models.
  - Specialize events for every situation. (last and non-last car in bridge example)
- Separate events handle screen according to state variables.
- But: additional interleavings, more error possibilities!
- Risky if not verified!

Introducing the model

- Refine m1 into m2.
- New variables and their types:

$$\begin{aligned} \textit{in\_corridor} &\in \{0,1\} \\ \textit{wait} &\in \mathrm{BOOL} \\ \textit{next\_counter} &\in \textit{COUNTERS} \end{aligned}$$

- Initialization:

$$\begin{aligned} \textit{in\_corridor} &:= 0 \\ \textit{wait} &:= \mathrm{FALSE} \\ \textit{next\_counter} &:\in \textit{COUNTERS} \end{aligned}$$

Why $\textit{in\_corridor} \in \{0,1\}$ instead of $\textit{in\_corridor} \in \mathrm{BOOL}$?

Additional security. $\mathrm{in\_corridor} := \mathrm{TRUE}$ may overwrite a previous value of $\mathrm{in\_corridor} = \mathrm{TRUE}$. However, an incorrect $\mathrm{in\_corridor} := \mathrm{in\_corridor} + 1$ will be detected

Requirements and invariants

REQ 0	When the corridor is not empty, the screen displays "WAIT".

$$in\_corridor = TRUE \Rightarrow wait = TRUE$$

REQ 0	When no counter is free, the screen displays "WAIT".

$$busy = COUNTERS \Rightarrow wait = TRUE$$

REQ 0	When access to the corridor is possible, the screen displays the identifier of one of the available counters.

$$wait = FALSE \Rightarrow next\_counter \notin busy$$

Enter them!

The new *enter* and refined *arrive* and *leave*

- leave does not need to be changed.
- A client (can) enter when there is no need to wait.
- The corridor has one more person.
- Other clients have to wait

```
Event enter
  when wait = FALSE
  then
    in_corridor := in_corridor + 1
    wait := TRUE
end
```

Type in “enter”

Refining *arrive*

- *next_counter*: see next slide.

```
Event arrive (abstract)
  refines arrive
  any c
  where
    c ∈ COUNTERS
    c ∉ busy
  then
    busy := busy ∪ {c}
  end
```

- GRD not discharged.

```
Event arrive (concrete)
  refines arrive
  when in_corridor > 0
  with c: c = next_counter
  then
      in_corridor := in_corridor − 1
      busy := busy ∪ {next_counter}
  end
```

- Parameter c disappeared: need to state concrete value for it.
- Modify "arrive"
- GRD needs to relate guards: prove $in\_corridor > 0 \Rightarrow next\_counter \notin busy$

- If it was a gluing invariant, GRD would be proven.
- It is! Add it and GRD should be proven.
- Not a requirement, but (a) necessary lemma and (b) sensible.

Screen management

- Display is set to “WAIT” when a client enters.
- We only need to decide whether we allow more clients to enter.

Event screen_num
  when
    COUNTERS $\neq$ busy
    in_corridor $=$ 0
    wait $=$ TRUE
  then
    next_counter $:\in$ COUNTERS $\setminus$ busy
    wait $:=$ FALSE
  end

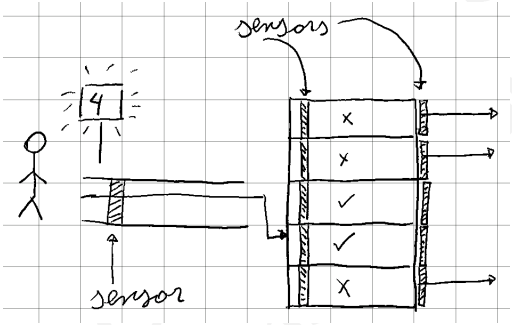
Type them in

All POs should be fine now.

Stages

1. Initial model: just number of clients
2. First refinement: distinguish checkout desks
3. Second refinement: entrance corridor and screen
4. Third refinement: sensors
5. Variant: sets instead of indicator functions

High-level view



- Keep previous “logical” model.
- Add physical model on top, connect with logical model.

- Separate environment and system variables / events.
- Keep interactions clear!
- Guidelines:
  - Some events simulate environment (clients).
  - They react to environment variables and act on sensors.
  - Events that represent the controller.
  - They react to sensors and act on environment variables.

How sensors work

- Not necessarily *real* sensors.
- Client presence activates sensor (a BOOL).
  - Stays on until deactivated by controller.
- Modeling sensor arrays:
  - First idea: use booleans, functions.

$$\begin{aligned} S\_E &\in \mathrm{BOOL} \\ S\_A &\in COUNTER \to \mathrm{BOOL} \\ S\_L &\in COUNTER \to \mathrm{BOOL} \end{aligned}$$

  - S_E sensor entry; S_A sensor arrival; S_L sensor for leaving.
- However, two last ones are indicator sets.
- We can use the **set** of activated sensors.

$$S\_A, S\_L \subseteq COUNTER$$

Using sensors in refined model

- enter, arrive, leave refined.
- **New** events enter_s, arrive_s, leave_s.
  - Note: we will not show leave_s. It is of little interest.
- *_s represent people; they react to environment variables, trigger changes in sensors.
- Modeling agent behavior: variables that represent what people can see, do.

$SCREEN\_CNT \in \{WAIT, NOWAIT\}$	What the screen displays (WAIT or a number)
$CROSSING\_E \in \mathrm{BOOL}$	A person is crossing the corridor sensor
$IN\_CORRIDOR \in \{0, 1\}$	Number of people in the corridor

- *IN_CORRIDOR* could be BOOL. We would then need a gluing invariant with *in_corridor*. Keeping it in $\{0, 1\}$ is easier.

Using sensors in refined model

```
Event enter (abstract)
    refines enter
  when wait = FALSE
  then
      in_corridor := TRUE
      wait := TRUE
  end
```

CROSSING_E in enter_s: a physical person is crossing. Others can see it. We behave correctly.
In enter: controller events should not update environment variables. But we (exceptionally?) model assumption that controllers so fast that when a person has physically crossed, controller has already updated state.

```
Event enter_s
  when SCREEN_CNT = NOWAIT
       CROSSING_E = FALSE
  then
    CROSSING_E := TRUE
    S_E := TRUE
    IN_CORRIDOR := IN_CORRIDOR + 1
  end
```

```
Event enter
    refines enter
  when S_E = TRUE // Only look at sensor
  then // abstract actions plus ...
    S_E := FALSE;
    CROSSING_E := FALSE // See explanation
    SCREEN_CNT = WAIT
  end
```

Using sensors in refined model

```
Event arrive (abstract)
  refines arrive
  when in_corridor > 0
  with c: c = next_counter
  then
    in_corridor := FALSE
    busy := busy ∪ {next_counter}
  end
```

CROSSING_E is used here to ensure that a person has actually crossed the entrance and is in the corridor.

```
Event arrive_s
  when IN_CORRIDOR > 0
       CROSSING_E = FALSE // State update
  then
    IN_CORRIDOR := IN_CORRIDOR − 1
    S_A:= S_A ∪ {next_counter}
  end
```

```
Event arrive
  refines arrive
  when next_counter ∈ S_A
  then
    in_corridor := in_corridor − 1
    busy := busy ∪ {next_counter}
    S_A:= S_A \ {next_counter}
end
```

Physical invariants

- Invariants for environment emulation.

$$
\begin{aligned}
&\text{inv1: } SCREEN\_CNT \in SCREEN \\
&\text{inv2: } IN\_CORRIDOR \in \{0,1\} \\
&\text{inv3: } CROSSING\_E \in BOOL \\
&\text{inv4: } S\_E \in BOOL \\
&\text{inv5: } S\_A \subseteq COUNTERS
\end{aligned}
$$

- We ought to state requirements in the physical model as well (that is what happens in reality).
- We will skip stating requirements in physical model – only for brevity!
- They should be reflected here as well.

Proof obligations

- In my case: pending to discharge
  - enter_s/inv2/INV (IN_CORRIDOR $\in$ {0,1})
  - enter/grd2/GRD (S_E = TRUE $\Rightarrow$ wait = FALSE)
  - arrive/grd1/GRD (next_counter $\in$ S_A $\Rightarrow$ in_corridor > 0)
- We will need additional helping invariants to prove them.
- We will use a new approach: see how the system behaves dynamically.
- Check variable values for possible invariants.
- Try to prove that they are inductive invariants and see if they help proving things.

Animating a model with ProB

- Install ProB from the “Install new software” dialog.
- Check the default values in the Preferences dialog.
- I would increase the size of deferred sets to 5 or 6.
- And set the boundaries for integers to the range -10 to 10.
- Right-click on model ’m3’.
- Drive execution by clicking on the events in the left pane.
- You can see the changes in variables in the pane in the middle.

Animating a model with ProB

- Animating the model shows that it is, fundamentally, an event sequence that enables either leave or screen_num or both at the end.
- It starts again after that.
- We can make a chart of the state of variables after every event.

	INIT	enter_s	enter	arrive_s	arrive
SCREEN_CNT	NOWAIT	NOWAIT	WAIT	WAIT	WAIT
IN_CORRIDOR	0	1	1	0	0
S_E	$\bot$	$\top$	$\bot$	$\bot$	$\bot$
CROSSING_E	$\bot$	$\top$	$\bot$	$\bot$	$\bot$
S_A	$\varnothing$	$\varnothing$	$\varnothing$	$\{n\_c\}$	$\varnothing$
in_corridor	0	0	1	1	0
wait	$\bot$	$\bot$	$\top$	$\top$	$\top$
busy	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\{\ldots\}$

First impressions

- S_E and CROSSING_E have the same values.
- The model can be simplified.
- But: **this** model is slightly oversimplified.
- In a more realistic model, they might be different.
- However, in out current situation we can take advantage of this.
- Add inv6: S_E = CROSSING_E.
- It is inductive and immediately discharged
- It gives additional hypotheses, relationships among variables useful for later proofs.
- Does not immediately help with pending proofs.

First impressions

- The next observation is that S_A is either $\varnothing$ or next_counter .
- That makes sense w.r.t. the expected behavior of the model:
  - Only one person in the corridor.
  - Can enter the corridor only when the corridor is empty.
  - That happens when no one is in the corridor, arrival sensors.
- So we can add inv7: S_A = $\varnothing \vee$ S_A = {next_counter} .
- Does not seem to help anything.
- And inv7/INV not discharged for screen_num .
- However, screen_num does not change S_A, so inv7 should be preserved.
- We will deal with it later.

enter_s/inv2/INV

- Seems in_corridor = 0 before and after enter_s.
- Nobody should be in corridor when a person can enter.
- Also, in_corridor (in the controller) changes with a delay w.r.t. physical environment.
- After INITIALIZATION: SCREEN_CNT = NOWAIT and S_E = FALSE.
- Only state moment we see this.
- We posit the invariant
  inv8: (SCREEN_CNT = NOWAIT $\vee$ S_E = FALSE) $\Rightarrow$ IN_CORRIDOR = 0
- enter_s/inv2/INV can be proven:
  - Remove $\in$ in IN_CORRIDOR $\in$ {0,1} goal (generates disjunction), and
  - Forcing one of the disjunction components to evaluate numerically.
- screen_num/inv8/INV is however not discharged.

- The proof obligation is (S_E = TRUE $\Rightarrow$ wait = FALSE).
- Let us posit it is an invariant.
  - That will discharge GRD automatically.
  - And we can see in the table that S_E = FALSE $\vee$ wait = FALSE seems to hold.
- Add inv9: S_E = FALSE $\vee$ wait = FALSE
- enter/grd2/GRD is now proved.
- enter_s/inv9/INV not discharged.
- We will deal with it later.

- arrive/grd1/GRD pending.
- Let us posit it is an invariant.
- Add inv10: next_counter $\in$ S_A $\Rightarrow$ in_corridor = 1 .
- GRD immediately proven.
- One pending proof from screen_num also discharged.

Linking levels of specification

- We do not have invariants that link the state of the controller and the environment.
- Without them, the model is not really checking thay they agree.
- Let us start linking the status of the corridor.
- If someone is physically in, and the sensor does not register a person, the control has already registered that person.
- (Can also be deduced from the table)
- inv11: IN_CORRIDOR = 1 $\wedge$ S_E = FALSE $\Rightarrow$ in_corridor = 1
- That discharges screen_num/inv8/INV
- arrive/inv11/INV pending to be discharged; we will deal with it later.

Linking levels of specification

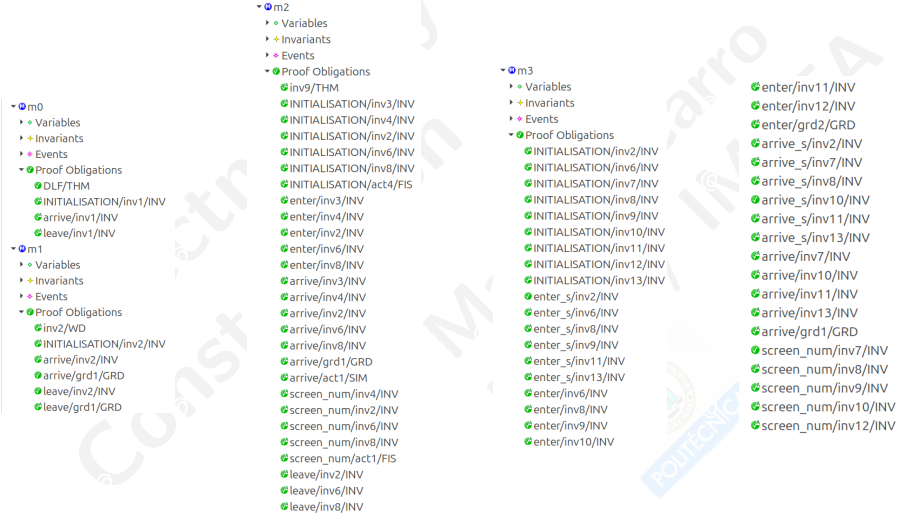
- Also, the values of SCREEN_CNT and wait match (although they have different types)
- Introduce inv12: SCREEN_CNT = NOWAIT $\Leftrightarrow$ WAIT = *FALSE*
- Pending enter_s/inv9/INV is discharged.
- Finally, if there is someone in the corridor, there is no one in the checkout counter sensor
- inv13: IN_CORRIDOR = 1 $\Rightarrow$ S_A = $\varnothing$.
- This discharges arrive/inv11/INV.

Linking levels of specification

- Two pending POs in my case:
  - arrive/inv11/INV
  - screen_num/inv7/INV
- Both can be solved similarly:
  - Click on them, go to the proving perspective.
  - On the proof tree on the left:
    - Right-click on the root node.
    - Select “Prune” to clean the proof attempted so fat.
    - In the proof control window, click on “Lasso” to bring related hypotheses from the available ones.
- That should allow the automatic provers discharge the proof.

Summary of invariants

inv1: $SCREEN\_CNT \in SCREEN$
inv2: $IN\_CORRIDOR \in \{0,1\}$
inv3: $CROSSING\_E \in BOOL$
inv4: $S\_E \in BOOL$
inv5: $S\_A \subseteq COUNTERS$
inv6: $S\_E = CROSSING\_E$
inv7: $S\_A = \varnothing \lor S\_A = \{next\_counter\}$
inv8: $(SCREEN\_CNT = NOWAIT \lor S\_E = FALSE) \Rightarrow IN\_CORRIDOR = 0$
inv9: $S\_E = FALSE \lor wait = FALSE$
inv10: $next\_counter \in S\_A \Rightarrow in\_corridor = 1$
inv11: $IN\_CORRIDOR = 1 \land S\_E = FALSE \Rightarrow in\_corridor = 1$
inv12: $SCREEN\_CNT = NOWAIT \Leftrightarrow WAIT = \mathit{FALSE}$
inv13: $IN\_CORRIDOR = 1 \Rightarrow S\_A = \varnothing$


m0
Variables
Invariants
Events
Proof Obligations
DLF/THM
INITIALISATION/inv1/INV
arrive/inv1/INV
leave/inv1/INV
m1
Variables
Invariants
Events
Proof Obligations
inv2/WD
INITIALISATION/inv2/INV
arrive/inv2/INV
arrive/grd1/GRD
leave/inv2/INV
leave/grd1/GRD
m2
Variables
Invariants
Events
Proof Obligations
inv9/THM
INITIALISATION/inv3/INV
INITIALISATION/inv4/INV
INITIALISATION/inv2/INV
INITIALISATION/inv6/INV
INITIALISATION/inv8/INV
INITIALISATION/act4/FIS
enter/inv3/INV
enter/inv4/INV
enter/inv2/INV
enter/inv6/INV
enter/inv8/INV
arrive/inv3/INV
arrive/inv4/INV
arrive/inv2/INV
arrive/inv6/INV
arrive/inv8/INV
arrive/grd1/GRD
arrive/act1/SIM
screen_num/inv4/INV
screen_num/inv2/INV
screen_num/inv6/INV
screen_num/inv8/INV
screen_num/act1/FIS
leave/inv2/INV
leave/inv6/INV
leave/inv8/INV
m3
Variables
Invariants
Events
Proof Obligations
INITIALISATION/inv2/INV
INITIALISATION/inv6/INV
INITIALISATION/inv7/INV
INITIALISATION/inv8/INV
INITIALISATION/inv9/INV
INITIALISATION/inv10/INV
INITIALISATION/inv11/INV
INITIALISATION/inv12/INV
INITIALISATION/inv13/INV
enter_s/inv2/INV
enter_s/inv6/INV
enter_s/inv8/INV
enter_s/inv9/INV
enter_s/inv11/INV
enter_s/inv13/INV
enter/inv6/INV
enter/inv8/INV
enter/inv9/INV
enter/inv10/INV
enter/inv11/INV
enter/inv12/INV
enter/grd2/GRD
arrive_s/inv2/INV
arrive_s/inv7/INV
arrive_s/inv8/INV
arrive_s/inv10/INV
arrive_s/inv11/INV
arrive_s/inv13/INV
arrive/inv7/INV
arrive/inv10/INV
arrive/inv11/INV
arrive/inv13/INV
arrive/grd1/GRD
screen_num/inv7/INV
screen_num/inv8/INV
screen_num/inv9/INV
screen_num/inv10/INV
screen_num/inv12/INV