A Market Compliant with COVID-19 Regulations

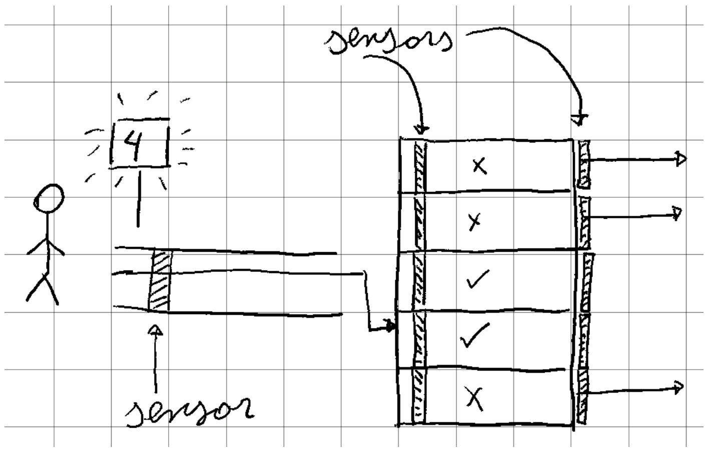
Manuel Carro
manuel.carro@upm.es

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.

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.

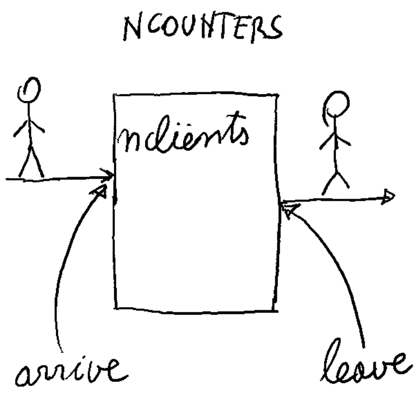
Modeling approach

- 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 $\in$??

Model

Context c0

CONSTANTS NCOUNTERS
AXIOMS NCOUNTERS $\in$??

Machine m0

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

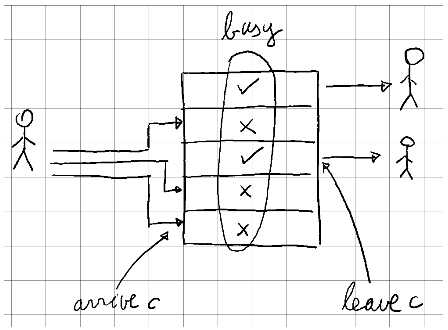
Event arrive
when $nclients < NCOUNTERS$
then
$nclients := nclients + 1$
end

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?

Model state

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

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

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!

Model state: context and invariants

Context c1

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

Create it!

- WD PO not discharged!

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)

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 ???

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
$busy \subseteq COUNTERS$

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, $busy =$

Events

- Initially, $busy = \varnothing$

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

  then

Event leave
  refines leave
  any c
  where

  then

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

Event leave
  refines leave
  any c
  where

  then

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

  then

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

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
    $busy := busy \setminus \{c\}$

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
    $busy := busy \cup \{c\}$

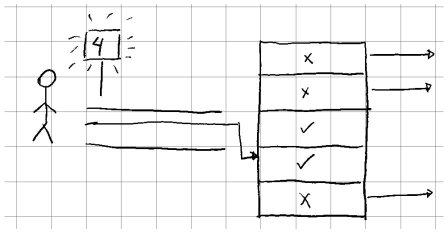
Event leave
  refines leave
  any c
  where
    $c \in busy$
  then
    $busy := busy \setminus \{c\}$

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

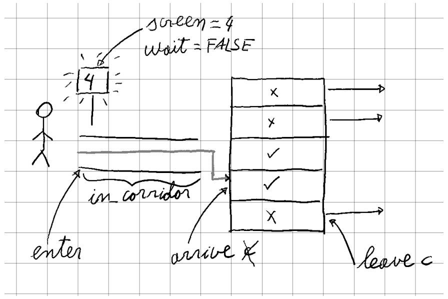


- Will introduce several components.
- Screen: tells clients what to do (controls entrance to corridor).

- 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



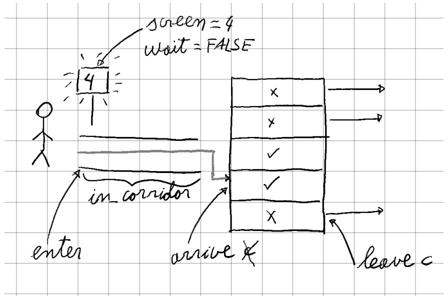
Two variables for display, one for corridor:

- *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:

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

- Initialization:

$$\begin{aligned} \mathit{in\_corridor} &:= \\ \mathit{wait} &:= \\ \mathit{next\_counter} &:\in \end{aligned}$$

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

Introducing the model

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

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

- Initialization:

$$\begin{aligned} \mathit{in\_corridor} &:= \\ \mathit{wait} &:= \\ \mathit{next\_counter} &:\in \end{aligned}$$

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

Additional security. in_corridor := TRUE may overwrite a previous value of in_corridor = TRUE. However, an incorrect in_corridor := in_corridor + 1 will be detected

Introducing the model

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

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

- Initialization:

$$
\begin{aligned}
\mathit{in\_corridor} &:= 0\\
\mathit{wait} &:=\\
\mathit{next\_counter} &:\in
\end{aligned}
$$

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

Additional security. in_corridor := TRUE may overwrite a previous value of in_corridor = TRUE. However, an incorrect in_corridor := in_corridor + 1 will be detected

Introducing the model

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

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

- Initialization:

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

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

Additional security. in_corridor := TRUE may overwrite a previous value of in_corridor = TRUE. However, an incorrect in_corridor := in_corridor + 1 will be detected

Introducing the model

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

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

- Initialization:

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

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

Additional security. in_corridor := TRUE may overwrite a previous value of in_corridor = TRUE. However, an incorrect in_corridor := in_corridor + 1 will be detected

Requirements and invariants

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

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".

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.

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) enters 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
```

- Arrive at counter previously shown in screen, counter becomes busy.

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

- Note "c:" in the label of "with": it is necessary (next slide)!
- Type in "enter", modify "arrive

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.
- 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


    wait = TRUE
  then
    next_counter :∈ COUNTERS \ busy
    wait := FALSE
  end
```

Type them in
All POs should be fine now.

Screen management

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

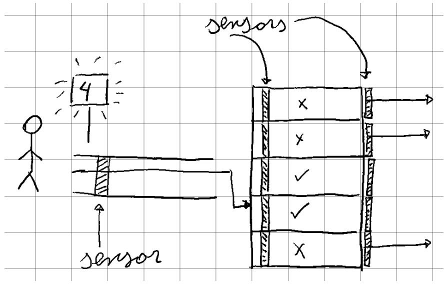
```
Event screen_num
  when
    COUNTERS ≠ busy
    in_corridor = 0
    wait = TRUE
  then
    next_counter :∈ COUNTERS \ busy
    wait := FALSE
  end
```

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. $\boxed{\text{Third refinement: sensors}}$
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 updated
  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
```

Proof obligations

- Some additional work regarding POs needs to be done.
- IN_CORRIDOR ∈ {0,1} invariant for enter_s.
- GRD for enter, arrive.
- Plus we will introduce a sensible invariant: only one sensor is active at a time:

  inv_sens_arr: ???
- Needs to be discharged for arrive_s

Proof obligations

- Some additional work regarding POs needs to be done.
- IN_CORRIDOR ∈ {0,1} invariant for enter_s.
- GRD for enter, arrive.
- Plus we will introduce a sensible invariant: only one sensor is active at a time:

  inv_sens_arr: card(S_A) ≤ 1
- Needs to be discharged for arrive_s

card(S_A) ≤ 1

The (minimal) sequent to discharge (see proving perspective – goal slightly simplified) is

$card(S\_A) \leq 1, IN\_CORRIDOR > 0, CROSSING\_E = FALSE$
$\vdash card(S\_A) \leq card(S\_A \bigcap \{next\_counter\})$

Can be proven if S_A = ∅. Note we have IN_CORRIDOR > 0 and it makes sense that no one is entering the counter if there is a person in the corridor (see arrive_s). Therefore the invariant

$$IN\_CORRIDOR > 0 \Rightarrow S\_A = \varnothing$$

(if provable) would be helpful. After adding it, proving cardinality is possible with lasso + "remove membership" in the hypothesis IN_CORRIDOR ∈ {0, 1} (click on membership symbol).

$IN\_CORRIDOR > 0 \Rightarrow S\_A = \varnothing$

Invariant needs discharging now in enter_s.
We will delay it.

GRD POs

- GRD POs for enter and arrive are pending.
- They would be

$$next\_counter \in S\_A \Rightarrow in\_corridor > 0$$

for arrive and

$$S\_E = TRUE \Rightarrow wait = FALSE$$

for enter. We will start with the latter.

GRD of enter

- PO for guard strengthening: $S\_E = TRUE \Rightarrow wait = FALSE$.
- After positing it as invariant, GRD is proven but the new invariant remains to be proven.
- $SCREEN\_CNT = NOWAIT \Rightarrow wait = FALSE$ as invariant can be proven and helps prove the previous one.

GRD of arrive

- PO for guard strengthening: $next\_counter \in S\_A \Rightarrow in\_corridor > 0$.
- Add as invariant. GRD is proven.
- New invariant needs to be discharged for arrive_s.
- Another, intermediate invariant helps prove it: $(IN\_CORRIDOR = 1 \wedge CROSSING\_E = FALSE) \Rightarrow in\_corridor = 1$
- At this point, all POs but one should be discharged.

Origin of $(IN\_CORRIDOR = 1 \wedge CROSSING\_E = FALSE) \Rightarrow in\_corridor = 1$

- The PO in the prover view needs to discharge $S\_A \neq 0 \Rightarrow in\_corridor = 1$.
- Inspecting the hypothesis we have $S\_A \neq 0$. So we need to deduce that $in\_corridor = 1$.
- The rest of the "facts" that we have among the hypotheses are $IN\_CORRIDOR = TRUE$ and $CROSSING\_E = FALSE$.
- Perhaps we can use them to infer $in\_corridor = 1$.

Introducing Model Checking, ProB, and Animations

Origin of $(IN\_CORRIDOR = 1 \wedge CROSSING\_E = FALSE) \Rightarrow in\_corridor = 1$

- Animating the model shows that it is, fundamentally, an event sequence that can fire either leave or screen_num at the end.
- 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	$\perp$	$\top$	$\top$	$\perp$	$\perp$
S_E, CROSSING_E	$\perp$	$\top$	$\perp$	$\perp$	$\perp$
S_A	$\varnothing$	$\varnothing$	$\varnothing$	$\{n\_c\}$	$\varnothing$
in_corridor	0	0	1	1	0
wait	$\perp$	$\perp$	$\top$	$\top$	$\top$
busy	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\{n\_c\}$

The two facts we have in our hypotheses ($IN\_CORRIDOR = TRUE$ and $CROSSING\_E = FALSE$) are true only after enter (the state in which arrive_s is executed) and in_corridor = 1. The implication is then a true invariant. Fortunately, it is also an inductive invariant.

Last PO

- Intermediate invariants helped prove pending POs.
- Pending: $S\_A \neq \varnothing \Rightarrow in\_corridor > 0$.
- Simplifying, it requires proving:

$$in\_corridor \in \{0, 1\}, next\_counter \in S\_A, \neg S\_A \subseteq \{next\_counter\}, card(S\_A) \leq 1 \vdash in\_corridor > 1$$

- I was not able to discharge it automatically.
- However, some Rodin user at the Event-B mailing list stated he could.
- But his (quite simple) strategy did not work for me.
- But it should be true – see why in the next slide.

Last PO (Cont.)

To prove (abstracting variable names and adding additional axioms):

$$v \in \{0,1\}, \mathrm{finite}(S), c \in S, \neg S \subseteq \{c\}, card(S) \leq 1 \vdash v > 1$$

- $v > 1$ cannot be inferred, as $v \in \{0,1\}$.
- Then: prove inconsistency in LHS.
- Since $card(S) \leq 1$, S has either zero or one elements.
- Since $c \in S$, S has at least one element.
- Then, $card(S) = 1$ and $S = \{c\}$.
- But we have $\neg(S \subseteq \{c\})$.
- That would mean that $\neg(\{c\} \subseteq \{c\})$.
- We have a contradiction in the LHS.
- Therefore the sequent is proven.
- I have left it as reviewed.
- Model checking can't find a counterexample, either.