A Market Compliant with COVID-19 Regulations

Manuel Carro
manuel.carro@upm.es

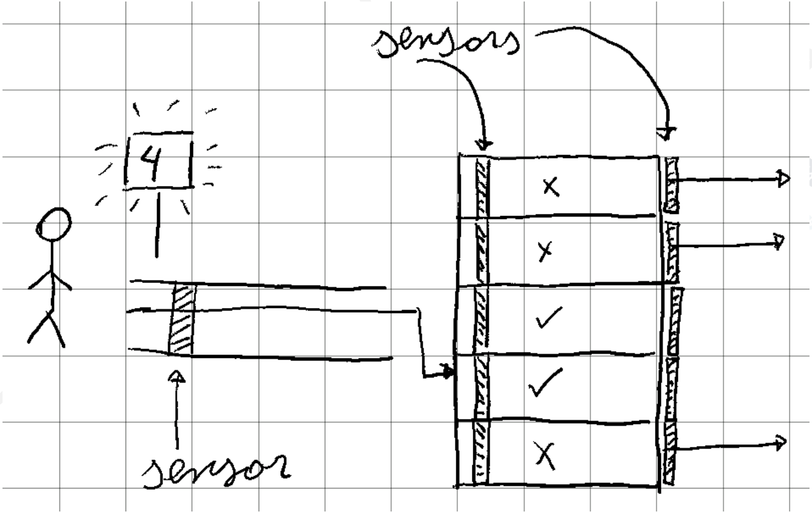
Universidad Politécnica de Madrid &
IMDEA Software Institute

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.

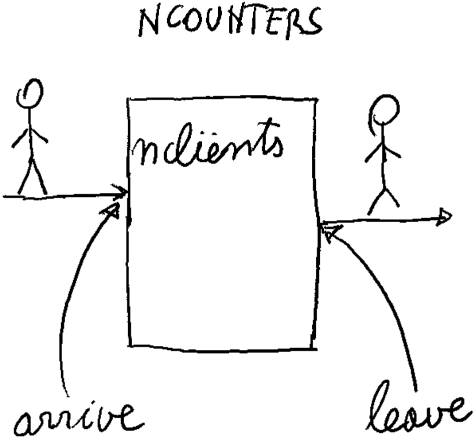
Modeling approach

- 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

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

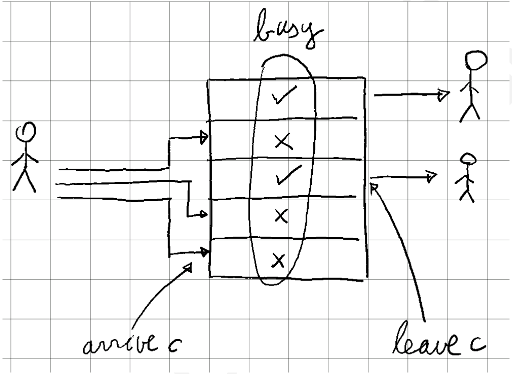
Event arrive
where $nclients < NCOUNTERS$
then
$nclients := nclients + 1$
end

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

Stages

1. Initial model: just number of clients
2. $\boxed{\text{First refinement: distinguish checkout desks}}$
3. Second refinement: entrance corridor and screen
4. Third refinement: sensors

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?

$$\mathit{available} \in 1..\mathit{NCOUNTERS} \rightarrow \mathrm{BOOL}$$

Model state

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

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

- But a function $A \rightarrow \mathrm{BOOL}$ 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, $busy = \emptyset$

Events

- Initially, $\mathit{busy} = \varnothing$
- 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 ∈ COUNTERS
    c ∉ 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
    $busy := busy \cup \{c\}$

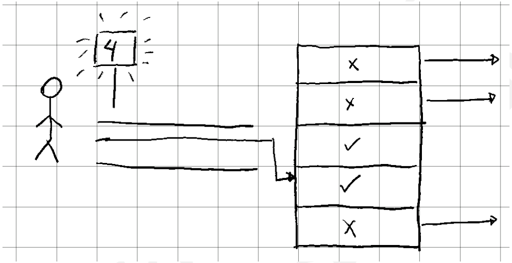
Event leave
  refines leave
  any c
  where
    $c \in busy$
  then
    $busy := busy \setminus \{c\}$

Fill in the Rodin model. POs should become green (otherwise, lasso + P0/ML)
arrive/grd1/GRD may need simplifying comparison in goal

Stages

1. Initial model: just number of clients
2. First refinement: distinguish checkout desks
3. $\boxed{\text{Second refinement: entrance corridor and screen}}$
4. Third refinement: sensors

High-level view

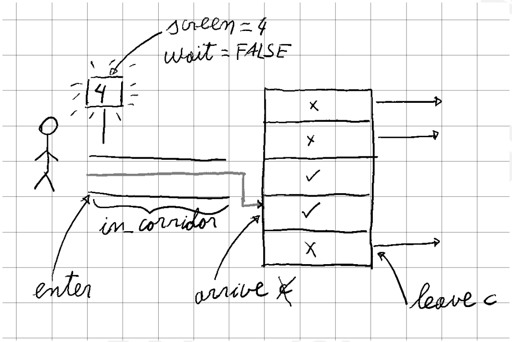


- Will introduce several components.
- Screen: tells clients what to do (controls entrance to corridor).

- 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



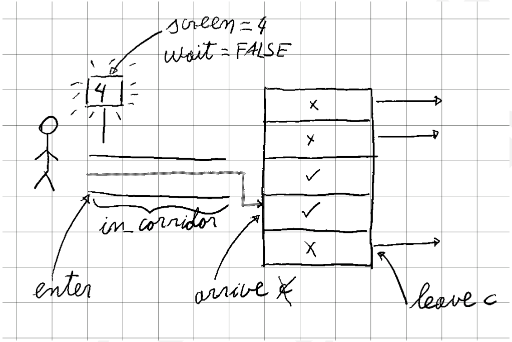
Two variables for display, one for corridor:

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

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:

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

- Initialization:

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

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

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

Introducing the model

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

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

- Initialization:

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

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

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".

Requirements and invariants

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

$$in\_corridor = 1 \Rightarrow wait = TRUE$$

(Note: this formula is equivalent to the IMPOSSIBLE line in slide 33)

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”.

$$in\_corridor = 1 \Rightarrow wait = TRUE$$

(Note: this formula is equivalent to the IMPOSSIBLE line in slide 33)

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".

$$in\_corridor = 1 \Rightarrow wait = TRUE$$

(Note: this formula is equivalent to the IMPOSSIBLE line in slide 33)

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
  where 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
  where 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
  $in\_corridor > 0 \Rightarrow next\_counter \notin busy$

- If invariant $\Rightarrow$ GRD 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
  where

    wait $=$ TRUE
  then
    next_counter $:\in$ COUNTERS $\setminus$ 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
  where
    $\mathsf{COUNTERS} \neq \mathsf{busy}$
    $\mathsf{in\_corridor} = 0$
    $\mathsf{wait} = \mathsf{TRUE}$
  then
    $\mathsf{next\_counter} :\in \mathsf{COUNTERS} \setminus \mathsf{busy}$
    $\mathsf{wait} := \mathsf{FALSE}$
  end

Type them in
All POs should be fine now.

Rationale for screen management

- Hybrid approach
  - From NOWAIT to WAIT in “enter” event.
  - From WAIT to NOWAIT in specific event.
- NOWAIT $\Rightarrow$ WAIT can only happen when a person enters corridor.
  - enter is appropiate.
  - Plus (for safety), the screen should turn to WAIT immediately when a person entering corridor is detected.
  - Separate event $\Rightarrow$ interleaving of other events possible, unless additional logic (& complexity) is added.

Rationale for screen management

- WAIT $\Rightarrow$ NOWAIT could happen after arrive or leave.
- Related logic in two events.
- In arrive:
  - Only if there are available counters.
  - Needs two variants of arrive (as in the "Cars in a narrow bridge" example).
- In leave:
  - Only if the corridor is empty.
  - Needs two variants of leave.
- All that logic can be put in a single separate event (screen_num).
- Having another event activated before screen_num is safe: it would only delay more clients entering the corridor.

Deadlock freedom

- As usual, disjunction of guards.
- Events with parameters need special treatment.

Event leave
  any c
  where
    $c \in busy$
  then
    $\ldots$

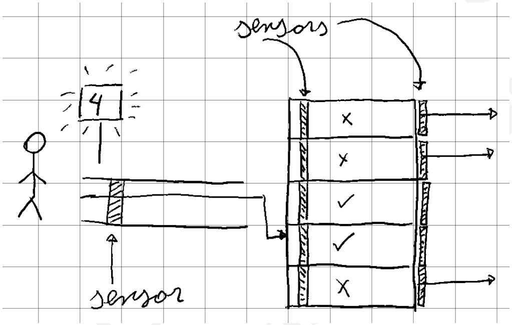
- Logical reading: the event is enabled if there is some c such that $c \in busy \wedge \ldots$.
- DLF:

$$
\begin{aligned}
wait = FALSE \vee \\
in\_corridor > 0 \vee \\
(\exists x \cdot x \in busy) \vee \\
(COUNTERS \neq busy \wedge \\
in\_corridor = 0 \wedge \\
wait = TRUE)
\end{aligned}
$$

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

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.

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

  - 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.

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

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)
$CROSSING\_E \in \mathrm{BOOL}$	Sensor: a person enters the corridor
$IN\_CORRIDOR \in \{0, 1\}$	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
  where 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?) modelthe assumption that the controller is fast enough to update its state in zero time after a person physically crosses the sensor.

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

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

Using sensors in refined model

```
Event arrive (abstract)
  refines arrive
  where 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
  where 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
  where next_counter ∈ S_A
  then
    in_corridor := in_corridor − 1
    busy := busy ∪ {next_counter}
    S_A:= S_A \ {next_counter}
end
```

Using sensors in refined model

Event screen_num (abstract)
  where wait = TRUE
    COUNTERS $\neq$ busy
    in_corridor = 0
  then
    next_counter :$\in$ COUNTERS $\setminus$ busy
    wait := FALSE
  end

Event screen_num (concrete)
  where wait = TRUE
    COUNTERS $\neq$ busy
    in_corridor = 0
  then
    next_counter :$\in$ COUNTERS $\setminus$ busy
    wait := FALSE
    SCREEN_CNT := NOWAIT
  end

Physical invariants

- Invariants for environment emulation.

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

- 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.

Changes to model

- Extend context c1 into c2.
- Add set $SCREEN$, constants $WAIT$, $NOWAIT$.
- Axioms: $SCREEN = \{WAIT, NOWAIT\}$, $WAIT \neq NOWAIT$.

- Refine m2 into m3, should *see* c2.
- Add variables $SCREEN\_CNT$, $IN\_CORRIDOR$, $CROSSING\_E$, S_E, S_A
- Add invariants:

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

- Add / modify events (next two slides)

Changes to model (Cont.)

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

Event enter
   refines  enter
  where S_E = TRUE // Only look at sensor
  then
    in_corridor := in_corridor + 1
    wait := TRUE
    S_E := FALSE;
    CROSSING_E := FALSE
    SCREEN_CNT = WAIT
  end

Event arrive_s
  where 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
  where next_counter ∈ S_A
  then
    in_corridor := in_corridor − 1
    busy := busy ∪ {next_counter}
    S_A:= S_A \ {next_counter}
end
```

Changes to model (Cont.)

$$
\begin{array}{l}
\textbf{Event } screen\_num \\
\quad \textbf{where } wait = TRUE \\
\qquad\qquad COUNTERS \neq busy \\
\qquad\qquad in\_corridor = 0 \\
\quad \textbf{then} \\
\qquad next\_counter :\in COUNTERS \setminus busy \\
\qquad wait := FALSE \\
\qquad SCREEN\_CNT := NOWAIT \\
\textbf{end}
\end{array}
$$

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

- Animation: fundamentally, 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	$\perp$	$\top$	$\perp$	$\perp$	$\perp$
CROSSING_E	$\perp$	$\top$	$\perp$	$\perp$	$\perp$
S_A	$\varnothing$	$\varnothing$	$\varnothing$	$\{n\_c\}$	$\varnothing$
in_corridor	0	0	1	1	0
wait	$\perp$	$\perp$	$\top$	$\top$	$\top$

- Useful to infer patterns.
- Must be proven (intuition / separate simulations not conclusive)!

First impressions

- S_E and CROSSING_E **seem to have** the same values.
- Inspect the events

First impressions

- S_E and CROSSING_E **seem to have** the same values.
- Inspect the events
- Can be fused, but **this** model is oversimplified.
  - More realistic model $\Rightarrow$ they might differ.
- We can however reflect 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 apparently 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.
- Inspect events.

First impressions

- The next observation is that apparently 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.
- Inspect events.
- So it seems we could add inv7: S_A = $\varnothing$ $\vee$ S_A = {next_counter} .
  - Does not seem to help.
  - And inv7/INV not discharged for screen_num .
  - screen_num does not change S_A, but it changes next_counter.
    - S_A should be $\varnothing$ after screen_num.
    - Since it is $\varnothing$ after arrive, and leave does not change it, it seems it should be so (see animation for intuition).

screen_num/inv8/INV

- Checking event: screen_num requires in_corridor = 0 (and does not change it)
- Checking chart: whenever in_corridor = 0, S_A = $\varnothing$.
- Se we can posit inv8: $in\_corridor = 0 \Rightarrow S\_A = \varnothing$.
- screen_num/inv7/INV immediately proven.
- And arrive/grd1/GRD also!
  - arrive/grd1/GRD PO is next_counter $\in$ S_A $\Rightarrow$ in_corridor = 1.
  - Since we had S_A = $\varnothing$ $\vee$ S_A = {next_counter}, the GRD PO is equivalent to inv8.
- But: inv8 PO unproven for two events.

enter_s/inv2/INV

- IN_CORRIDOR $\in \{0, 1\}$.
- enter_s increments IN_CORRIDOR.
- Prove that IN_CORRIDOR = 0 whenever enter_s is enabled.
  - Guard: SCREEN_CNT = NOWAIT $\wedge$ CROSSING_E = FALSE
- From the chart: if SCREEN_CNT = NOWAIT $\wedge$ CROSSING_E = FALSE, then IN_CORRIDOR = 0.
  - Intuition: the corridor should be empty when a person can enter.
- We posit the invariant
  inv9: (SCREEN_CNT = NOWAIT $\wedge$ CROSSING_E = FALSE) $\Rightarrow$
  IN_CORRIDOR = 0
- enter_s/inv2/INV is proven. If not:
  - Remove $\in$ in IN_CORRIDOR $\in$ {0,1} goal (generates disjunction), and
  - Forcing one of the disjunction components to evaluate numerically.
- screen_num/inv9/INV is however not discharged.

enter/grd1/GRD

- 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 = TRUE $\Rightarrow$ wait = FALSE seems to hold.
- Add inv10: S_E = TRUE $\Rightarrow$ wait = FALSE
- enter/grd1/GRD is now proved.
- enter/inv8/INV discharged as well.
- enter_s/inv10/INV not discharged.
- We will deal with it later.

Why does adding a PO (GRD, SIM, ...) as invariant helps?

- If the PO already generated the formula to be proven, why adding it explicitly can be good?
- Adding it as an invariant makes the prover discharge the PO immediately.
  - (But it has to be proven to hold for every event activation).
- And (as we have seen), making it explicit as an invariant may help other POs to be discharged.

enter_s/inv10/INV

- inv10: $S\_E = TRUE \Rightarrow wait = FALSE$
- enter_s sets S_E = TRUE.
- Its guard is $SCREEN\_CNT = NOWAIT \wedge CROSSING\_E = FALSE$.
- Try to infer a relationship between the guard and the value of 'wait' that can be an invariant.
- enter_s changes CROSSING_E , so we cannot use it.
- It seems that the values of SCREEN_CNT and wait match (although they have different types)
- Introduce inv11: $SCREEN\_CNT = NOWAIT \Leftrightarrow wait = \mathit{FALSE}$
- Pending enter_s/inv10/INV is discharged.
- And inv11 is proven as invariant.

screen_num/inv9/INV

- inv9: $(SCREEN\_CNT = NOWAIT \wedge CROSSING\_E = FALSE) \Rightarrow IN\_CORRIDOR = 0$
- screen_num does not change IN_CORRIDOR.
- Try to identify and add an invariant related to IN_CORRIDOR that uses the state in which screen_num can be enabled.
- Guard: $in\_corridor = 0 \wedge wait = TRUE$.
- Chart: seems that if these are true, then $IN\_CORRIDOR = 0$.
- Let us posit the invariant
  inv12: $(in\_corridor = 0 \wedge wait = TRUE) \Rightarrow IN\_CORRIDOR = 0$
- Intuition: if controller registers corridor empty and people have to wait, there must not (physically) be anyone in the corridor.
- screen_num/inv9/INV is discharged.
- But arrive/inv12/INV to be discharged.

arrive/inv12/INV

- inv12: $(in\_corridor = 0 \wedge wait = TRUE) \Rightarrow IN\_CORRIDOR = 0$
- arrive does not change IN_CORRIDOR.
- But it requires $S\_A \neq \varnothing$.
- Let us try to link S_A with IN_CORRIDOR.
- From the chart, it seems that if $S\_A \neq \varnothing$, then $IN\_CORRIDOR = 0$.
- Let us posit
  inv13: $S\_A \neq \varnothing \Rightarrow IN\_CORRIDOR = 0$.
- It is an invariant and it discharges arrive/inv12/INV.

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: $in\_corridor = 0 \Rightarrow S\_A = \varnothing$
inv9: $(SCREEN\_CNT = NOWAIT \land CROSSING\_E = FALSE) \Rightarrow IN\_CORRIDOR = 0$
inv10: $S\_E = TRUE \Rightarrow wait = FALSE$
inv11: $SCREEN\_CNT = NOWAIT \Leftrightarrow WAIT = \mathit{FALSE}$
inv12: $(in\_corridor = 0 \land wait = TRUE) \Rightarrow IN\_CORRIDOR = 0$
inv13: $S\_A \neq \varnothing \Rightarrow IN\_CORRIDOR = 0$

Deadlock freedom

- Proofs somewhat complex.
  - Additional invariants needed.
- Model checker did not detect deadlocks.
- But limited reach.
- Left as an exercise!