

# One-Way Bridge<sup>1</sup>

Manuel Carro

[manuel.carro@upm.es](mailto:manuel.carro@upm.es)

Universidad Politécnica de Madrid &  
IMDEA Software Institute

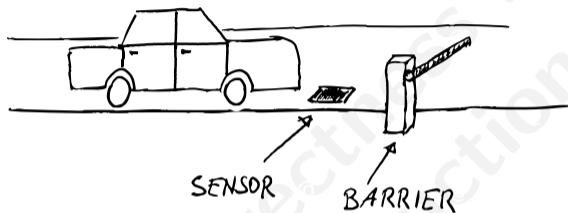
---

<sup>1</sup>Example and several slides from J. R. Abrial book *Modeling in Event-B: system and software engineering*.

Goals .....	s. 3
Requirements .....	s. 7
Initial model .....	s. 17
First refinement: one-way bridge .....	s. 29
Second refinement: traffic lights .....	s. 56
Third refinement: sensors .....	s. 114

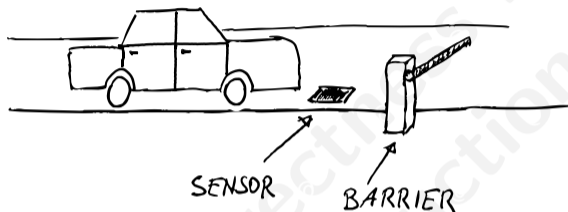
- Example of reactive system development.
- Including modeling the environment.
- Invariants: **capture requirements**.
  - Invariant preservation will prove that requirements are respected.
- Increasingly accurate models (refinement).
- Refinements “zoom in”, see more details.
- Models separately proved correct.
  - Final system: correct by construction.
- Correctness criteria: proof obligations.
- Proofs: helped by theorem provers working on sequent calculus.

- Previous examples were *transformational*.
  - Input  $\Rightarrow$  transformation  $\Rightarrow$  output.
- Current example:
  - Interaction with **environment**.
- Sensors and communication channels:
  - Hardware sensors modeled with events.
  - Channels modeled with variables.



- **Control software reads sensor, raises barrier.**
  - If conditions allow it.

- Software behavior **relies on environment:**
  - Cars **stop** on a closed barrier.
  - Cars drive **over** sensor.
  - ...
- **Correctness proofs:** take this behavior into account.



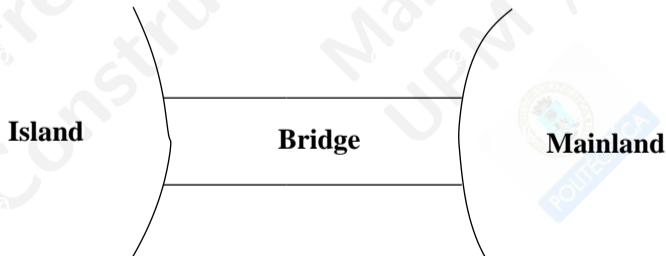
- **Control software reads sensor, raises barrier.**
  - If conditions allow it.

- Software behavior **relies on environment**:
  - Cars **stop** on a closed barrier.
  - Cars drive **over** sensor.
  - ...
- **Correctness proofs**: take this behavior into account.
  - Model external actions as **events**.
    - E.g., sensor signal raised by event.
    - Following expected behavior.
  - Software control also events.
  - Everything subject to proofs.

- Sequential systems specified through  $\{Pre\} P \{Post\}$ .
  - Considerably more difficult in case of (a) large real-world and (b) reactive systems.
  - Building it piece-wise, modeling (natural-language) requirements and ensuring they are respected: a way to ensure we have a detailed system specification that is provable correct.
- 
- Two kinds of requirements:
    - Concerned with the equipment (EQP).
    - Concerned with system functionality (FUN).
  - Objective: control cars on a narrow bridge.
  - Bridge links the mainland to (small) island.

The system is controlling cars on a bridge between the mainland and an island	FUN-1
---	-------

- This can be illustrated as follows





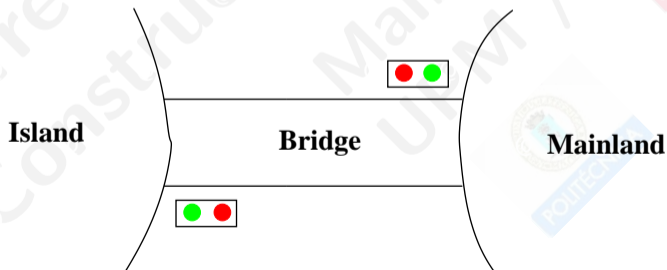
- The controller is equipped with two traffic lights with two colors.

The system has two traffic lights with two colors: green and red

EQP-1

## Requirements

- One of the traffic lights is situated on the mainland and the other one on the island. Both are close to the bridge.
- This can be illustrated as follows



The traffic lights control the entrance to the bridge at both ends of it

EQP-2

- Drivers are supposed to obey the traffic light by not passing when a traffic light is red.

Cars are not supposed to pass on a red traffic light, only on a green one

EQP-3

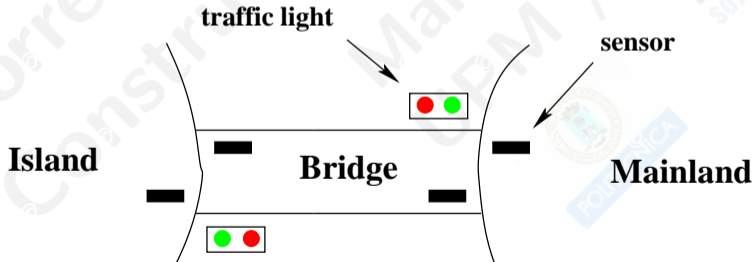
- There are also some car sensors situated at both ends of the bridge.
- These sensors are supposed to detect the presence of cars intending to enter or leave the bridge.
- There are four such sensors. Two of them are situated on the bridge and the other two are situated on the mainland and on the island.

The system is equipped with four car sensors each with two states: on or off

EQP-4

<p>The sensors are used to detect the presence of cars entering or leaving the bridge</p>	<p>EQP-5</p>
---	--------------

- The pieces of equipment can be illustrated as follows:



- This system has two main constraints: the number of cars on the bridge and the island is limited and the bridge is one way.

The number of cars on the bridge and the island is limited	FUN-2
--	-------

The bridge is one way or the other, not both at the same time	FUN-3
---	-------

- Software controller has model of the world.
  - In some sense, it partially simulates it.
  - Knowledge of world through sensors.
  - Incrementally adding requirements, proving they are implemented.
- When finished, an additional software layer (= more events) simulate the “real world”.
  - “Real world” simulation only interacts with controller through sensors, actuators.
  - Proof that controller + simulation follow requirements.
- Real implementation: strip “Real world” layer, derive code from software controller.

**Initial model** Limiting the number of cars (FUN-2).

**First refinement** Introducing the one-way bridge (FUN-3).

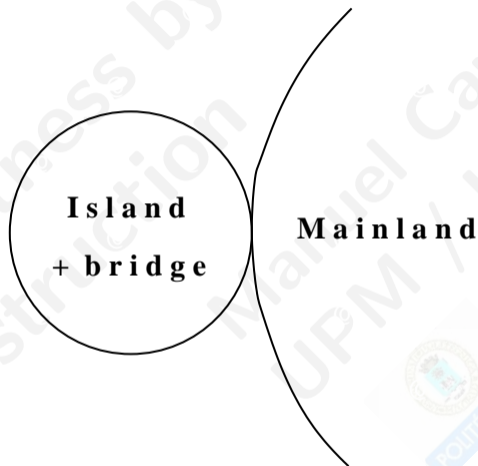
**Second refinement** Introducing the traffic lights (EQP-1,2,3)

**Third refinement** Introducing the sensors (EQP-4,5)

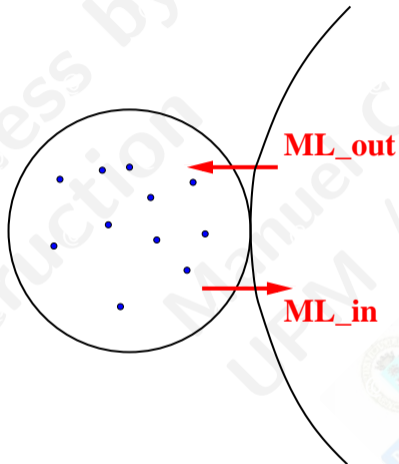


- We ignore the equipment (traffic lights and sensors).
- We do not consider the bridge.
- We focus on the pair island + bridge.
- FUN-2: limit number of cars on island + bridge.

## Situation from the sky



## Situation from the sky



## Formalization of state

✓ *Create project Cars, context  $c0$ , machine  $m0$ , add constant, axiom, variable, invariants, initialization*

Static part (context):

**constant:**  $d$

**axm0\_1:**  $d \in \mathbb{N}$

$d$  is the maximum number of cars allowed in island + bridge.

- **Labels** axm0\_1, inv0\_1, chosen **systematically**.
- Label **axm**, **inv** recalls purpose.
- **0** (as in **inv0\_1**): initial model.

Dynamic part (machine):

**variable:**  $n$

**inv0\_1:**  $n \in \mathbb{N}$

**inv0\_2:**  $n \leq d$

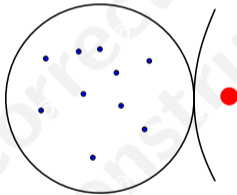
$n$  number of cars in island + bridge

Always smaller than or equal to  $d$  (**FUN\_2**)

- Later: **inv1\_1** for invariant 1 of refinement 1, etc.
- Any **systematic** convention is valid.

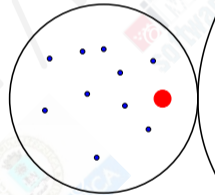
## Situation from the sky

- This is the **first transition** (or event) that can be **observed**
- A car is leaving the mainland and entering the Island-Bridge



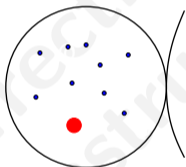
**Before**

**ML\_out**



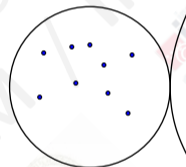
**After**

- We can also observe a **second transition** (or event)
- A car leaving the Island-Bridge and re-entering the mainland



Before

ML\_in



After

- The **number of cars** in the Island-Bridge is **decremented**

✓ Create events `ML_out`, `ML_in`. Add actions. Guards?

- Event `ML_out` increments the number of cars

**ML\_out**  
 $n := n + 1$

- Event `ML_in` decrements the number of cars

**ML\_in**  
 $n := n - 1$

- An event is denoted by its name and its action (an assignment)

## INITIALISATION

 $n := 0$ 

Event ML\_out

where

 $n < d$ 

then

 $n := n + 1$ 

end

Event ML\_in

where

 $0 < n$ 

then

 $n := n - 1$ 

end

ML\_out/inv0\_1/INV

 $d \in \mathbb{N}, n \in \mathbb{N}, n \leq d, n < d \vdash n + 1 \in \mathbb{N}$ 

ML\_out/inv0\_2/INV

 $d \in \mathbb{N}, n \in \mathbb{N}, n \leq d, n < d \vdash n + 1 \leq d$ 

ML\_in/inv0\_1/INV

 $d \in \mathbb{N}, n \in \mathbb{N}, n \leq d, 0 < n \vdash n - 1 \in \mathbb{N}$ 

ML\_in/inv0\_2/INV

 $d \in \mathbb{N}, n \in \mathbb{N}, n \leq d, n < d \vdash n - 1 < d$



- It is common to require that physical systems progress.
- We want cars to be able to either enter or exit.
- Therefore, (some) event(s) have to always be enabled.
- Depends on guards: *deadlock freedom*.

$$A_{1\dots l}, I_{1\dots m} \vdash \bigvee_{i=1}^n G_i(v, c)$$

- In our case:

$$d \in \mathbb{N}, n \in \mathbb{N}, n \leq d \vdash n < d \vee 0 < n$$

- ✓ *Add invariant at the end, mark as theorem.*

- It is common to require that physical systems progress.
- We want cars to be able to either enter or exit.
- Therefore, (some) event(s) have to always be enabled.
- Depends on guards: *deadlock freedom*.

$$A_{1\dots l}, I_{1\dots m} \vdash \bigvee_{i=1}^n G_i(v, c)$$

- In our case:

$$d \in \mathbb{N}, n \in \mathbb{N}, n \leq d \vdash n < d \vee 0 < n$$

- ✓ *Add invariant at the end, mark as theorem.*
- *Cannot be proven!*

- It is common to require that physical systems progress.
- We want cars to be able to either enter or exit.
- Therefore, (some) event(s) have to always be enabled.
- Depends on guards: *deadlock freedom*.

$$A_{1\dots l}, I_{1\dots m} \vdash \bigvee_{i=1}^n G_i(v, c)$$

- In our case:

$$d \in \mathbb{N}, n \in \mathbb{N}, n \leq d \vdash n < d \vee 0 < n$$

- ✓ *Add invariant at the end, mark as theorem.*
- *Cannot be proven!*
- Why? Let us find out in which cases events may be in deadlock.
- *Solve  $\neg(n > 0 \vee n < d)$ .*

- It is common to require that physical systems progress.
- We want cars to be able to either enter or exit.
- Therefore, (some) event(s) have to always be enabled.
- Depends on guards: *deadlock freedom*.

$$A_{1\dots l}, I_{1\dots m} \vdash \bigvee_{i=1}^n G_i(v, c)$$

- In our case:

$$d \in \mathbb{N}, n \in \mathbb{N}, n \leq d \vdash n < d \vee 0 < n$$

- ✓ *Add invariant at the end, mark as theorem.*
- **Cannot be proven!**
- Why? Let us find out in which cases events may be in deadlock.
- **Solve**  $\neg(n > 0 \vee n < d)$ .
- If  $d = 0$ , no car can enter! **Missing axiom:**  $0 < d$ . Add it.
- Note that we are **calculating** the model.

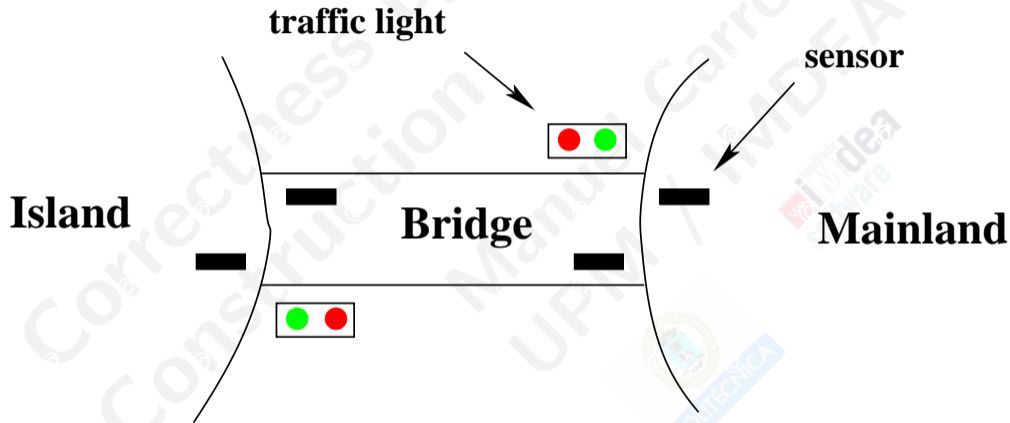
**Initial model** Limiting the number of cars (FUN-2).

**First refinement** Introducing the one-way bridge (FUN-3).

**Second refinement** Introducing the traffic lights (EQP-1,2,3)

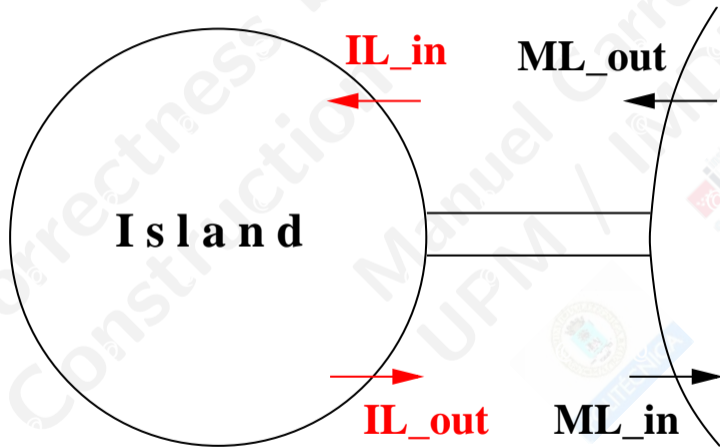
**Third refinement** Introducing the sensors (EQP-4,5)

## Physical system (reminder)



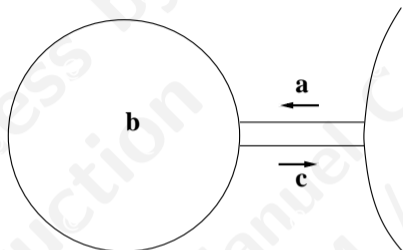
- We introduce the bridge.
- We refine the state and the events.
- We also add two new events: IL\_in and IL\_out.
- We are focusing on FUN-3: one-way bridge.

# One-way bridge





## One-way bridge



- $a$  denotes the number of cars on bridge going to island
- $b$  denotes the number of cars on island
- $c$  denotes the number of cars on bridge going to mainland
- $a$ ,  $b$ , and  $c$  are the concrete variables

## Refining state: invariants

Cars on bridge going to island

Cars on island

Cars on bridge to mainland

Linking new variables to previous model

Formalization of **one-way bridge** (FUN-3)

inv1\_1  $a \in \mathbb{N}$

inv1\_2  $b \in \mathbb{N}$

inv1\_3  $c \in \mathbb{N}$

inv1\_4 ??

inv1\_5 ??

## Refining state: invariants

Cars on bridge going to island

inv1\_1  $a \in \mathbb{N}$

Cars on island

inv1\_2  $b \in \mathbb{N}$

Cars on bridge to mainland

inv1\_3  $c \in \mathbb{N}$

Linking new variables to previous model

inv1\_4  $a + b + c = n$

Formalization of **one-way bridge** (FUN-3)

inv1\_5 ??

inv1\_4 **glues** the **abstract state**  $n$  with the **concrete state**  $a, b, c$

## Refining state: invariants

Cars on bridge going to island

Cars on island

Cars on bridge to mainland

Linking new variables to previous model

Formalization of **one-way bridge** (FUN-3)

inv1\_1  $a \in \mathbb{N}$

inv1\_2  $b \in \mathbb{N}$

inv1\_3  $c \in \mathbb{N}$

inv1\_4  $a + b + c = n$

inv1\_5  $a = 0 \vee c = 0$

Cars on bridge going to island

Cars on island

Cars on bridge to mainland

Linking new variables to previous model

Formalization of **one-way bridge** (FUN-3)

inv1\_1  $a \in \mathbb{N}$

inv1\_2  $b \in \mathbb{N}$

inv1\_3  $c \in \mathbb{N}$

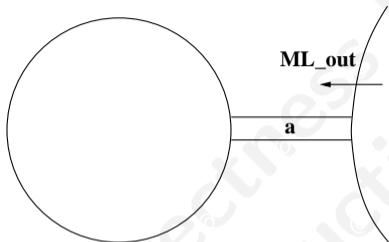
inv1\_4  $a + b + c = n$

inv1\_5  $a = 0 \vee c = 0$

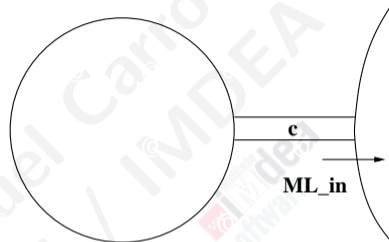
### A new class of invariant

Note that we are not finding an invariant to **prove** the correctness (= postcondition) of a loop. We are establishing an invariant to capture a requirement and we want the model to preserve the invariant, therefore implementing **correctly** that requirement.

# Event refinement proposal

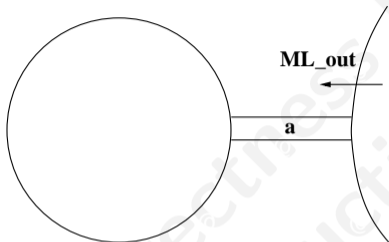


Event ML\_out  
where  
    ????  
  
then  
    ????  
  
end

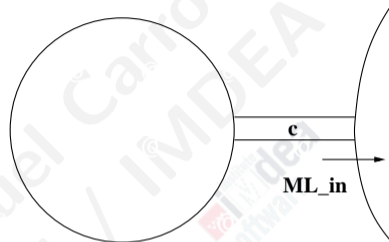


Event ML\_in  
where  
    ????  
  
then  
    ????  
  
end

# Event refinement proposal

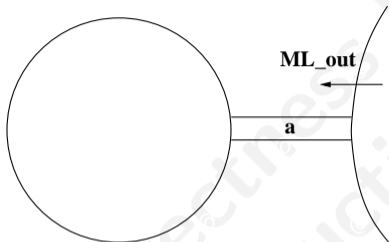


Event  $ML\_out$   
where  
 $a + b < d$   
 $c = 0$   
then  
 $a := a + 1$   
end

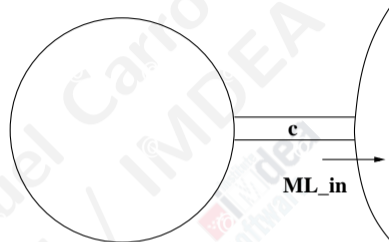


Event  $ML\_in$   
where  
????  
then  
????  
end

# Event refinement proposal



Event  $ML\_out$   
where  
 $a + b < d$   
 $c = 0$   
then  
 $a := a + 1$   
end



Event  $ML\_in$   
where  
 $0 < c$   
then  
 $c := c - 1$   
end



- Right-click on  $m0$ .
- Select **Refine**.
- Name it ( $m1$ ).
- Remove variable  $n$ .
- Introduce variables, invariants.
- Edit existing events by changing them from “extended” to “not extended” (mouse click).

$a \in \mathbb{N}$   
 $b \in \mathbb{N}$   
 $c \in \mathbb{N}$   
 $a + b + c = n$   
 $a = 0 \vee c = 0$

Event  $ML\_out$   
where  
     $a + b < d$   
     $c = 0$   
then  
     $a := a + 1$   
end

Event  $ML\_in$   
where  
     $0 < c$   
then  
     $c := c - 1$   
end

- Every concrete guard is **stronger** than abstract guard.
- Every concrete action is **simulated** by abstract action.

ML\_out / GRD:

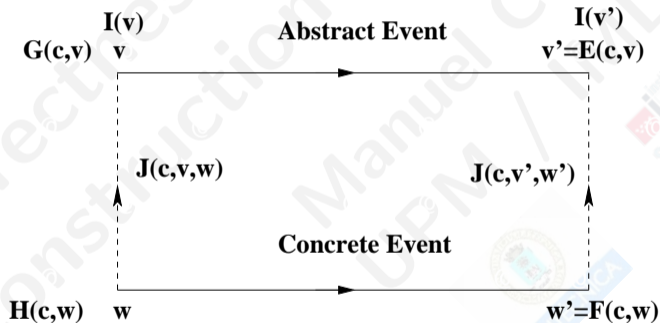
$$d \in \mathbb{N}, 0 < d, n \in \mathbb{N}, n \leq d, a \in \mathbb{N}, b \in \mathbb{N}, c \in \mathbb{N}, a + b + c = n, a = 0 \vee c = 0, a + b < d, c = 0 \quad \vdash n < d$$

ML\_in / GRD:

$$d \in \mathbb{N}, 0 < d, n \in \mathbb{N}, n \leq d, a \in \mathbb{N}, b \in \mathbb{N}, c \in \mathbb{N}, a + b + c = n, a = 0 \vee c = 0, 0 < c \quad \vdash 0 < n$$

- Gluing invariant needed to relate variables in different models!

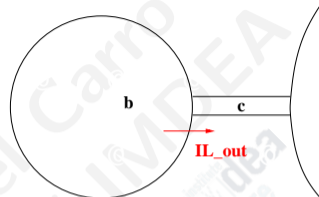
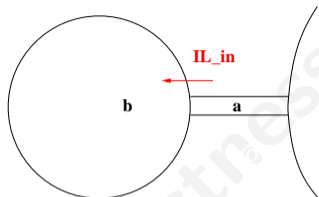
- Gluing invariant  $J(c, v, w)$  has to be **preserved** for every transition.
- Defined in terms of concrete and abstract variables.



**INV:**  $A(c), I(c, v), J(c, v, w), H(c, w) \vdash J_j(c, E(c, v), F(c, w))$

- New events add transitions **without abstract counterpart**.
- Refining **skip**.
- Can be seen as **internal steps** (w.r.t. abstract model).
- Only perceived by observer who is **zooming in**.

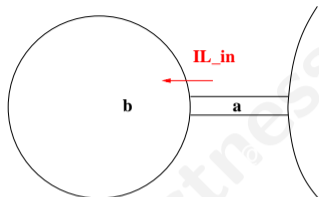
# Proposal for new events



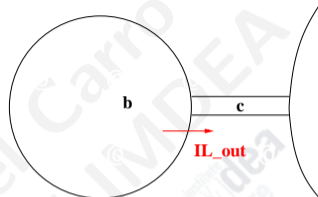
```
Event IL_in
where
  ???
then
  ???
end
```

```
Event IL_out
where
  ???
then
  ???
end
```

# Proposal for new events

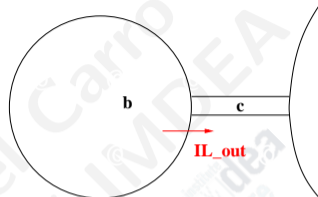
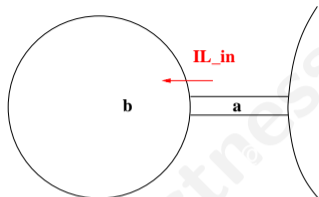


```
Event IL_in
where
  0 < a
then
  a := a - 1
  b := b + 1
end
```



```
Event IL_out
where
  ???
then
  ???
end
```

## Proposal for new events



```
Event  $IL\_in$ 
  where
     $0 < a$ 
  then
     $a := a - 1$ 
     $b := b + 1$ 
  end
```

```
Event  $IL\_out$ 
  where
     $0 < b$ 
     $a = 0$ 
  then
     $c := c + 1$ 
     $b := b - 1$ 
  end
```

✓ Add them!

- **New** events refine **implicit** “void” event
  - *skip* action:  $n' = n$ .
  - No *previous history* to respect.
  - *True* guards.
    - Guard strengthening (GR): trivial (implicit event has *true* guards).
    - Simulation (SIM) trivial: the updates to  $a, b, c$  do not change  $n \Rightarrow$  **no new abstract states introduced**.
  - Need to prove invariants.
- **Termination**: meaningful events are eventually not eligible any more.
  - *Finish* event: artifact to mark when computation is successful.
- **Convergence**: a generalization of termination.
  - Events from a subset of (convergent) events are eligible for a bounded time.
  - Right after this, only events outside this subset are eligible.
  - Then, convergent events can be eligible again.
  - Avoid liveness  $\Rightarrow$  computation progress.



## Convergence of new events

- ML\_in, ML\_out should not alternate forever (see guards / actions).
- IL\_in, IL\_out should not alternate forever (ensure we model real world).
- **New events** must not diverge:
  - IL\_in, IL\_out not continuously enabled.
  - Not physically observable.
  - It should not happen in our model.
  - Ensure that without forcing scheduling restrictions.
- Create variant that proves that IL\_in, IL\_out are not indefinitely enabled.

- Reminder:

<u>IL_in</u>	<u>IL_out</u>
a := a - 1	c := c + 1
b := b + 1	b := b - 1

- ML\_in, ML\_out should not alternate forever (see guards / actions).
- IL\_in, IL\_out should not alternate forever (ensure we model real world).
- **New events** must not diverge:
  - IL\_in, IL\_out not continuously enabled.
  - Not physically observable.
  - It should not happen in our model.
  - Ensure that without forcing scheduling restrictions.
- Create variant that proves that IL\_in, IL\_out are not indefinitely enabled.

- Reminder:

<u>IL_in</u>	<u>IL_out</u>
$a := a - 1$	$c := c + 1$
$b := b + 1$	$b := b - 1$

- We need an  $f$  s.t.:

$$f(a, b, c) > f(a - 1, b + 1, c)$$

$$f(a, b, c) > f(a, b - 1, c + 1)$$

Any proposal?

- ML\_in, ML\_out should not alternate forever (see guards / actions).
- IL\_in, IL\_out should not alternate forever (ensure we model real world).
- **New events** must not diverge:
  - IL\_in, IL\_out not continuously enabled.
  - Not physically observable.
  - It should not happen in our model.
  - Ensure that without forcing scheduling restrictions.
- Create variant that proves that IL\_in, IL\_out are not indefinitely enabled.

- Reminder:

$$\begin{array}{ll} \underline{\text{IL\_in}} & \underline{\text{IL\_out}} \\ a := a - 1 & c := c + 1 \\ b := b + 1 & b := b - 1 \end{array}$$

- We need an  $f$  s.t.:

$$\begin{array}{l} f(a, b, c) > f(a - 1, b + 1, c) \\ f(a, b, c) > f(a, b - 1, c + 1) \end{array}$$

Any proposal?

- Calculate it! ✓ **Add variant!**  
(sketch of calculation in next slide)

Note: ignoring guards here – not necessary.  
Other cases may need them. See PO scheme in *Search* slides.

## Calculating a variant

- In general, convergent variants cannot be automatically determined.
- Making some educated guesses help.
- Let us suppose that  $f$  is a linear function of the variables involved:

$$f(a, b, c) = k_1a + k_2b + k_3c$$

- Therefore:

$$\begin{aligned}k_1a + k_2b + k_3c &> k_1(a - 1) + k_2(b + 1) + k_3c \\k_1a + k_2b + k_3c &> k_1a + k_2(b - 1) + k_3(c + 1)\end{aligned}$$

- Simplifying and solving:

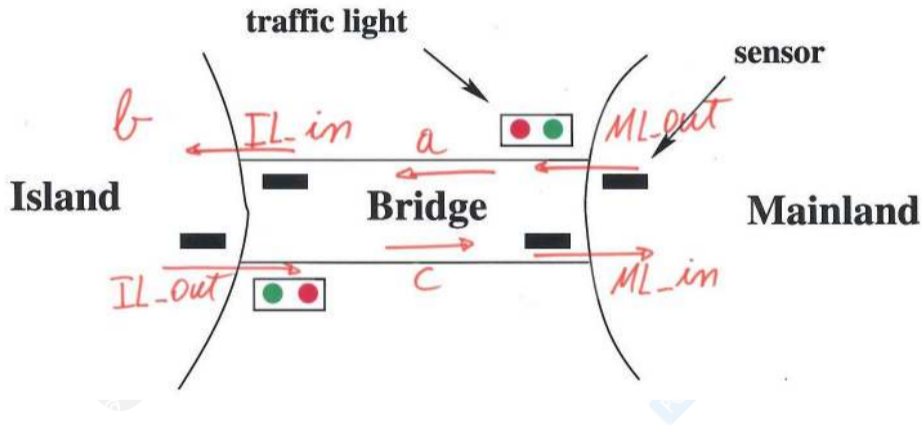
$$k_1 > k_2 > k_3$$

- The simplest selection:

$$k_1 = 2, k_2 = 1, k_3 = 0$$

- **VAR:**  $2a + b$
- Moreover: if  $a \in \mathbb{N}, b \in \mathbb{N}$ , then  $2a + b \in \mathbb{N}$ .

# Bridge after first refinement



- Ensure **no new deadlocks introduced**.
- If concrete model deadlocks, it is because abstract model **already did**.
- $G_i(c, v)$  **abstract** guards,  $H_i(c, v)$  **concrete** guards:

$$A_{1\dots l}(c), I_{1\dots m}(c, v), \bigvee_{i=1}^n G_i(c, v) \vdash \bigvee_{i=1}^p H_i(c, v)$$

- Optional PO (depends on system).

- ✓ *Add invariant:*

$$\bigvee_{i=1}^n G_i(c, v) \Rightarrow \bigvee_{i=1}^p H_i(c, v)$$

- ✓ *Mark as theorem.*

No need to check per event.

- Invariant preservation will generate the right PO.

### Complete sequent

$$d \in \mathbb{N}, 0 < d, n \in \mathbb{N}, n \leq d, a \in \mathbb{N}, b \in \mathbb{N}, c \in \mathbb{N}, a + b + c = n, a = 0 \vee c = 0, 0 < n \vee n < d$$

$$\vdash (a + b < d \wedge c = 0) \vee c > 0 \vee a > 0 \vee (b > 0 \wedge a = 0)$$

## ✔ Proof Obligations

✔<sup>A</sup> thm1/THM

✔ thm2/THM

✔<sup>A</sup> INITIALISATION/inv1/INV

✔<sup>A</sup> INITIALISATION/inv2/INV

✔<sup>A</sup> INITIALISATION/inv3/INV

✔<sup>A</sup> INITIALISATION/inv4/INV

✔<sup>A</sup> INITIALISATION/inv5/INV

✔<sup>A</sup> ML\_out/inv1/INV

✔<sup>A</sup> ML\_out/inv4/INV

✔<sup>A</sup> ML\_out/inv5/INV

✔<sup>A</sup> ML\_out/grd1/GRD

✔<sup>A</sup> IL\_in/inv1/INV

✔<sup>A</sup> IL\_in/inv2/INV

✔<sup>A</sup> IL\_in/inv4/INV

✔<sup>A</sup> IL\_in/inv5/INV

✔<sup>A</sup> IL\_in/VAR

✔<sup>A</sup> IL\_in/NAT

✔<sup>A</sup> IL\_out/inv2/INV

✔<sup>A</sup> IL\_out/inv3/INV

✔<sup>A</sup> IL\_out/inv4/INV

✔<sup>A</sup> IL\_out/inv5/INV

✔<sup>A</sup> IL\_out/VAR

✔<sup>A</sup> IL\_out/NAT

✔<sup>A</sup> ML\_in/inv3/INV

✔<sup>A</sup> ML\_in/inv4/INV

✔<sup>A</sup> ML\_in/inv5/INV

✔<sup>A</sup> ML\_in/grd1/GRD

**Initial model** Limiting the number of cars (FUN-2).

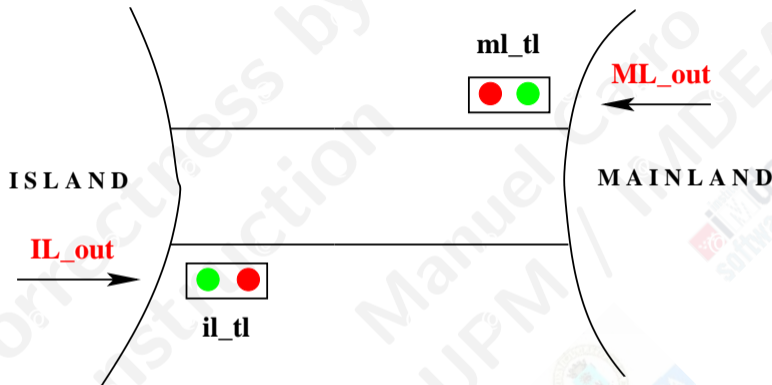
**First refinement** Introducing the one-way bridge (FUN-3).

**Second refinement** Introducing the traffic lights (EQP-1,2,3)

**Third refinement** Introducing the sensors (EQP-4,5)



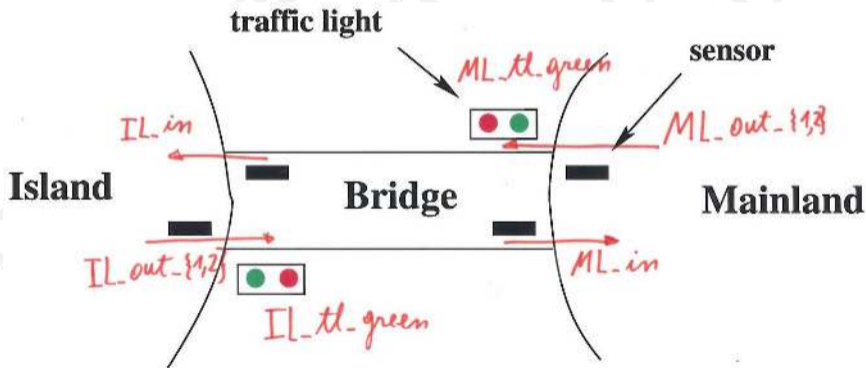
## Introducing traffic lights



Cars cannot magically know the state of the system

## At the end of the refinement...

- For pedagogical reasons: this is where we will end in this refinement.



**set:** *COLOR*

**constants:** *red, green*

**axm2\_1:** *COLOR = {green, red}*

**axm2\_2:** *green  $\neq$  red*

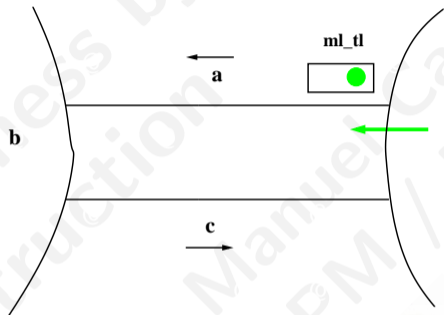
- ✓ *Create context COLORS*
- ✓ *Introduce in context: set, constants, axioms.*
- ✓ *Refine machine m1, create m2*
- ✓ *Make m2 see COLORS*

$$il\_tl \in COLOR$$
$$ml\_tl \in COLOR$$

Remark: Events **IL\_in** and **ML\_in** are **not modified** in this refinement

✓ Add variables, invariants to  $m2$

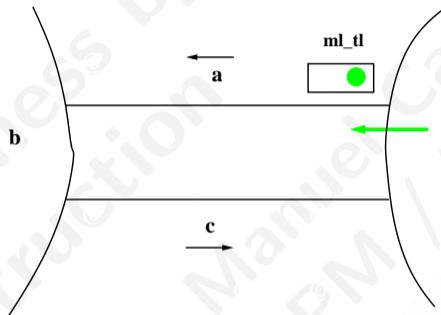
## Introducing traffic lights: leaving mainland



- A green **mainland traffic light** implies **safe access** to the bridge

Invariant?

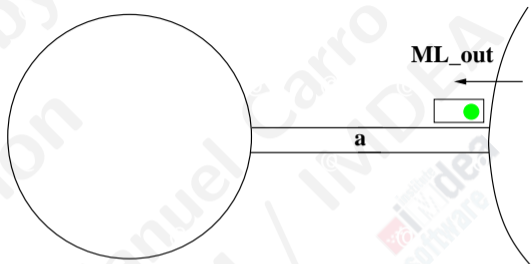
## Introducing traffic lights: leaving mainland



- A green mainland traffic light implies safe access to the bridge

Invariant:  $ml\_tl = green \Rightarrow c = 0 \wedge a + b < d$

- ML\_out was enabled depending on # of cars in system.
- But in reality a car cannot now that.
- It will now depend on state of traffic light.



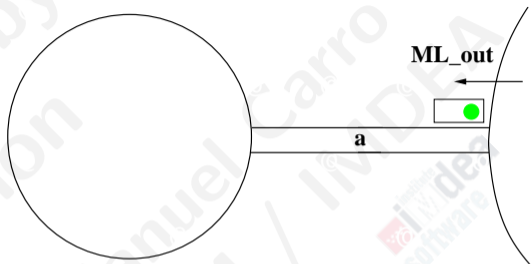
### Abstract

```
Event ML_out
where
  c = 0
  a + b < d
then
  a := a + 1
end
```

### Concrete

```
Event ML_out
where
  ??????
then
  ??????
end
```

- ML\_out was enabled depending on # of cars in system.
- But in reality a car cannot now that.
- It will now depend on state of traffic light.



### Abstract

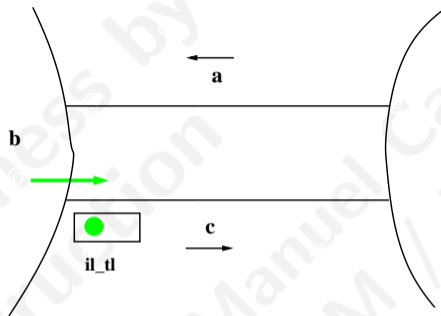
```
Event ML_out
where
  c = 0
  a + b < d
then
  a := a + 1
end
```

### Concrete

```
Event ML_out
where
  mt_tl = green
then
  a := a + 1
end
```



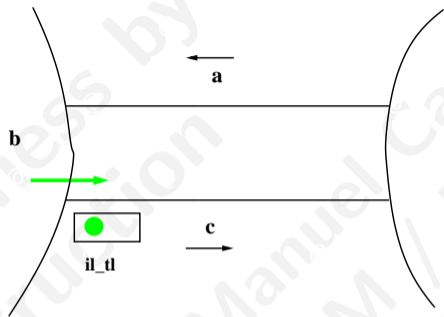
## Introducing traffic lights: leaving island



- A green island traffic light implies safe access to the bridge

Invariant?

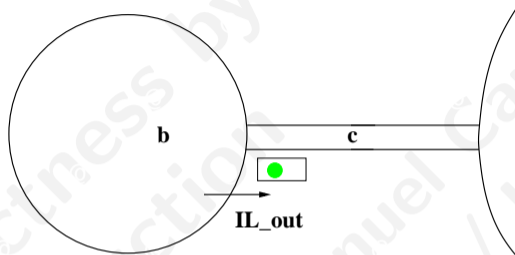
## Introducing traffic lights: leaving island



- A green **island traffic light** implies **safe access** to the bridge

Invariant:  $il\_tl = green \Rightarrow a = 0 \wedge b > 0$

A note on  $b > 0$ :  $il\_tl$  green signals cars in island they may pass. It does not make sense to let them pass if there is no car in the island; it would not align with intention of  $IL\_out$ . The invariant helps check / guarantee that the light does not turn green if the island is empty.

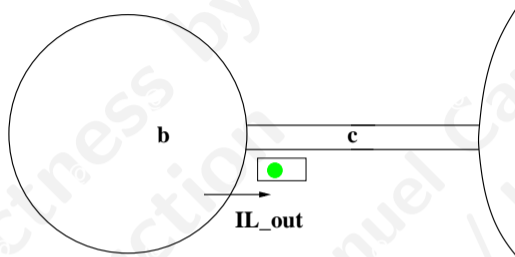


## Abstract

```
Event IL_out
where
  a = 0
  b > 0
then
  b := b - 1
  c := c + 1
end
```

## Concrete

```
Event IL_out
where
  ??????
then
  ??????
end
```



## Abstract

```
Event IL_out
where
   $a = 0$ 
   $b > 0$ 
then
   $b := b - 1$ 
   $c := c + 1$ 
end
```

## Concrete

```
Event IL_out
where
   $il\_tl = green$ 
then
   $b := b - 1$ 
   $c := c + 1$ 
end
```

$il\_tl \in COLOR$   
 $ml\_tl \in COLOR$   
 $il\_tl = green \Rightarrow a = 0 \wedge b > 0$   
 $ml\_tl = green \Rightarrow c = 0 \wedge a + b < d$

- ✓ *Add invariants.*
- ✓ *Change initialization, ML\_out, IL\_out to “non extended”.*
- ✓ *INITIALIZE variables, change guards.*

- **Several INV not proven.**
- We will come back to them.

```
Event ML_out
  where
    ml_tl = green
  then
    a := a + 1
  end
```

```
Event IL_out
  where
    il_tl = green
  then
    b := b - 1
    c := c + 1
  end
```

## Changing traffic lights

- Car entering event visible when traffic light so allows.
  - We will eventually **control** traffic lights.
- When do traffic lights change?
- First approximation: correct simulation.
  - Traffic lights **may** change at any moment it is **not wrong** to do so.
  - We are removing wrong behaviors.
- We can **observe** traffic light changes with associated events.

```
Event ML_tl_green  
  where // Mainland traf. light  
        ?????
```

```
then  
  ml_tl := green  
end
```

```
Event IL_tl_green  
  where // Island traf. light  
        ?????
```

```
then  
  il_tl := green  
end
```

## Changing traffic lights

- Car entering event visible when traffic light so allows.
  - We will eventually **control** traffic lights.
- When do traffic lights change?
- First approximation: correct simulation.
  - Traffic lights **may** change at any moment it is **not wrong** to do so.
  - We are removing wrong behaviors.
- We can **observe** traffic light changes with associated events.

```
Event ML_tl_green
  where // Mainland traf. light
    ml_tl = red
    c = 0
    a + b < d
  then
    ml_tl := green
  end
```

```
Event IL_tl_green
  where // Island traf. light
    ?????
  then
    il_tl := green
  end
```

## Changing traffic lights

- Car entering event visible when traffic light so allows.
  - We will eventually **control** traffic lights.
- When do traffic lights change?
- First approximation: correct simulation.
  - Traffic lights **may** change at any moment it is **not wrong** to do so.
  - We are removing wrong behaviors.
- We can **observe** traffic light changes with associated events.
- ✓ *Add new events.*

```
Event ML_tl_green
  where // Mainland traf. light
    ml_tl = red
    c = 0
    a + b < d
  then
    ml_tl := green
  end
```

```
Event IL_tl_green
  where // Island traf. light
    il_tl = red
    a = 0
    b > 0
  then
    il_tl := green
  end
```



### Variables, invariants

variables:  $a, b, c, ml\_tl, il\_tl$

inv2\_1:  $ml\_tl \in COLOR$

inv2\_2:  $il\_tl \in COLOR$

inv2\_3:  $il\_tl = green \Rightarrow a = 0 \wedge b > 0$

inv2\_4:  $ml\_tl = green \Rightarrow$   
 $c = 0 \wedge a + b < d$

### Pending refinement proofs

- Simulation (SIM).
  - **Nothing to do**: refined events have same actions.
- Guard strengthening (GRD).
  - Guards have changed.
  - **Easy**: invariants directly imply GRD.
- Invariant establishment and preservation (INV).
  - New invariants, new events.

- Some INV POs were not discharged.
- Some look like

H ⊥ ⊥

- Some INV POs were not discharged.
- Some look like  $H \vdash \perp$
- Would be discharged if  $H$  was **inconsistent**.

- Some INV POs were not discharged.
- Some look like

$$H \vdash \perp$$

- Would be discharged if  $H$  was **inconsistent**.
- Further examination:
  - Some  $H$  contains  $ml\_tl = green$  and  $il\_tl = green$ .
  - I.e., both traffic lights are green.
  - That should not be allowed.
  - Or require inferring  $ml\_tl = green$  when  $il\_tl = green$  (equivalent).

- Some INV POs were not discharged.
- Some look like
$$H \vdash \perp$$
- Would be discharged if  $H$  was **inconsistent**.
- Further examination:
  - Some  $H$  contains  $ml\_tl = green$  and  $il\_tl = green$ .
  - I.e., both traffic lights are green.
  - That should not be allowed.
  - Or require inferring  $ml\_tl = green$  when  $il\_tl = green$  (equivalent).

- We are missing an invariant

- This allows some proofs to be completed.

✓ Add it

- Some INV POs were not discharged.
- Some look like

$$H \vdash \perp$$

- Would be discharged if  $H$  was **inconsistent**.
- Further examination:
  - Some  $H$  contains  $ml\_tl = green$  and  $il\_tl = green$ .
  - I.e., both traffic lights are green.
  - That should not be allowed.
  - Or require inferring  $ml\_tl = green$  when  $il\_tl = green$  (equivalent).

- We are missing an invariant

$$inv2\_5 : ml\_tl = red \vee il\_tl = red$$

(FUN-3 and EQP-3)

- This allows some proofs to be completed.

✓ Add it

### Done

ML\_out / inv2\_4 / INV

IL\_out / inv2\_3 / INV

### Pending

ML\_out / inv2\_3 / INV

IL\_out / inv2\_4 / INV

ML\_tl\_green / inv2\_5 / INV

IL\_tl\_green / inv2\_5 / INV

```
Event ML_out
  where
    ml_tl = green
  then
    a := a + 1
  end
```

- Preservation of  $a + b < d, ml\_tl = green \vdash a + 1 + b < d$  fails.



```
Event ML_out
  where
    ml_tl = green
  then
    a := a + 1
  end
```

- Preservation of  $a + b < d, ml\_tl = green \vdash a + 1 + b < d$  fails.
- The  $n^{\text{th}}$  car to enter the island should **force** traffic light to become red.

✓ *Split event corresponding to car entering bridge into two different cases: last car and non-last car.*

```
Event ML_out_1
  where
    ml_tl = green
    a + 1 + b < d
  then
    a := a + 1
  end
```

```
Event ML_out_2
  where
    ml_tl = green
    a + 1 + b = d
  then
    a := a + 1
    ml_tl := red
  end
```

```
Event IL_out
  where
    il_tl = green
  then
    b := b - 1
    c := c + 1
  end
```

- IL\_out / inv2\_4 / INV fails.
- $0 < b \vdash 0 < b - 1$ .
- The last car to leave the island should turn the island traffic light red.
- Again, two different cases.

✓ *Add to the model.*

```
Event IL_out_1
  where
    il_tl = green
    b ≠ 1
  then
    b, c := b - 1, c + 1
  end
```

```
Event IL_out_2
  where
    il_tl = green
    b = 1
  then
    b, c := b - 1, c + 1
    il_tl := red
  end
```

### Done

ML\_out / inv2\_4 / INV

IL\_out / inv2\_3 / INV

ML\_out\_{1,2} / inv2\_3 / INV

IL\_out\_{1,2} / inv2\_4 / INV

### Pending

ML\_tl\_green / inv2\_5 / INV

IL\_tl\_green / inv2\_5 / INV

**inv2\_5:**  $ml\_tl = red \vee il\_tl = red$

- Not preserved by ML\_tl\_green, IL\_tl\_green.
- There is an state where ML\_tl\_green and IL\_tl\_green can fire sequentially.

Event ML\_tl\_green

where

$ml\_tl = red$

$a + b < d$

$c = 0$

then

$ml\_tl := green$

??????

end

Event IL\_tl\_green

where

$il\_tl = red$

$0 < b$

$a = 0$

then

$il\_tl := green$

??????

end

**inv2\_5:**  $ml\_tl = red \vee il\_tl = red$

- Not preserved by ML\_tl\_green, IL\_tl\_green.
- There is an state where ML\_tl\_green and IL\_tl\_green can fire sequentially.

Event ML\_tl\_green

where

$ml\_tl = red$

$a + b < d$

$c = 0$

then

$ml\_tl := green$

$il\_tl := red$

end

Event IL\_tl\_green

where

$il\_tl = red$

$0 < b$

$a = 0$

then

$il\_tl := green$

??????

end

**inv2\_5:**  $ml\_tl = red \vee il\_tl = red$

- Not preserved by ML\_tl\_green, IL\_tl\_green.
- There is an state where ML\_tl\_green and IL\_tl\_green can fire sequentially.

Event ML\_tl\_green

where

$ml\_tl = red$

$a + b < d$

$c = 0$

then

$ml\_tl := green$

$il\_tl := red$

end

Event IL\_tl\_green

where

$il\_tl = red$

$0 < b$

$a = 0$

then

$il\_tl := green$

$ml\_tl := red$

end

**inv2\_5:**  $ml\_tl = red \vee il\_tl = red$

- Not preserved by ML\_tl\_green, IL\_tl\_green.
- There is an state where ML\_tl\_green and IL\_tl\_green can fire sequentially.

Event ML\_tl\_green

where

$ml\_tl = red$

$a + b < d$

$c = 0$

then

$ml\_tl := green$

$il\_tl := red$

end

Event IL\_tl\_green

where

$il\_tl = red$

$0 < b$

$a = 0$

then

$il\_tl := green$

$ml\_tl := red$

end

At this point, all invariants for requirements in this refinement are preserved (safety). We can think about liveness.

- Event firing may happen without leading to system progress.
- Other (necessary) events may not take place.
  - Called “livelock” in concurrent programming.
- Events that do not clearly change a bounded expression or variable<sup>a</sup> are suspicious.
- **New** events in particular — remember we already proved convergence of IL\_in and IL\_out

---

<sup>a</sup>“Clearly” does not ensure; properties should anyway be proven.



Event ML\_tl\_green

where

$ml\_tl = red$

$a + b < d$

$c = 0$

then

$ml\_tl := green$

$il\_tl := red$

end

- Guards depend on  $a$ ,  $b$ ,  $c$  and traffic lights.
- $ml\_tl = red$  and  $il\_tl = red$  (in guards) alternatively set by the other event.

Event IL\_tl\_green

where

$il\_tl = red$

$0 < b$

$a = 0$

then

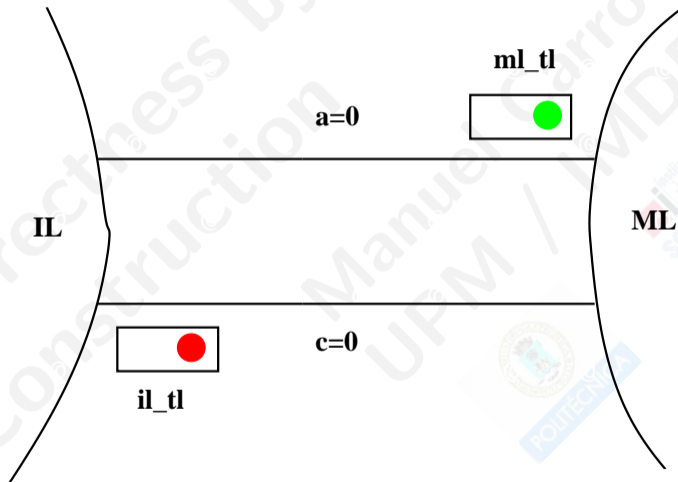
$il\_tl := green$

$ml\_tl := red$

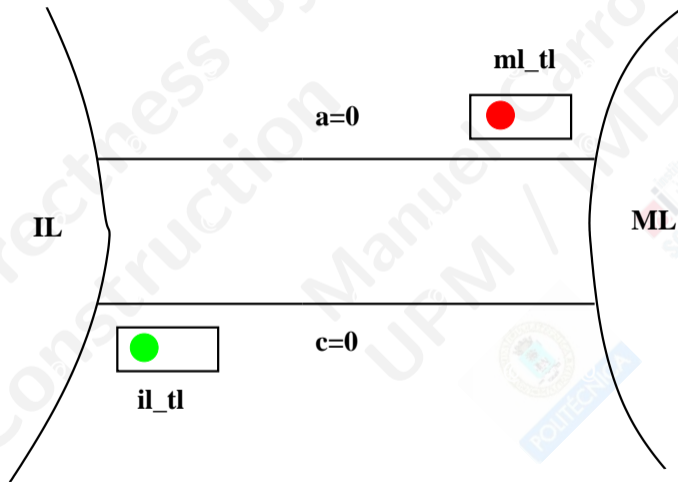
end

- The rest of the guards are simultaneously true when  $a = c = 0, 0 < b < d$ .
- Traffic lights could alternatively change colors w.o. control.

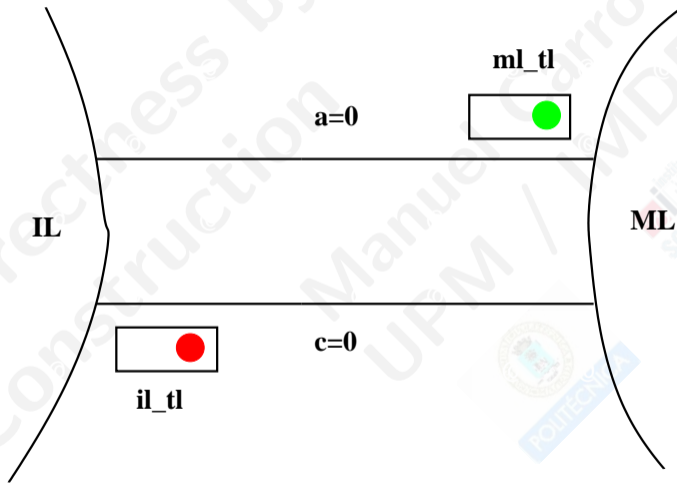
# Alternating traffic lights



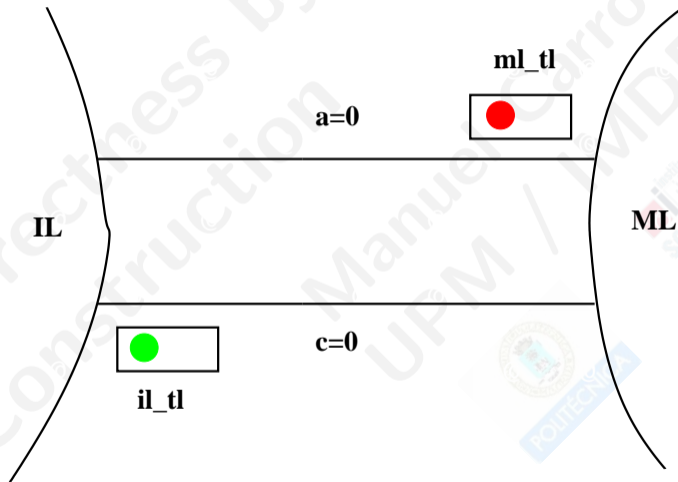
# Alternating traffic lights



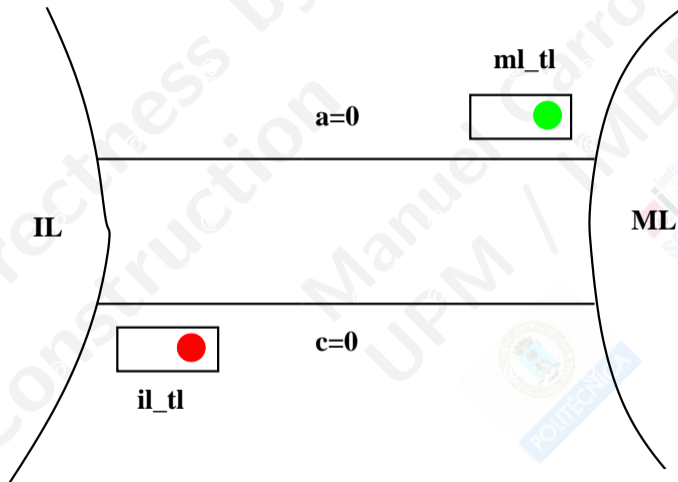
# Alternating traffic lights



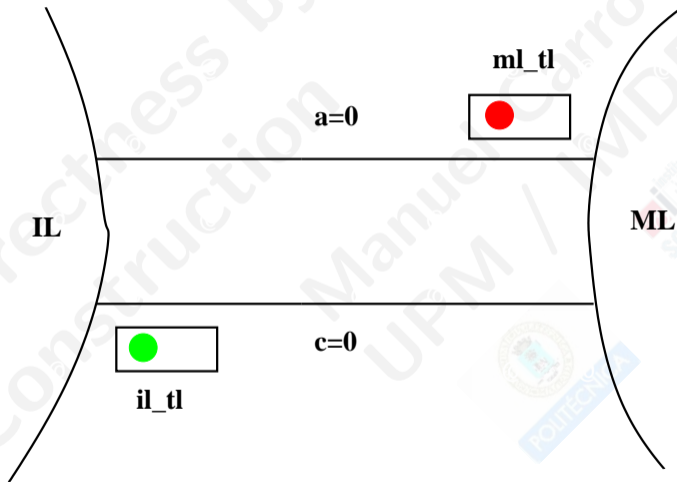
# Alternating traffic lights



# Alternating traffic lights



# Alternating traffic lights



## Prove convergence: variant

- We have seen that there is divergence.
- Adding a variant does not help: it does not change behavior (just checks it!).
- We need to add a way to control when events are enabled.



- We have seen that there is divergence.
- Adding a variant does not help: it does not change behavior (just checks it!).
- We need to add a way to control when events are enabled.
- Allow lights to turn green only when a car has passed in the other direction since it turned red.
- Two additional variables:
  - inv2\_6:**  $ml\_pass \in \{0, 1\}$
  - inv2\_7:**  $ll\_pass \in \{0, 1\}$
- We update them when cars go out of mainland and out of the island.

### Concerns:

- Is it safe?

- We have seen that there is divergence.
- Adding a variant does not help: it does not change behavior (just checks it!).
- We need to add a way to control when events are enabled.
- Allow lights to turn green only when a car has passed in the other direction since it turned red.
- Two additional variables:
  - inv2\_6:**  $ml\_pass \in \{0, 1\}$
  - inv2\_7:**  $ll\_pass \in \{0, 1\}$
- We update them when cars go out of mainland and out of the island.

### Concerns:

- Is it safe?
  - Yes. We are not letting traffic lights be green when inadequate. Other invariants will be not provable otherwise.

- We have seen that there is divergence.
- Adding a variant does not help: it does not change behavior (just checks it!).
- We need to add a way to control when events are enabled.
- Allow lights to turn green only when a car has passed in the other direction since it turned red.
- Two additional variables:
  - inv2\_6:**  $ml\_pass \in \{0, 1\}$
  - inv2\_7:**  $ll\_pass \in \{0, 1\}$
- We update them when cars go out of mainland and out of the island.

### Concerns:

- Is it safe?
  - Yes. We are not letting traffic lights be green when inadequate. Other invariants will be not provable otherwise.
- Won't traffic stop circulating?

- We have seen that there is divergence.
- Adding a variant does not help: it does not change behavior (just checks it!).
- We need to add a way to control when events are enabled.
- Allow lights to turn green only when a car has passed in the other direction since it turned red.
- Two additional variables:
  - inv2\_6:**  $ml\_pass \in \{0, 1\}$
  - inv2\_7:**  $ll\_pass \in \{0, 1\}$
- We update them when cars go out of mainland and out of the island.

### Concerns:

- Is it safe?
  - Yes. We are not letting traffic lights be green when inadequate. Other invariants will be not provable otherwise.
- Won't traffic stop circulating?
  - Assuming that the system progresses (there are cars going in and out), it should not. At the moment, since we do not have sensors yet, we cannot do anything better.

# Modifications to avoid divergence

```
Event ML_out_1
  where
    ml_tl = green
    a + 1 + b < d
  then
    a := a + 1
    ml_pass := 1
  end
```

```
Event ML_out_2
  where
    ml_tl = green
    a + 1 + b = d
  then
    a := a + 1
    ml_tl := red
    ml_pass := 1
  end
```

```
Event ML_tl_green
  where
    ml_tl = red
    a + b < d
    c = 0
    il_pass = 1
  then
    ml_tl := green
    il_tl := red
    ml_pass := 0
  end
```

```
Event IL_out_1
  where
    il_tl = green
    b ≠ 1
  then
    b := b - 1
    c := c + 1
    il_pass := 1
  end
```

```
Event IL_out_2
  where
    il_tl = green
    b = 1
  then
    b := b - 1
    c := c + 1
    il_tl := red
    il_pass := 1
  end
```

```
Event IL_tl_green
  where
    il_tl = red
    0 < b
    a = 0
    ml_pass = 1
  then
    il_tl := green
    ml_tl := red
    il_pass := 0
  end
```

- Proving non-divergence (✓ *Add VARIANT to model*):

variant\_2 :  $ml\_pass + il\_pass$

- Convergence proofs (for ML\_tl\_green and IL\_tl\_green):

$ml\_tl = red, il\_pass = 1, \dots \vdash il\_pass + 0 < ml\_pass + il\_pass$

$il\_tl = red, ml\_pass = 1, \dots \vdash ml\_pass + 0 < ml\_pass + il\_pass$

- Proving non-divergence (✓ *Add VARIANT to model*):

variant\_2 :  $ml\_pass + il\_pass$

- Convergence proofs (for ML\_tl\_green and IL\_tl\_green):

$ml\_tl = red, il\_pass = 1, \dots \vdash il\_pass + 0 < ml\_pass + il\_pass$

$il\_tl = red, ml\_pass = 1, \dots \vdash ml\_pass + 0 < ml\_pass + il\_pass$

- Cannot be proven as they are.

- Proving non-divergence (✓ *Add VARIANT to model*):

$\text{variant\_2} : ml\_pass + il\_pass$

- Convergence proofs (for ML\_tl\_green and IL\_tl\_green):

$ml\_tl = red, il\_pass = 1, \dots \vdash il\_pass + 0 < ml\_pass + il\_pass$

$il\_tl = red, ml\_pass = 1, \dots \vdash ml\_pass + 0 < ml\_pass + il\_pass$

- Cannot be proven as they are.
- Suggestion: posit the invariants (✓ *Add them*)

**inv2\_8:**  $ml\_tl = red \Rightarrow ml\_pass = 1$

**inv2\_9:**  $il\_tl = red \Rightarrow il\_pass = 1$

- Note: we are **not** forcing  $ml\_pass = 1$  when  $ml\_tl = red$ .
- But if it is true ( $\Rightarrow$  invariant preservation), then we can prove non-divergence.



All axioms, invariants,  
theorems

⊢

$$(ml\_tl = green \wedge a + b + 1 < d) \quad \checkmark$$

$$(ml\_tl = green \wedge a + b + 1 = d) \quad \checkmark$$

$$(il\_tl = green \wedge b > 1) \vee (il\_tl = green \wedge b = 1) \quad \checkmark$$

$$(ml\_tl = red \wedge a + b < d \wedge c = 0 \wedge il\_pass = 1) \quad \checkmark$$

$$(il\_tl = red \wedge 0 < b \wedge a = 0 \wedge ml\_pass = 1) \quad \checkmark$$

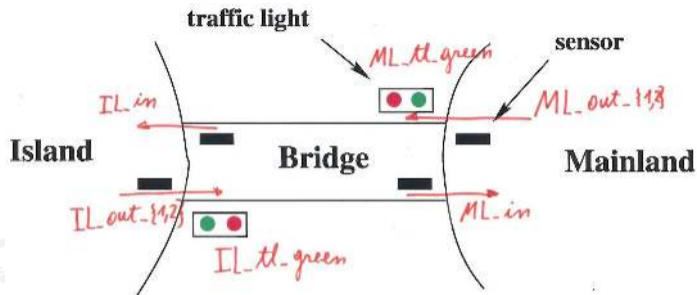
$$0 < a \vee 0 < c$$

- Lengthy, but mechanical.
- Copy and paste from guards, add invariant, mark as theorem.
- Left as exercise! (but use the guards in your model, in case they differ from the ones above)

## Conclusion of second refinement

- We discovered four errors.
- We introduced several additional invariants.
- We corrected four events.
- We introduced two more variables to model the system.
- An two additional variables to control divergence.

## Analysis of second refinement



$ML\_in$  Car leaves bridge to mainland.

$\{M,I\}L\_out_{\{1,2\}}$  Cars enter bridge.

$IL\_in$  Car bridge leaves to island.

- Depending on traffic light.
- Traffic light, turn changes depending on # of cars.

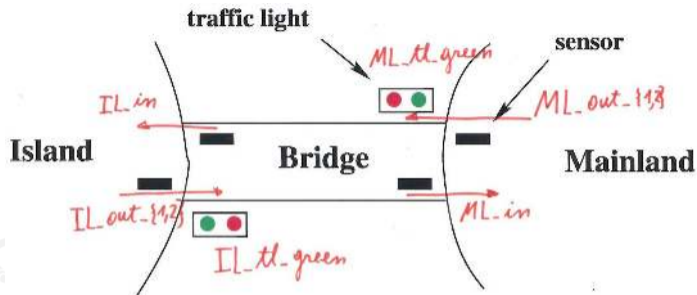
$ML\_tl\_green$  Controls ML traffic light.

- Dep. on # of cars, turn.

- How do we know # of cars?

$IL\_tl\_green$  Same for island traffic light.

## Analysis of second refinement



$ML\_in$  Car leaves bridge to mainland.

$\{M,I\}L\_out_{\{1,2\}}$  Cars enter bridge.

$IL\_in$  Car bridge leaves to island.

- Depending on traffic light.
- Traffic light, turn changes depending on # of cars.

$ML\_tl\_green$  Controls ML traffic light.

- Dep. on # of cars, turn.

• How do we know # of cars?

$IL\_tl\_green$  Same for island traffic light.

• Sensors!

## Invariant / variant summary

$ml\_tl \in \{red, green\}$

$il\_tl \in \{red, green\}$

$ml\_tl = green \Rightarrow a + b < d \wedge c = 0$

$il\_tl = green \Rightarrow 0 < b \wedge a = 0$

$ml\_tl = red \vee il\_tl = red$

$ml\_pass \in \{0, 1\}$

$il\_pass \in \{0, 1\}$

$ml\_tl = red \Rightarrow ml\_pass = 1$

$il\_tl = red \Rightarrow il\_pass = 1$

variant:  $ml\_pass + il\_pass$

Possible colors .

Possible colors.

If TL to enter island is green, there is space in the island and no car is leaving.

If TL to exit island is green, at least on car is in the island and no car is coming in through the bridge.

Both traffic lights cannot be green at the same time.  
A car entered bridge from ML since ML TL turned green.

A car entered bridge from IL since IL TL turned green.

Captures *technical* invariant

Captures *technical* invariant

To ensure that traffic lights do not alternate forever.

## Summary of events (1)

Event ML\_out\_1

where

ml\_tl = green

$a + 1 + b < d$

then

$a := a + 1$

ml\_pass := 1

end

Event ML\_out\_2

where

ml\_tl = green

$a + 1 + b = d$

then

$a := a + 1$

ml\_pass := 1

ml\_tl := red

end

## Summary of events (2)

Event IL\_out\_1

where

il\_tl = green

b  $\neq$  1

then

b := b - 1

c := c + 1

il\_pass := 1

end

Event IL\_out\_2

where

il\_tl = green

b = 1

then

b := b - 1

c := c + 1

il\_pass := 1

il\_tl := red

end

## Summary of events (3)

Event ML\_tl\_green

where

ml\_tl = red

$a + b < d$

$c = 0$

il\_pass = 1

then

ml\_tl := green

il\_tl := red

ml\_pass := 0

end

Event IL\_tl\_green

where

il\_tl = red

$0 < b$

$a = 0$

ml\_pass = 1

then

il\_tl := green

ml\_tl := red

il\_pass := 0

end



These are identical to their abstract versions

```
Event ML_in
  where
     $0 < c$ 
  then
     $c := c - 1$ 
  end
```

```
Event IL_in
  where
     $0 < a$ 
  then
     $a := a - 1$ 
     $b := b + 1$ 
  end
```

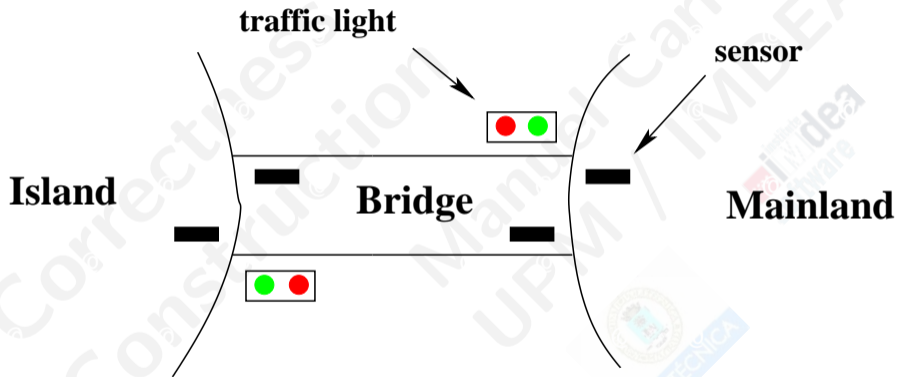
**Initial model** Limiting the number of cars (FUN-2).

**First refinement** Introducing the one-way bridge (FUN-3).

**Second refinement** Introducing the traffic lights (EQP-1,2,3).

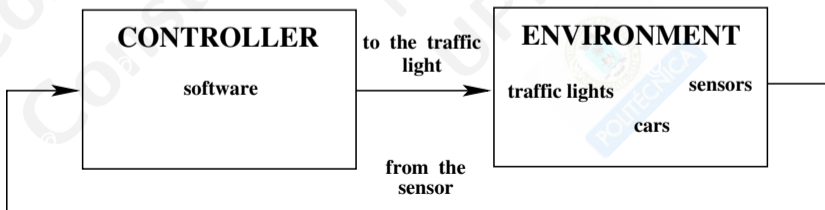
**Third refinement** Introducing the sensors (EQP-4,5).

## Reminder of system



We need to identify:

- The **controller**.
  - The **environment**.
  - The **communication channels**.
- Environment: deals with **physical** cars.
  - Controller: deals with **logical** cars.
  - Communication channels: keep relationship among them.
    - Physical reality / logical view not always in sync!



Controller variables  
(used to decide traffic light colors)

*a,*  
*b,*  
*c,*  
*ml\_pass,*  
*il\_pass*

Environment variables  
(denote **physical** objects):

*A,*  
*B,*  
*C,*  
*ML\_OUT\_SR,*  
*ML\_IN\_SR,*  
*IL\_OUT\_SR,*  
*IL\_IN\_SR*

- *A, B, C*: physical cars.
- *\*\_\*\_SR*: state of physical sensors.

Output channels  
(send state / signal to traffic lights)

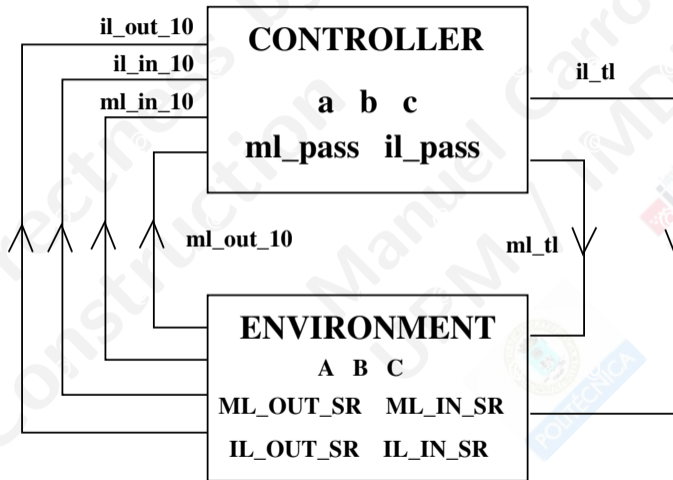
*ml\_tl,*  
*il\_tl*

Input channels  
(receive signals from sensors):

*ml\_out\_10,*  
*ml\_in\_10,*  
*il\_out\_10,*  
*il\_in\_10*

Sensors: a message is sent when it changes from *on* to *off*.





- The possible states of a sensor:

**Carrier sets:**  $\dots, \text{SENSOR}$ .

**Constants:**  $on, off$ .

**axm3\_1:**  $\text{SENSOR} = \{on, off\}$

**axm3\_2:**  $on \neq off$

- Type invariants:

**inv3\_1:**  $ML\_OUT\_SR \in \text{SENSOR}$

**inv3\_2:**  $ML\_IN\_SR \in \text{SENSOR}$

**inv3\_3:**  $IL\_OUT\_SR \in \text{SENSOR}$

**inv3\_4:**  $IL\_IN\_SR \in \text{SENSOR}$

**inv3\_5:**  $A \in \mathbb{N}$

**inv3\_6:**  $B \in \mathbb{N}$

**inv3\_7:**  $C \in \mathbb{N}$

**inv3\_8:**  $ml\_out\_10 \in \text{BOOL}$

**inv3\_9:**  $ml\_in\_10 \in \text{BOOL}$

**inv3\_10:**  $il\_out\_10 \in \text{BOOL}$

**inv3\_11:**  $il\_in\_10 \in \text{BOOL}$

BOOL is a built-in set:  $\text{BOOL} = \{\text{TRUE}, \text{FALSE}\}$ .

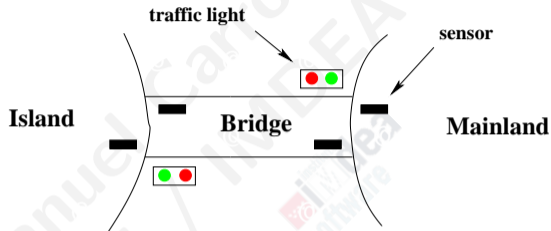


When sensors are on, there are cars on them:

**inv3\_12:**  $IL\_IN\_SR = on \Rightarrow A > 0$

**inv3\_13:**  $IL\_OUT\_SR = on \Rightarrow B > 0$

**inv3\_14:**  $ML\_IN\_SR = on \Rightarrow C > 0$



The sensors are used to detect the presence of cars entering or leaving the bridge

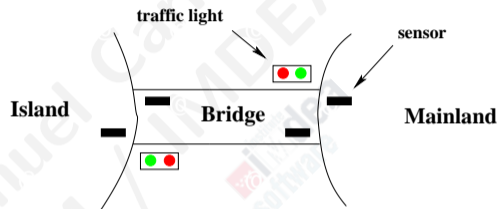
EQP-5

(We do not count / control cars in mainland)

Drivers obey traffic lights (e.g., they cross with green traffic light):

**inv3\_15:**  $ml\_out\_10 = \text{TRUE} \Rightarrow ml\_tl = \text{green}$

**inv3\_16:**  $il\_out\_10 = \text{TRUE} \Rightarrow il\_tl = \text{green}$



Cars are supposed to pass only on a green traffic light

EQP-3

When sensor *on*, its logical representation should have been updated.  
Note: this does **not** update variables – it only checks they were.

**inv3\_17:**  $IL\_IN\_SR = on \Rightarrow il\_in\_10 = FALSE$

**inv3\_18:**  $IL\_OUT\_SR = on \Rightarrow il\_out\_10 = FALSE$

**inv3\_19:**  $ML\_IN\_SR = on \Rightarrow ml\_in\_10 = FALSE$

**inv3\_20:**  $ML\_OUT\_SR = on \Rightarrow ml\_out\_10 = FALSE$



The controller must be fast enough so as to be able to treat all the information coming from the environment

FUN-5

**inv3\_21:**  $il\_in\_10 = \text{TRUE} \wedge ml\_out\_10 = \text{TRUE} \Rightarrow A = a$

**inv3\_22:**  $il\_in\_10 = \text{FALSE} \wedge ml\_out\_10 = \text{TRUE} \Rightarrow A = a + 1$

**inv3\_23:**  $il\_in\_10 = \text{TRUE} \wedge ml\_out\_10 = \text{FALSE} \Rightarrow A = a - 1$

**inv3\_24:**  $il\_in\_10 = \text{FALSE} \wedge ml\_out\_10 = \text{FALSE} \Rightarrow A = a$

**inv3\_25:**  $il\_in\_10 = \text{TRUE} \wedge il\_out\_10 = \text{TRUE} \Rightarrow B = b$

**inv3\_26:**  $il\_in\_10 = \text{TRUE} \wedge il\_out\_10 = \text{FALSE} \Rightarrow B = b + 1$

**inv3\_27:**  $il\_in\_10 = \text{FALSE} \wedge il\_out\_10 = \text{TRUE} \Rightarrow B = b - 1$

**inv3\_28:**  $il\_in\_10 = \text{FALSE} \wedge il\_out\_10 = \text{FALSE} \Rightarrow B = b$

**inv3\_29:**  $il\_out\_10 = \text{TRUE} \wedge ml\_out\_10 = \text{TRUE} \Rightarrow C = c$

**inv3\_30:**  $il\_out\_10 = \text{TRUE} \wedge ml\_out\_10 = \text{FALSE} \Rightarrow C = c + 1$

**inv3\_31:**  $il\_out\_10 = \text{FALSE} \wedge ml\_out\_10 = \text{TRUE} \Rightarrow C = c - 1$

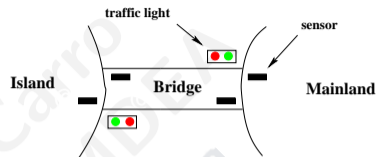
**inv3\_32:**  $il\_out\_10 = \text{FALSE} \wedge ml\_out\_10 = \text{FALSE} \Rightarrow C = c$

**inv3\_21:**  $il\_in\_10 = \text{TRUE} \wedge ml\_out\_10 = \text{TRUE} \Rightarrow A = a$

**inv3\_22:**  $il\_in\_10 = \text{FALSE} \wedge ml\_out\_10 = \text{TRUE} \Rightarrow A = a + 1$

**inv3\_23:**  $il\_in\_10 = \text{TRUE} \wedge ml\_out\_10 = \text{FALSE} \Rightarrow A = a - 1$

**inv3\_24:**  $il\_in\_10 = \text{FALSE} \wedge ml\_out\_10 = \text{FALSE} \Rightarrow A = a$



- $A$ : physical # cars. Updated by events representing cars entering.
- $a$ : *controller* (logical) view.
- When  $ml\_out\_10 = \text{TRUE}$ : other

events will update logical # of cars, set  $ml\_out\_10 = \text{FALSE}$ .

- In the meantime, logical and physical # cars may be out of sync.

One event represents car entering bridge. Increases  $A$ . Simulates sensor  $ML\_OUT$  going from *off* to *on*. Another even registers change. Sets **logical**  $ml\_out\_10$  to TRUE. **Here,  $A = a + 1$**  Then another event sees  $ml\_out\_10 = \text{FALSE}$  and updates  $a$ . **Here  $A = a$ .**

When  $ml\_out\_10 = \text{TRUE} \wedge il\_out\_10 = \text{TRUE}$ , they balance each other.

## New (physical) events (examples)

```
Event ML_out_arr
  where // No car on sensor
    ML_OUT_SR = off
    ml_out_10 = FALSE
  then
    ML_OUT_SR := on
  end
```

```
Event ML_out_dep
  where
    ML_OUT_SR = on
    ml_tl = green
  then
    ML_OUT_SR := off
    ml_out_10 := TRUE
    A := A + 1
  end
```

```
Event IL_in_arr
  where
    IL_IN_SR = off
    il_in_10 = FALSE
    A > 0
  then
    IL_IN_SR := on
  end
```

```
Event IL_in_dep
  where
    IL_IN_SR = on
  then
    IL_IN_SR := off
    il_in_10 := TRUE
    A := A - 1
    B := B + 1
  end
```

## Refining abstract events (example)

Event ML\_out\_1 (abstract)

where

ml\_tl = green

$a + b + 1 \neq d$

then

$a := a + 1$

ml\_pass := 1

end

Event ML\_out\_1

where

ml\_out\_10 = TRUE

$a + b + 1 \neq d$

then

$a := a + 1$

ml\_pass := 1

ml\_out\_10 := FALSE

end

# A taste of the dynamic behavior

```
Event ML_out_arr
  where // No car on sensor
    ML_OUT_SR = off
    ml_out_10 = FALSE
  then
    ML_OUT_SR := on
  end
```

```
Event ML_out_dep
  where
    ML_OUT_SR = on
    ml_tl = green
  then
    ML_OUT_SR := off
    ml_out_10 := TRUE
    A := A + 1
  end
```

```
Event ML_out_1
  where
    ml_out_10 = TRUE
    a + b + 1 ≠ d
  then
    a := a + 1
    ml_pass := 1
    ml_out_10 := FALSE
  end
```

- A car can be on the sensor **when there is no other car on it.**
- The signal to the controller has been received and cleared.
  - **Cannot be done in reality. Models hardware + software is fast.**
- We record that a car is on the sensor.
- Car on sensor can leave when traffic light green.
- Record it's not on sensor any more.
- **Set signal to controller.**
- Increase # physical cars ( **$A \neq a$  now**).
- Signal from sensor read.
- # cars in bridge+island checked.
- Increase # cars recorded by controller ( **$A = a$  now**).
- **Reset signal from sensor.**
- Update controller variables as seen.



**inv3\_33:**  $A = 0 \vee C = 0$

**inv3\_34:**  $A + B + C \leq d$

The number of cars on the bridge and the island is limited

FUN-2



The bridge is one-way

FUN-3

- Ensure **new** events converge.

Correctness by  
Construction

Manuel Carro  
UPM / IMDEA



- Ensure **new** events converge.
- The (somewhat surprising) variant expression is

$$12 - (ML\_OUT\_SR + ML\_IN\_SR + IL\_OUT\_SR + IL\_IN\_SR + 2 \times (ml\_out\_10 + ml\_in\_10 + il\_out\_10 + il\_in\_10))$$

- Note: formally incorrect. Booleans have to be converted to integers in the usual way.

# Final structure

