One-Way Bridge[1]

Manuel Carro
manuel.carro@upm.es

Universidad Politécnica de Madrid &
IMDEA Software Institute

[1]Example and many slides borrowed from J. R. Abrial

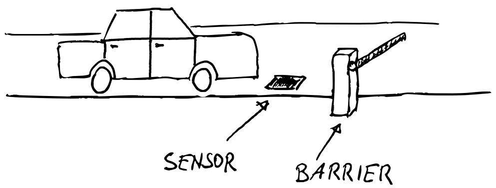
Goals of this chapter

- 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.

Difference with previous examples

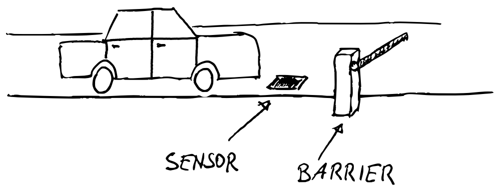
- Previous examples were *transformational*.
  - Input ⇒ transformation ⇒ output.
- Current example:
  - Interaction with environment.
- Sensors and communication channels:
  - Hardware sensors modeled with events.
  - Channels modeled with variables.

Correctness within an environment



- 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.

Correctness within an environment



- 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.

Requirements document

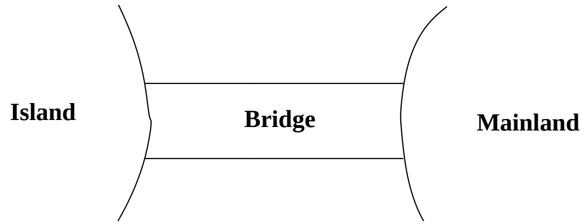
- Large reactive systems difficult to specify from the outset.
- Building it piece-wise, ensuring it meets (natural-language) requirements: a way towards ensuring we have a detailed system specification that is provable correct.

- Two kinds of requirements:
  - Concerned with the equipment (EQP).
  - Concerned with function of system (FUN).
- Objective: control cars on a narrow bridge.
- Bridge links the mainland to (small) island.

Requirements

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

- This can be illustrated as follows



Requirements

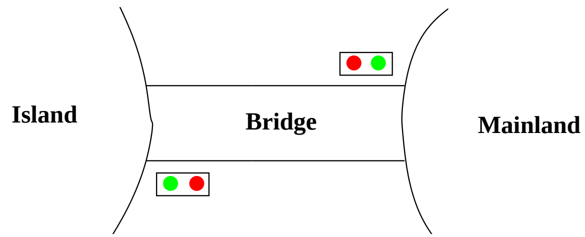
- The controller is equipped with two traffic lights with two colors.

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



Requirements

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

Requirements

- 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.

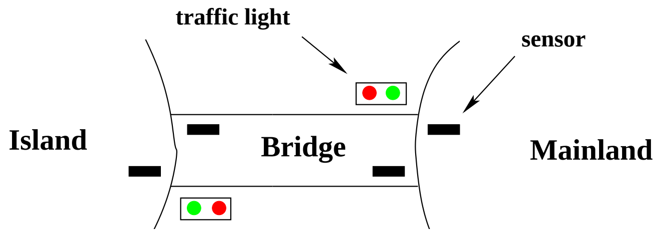
- There are four such sensors. Two of them are situated on the bridge and the other two are situated on the mainland and on the island.

The system is equipped with four car sensors each with two states: on or off	EQP-4

Requirements

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

- The pieces of equipment can be illustrated as follows:



Requirements

- 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

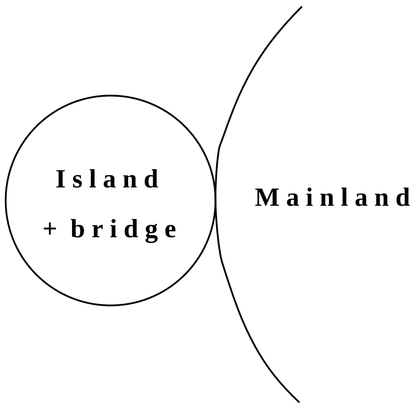
Strategy

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)

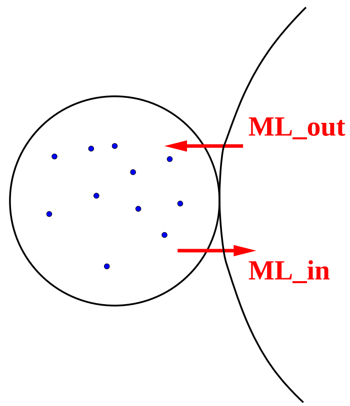
Initial model

- 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.

Dynamic part (machine):

variable: n

inv0_1: $n \in \mathbb{N}$

inv0_2: $n \leq d$

n number of cars in island + bridge

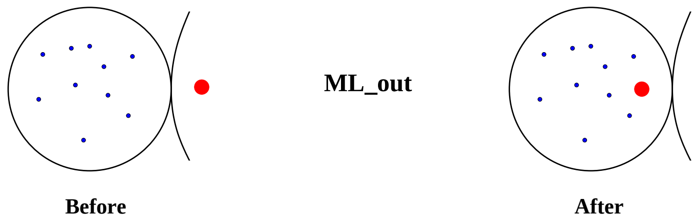
Always smaller than or equal to d (FUN_2)

- Labels axm0_1, inv0_1, chosen systematically.
- Label **axm**, **inv** recalls purpose.
- 0 (as in inv0_1): initial model.
- 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

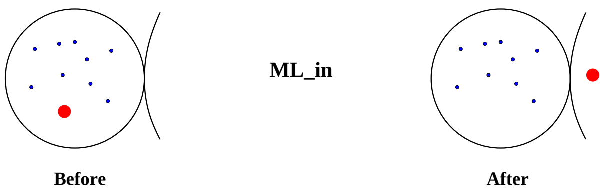
- A car is leaving the mainland and entering the Island-Bridge



Situation from the sky

- We can also observe a second transition (or event)

- A car leaving the Island-Bridge and re-entering the mainland



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

Situation from the sky

✓ ***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)

Events

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

Progress

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

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

- 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.*

Progress

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

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

- 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!

Progress

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

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

- 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 $\neg(n > 0 \vee n < d)$.

Progress

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

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

- In our case:

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

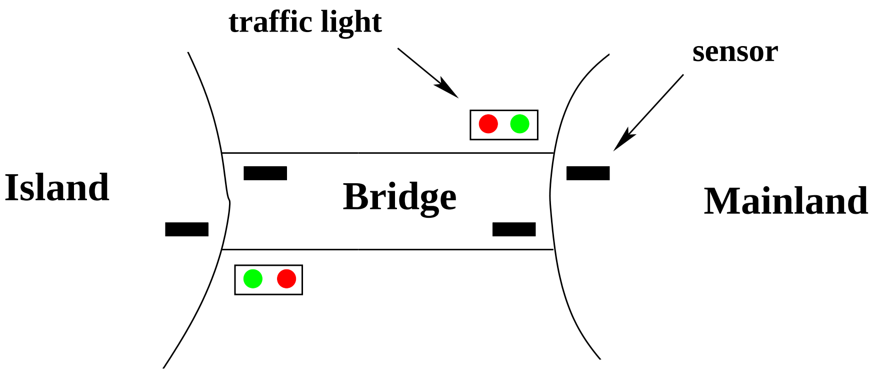
- ✓ *Add invariant at the end, mark as theorem.*
- Cannot be proven!
- Why? Let us find out in which cases events may be in deadlock.
- Solve $\neg(n > 0 \vee n < d)$.

- If $d = 0$, no car can enter! Missing axiom: $0 < d$. Add it.
- Note that we are calculating the model.

Strategy

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)

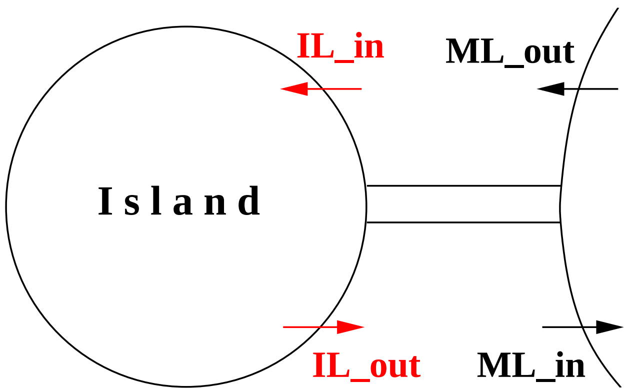
Physical system (reminder)



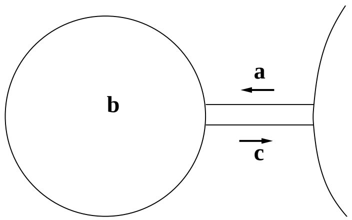
One-way bridge

- 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	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	??
Formalization of one-way bridge (FUN-3)	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	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	$a = 0 \vee c = 0$

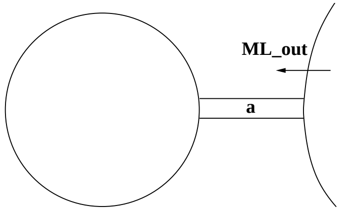
Refining state: invariants

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	$a = 0 \vee c = 0$

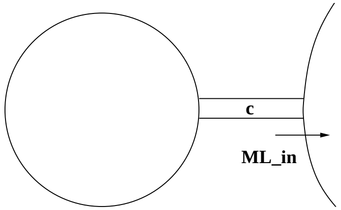
A new class of invariant

Note that we are not finding an invariant to justify 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 the requirement.

Event refinement proposal

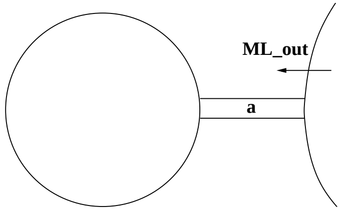


Event ML_out
  where
    ????
  then
    ????
  end

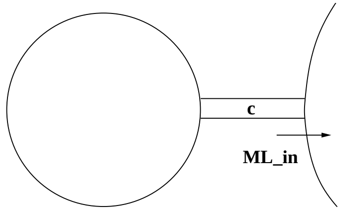


Event ML_in
  where
    ????
  then
    ????
  end

Event refinement proposal

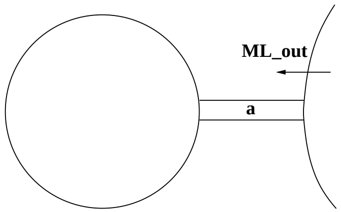


Event ML_out
  where
    $a + b < d$
    $c = 0$
  then
    $a := a + 1$
  end

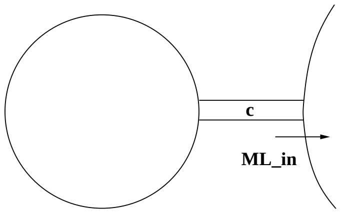


Event ML_in
  where
    ????
  then
    ????
  end

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

In Rodin...

- 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).

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

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

Refinement POs (reminder)

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

ML_out / GRD:

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

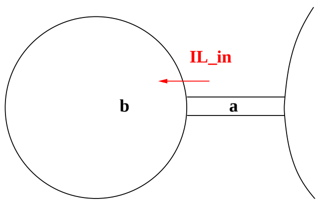
ML_in / GRD:

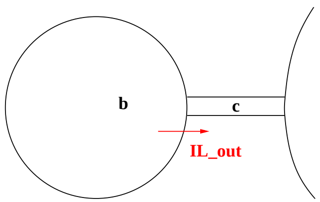
$d \in \mathbb{N}, 0 < d, n \in \mathbb{N}, n \leq d, a \in \mathbb{N}, b \in \mathbb{N}, c \in \mathbb{N}, a + b + c = n, a = 0 \vee c = 0, 0 < c \vdash 0 < n$

New events

- 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

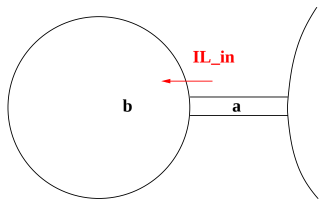


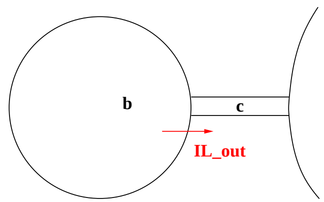



```
Event IL_in
    where
        ????
    then
        ????

    end
```

```
Event IL_out
    where
        ????

    then
        ????

    end
```

Proposal for new events

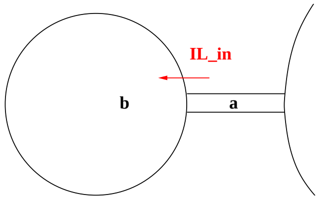


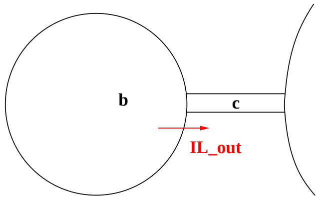


Event IL_in
  where
    $0 < a$
  then
    $a := a - 1$
    $b := b + 1$
  end

```
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

POs and convergence of new events

- New events refine implicit "void" event (*skip* action, *true* guards).
  - No *previous history* to respect.
    - 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 lifelocks $\Rightarrow$ computation progress.

Convergence of new events

- Events ML_in and ML_out can alternate *ad infinitum*.
- But **new events** must not diverge:
  - IL_in, IL_out should not be enabled without limits.
  - Not physically observable.
  - It should not happen in our model.
  - Ensure it does not happen without imposing unnecessary scheduling restrictions? (Dangerous!)
- Idea: create a variant that ensures IL_in, IL_out not indefinitely enabled.
- Reminder:

IL_in	IL_out
a := a − 1	c := c + 1
b := b + 1	b := b − 1

Convergence of new events

- Events ML_in and ML_out can alternate *ad infinitum*.
- But **new events** must not diverge:
  - IL_in, IL_out should not be enabled without limits.
  - Not physically observable.
  - It should not happen in our model.
  - Ensure it does not happen without imposing unnecessary scheduling restrictions? (Dangerous!)
- Idea: create a variant that ensures IL_in, IL_out not indefinitely enabled.
- Reminder:

IL_in	IL_out
a := a − 1	c := c + 1
b := b + 1	b := b − 1

- We need an f s.t.:

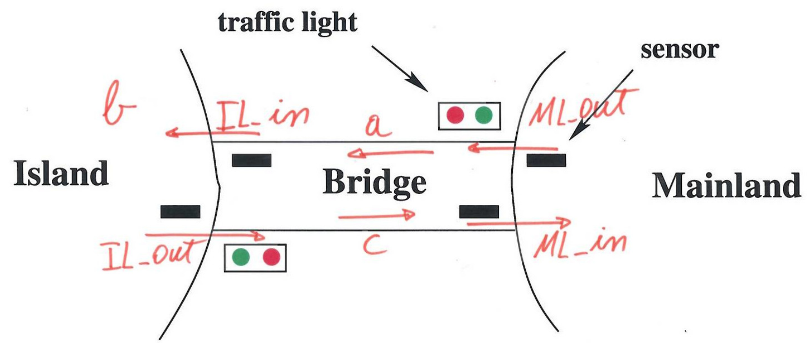
$$f(a,b,c) > f(a-1,b+1,c)$$
$$f(a,b,c) > f(a,b-1,c+1)$$

- Calculate it! ✓ *Add variant!*

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

- We can posit a simple f: $f(a,b,c) = k_1 a + k_2 b + k_3 c$
- Therefore:
$$k_1 a + k_2 b + k_3 c > k_1(a-1) + k_2(b+1) + k_3 c$$
$$k_1 a + k_2 b + k_3 c > k_1 a) + k_2(b-1) + k_3(c+1)$$
- 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



Progress: (relative) deadlock freedom

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

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

- 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

$$d \in \mathbb{N}, 0 < d, n \in \mathbb{N}, n \leq d, a \in \mathbb{N}, b \in \mathbb{N}, c \in \mathbb{N}, a + b + c = n, a = 0 \lor c = 0, 0 < n \lor n < d \vdash (a + b < d \land c = 0) \lor c > 0 \lor a > 0 \lor (b > 0 \land a = 0)$$

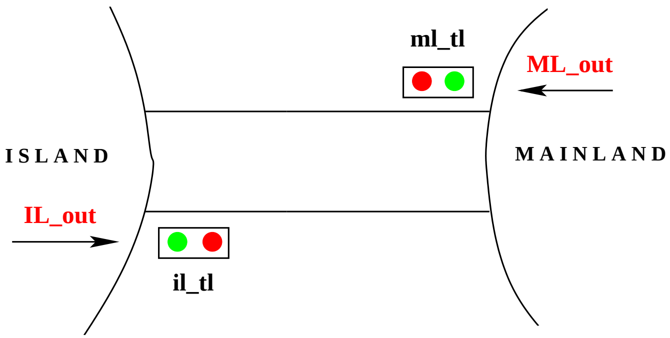
Discharged POs

- Proof Obligations
  - thm1/THM
  - thm2/THM
  - INITIALISATION/inv1/INV
  - INITIALISATION/inv2/INV
  - INITIALISATION/inv3/INV
  - INITIALISATION/inv4/INV
  - INITIALISATION/inv5/INV
  - ML_out/inv1/INV
  - ML_out/inv4/INV
  - ML_out/inv5/INV
  - ML_out/grd1/GRD
  - IL_in/inv1/INV
  - IL_in/inv2/INV
  - IL_in/inv4/INV
  - IL_in/inv5/INV
  - IL_in/VAR
  - IL_in/NAT
  - IL_out/inv2/INV
  - IL_out/inv3/INV
  - IL_out/inv4/INV
  - IL_out/inv5/INV
  - IL_out/VAR
  - IL_out/NAT
  - ML_in/inv3/INV
  - ML_in/inv4/INV
  - ML_in/inv5/INV
  - ML_in/grd1/GRD

Strategy

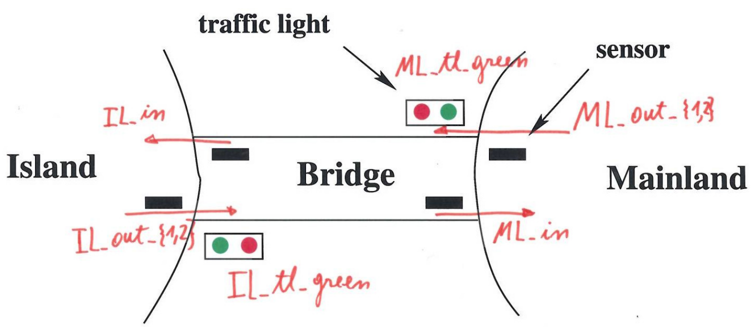
Initial model Limiting the number of cars (FUN-2).
First refinement Introducing the one-way bridge (FUN-3).
Second refinement Introducing the traffic lights (EQP-1,2,3)
Third refinement Introducing the sensors (EQP-4,5)

Introducing traffic lights



At the end of the refinement...

- When developing, we often do not know where we are going.
- For pedagogical reasons: this is where we will end in this refinement.



Introducing traffic lights

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`

Introducing traffic lights

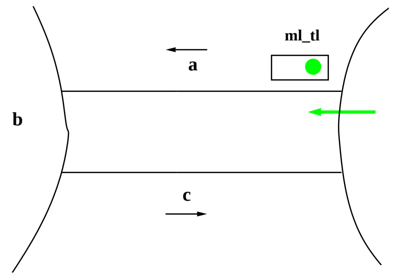
$$il\_tl \in COLOR$$

$$ml\_tl \in COLOR$$

Remark: Events IL_in and ML_in are not modified in this refinement

✓ *Add variables, invariants to* `m2`

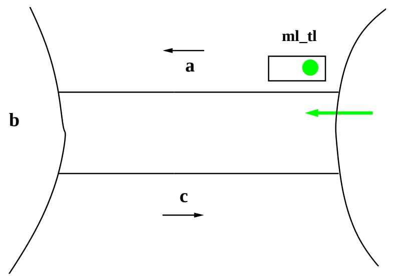
Introducing traffic lights: leaving mainland



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

Invariant?

Introducing traffic lights: leaving mainland

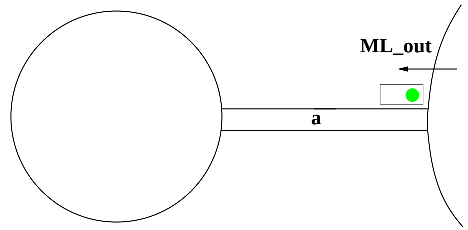


- A green mainland traffic light implies safe access to the bridge

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

Refining ML_out

- 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

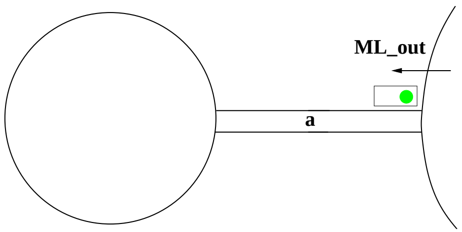
```
Event  ML_out
   where
         c = 0
         a + b < d
   then
         a := a + 1
   end
```

Concrete

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

Refining ML_out

- 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.



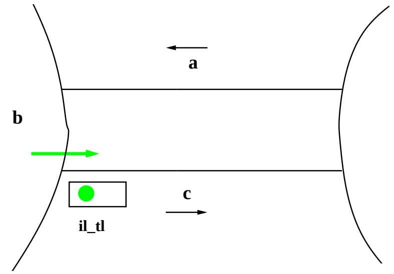
Abstract

```
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
```

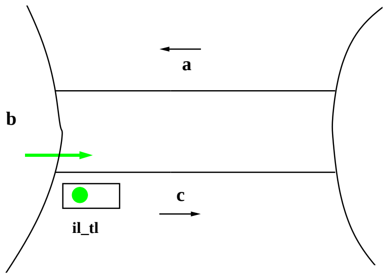
Introducing traffic lights: leaving island



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

Invariant?

Introducing traffic lights: leaving island

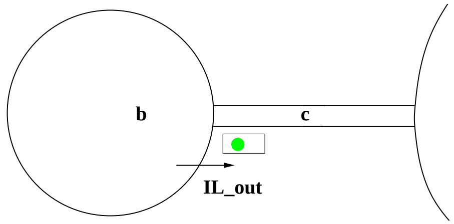


- A green island traffic light implies safe access to the bridge

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 ensure that the light does not turn green if the island is empty.

Refining IL_out



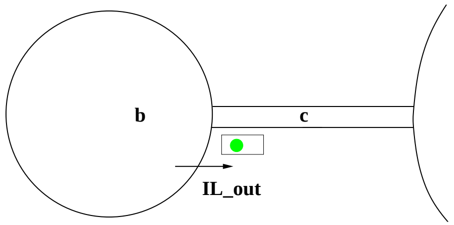
Abstract

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

Concrete

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

Refining IL_out



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
```

Status so far

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

✓ *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
```

Summary of refinement so far

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.

Issues in POs

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

$$H \vdash \bot$$

- Would be discharged if H were 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*

Issues in POs

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

$$H \vdash \bot$$

- Would be discharged if H were 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*

Status of proofs

Done	Pending
ML_out / inv2_4 / INV	ML_out / inv2_3 / INV
IL_out / inv2_3 / INV	IL_out / inv2_4 / INV
	ML_tl_green / inv2_5 / INV
	IL_tl_green / inv2_5 / INV

Issues in POs

```
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^{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
```

Issues in POs

```
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
```

Status of proofs

Done	Pending
ML_out / inv2_4 / INV	ML_tl_green / inv2_5 / INV
IL_out / inv2_3 / INV	IL_tl_green / inv2_5 / INV
ML_out_{1,2} / inv2_3 / INV	
IL_out_{1,2} / inv2_4 / INV	

Proving inv2_5

inv2_5: $ml\_tl = red \lor il\_tl = red$

- 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

Proving inv2_5

inv2_5: $ml\_tl = red \lor il\_tl = red$

- 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

Proving inv2_5

inv2_5: $ml\_tl = red \lor il\_tl = red$

- 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

Proving inv2_5

inv2_5: $ml\_tl = red \lor il\_tl = red$

- 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

✓ *Add actions*

Divergence

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[a] are suspicious.
- New events in particular — remember we already proved convergence of IL_in and IL_out

[a] “Clearly” does not ensure; properties should anyway be proven.

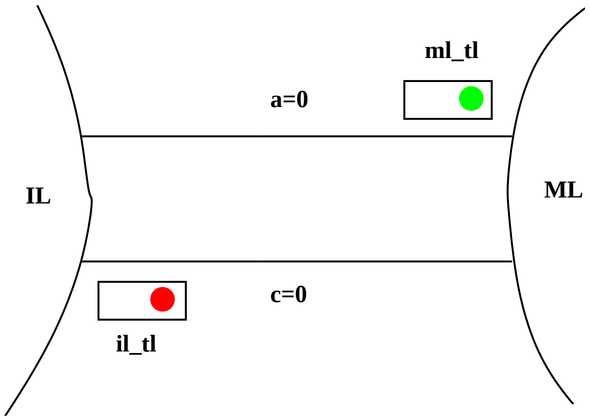
Divergence

Event ML_tl_green
  where
    $ml\_tl = red$
    $a + b < d$
    $c = 0$
  then
    $ml\_tl := green$
    $il\_tl := red$
  end

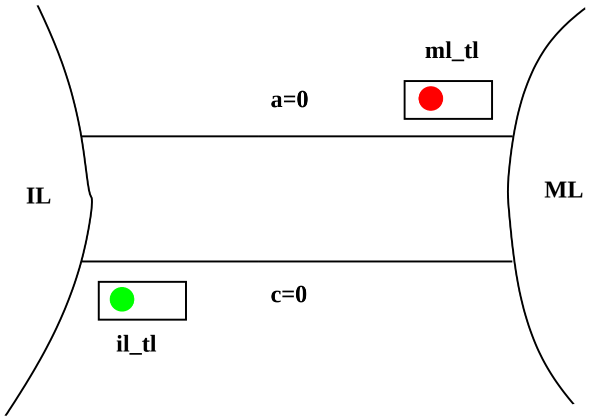
Event IL_tl_green
  where
    $il\_tl = red$
    $0 < b$
    $a = 0$
  then
    $il\_tl := green$
    $ml\_tl := red$
  end

- Guards depend on a, b, c and traffic lights.
- $ml\_tl = red$ and $il\_tl = red$ (in guards) alternatively set by the other event.
- The rest of the guards are simultaneously true when $a = c = 0, 0 < b < d$.
- Traffic lights could alternatively change colors w.o. control.

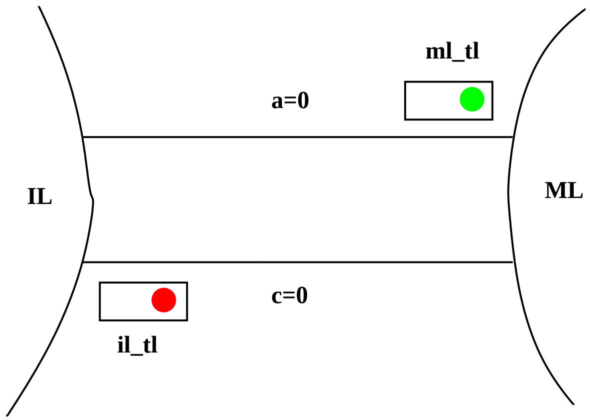
Alternating traffic lights



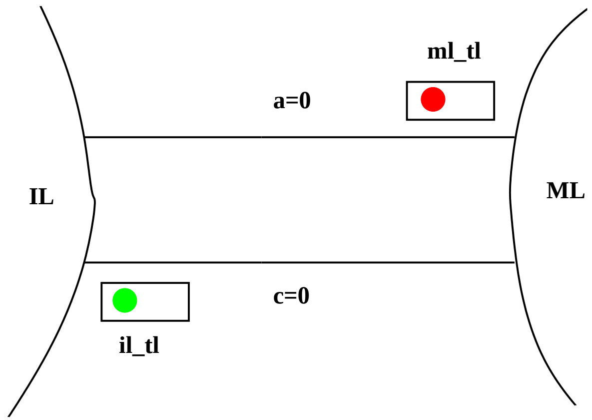
Alternating traffic lights



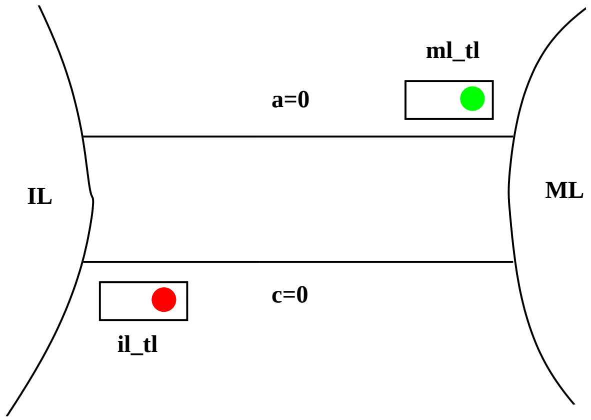
Alternating traffic lights



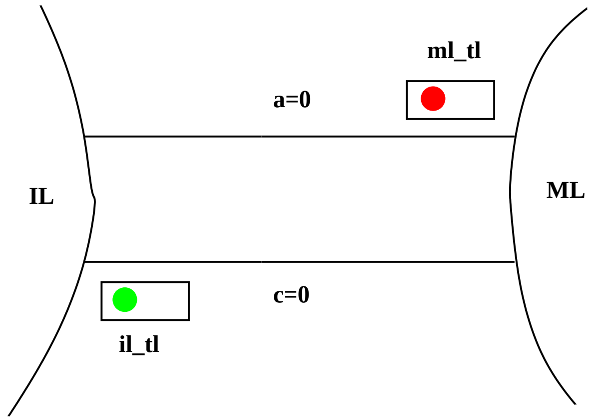
Alternating traffic lights



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.

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.
- 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: $il\_pass \in \{0, 1\}$
- We update them when cars go out of mainland and out of the island.

Concerns:

- Is it safe?

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.
- 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: $il\_pass \in \{0, 1\}$
- 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.

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.
- 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: $il\_pass \in \{0, 1\}$
- 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.
- Isn't traffic going to stop circulating?

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.
- 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: $il\_pass \in \{0, 1\}$
- 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.
- Isn't traffic going to stop circulating?
- Perhaps. Anyway we were letting traffic lights change color, and stating when it is not safe to do so. We will deal with that.

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
```

Divergence, once more

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

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

- Convergence proofs (for ML_tl_green and IL_tl_green):

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

Divergence, once more

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

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

- Convergence proofs (for ML_tl_green and IL_tl_green):

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

- Cannot be proven as they are.

Divergence, once more

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

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

- Convergence proofs (for ML_tl_green and IL_tl_green):

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

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

$$\textbf{inv2\_8:} \quad ml\_tl = red \Rightarrow ml\_pass = 1$$
$$\textbf{inv2\_9:} \quad il\_tl = red \Rightarrow il\_pass = 1$$

- 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.

No-deadlock

All axioms, invariants, theorems $\vdash$

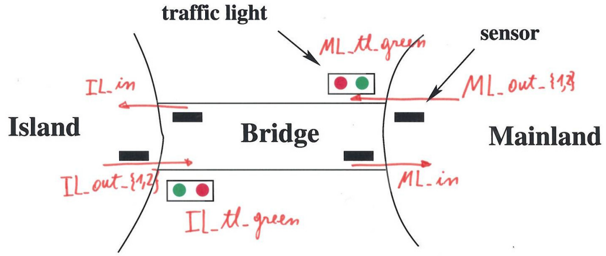
$$\begin{array}{ll}
(ml_tl = green \wedge a + b + 1 < d) & \vee \\
(ml_tl = green \wedge a + b + 1 = d) & \vee \\
(il_tl = green \wedge b > 1) \vee (il_tl = green \wedge b = 1) & \vee \\
(ml_tl = red \wedge a + b < d \wedge c = 0 \wedge il_pass = 1) & \vee \\
(il_tl = red \wedge 0 < b \wedge a = 0 \wedge ml_pass = 1) & \vee \\
0 < a \vee 0 < c &
\end{array}$$

- 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



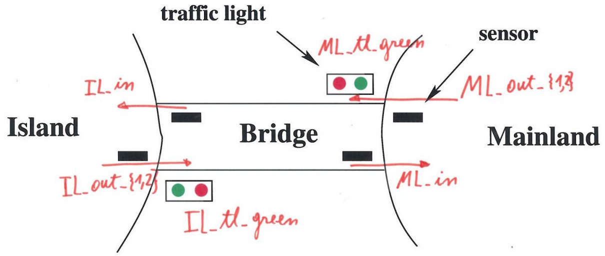
ML_in	Car leaves bridge to mainland.
IL_in	Car bridge leaves to island.
ML_tl_green	Controls ML traffic light.
	• Dep. on # of cars, turn.
IL_tl_green	Same for island traffic light.

{M,I}L_out_{1,2} Cars enter bridge.

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

- How do we know # of cars?

Analysis of second refinement



ML_in Car leaves bridge to mainland.

IL_in Car bridge leaves to island.

ML_tl_green Controls ML traffic light.

- Dep. on # of cars, turn.

IL_tl_green Same for island traffic light.

{M,I}L_out_{1,2} Cars enter bridge.

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

- How do we know # of cars?
- Sensors!

Invariant / variant summary

$ml\_tl \in \{red, green\}$	Possible colors .
$il\_tl \in \{red, green\}$	Possible colors.
$ml\_tl = green \Rightarrow a + b < d \wedge c = 0$	If TL to enter island is green, there is space in the island and no car is leaving.
$il\_tl = green \Rightarrow 0 < b \wedge a = 0$	If TL to exit island is green, at least on car is in the island and no car is coming in through the bridge.
$ml\_tl = red \vee il\_tl = red$	Both traffic lights cannot be green at the same time.
$ml\_pass \in \{0, 1\}$	A car entered bridge from ML since ML TL turned green.
$il\_pass \in \{0, 1\}$	A car entered bridge from IL since IL TL turned green.
$ml\_tl = red \Rightarrow ml\_pass = 1$	Captures *technical* invariant
$il\_tl = red \Rightarrow il\_pass = 1$	Captures *technical* invariant
variant: $ml\_pass + il\_pass$	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 ≠ 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
```

Summary of events (4)

These are identical to their abstract versions

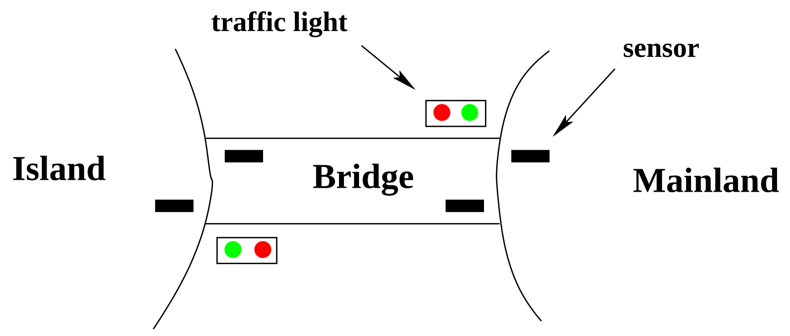
```
Event  ML_in
    where
        0 < c
    then
        c := c − 1
    end
```

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

Strategy

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

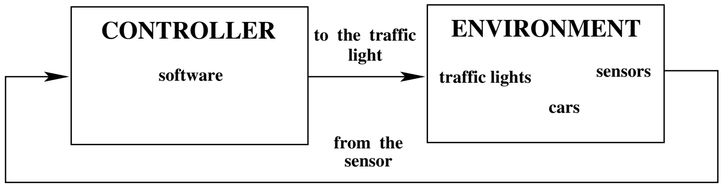


Environment and control

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 and environment variables

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.

Channels

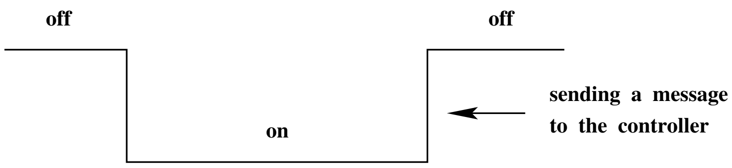
Output channels
(send state / signal to traffic lights)

ml_tl,
il_tl

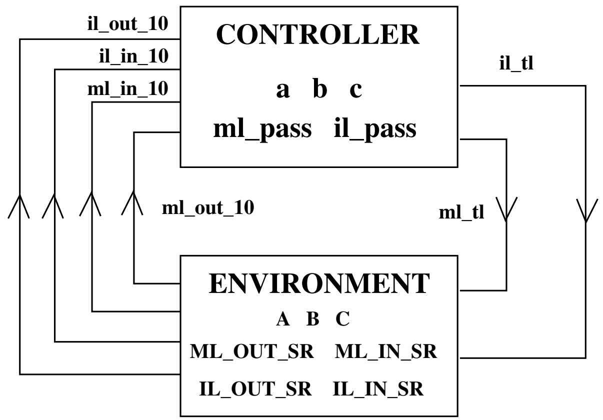
Input channels
(receive signals from sensors):

ml_out_10,
ml_in_10,
il_out_10,
il_in_10

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



Summary



Enlarging the refined model

- The possible states of a sensor:

  Carrier sets: $\ldots, SENSOR$.
  Constants: on, off.
  axm3_1: $SENSOR = \{on, off\}$
  axm3_2: $on \neq off$

- Type invariants:

  inv3_1: $ML\_OUT\_SR \in SENSOR$
  inv3_2: $ML\_IN\_SR \in SENSOR$

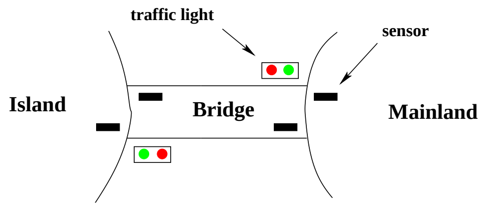
inv3_3: $IL\_OUT\_SR \in SENSOR$
inv3_4: $IL\_IN\_SR \in SENSOR$
inv3_5: $A \in \mathbb{N}$
inv3_6: $B \in \mathbb{N}$
inv3_7: $C \in \mathbb{N}$
inv3_8: $ml\_out\_10 \in \mathrm{BOOL}$
inv3_9: $ml\_in\_10 \in \mathrm{BOOL}$
inv3_10: $il\_out\_10 \in \mathrm{BOOL}$
inv3_11: $il\_in\_10 \in \mathrm{BOOL}$

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

Invariants capturing behavior, relationship with environment

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

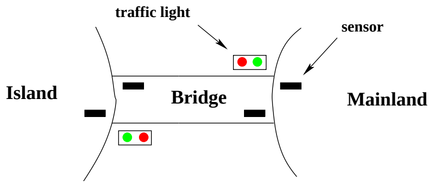


The sensors are used to detect the presence of cars entering or leaving the bridge	EQP-5

(We do not count / control cars in mainland)

Invariants capturing behavior, relationship with environment

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



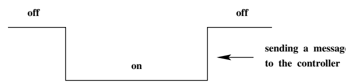
inv3_15: $ml\_out\_10 = \mathrm{TRUE} \Rightarrow ml\_tl = green$
inv3_16: $il\_out\_10 = \mathrm{TRUE} \Rightarrow il\_tl = green$

Cars are supposed to pass only on a green traffic light	EQP-3

Linking hardware sensor information and logical representation

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

inv3_17: $IL\_IN\_SR = on \Rightarrow il\_in\_10 = \mathrm{FALSE}$
inv3_18: $IL\_OUT\_SR = on \Rightarrow il\_out\_10 = \mathrm{FALSE}$
inv3_19: $ML\_IN\_SR = on \Rightarrow ml\_in\_10 = \mathrm{FALSE}$
inv3_20: $ML\_OUT\_SR = on \Rightarrow ml\_out\_10 = \mathrm{FALSE}$



The controller must be fast enough so as to be able to treat all the information coming from the environment	FUN-5

Physical and logical cars

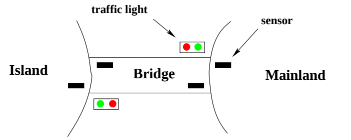
inv3_21: $il\_in\_10 = \mathrm{TRUE} \wedge ml\_out\_10 = \mathrm{TRUE} \Rightarrow A = a$
inv3_22: $il\_in\_10 = \mathrm{FALSE} \wedge ml\_out\_10 = \mathrm{TRUE} \Rightarrow A = a + 1$
inv3_23: $il\_in\_10 = \mathrm{TRUE} \wedge ml\_out\_10 = \mathrm{FALSE} \Rightarrow A = a - 1$
inv3_24: $il\_in\_10 = \mathrm{FALSE} \wedge ml\_out\_10 = \mathrm{FALSE} \Rightarrow A = a$

inv3_25: $il\_in\_10 = \mathrm{TRUE} \wedge il\_out\_10 = \mathrm{TRUE} \Rightarrow B = b$
inv3_26: $il\_in\_10 = \mathrm{TRUE} \wedge il\_out\_10 = \mathrm{FALSE} \Rightarrow B = b + 1$
inv3_27: $il\_in\_10 = \mathrm{FALSE} \wedge il\_out\_10 = \mathrm{TRUE} \Rightarrow B = b - 1$
inv3_28: $il\_in\_10 = \mathrm{FALSE} \wedge il\_out\_10 = \mathrm{FALSE} \Rightarrow B = b$

inv3_29: $il\_out\_10 = \mathrm{TRUE} \wedge ml\_out\_10 = \mathrm{TRUE} \Rightarrow C = c$
inv3_30: $il\_out\_10 = \mathrm{TRUE} \wedge ml\_out\_10 = \mathrm{FALSE} \Rightarrow C = c + 1$
inv3_31: $il\_out\_10 = \mathrm{FALSE} \wedge ml\_out\_10 = \mathrm{TRUE} \Rightarrow C = c - 1$
inv3_32: $il\_out\_10 = \mathrm{FALSE} \wedge ml\_out\_10 = \mathrm{FALSE} \Rightarrow C = c$

Rationale

inv3_21: $il\_in\_10 = \mathrm{TRUE} \wedge ml\_out\_10 = \mathrm{TRUE} \Rightarrow A = a$
inv3_22: $il\_in\_10 = \mathrm{FALSE} \wedge ml\_out\_10 = \mathrm{TRUE} \Rightarrow A = a + 1$
inv3_23: $il\_in\_10 = \mathrm{TRUE} \wedge ml\_out\_10 = \mathrm{FALSE} \Rightarrow A = a - 1$
inv3_24: $il\_in\_10 = \mathrm{FALSE} \wedge ml\_out\_10 = \mathrm{FALSE} \Rightarrow A = a$



- *A*: physical # cars. Updated by events representing cars entering.
- *a*: *controller* (logical) view.
- When $ml\_out\_10 = \mathrm{TRUE}$: other events will update logical # of cars, set $ml\_out\_10 = \mathrm{FALSE}$.
- 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 $ml\_out\_10 = \mathrm{FALSE}$ and updates *a*. Here $A = a$.

When $ml\_out\_10 = \mathrm{TRUE} \wedge il\_out\_10 = \mathrm{TRUE}$, 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 ≠ d
    then
        a := a + 1
        ml_pass := 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
```

Basic properties

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

Variant

- Ensure new events converge.

Variant

- 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