

# Practical System Modeling (2)

## Example: Controlling Cars on a Bridge

Jean-Raymond Abrial

May 2005

# The Reference Document (1)

---

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

FUN-1

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

# The Reference Document (2)

---

<p>The system has two traffic lights with two colors: green and red</p>	<p>EQP-1</p>
---	--------------

<p>The traffic lights control the entrance to the bridge at both ends of it</p>	<p>EQP-2</p>
---	--------------

<p>Cars are not supposed to pass on a red traffic light, only on a green one</p>	<p>EQP-3</p>
--	--------------

# The Reference Document (3)

---

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

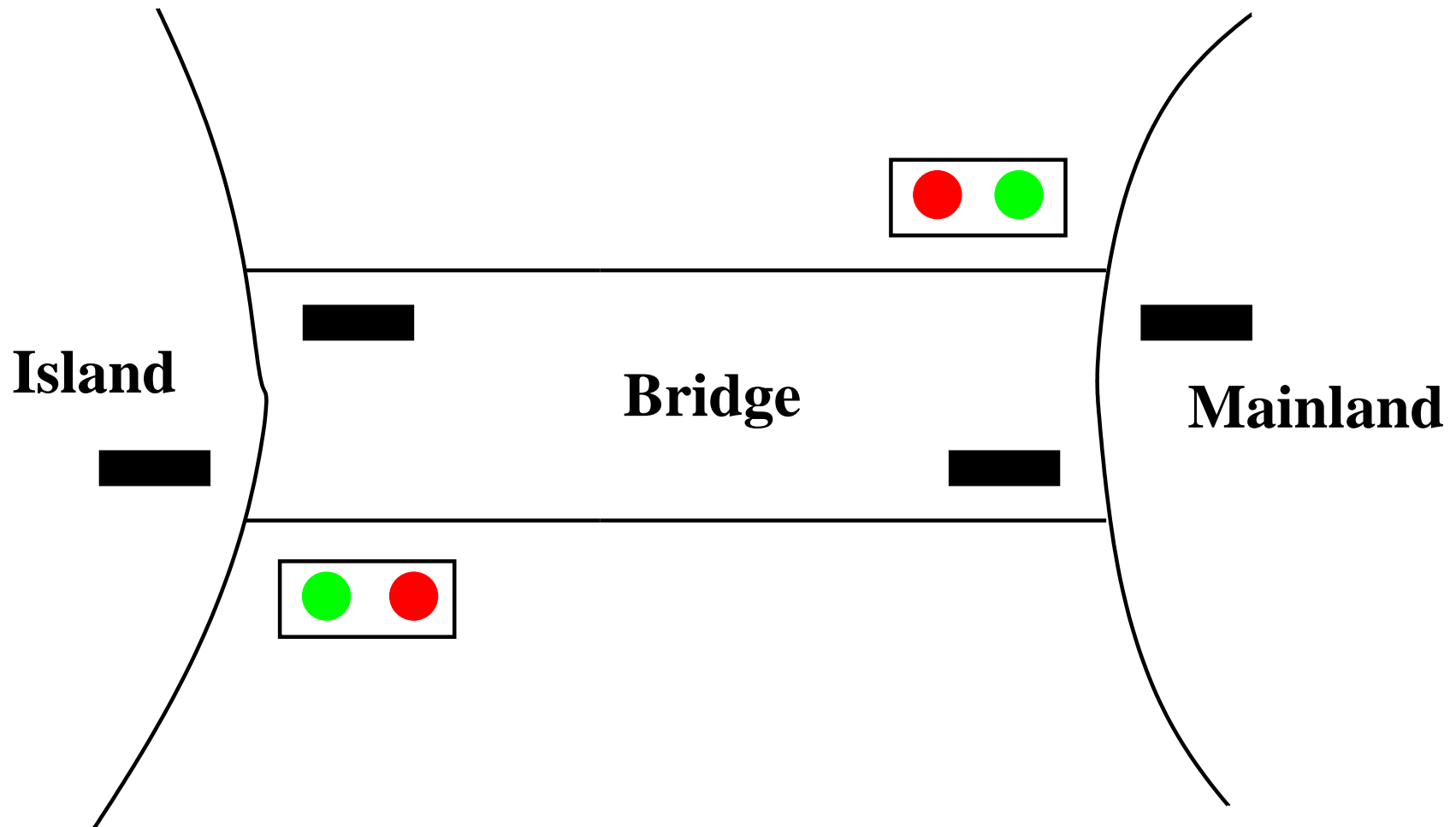
EQP-4

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

EQP-5

# Equipment

---



# Our Development Approach

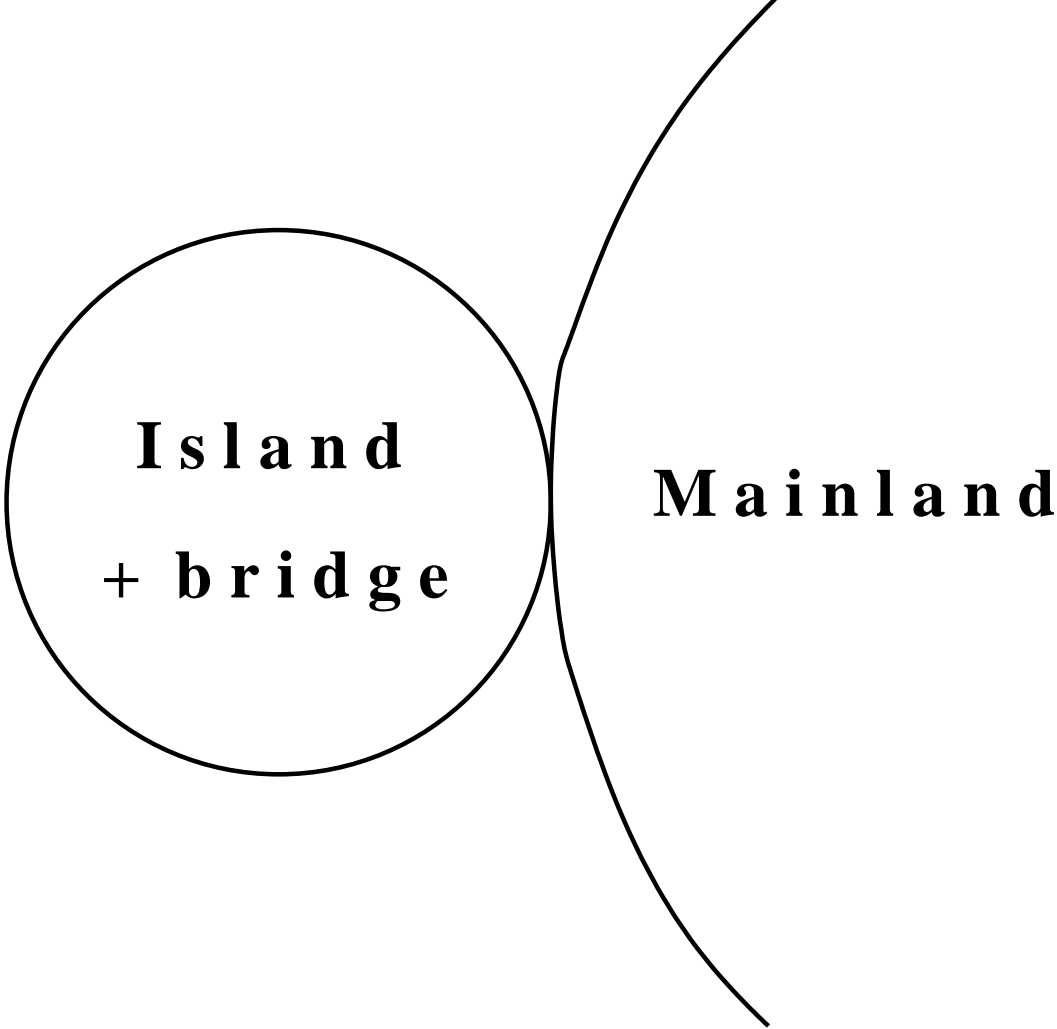
---

- **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)

- It is **very simple**
- We do not consider at all the equipment: traffic lights and sensors
- We even do not consider the bridge
- We are just interested in the **pair “island-bridge”**
- We are focusing on **FUN-2** (limited number of cars on island-bridge)

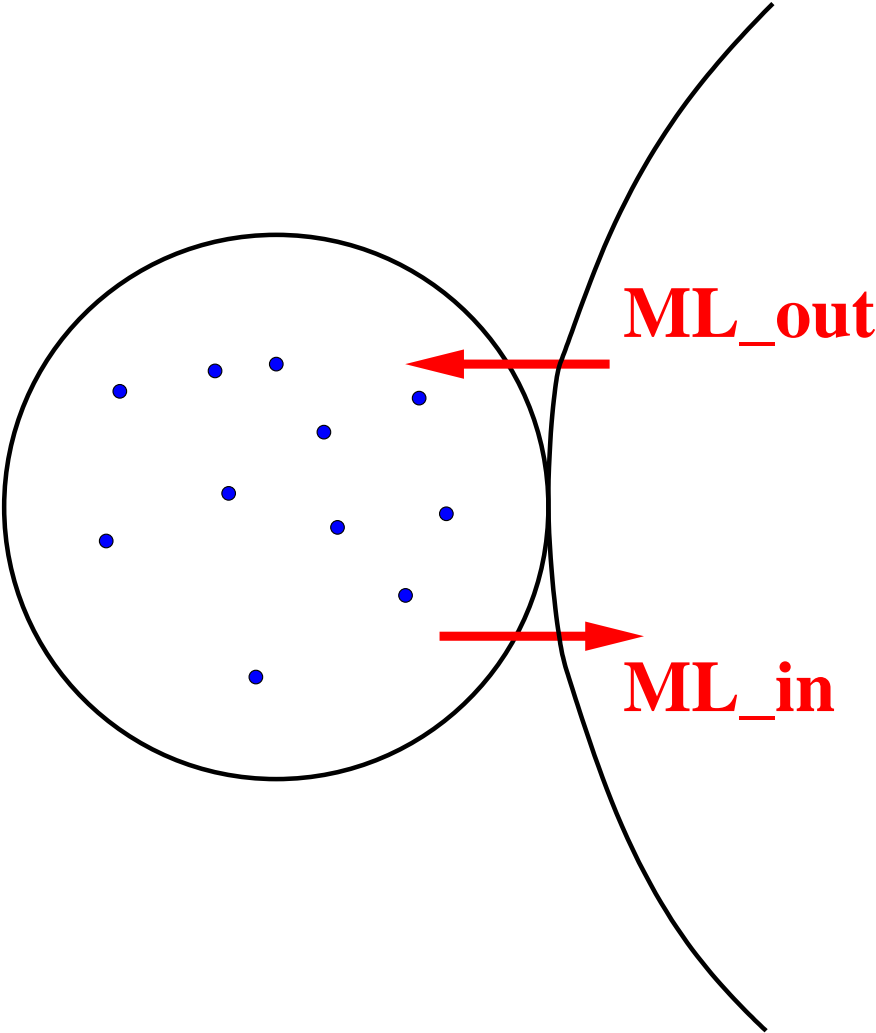
# A Situation as Seen from the Sky

---



# Two Events

---



# Formalizing the State

---

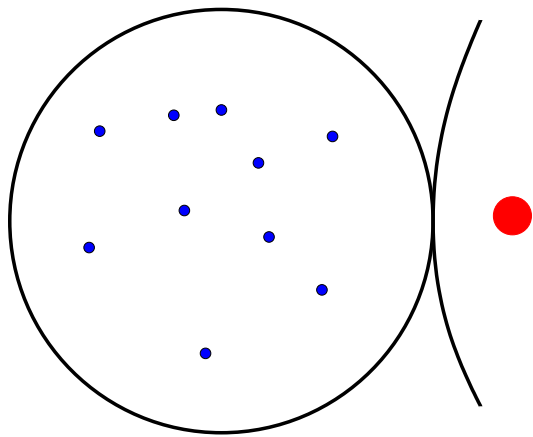
**constant:**  $d$

**variable:**  $n$

**prp0\_1:**  $d \in \mathbb{N}$

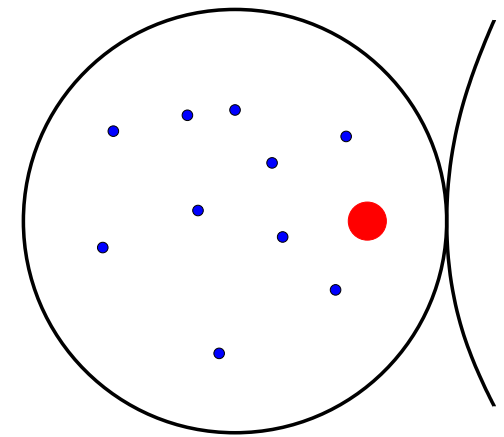
**inv0\_1:**  $n \in \mathbb{N}$

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

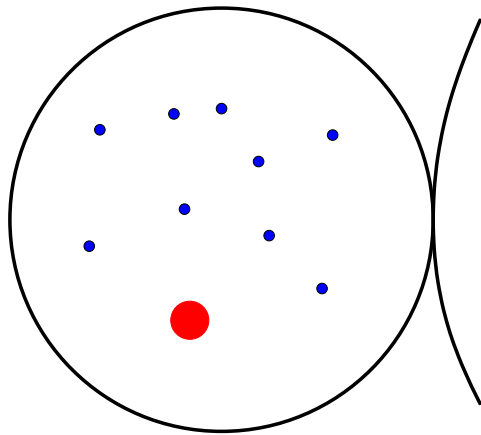


**Before**

**ML\_out**

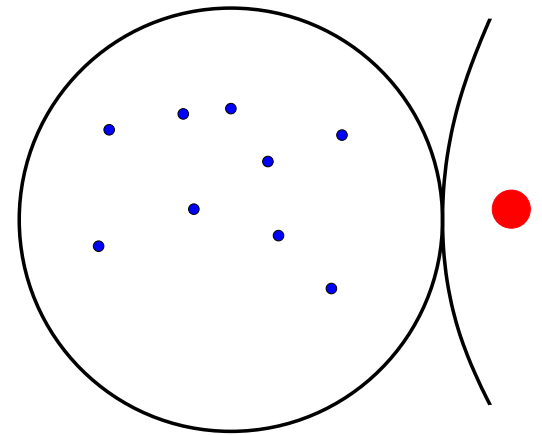


**After**



**Before**

**ML\_in**



**After**

# Formalizing the two Events and Initialization

---

- Initially there are no cars on the Island-Bridge
- Event ML\_out **increments** the number of cars
- Event ML\_in **decrements** the number of cars

**init**

$n := 0$

**ML\_out**

$n := n + 1$

**ML\_in**

$n := n - 1$

The Events

**init**

$$n := 0$$

**ML\_out**

$$n := n + 1$$

**ML\_in**

$$n := n - 1$$

The corresponding **before-after** predicates

$$n' = 0$$

$$n' = n + 1$$

$$n' = n - 1$$

These representations are equivalent

# Proving Invariant Preservation

---

- Given constants  $c$  with properties  $P(c)$
- And variables  $v$  with invariants  $I(c, v)$
- Given an event with before-after predicate  $v' = E(c, v)$
- We have to prove the following in order to preserve the invariant:

$P(c) \wedge I(c, v) \Rightarrow I(c, E(c, v))$	INV
---	-----

# Vertical Layout of the Invariant Rule

---

<p>Properties of the constants Invariants <math>\Rightarrow</math> Modified Invariants</p>	<p><math>P(c)</math> <math>I(c, v)</math> <math>\Rightarrow</math> <math>I(c, E(c, v))</math></p>	<p>INV</p>
--	---	------------

# What is to be proved

---

- We have **two events**

**ML\_out**

$$n := n + 1$$

**ML\_in**

$$n := n - 1$$

- And **two invariants**

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

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

- Thus **four statements have to be proved**

# To be Proved for event **ML\_out** (1)

---

**ML\_out**

$$n := n + 1$$

Properties of the constants

Invariant **inv0\_1**

Invariant **inv0\_2**

$\Rightarrow$

Modified Invariant **inv0\_1**

$$d \in \mathbb{N}$$

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

$$n \leq d$$

$\Rightarrow$

$$n + 1 \in \mathbb{N}$$

# To be Proved for event **ML\_out** (2)

---

**ML\_out**

$$n := n + 1$$

Properties of the constants

Invariant **inv0\_1**

Invariant **inv0\_2**

$\Rightarrow$

Modified Invariant **inv0\_2**

$$d \in \mathbb{N}$$

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

$$n \leq d$$

$\Rightarrow$

$$n + 1 \leq d$$

# To be Proved for event **ML\_in** (1)

---

**ML\_in**

$n := n - 1$

Properties of the constants

Invariant **inv0\_1**

Invariant **inv0\_2**

$\Rightarrow$

Modified Invariant **inv0\_1**

$d \in \mathbb{N}$

$n \in \mathbb{N}$

$n \leq d$

$\Rightarrow$

$n - 1 \in \mathbb{N}$

# To be Proved for event **ML\_in** (2)

**ML\_in**

$$n := n - 1$$

Properties of the constants

Invariant **inv0\_1**

Invariant **inv0\_2**

$\Rightarrow$

Modified Invariant **inv0\_2**

$$d \in \mathbb{N}$$

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

$$n \leq d$$

$\Rightarrow$

$$n - 1 \leq d$$

# Summary of What is to be Proved

---

$$\begin{array}{l} d \in \mathbb{N} \\ n \in \mathbb{N} \\ n \leq d \\ \Rightarrow \\ n + 1 \in \mathbb{N} \end{array}$$

$$\begin{array}{l} d \in \mathbb{N} \\ n \in \mathbb{N} \\ n \leq d \\ \Rightarrow \\ n + 1 \leq d \end{array}$$

$$\begin{array}{l} d \in \mathbb{N} \\ n \in \mathbb{N} \\ n \leq d \\ \Rightarrow \\ n - 1 \in \mathbb{N} \end{array}$$

$$\begin{array}{l} d \in \mathbb{N} \\ n \in \mathbb{N} \\ n \leq d \\ \Rightarrow \\ n - 1 \leq d \end{array}$$

# Two Proofs Fail (1)

---

$$\begin{array}{l} d \in \mathbb{N} \\ n \in \mathbb{N} \\ n \leq d \\ \Rightarrow \\ n + 1 \leq d \end{array}$$

- This proof fails when  $n = d$
- We must have the additional assumption:  $n < d$

## Two Proofs Fail (2)

---

$$\begin{array}{l} d \in \mathbb{N} \\ n \in \mathbb{N} \\ n \leq d \\ \Rightarrow \\ n - 1 \in \mathbb{N} \end{array}$$

- This proof fails when  $n = 0$
- We must have the additional assumption:  $0 < n$

```
ML_out
  when
     $n < d$ 
  then
     $n := n + 1$ 
  end
```

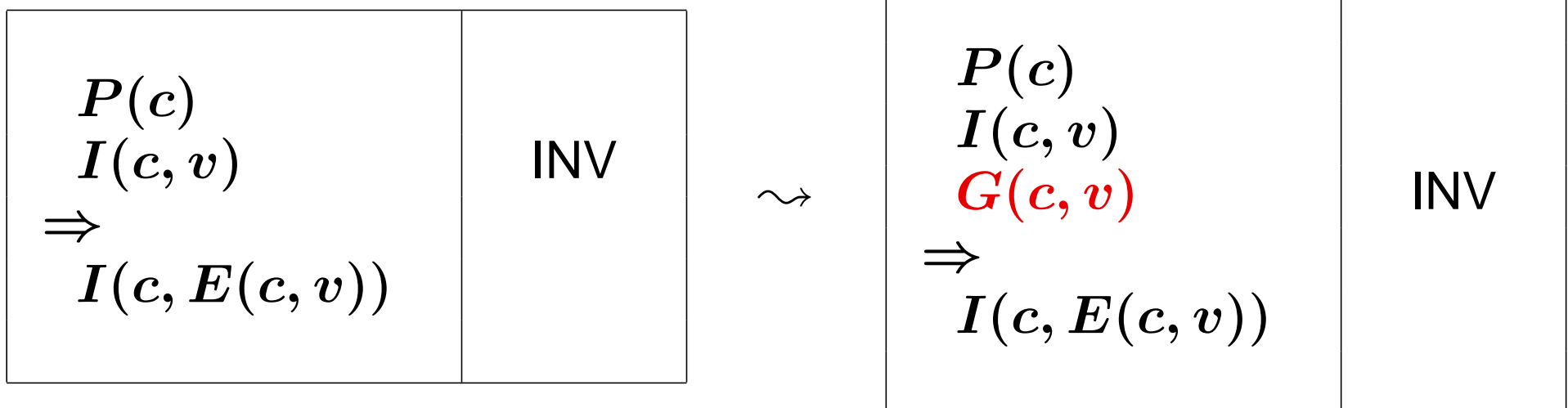
```
ML_in
  when
     $0 < n$ 
  then
     $n := n - 1$ 
  end
```

We are adding **guards** to the events

The guard is the **necessary condition** for an event to “occur”

# Adapting the Invariant Preservation Rule

- Given  $c$  with properties  $P(c)$  and  $v$  with invariants  $I(c, v)$
- Given an event with guard  $G(c, v)$  and b-a predicate  $v' = E(c, v)$
- We modify the Invariant Preservation Rule as follows:



Properties of the constants

Invariants

Guard of the event

$\Rightarrow$

Modified Invariant

$P(c)$

$I(c, v)$

$G(c, v)$

$\Rightarrow$

$I(c, E(c, v))$

INV

# Re-proving the Events: No Proofs Fail

$$\begin{array}{l} d \in \mathbb{N} \\ n \in \mathbb{N} \\ n \leq d \\ n < d \\ \Rightarrow \\ n + 1 \in \mathbb{N} \end{array}$$

$$\begin{array}{l} d \in \mathbb{N} \\ n \in \mathbb{N} \\ n \leq d \\ n < d \\ \Rightarrow \\ n + 1 \leq d \end{array}$$

$$\begin{array}{l} d \in \mathbb{N} \\ n \in \mathbb{N} \\ n \leq d \\ 0 < n \\ \Rightarrow \\ n - 1 \in \mathbb{N} \end{array}$$

$$\begin{array}{l} d \in \mathbb{N} \\ n \in \mathbb{N} \\ n \leq d \\ 0 < n \\ \Rightarrow \\ n - 1 \leq d \end{array}$$

- Our system must be **initialized** (with no car in the island-bridge)
- The initialization event is **never guarded**
- It does **not mention any variable** in the right hand side of  $:=$

**init**  
 $n := 0$

After predicate

$n' = 0$

# Invariant Establishment Rule

- Given  $c$  with properties  $P(c)$  and  $v$  with invariants  $I(c, v)$
- Given an init event with after predicate  $v' = K(c)$
- The Invariant Establishment Rule is the following:

Properties of the constants $\Rightarrow$ Modified Invariants
---

$P(c)$ $\Rightarrow$ $I(c, K(c))$	INI_INV
---	---------

# Applying the Invariant Establishment Rule

---

Property of the constant  $d$   
 $\Rightarrow$   
Modified Invariant

$$\begin{array}{l} d \in \mathbb{N} \\ \Rightarrow \\ 0 \in \mathbb{N} \end{array}$$

$$\begin{array}{l} d \in \mathbb{N} \\ \Rightarrow \\ 0 \leq d \end{array}$$

- Both statements hold trivially

# A Missing Requirement

---

- It is possible for the system to be blocked if both guards are false
- We do not want this to happen
- We figure out that one important requirement is missing

Once started, the system should work for ever
---

FUN-4
-------

# Proving Deadlock Freeness Rule

- Given  $c$  with properties  $P(c)$  and  $v$  with invariants  $I(c, v)$
- Given the guards  $G_1(c, v), \dots, G_m(c, v)$  of the events
- We have to prove the following:

$\begin{array}{l} P(c) \\ I(c, v) \\ \Rightarrow \\ G_1(c, v) \vee \dots \vee G_m(c, v) \end{array}$	$\text{DLF}$
--	--------------

Properties of constants

Invariants

$\Rightarrow$

Disjunction of guards

$d \in \mathbb{N}$

$n \in \mathbb{N}$

$n \leq d$

$\Rightarrow$

$n < d \vee 0 < n$

This cannot be proved when  $d = 0$ , because then  $n = 0$

since  $n \in \mathbb{N}$  and  $n \leq d$ . We have to add the following property:

**prp0\_2:**  $0 < d$

- Thanks to the **proofs**, we discovered **3 errors**
- They were corrected by:
  - adding guards to both events
  - adding an additional property

- We have seen three Proof Rules:
  - The **Invariant Establishment** Rule: INI\_INV
  - The **Invariant Preservation** Rule: INV
  - The **Deadlock Freeness** Rule (not mandatory): DLF

Properties of the constants $\Rightarrow$ Modified Invariants	INI_INV
---	---------

Properties of the constants Invariants Guard of the event $\Rightarrow$ Modified Invariants	INV
---	-----

<p>Property of the constant Invariants</p> <p><math>\Rightarrow</math></p> <p>Disjunction of the guards</p>	<p>DLF</p>
---	------------

# Summary of Initial Model

---

**constant:**  $d$

**variable:**  $n$

**prp0\_1:**  $d \in \mathbb{N}$

**prp0\_2:**  $d > 0$

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

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

init

$n := 0$

ML\_out

**when**

$n < d$

**then**

$n := n + 1$

**end**

ML\_in

**when**

$0 < n$

**then**

$n := n - 1$

**end**

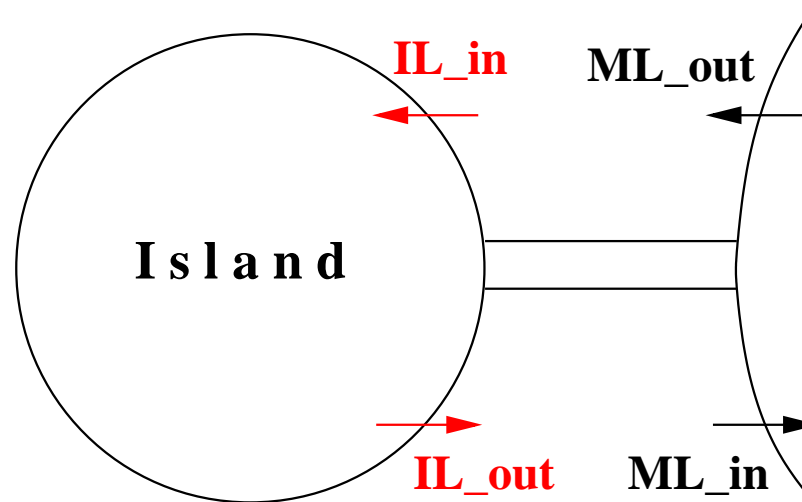
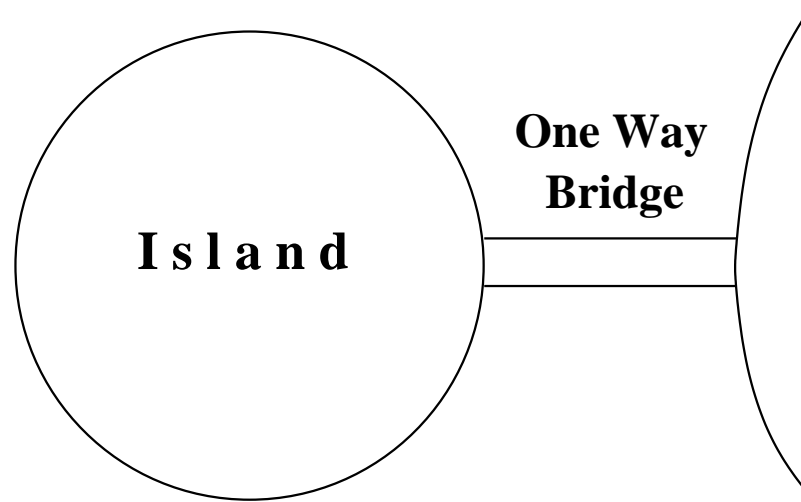
- **Initial model:** Limiting the number of cars (FUN\_2)
- **First refinement:** Introducing the one way bridge (FUN\_3)
- **Second refinement:** Introducing the traffic lights (EQP\_1,2,3)
- **Third refinement:** Introducing the sensors (EQP\_4,5)

# First Refinement: Introducing a One-Way Bridge

---

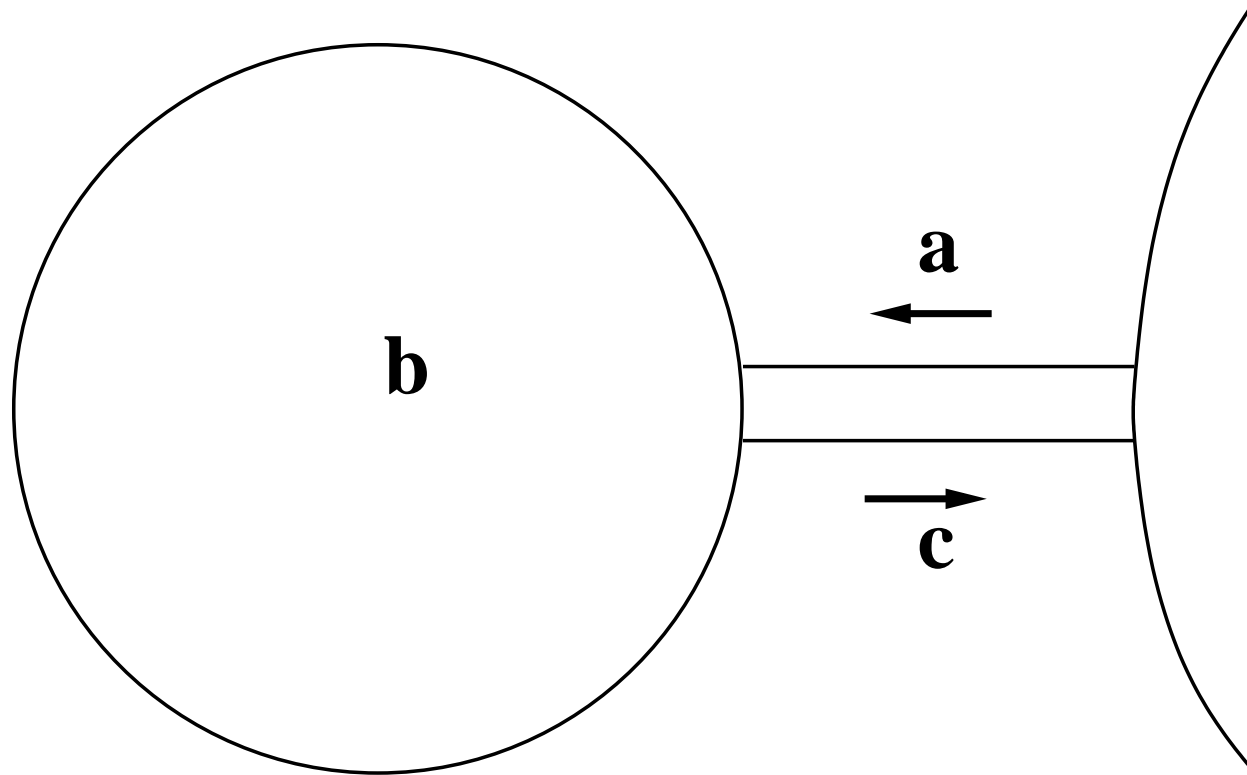
- Our **view** of the system gets **more accurate**
- We introduce the **bridge**
- We shall **refine the state**
- And also add **two new events**
- We are focusing on **FUN-3** (one way bridge)

# First Refinement: Introducing a one Way Bridge



# Introducing Three New Variables: $a$ , $b$ , and $c$

---



- $a$  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**

# Refining the State: Formalizing Variables $a$ , $b$ , and $c$

---

- Variables  $a$ ,  $b$ , and  $c$  denote **natural numbers**

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

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

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

# Refining the State: Introducing New Invariants

---

- Relating the **concrete state**  $(a, b, c)$  to the **abstract state**  $(n)$

$$a + b + c = n$$

- Formalizing the new invariant: **one way bridge**

$$a = 0 \quad \vee \quad c = 0$$

**constants:**  $d$

**variables:**  $a, b, c$

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

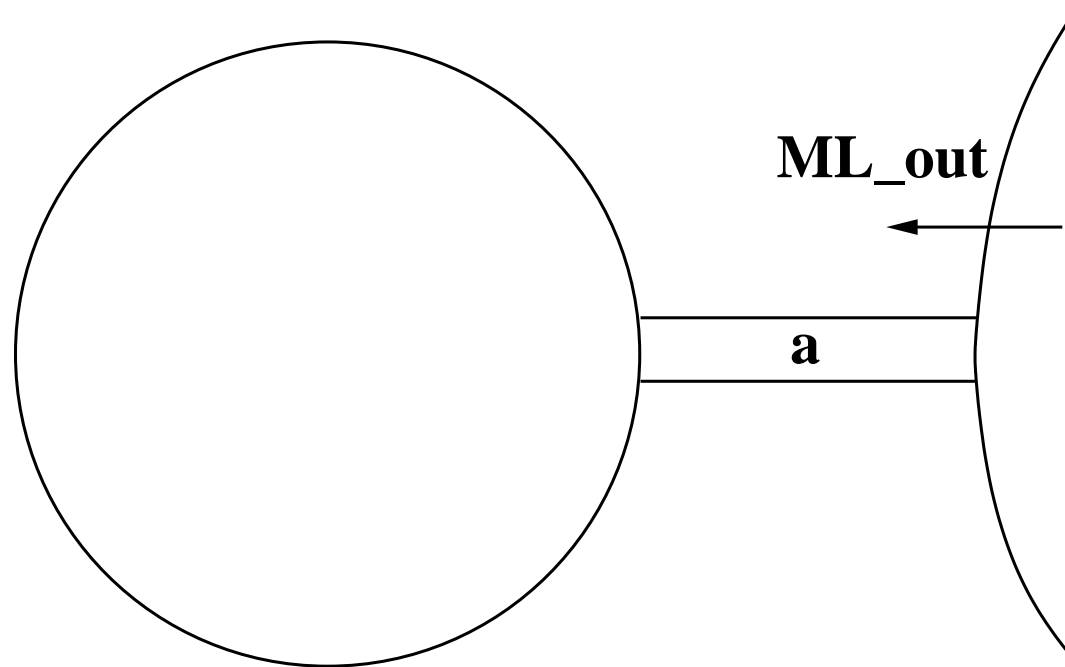
**inv1\_2:**  $b \in \mathbb{N}$

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

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

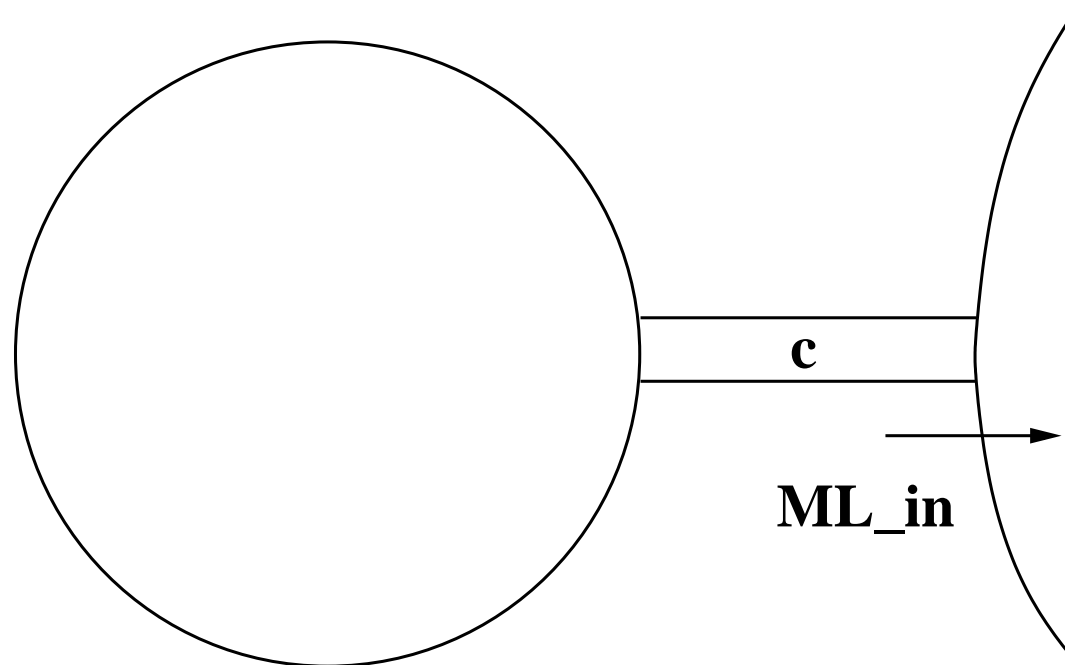
**inv1\_5:**  $a = 0 \vee c = 0$

- Invariants **inv1\_1** to **inv1\_5** are called the **concrete invariants**
- **inv1\_4** glues the **abstract state**,  $n$ , to the **concrete state**,  $a, b, c$



```
abstract_ML_out  
when  
   $n < d$   
then  
   $n := n + 1$   
end
```

```
concrete_ML_out  
when  
   $a + b < d \wedge c = 0$   
then  
   $a := a + 1$   
end
```



```
abstract_ML_in  
when  
   $0 < n$   
then  
   $n := n - 1$   
end
```

```
concrete_ML_in  
when  
   $0 < c$   
then  
   $c := c - 1$   
end
```

# Missing Variables are not modified

```
concrete_ML_out
when
   $a + b < d$ 
   $c = 0$ 
then
   $a := a + 1$ 
end
```

```
concrete_ML_in
when
   $0 < c$ 
then
   $c := c - 1$ 
end
```

Before-after predicates:

$$a' = a + 1 \quad \wedge \quad b' = b \quad \wedge \quad c' = c$$

$$a' = a \quad \wedge \quad b' = b \quad \wedge \quad c' = c - 1$$

Constants  $c$  with properties  $P(c)$

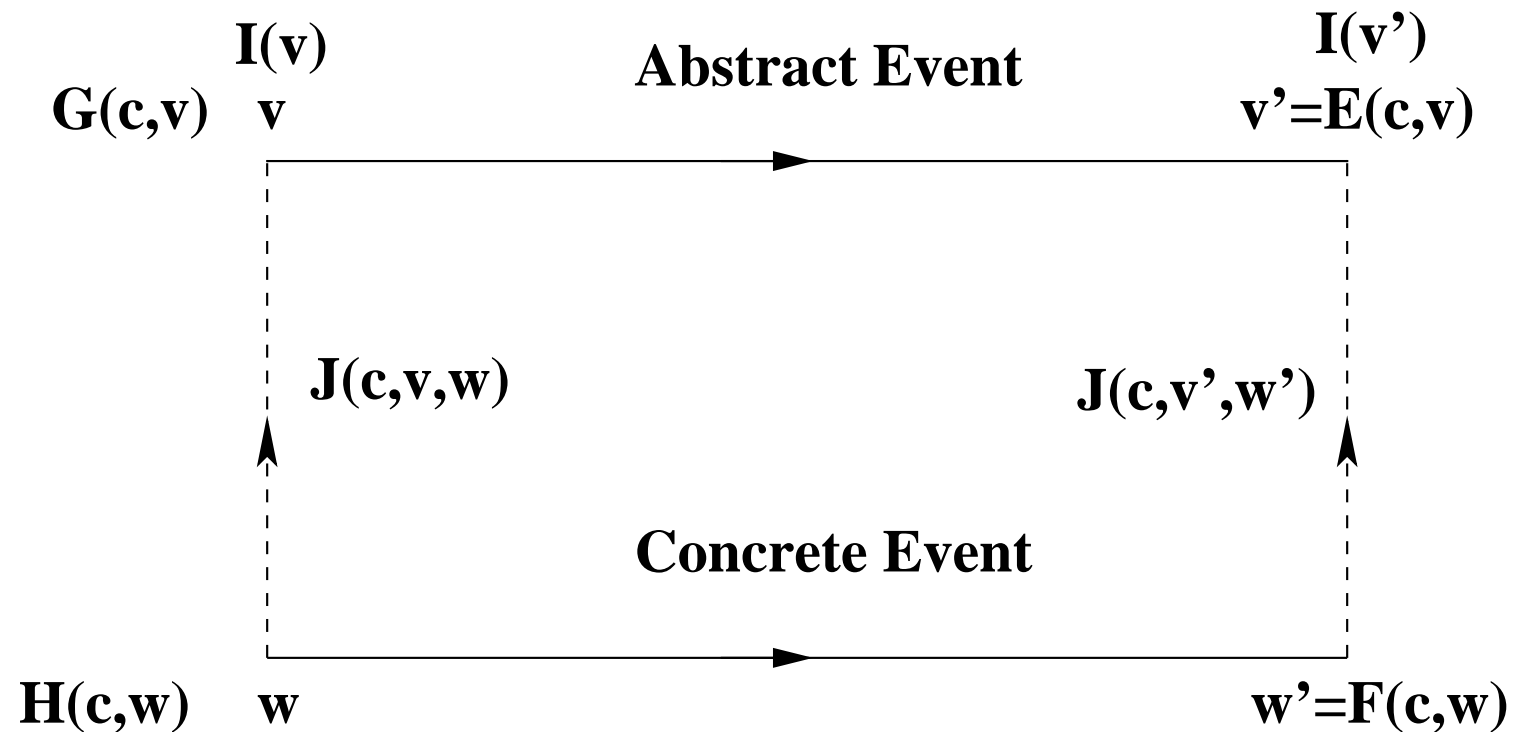
Abstract variables  $v$  with abstract invariant  $I(c, v)$

Concrete variables  $w$  with concrete invariant  $J(c, v, w)$

Abstract event with guard  $G(c, v)$  and b-a predicate  $v' = E(c, v)$

Concrete event with guard  $H(c, w)$  and b-a predicate  $w' = F(c, w)$

# Proving Correct Refinement: Invariant Refinement



Properties of Constants  
Abstract Invariant  
Concrete Invariant  
Concrete Guard

$\Rightarrow$

Abstract Guard

$P(c)$

$I(c, v)$

$J(c, v, w)$

$H(c, w)$

$\Rightarrow$

$G(c, v)$

GRD\_REF

Properties of Constants

Abstract Invariant

Concrete Invariant

Concrete Guard

$\Rightarrow$

Abstract Guard

$$d \in \mathbb{N}$$

$$0 < d$$

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

$$n \leq d$$

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

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

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

$$a + b + c = n$$

$$a = 0 \vee c = 0$$

$$a + b < d$$

$$c = 0$$

$\Rightarrow$

$$n < d$$

Properties of Constants

Abstract Invariant

Concrete Invariant

Concrete Guard

$\Rightarrow$

Abstract Guard

$$d \in \mathbb{N}$$

$$0 < d$$

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

$$n \leq d$$

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

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

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

$$a + b + c = n$$

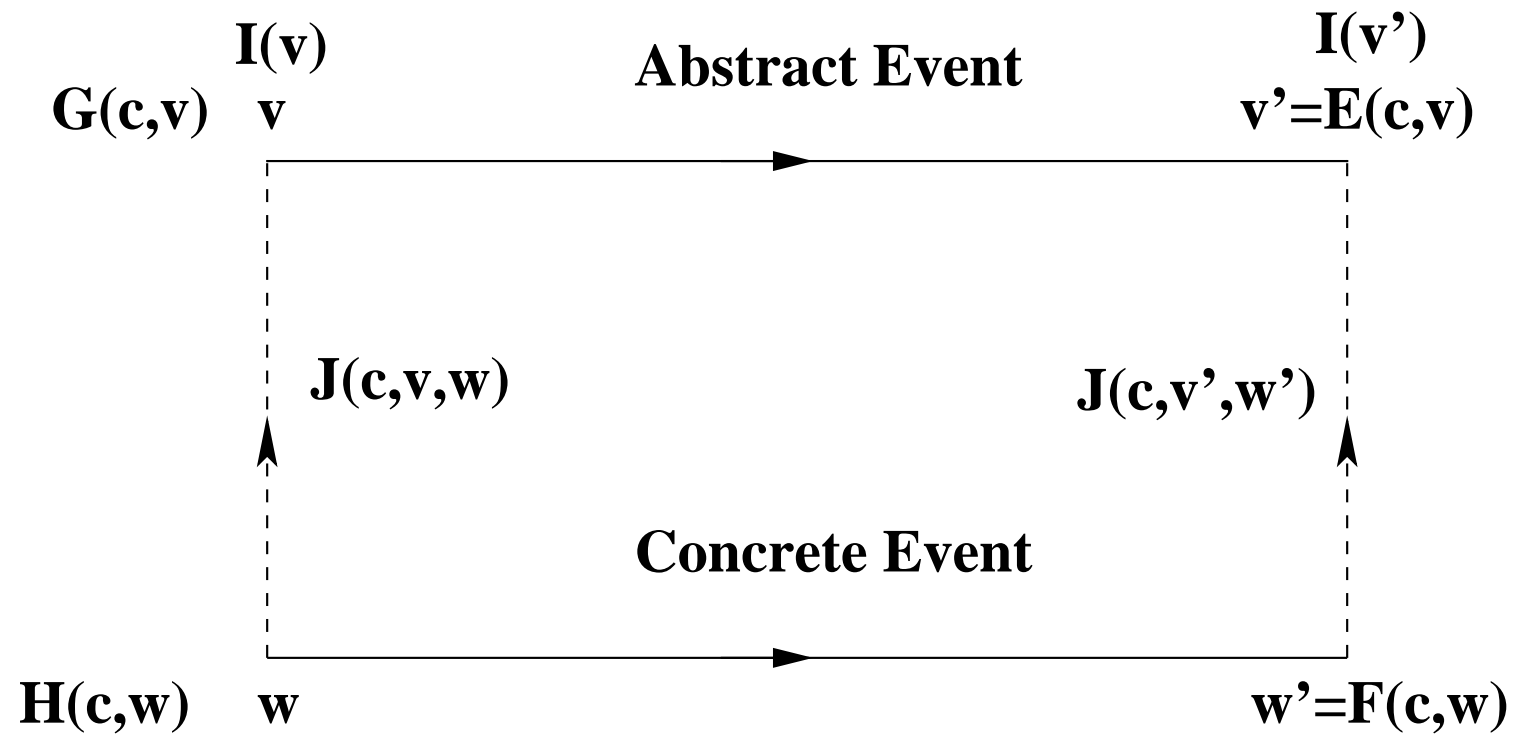
$$a = 0 \vee c = 0$$

$$0 < c$$

$\Rightarrow$

$$0 < n$$

# Proving Correct Refinement: Invariant Refinement



# Invariant Refinement Rule (Special Case)

<p>Properties of Constants Abstract Invariant Concrete Invariant Concrete Guard <math>\Rightarrow</math> New Concrete Invariant</p>	<p><math>P(c)</math> <math>I(c, v)</math> <math>J(c, v, w)</math> <math>H(c, w)</math> <math>\Rightarrow</math> <math>J(c, E(c, v), F(c, w))</math></p>	<p>INV_REF</p>
---	---	----------------

# Applying Invariant Refinement Rule to Event ML\_out

56

Properties of Constants

Abstract Invariants

Concrete Invariants

Concrete Guard

$\Rightarrow$

New Concrete Invariants

$$d \in \mathbb{N}$$

$$0 < d$$

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

$$n \leq d$$

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

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

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

$$a + b + c = n$$

$$a = 0 \vee c = 0$$

$$a + b < d$$

$$c = 0$$

$\Rightarrow$

$$a + 1 \in \mathbb{N}$$

$$a + 1 + b + c = n + 1$$

$$a + 1 = 0 \vee c = 0$$

Properties of Constants

Abstract Invariants

Concrete Invariants

Concrete Guard

$\Rightarrow$

New Concrete Invariants

$$d \in \mathbb{N}$$

$$0 < d$$

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

$$n \leq d$$

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

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

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

$$a + b + c = n$$

$$a = 0 \vee c = 0$$

$$0 < c$$

$\Rightarrow$

$$c - 1 \in \mathbb{N}$$

$$a + b + c - 1 = n - 1$$

$$a = 0 \vee c - 1 = 0$$

- Concrete initialization

$$\text{init}$$
$$a, b, c := 0, 0, 0$$

- Corresponding after predicate

$$a' = 0 \quad \wedge \quad b' = 0 \quad \wedge \quad c' = 0$$

# Refinement Rule for Initialization

Constants  $c$  with properties  $P(c)$

Concrete invariant  $J(c, v, w)$

Abstract initialization with after predicate  $v' = K(c)$

Concrete initialization with after predicate  $w' = L(c)$

Properties of the constants

$\Rightarrow$

Modified concrete invariants

$P(c)$

$\Rightarrow$

$J(c, K(c), L(c))$

INI\_INV\_REF

Properties of the constants

$\Rightarrow$

Modified concrete invariant **inv1\_4**  
 $(a + b + c = n)$

$$d \in \mathbb{N}$$

$$d > 0$$

$\Rightarrow$

$$0 + 0 + 0 = 0$$

Properties of the constants

$\Rightarrow$

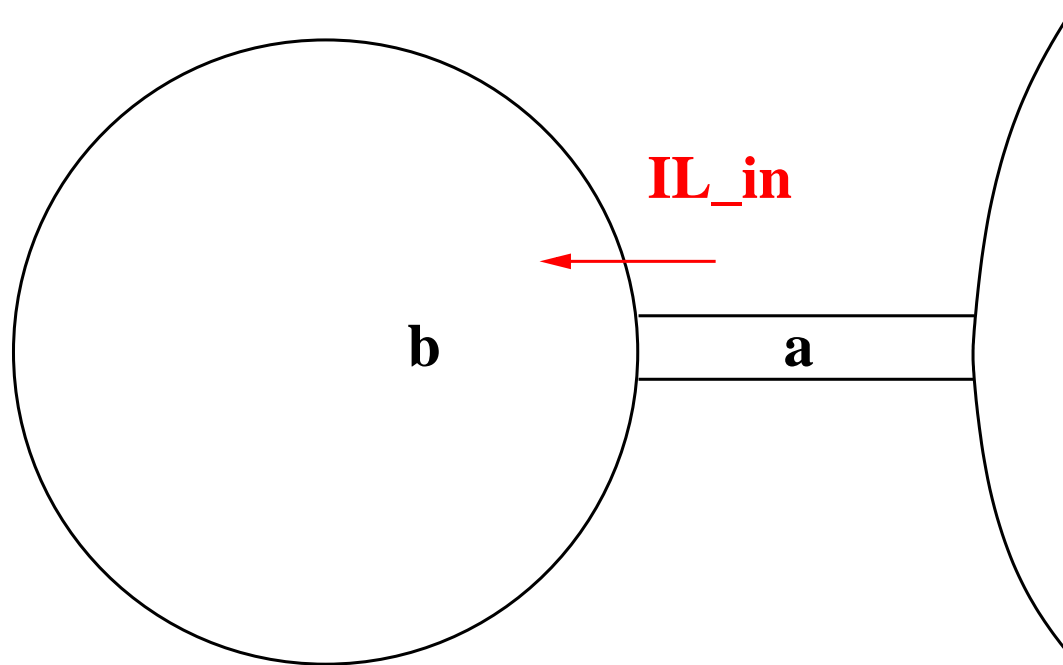
Modified concrete invariant **inv1\_5**  
 $(a = 0 \vee c = 0)$

$$d \in \mathbb{N}$$

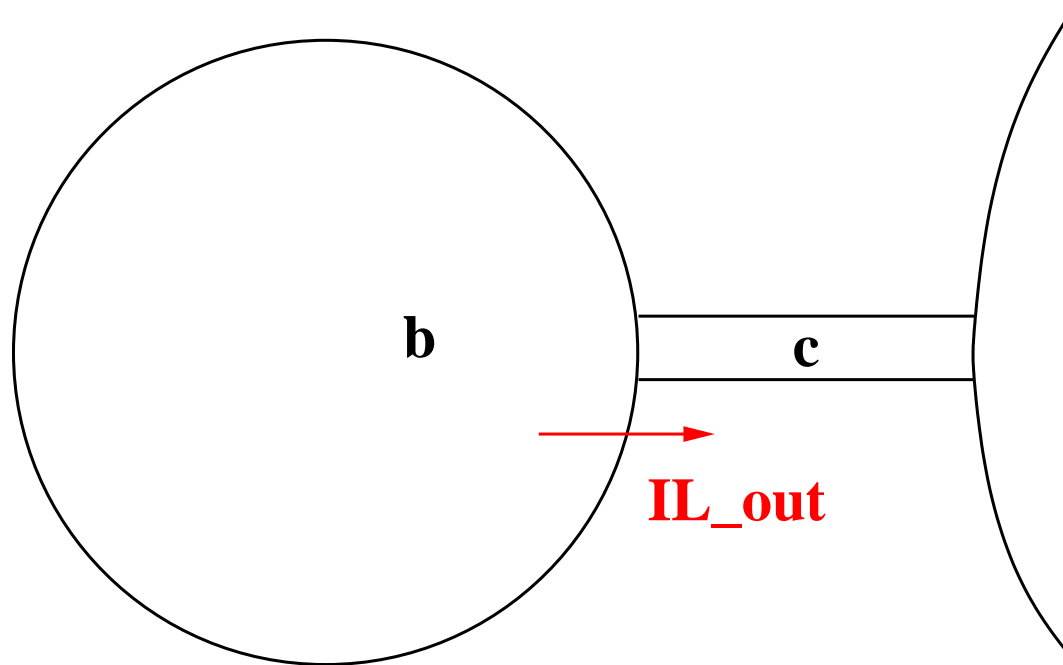
$$d > 0$$

$\Rightarrow$

$$0 = 0 \vee 0 = 0$$



```
IL_in  
  when  
     $0 < a$   
  then  
     $a, b := a - 1, b + 1$   
  end
```



```
IL_out
when
  when
     $0 < b \wedge a = 0$ 
  then
     $b, c := b - 1, c + 1$ 
  end
```

# Several Actions Done Together

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

```
IL_out
  when
     $0 < b$ 
     $a = 0$ 
  then
     $b := b - 1$ 
     $c := c + 1$ 
  end
```

Before-after predicates

$$a' = a + 1 \wedge b' = b + 1 \wedge c' = c$$

$$a' = a \wedge b' = b - 1 \wedge c' = c + 1$$

# The empty assignment **skip**

---

The before-after predicate of **skip** in the **initial model**

$$n' = n$$

The before-after predicate of **skip** in the **first refinement**

$$a' = a \quad \wedge \quad b' = b \quad \wedge \quad c' = c$$

- (1) A new event must **refine an implicit event**, made of a **skip action**
  
- (2) The new events **must not diverge**
  - For this one has to exhibit a **variant**
  - The variant is a **natural number** (could be more complicated)
  - Each new event must **decrease this variant**

Properties of the constants

Abstract invariants

Concrete invariants

Concrete guards

$\Rightarrow$

Modified invariant **inv1\_4**

$$d \in \mathbb{N}$$

$$0 < d$$

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

$$n \leq d$$

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

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

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

$$a + b + c = n$$

$$a = 0 \vee c = 0$$

$$0 < a$$

$\Rightarrow$

$$a - 1 + b + 1 + c = n$$

Properties of the constants

Abstract invariants

Concrete invariants

Concrete guards

$\Rightarrow$

Modified invariant **inv1\_5**

$$d \in \mathbb{N}$$

$$0 < d$$

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

$$n \leq d$$

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

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

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

$$a + b + c = n$$

$$a = 0 \vee c = 0$$

$$0 < a$$

$\Rightarrow$

$$a - 1 = 0 \vee c = 0$$

# Non-divergence of New Events (1)

Properties  $P(c)$ , invariants  $I(c, v)$ , concrete invariant  $J(c, v, w)$

New event with guard  $H(c, w)$

Variant  $V(c, w)$

Properties of the constants  
Abstract invariants  
Concrete invariants  
Concrete guard of a new event

$\Rightarrow$

Variant  $\in \mathbb{N}$

$P(c)$   
 $I(c, v)$   
 $J(c, v, w)$   
 $H(c, w)$

$\Rightarrow$

$V(c, w) \in \mathbb{N}$

WFD\_REF1

## Non-divergence of New Events (2)

Properties  $P(c)$ , invariants  $I(c, v)$ , concrete invariant  $J(c, v, w)$

New event with guard  $H(c, w)$  and b-a predicate  $w' = F(c, w)$

Variant  $V(c, w)$

Properties of cons.  
Abstract invariant  
Concrete invariant  
Concrete guard

$\Rightarrow$

Modified Var.  $<$  Var.

$P(c)$

$I(c, v)$

$J(c, v, w)$

$H(c, w)$

$\Rightarrow$

$V(c, F(c, w)) < V(c, w)$

WFD\_REF2

**variant\_1:**  $2 * a + b$

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

$$d \in \mathbb{N}$$

$$0 < d$$

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

$$n \leq d$$

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

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

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

$$a + b + c = n$$

$$a = 0 \vee c = 0$$

$$0 < a$$

$\Rightarrow$

$$2 * (a - 1) + b + 1 < 2 * a + b$$

# Decreasing of the Variant by Event IL\_out

```
IL_out
  when
    0 < b
    a = 0
  then
    b := b - 1
    c := c + 1
  end
```

$$d \in \mathbb{N}$$

$$0 < d$$

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

$$n \leq d$$

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

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

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

$$a + b + c = n$$

$$a = 0 \vee c = 0$$

$$0 < b$$

$$a = 0$$

$\Rightarrow$

$$2 * a + (b - 1) < 2 * a + b$$

# The Last Rule: No More Deadlock in Concrete Space

73

The  $G_i(c, v)$  are the abstract guards

The  $H_i(c, v)$  are the concrete guards

We have to prove the following Deadlock Freeness Rule

$\begin{array}{l} P(c) \\ I(c, v) \\ J(c, v, w) \\ G_1(c, v) \vee \dots \vee G_m(c, v) \\ \Rightarrow \\ H_1(c, w) \vee \dots \vee H_n(c, w) \end{array}$	$\text{DLF\_REF}$
---	-------------------

# Applying Deadlock Freeness Rule

$$d \in \mathbb{N}$$

$$0 < d$$

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

$$n \leq d$$

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

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

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

$$a + b + c = n$$

$$a = 0 \vee c = 0$$

$$\Rightarrow$$

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

$$c > 0 \vee$$

$$a > 0 \vee$$

$$(b > 0 \wedge a = 0)$$

$$\rightsquigarrow$$

$$d \in \mathbb{N}$$

$$0 < d$$

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

$$\Rightarrow$$

$$b < d \vee b > 0$$

- For old events:
  - Strengthening of guards: rule **GRD\_REF**
  - Concrete invariant preservation: rule **INV\_REF**
  
- For new events:
  - Refining the implicit event with a skip action
  - Absence of divergence: rules **WFD\_REF1** and **WFD\_REF2**
  
- For all events:
  - Relative deadlock freeness: rule **DLF\_REF**

<p>Properties of the constants Abstract invariants Concrete invariants Concrete guards ⇒ Abstract guards</p>	<p>GRD_REF</p>
--	----------------

<p>Properties of the constants Abstract invariant Concrete invariant Concrete guard <math>\Rightarrow</math> Modified concrete invariant</p>	<p>INV_REF</p>
--	----------------

# Refinement Rules (3)

---

<p>Properties of the constants <math>\Rightarrow</math> Modified concrete invariants</p>	<p>INI_INV_REF</p>
--	--------------------

<p>Properties of the constants Abstract invariants Concrete invariants Concrete guard of a new event <math>\Rightarrow</math> Variant <math>\in \mathbb{N}</math></p>	<p>WFD_REF1</p>
---	-----------------

# Refinement Rules (5)

---

Properties of the constants

Abstract invariants

Concrete invariants

Concrete guard of a new event

⇒

Modified variant < Variant

WFD\_REF2

Properties of the constants Abstract invariants Concrete invariants Disjunction of abstract guards $\Rightarrow$ Disjunction of concrete guards	DLF_REF
--	---------

**constants:**  $d$

**variables:**  $a, b, c$

**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:**  $a = 0 \vee c = 0$

**variant1:**  $2 * a + b$

init

$a := 0$

$b := 0$

$c := 0$

ML\_in

**when**

$0 < c$

**then**

$c := c - 1$

**end**

ML\_out

**when**

$a + b < d$

$c = 0$

**then**

$a := a + 1$

**end**

IL\_in

**when**

$0 < a$

**then**

$a := a - 1$

$b := b + 1$

**end**

IL\_out

**when**

$0 < b$

$a = 0$

**then**

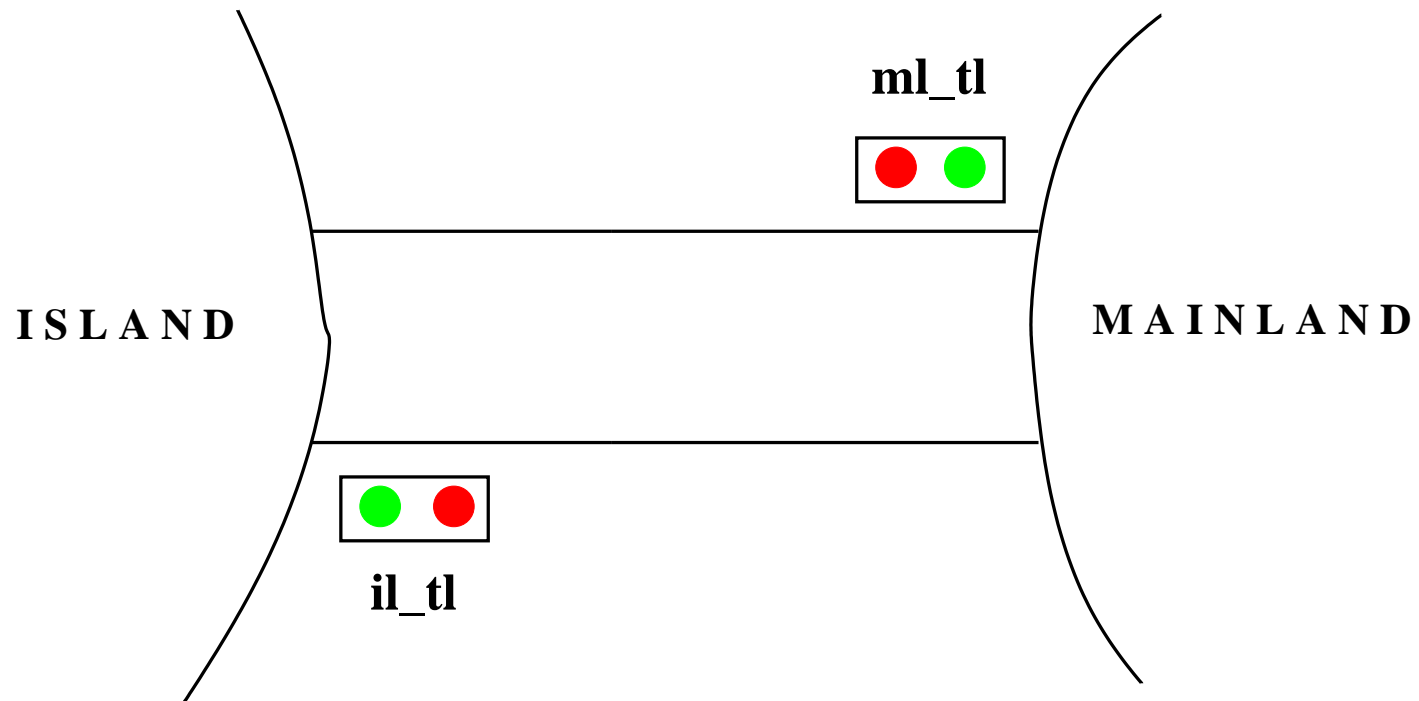
$b := b - 1$

$c := c + 1$

**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)

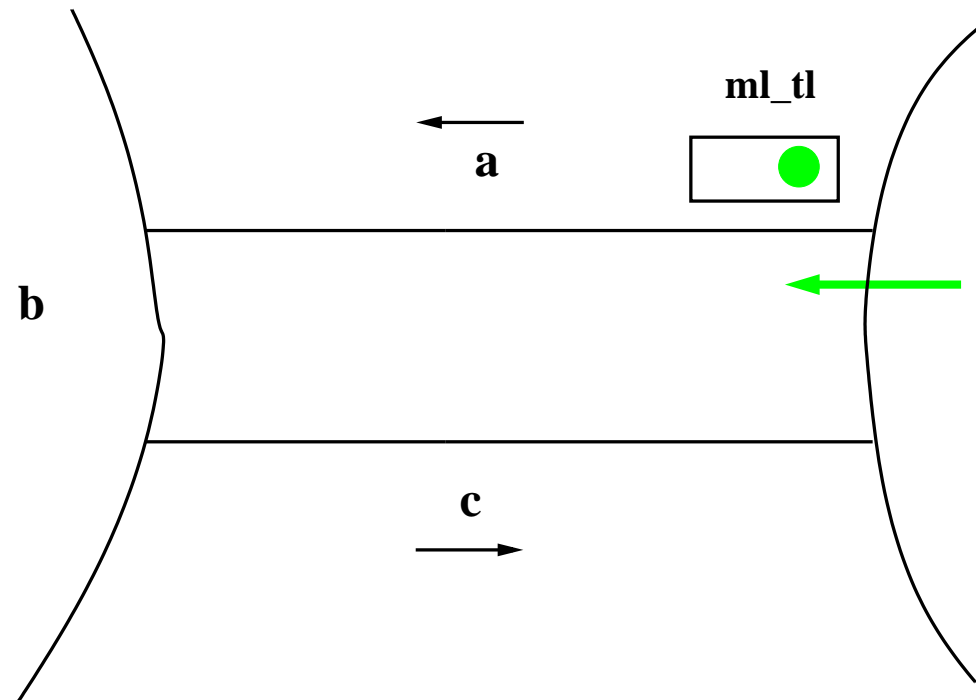
# Second Refinement: Introducing Traffic Lights



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

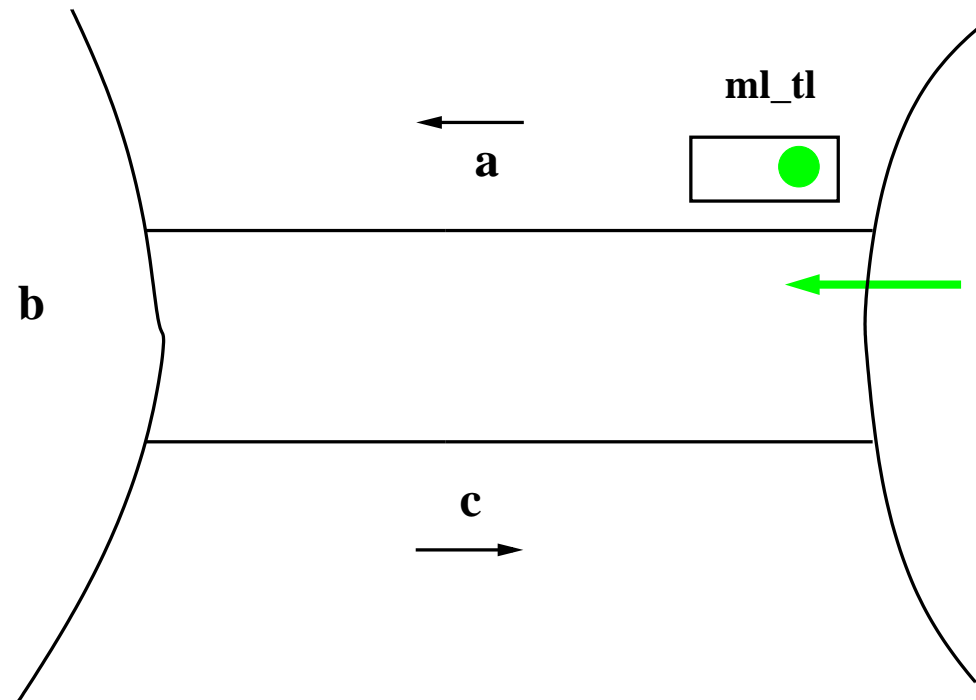
$ml\_tl \in \{\text{green, red}\}$

# Extending the Invariant (1)



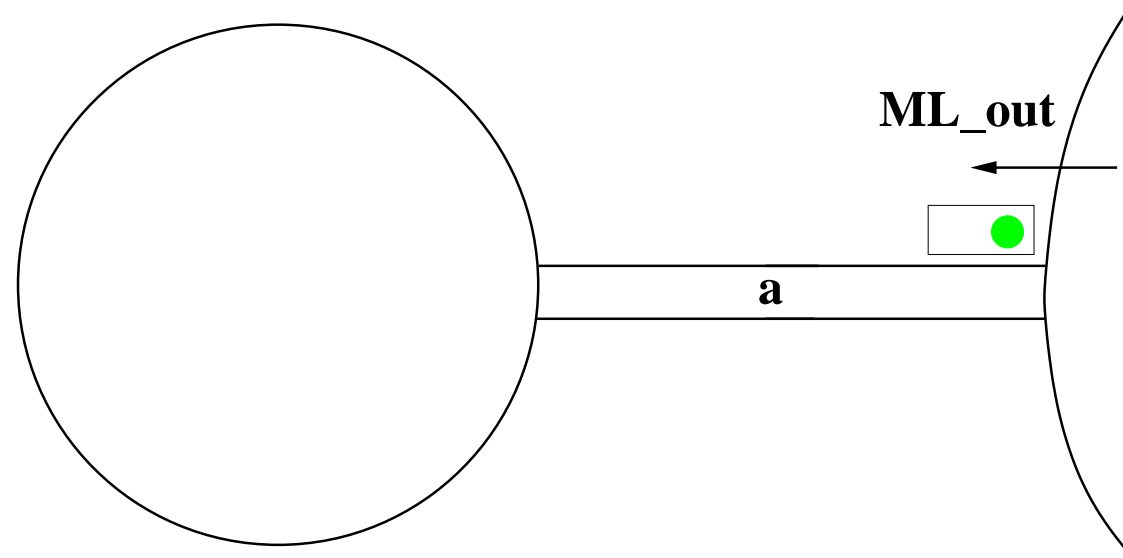
- A green **mainland traffic light** implies **safe access** to the bridge

# Extending the Invariant (1)



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

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



**abstract** ML\_out

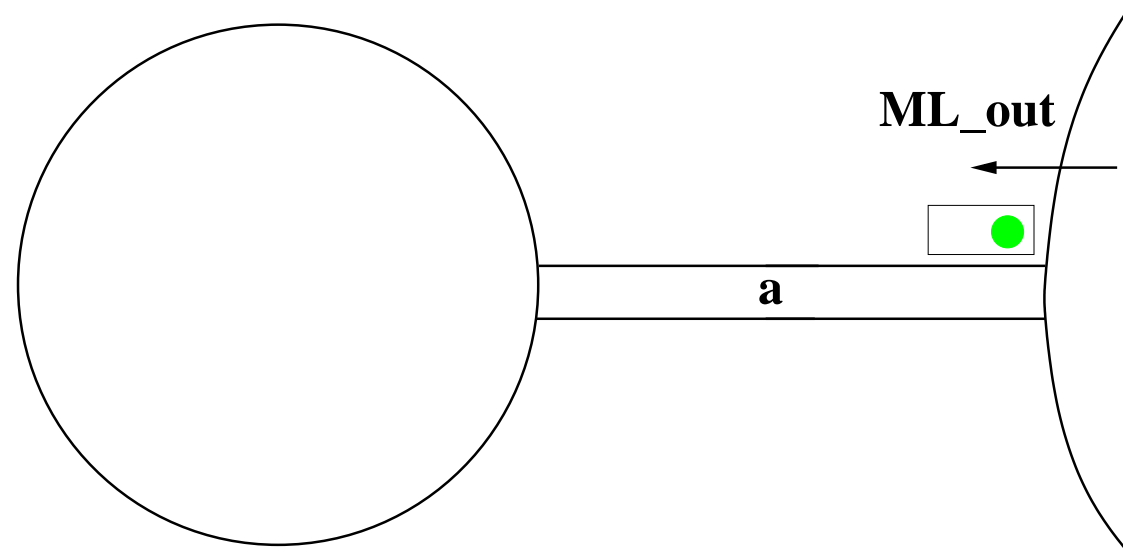
**when**

$$c = 0 \wedge a + b < d$$

**then**

$$a := a + 1$$

**end**



**abstract**\_ML\_out

**when**

$$c = 0 \wedge a + b < d$$

**then**

$$a := a + 1$$

**end**

**concrete**\_ML\_out

**when**

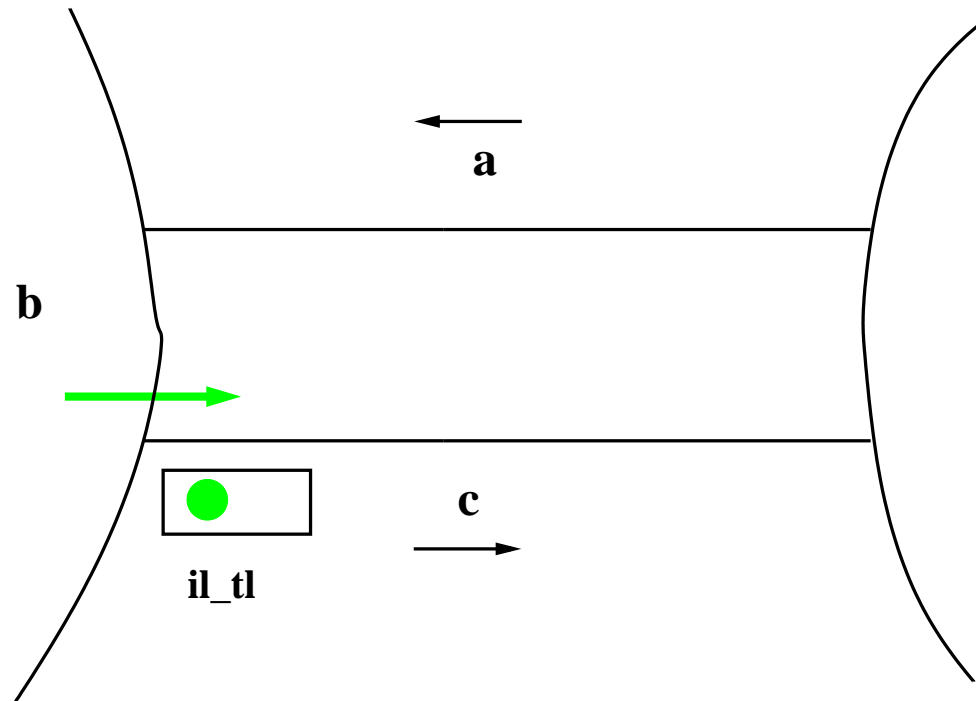
$$ml\_tl = \text{green}$$

**then**

$$a := a + 1$$

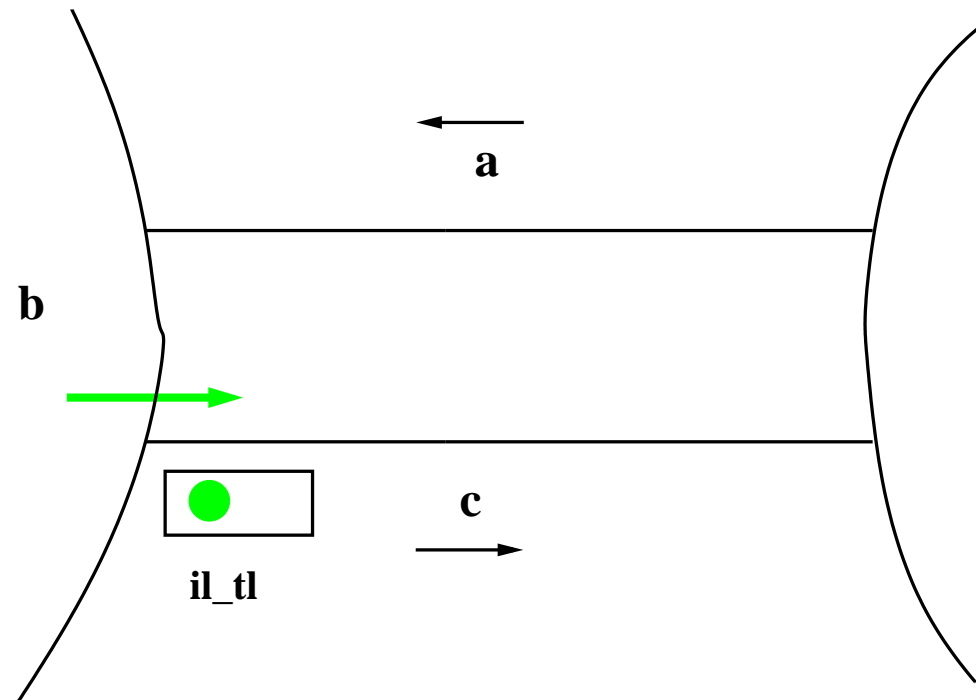
**end**

# Extending the Invariant (2)



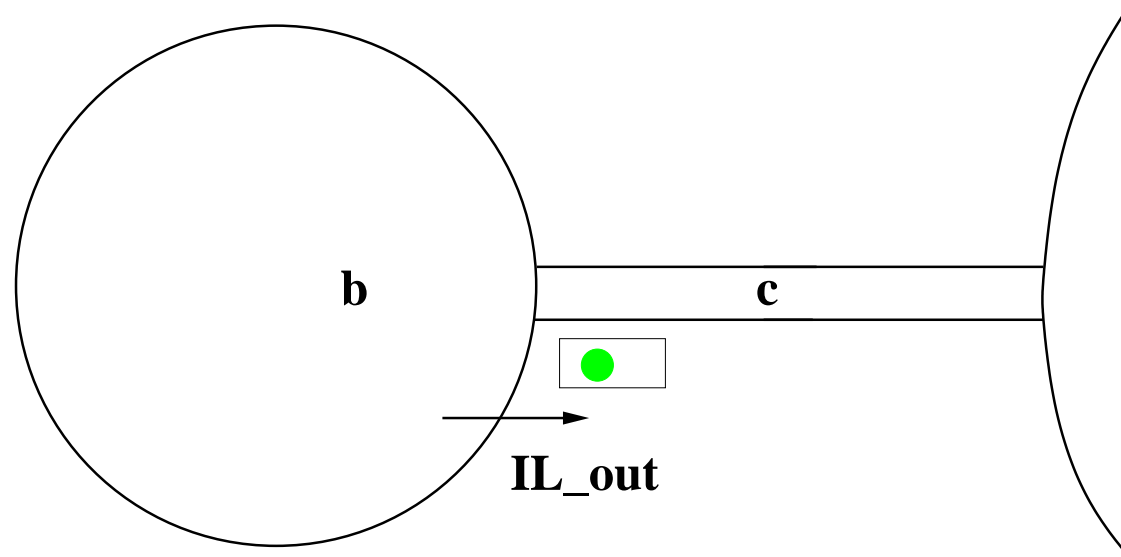
- A green **island traffic light** implies **safe access** to the bridge

## Extending the Invariant (2)



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

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



**abstract** IL\_out

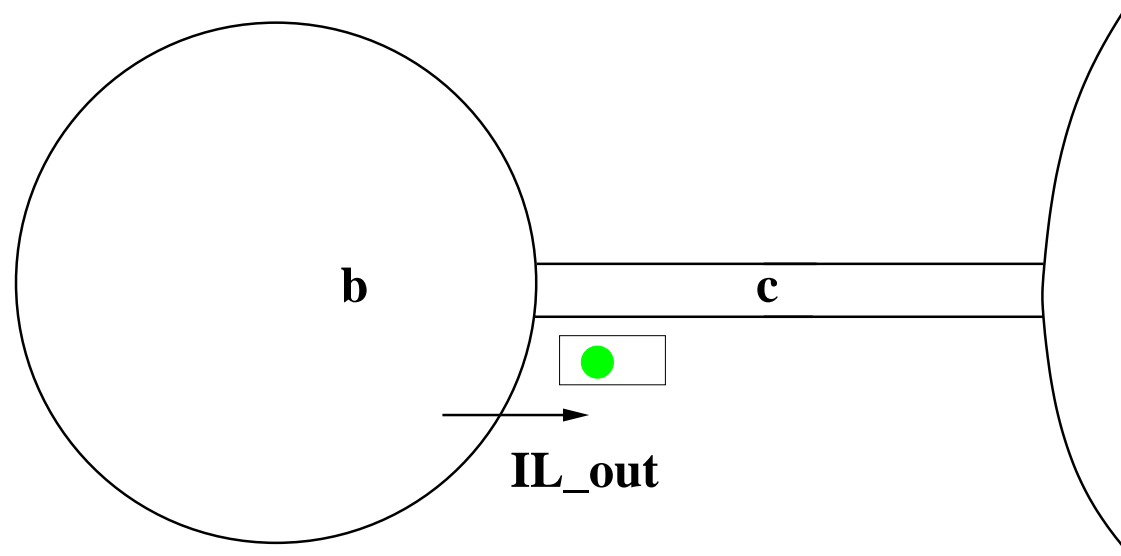
**when**

$a = 0 \wedge 0 < b$

**then**

$b, c := b - 1, c + 1$

**end**



**abstract** IL\_out

**when**

$a = 0 \wedge 0 < b$

**then**

$b, c := b - 1, c + 1$

**end**

**concrete** IL\_out

**when**

$il\_tl = \text{green}$

**then**

$b, c := b - 1, c + 1$

**end**

# New Events **ML\_tl\_green** and **IL\_tl\_green**

ML\_tl\_green

**when**

$ml\_tl = \text{red}$

$c = 0 \wedge a + b < d$

**then**

$ml\_tl := \text{green}$

**end**

IL\_tl\_green

**when**

$il\_tl = \text{red}$

$a = 0 \wedge 0 < b$

**then**

$il\_tl := \text{green}$

**end**

- Turning lights to **green** when **proper conditions hold**

**constants:**  $d$

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

**inv2\_1:**  $ml\_tl \in \{\text{red}, \text{green}\}$

**inv2\_2:**  $il\_tl \in \{\text{red}, \text{green}\}$

**inv2\_3:**  $ml\_tl = \text{green} \Rightarrow a + b < d \wedge c = 0$

**inv2\_4:**  $il\_tl = \text{green} \Rightarrow 0 < b \wedge a = 0$

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

- Variables  $a$ ,  $b$ , and  $c$  were present in the previous refinement
- This is called superposition
- However the abstract and concrete spaces must be disjoint
- We have thus to adapt rule INV\_REF

- Constant  $c$  and properties  $P(c)$
- Abstract variables  $u$  and  $v$  and abstract invariants  $I(c, u, v)$
- Concrete variables  $v$  and  $w$  and concrete invariants  $J(c, v, w)$
- Abstract b-a predicate  $u' = E(c, u, v)$  and  $v' = M(c, u, v)$
- Concrete guard  $H(c, v, w)$
- Concrete b-a predicate  $v' = N(c, v, w)$  and  $w' = F(c, v, w)$

# Superposition: Change of Variable

- We make a change of variable:  $v \rightsquigarrow v1$
- We add a concrete invariant:  $v1 = v$ . Rule INV\_REF becomes:

Properties of constants	$P(c)$
Abstract invariants	$I(c, u, v)$
Concrete invariants	$J(c, u, v1, w)$
	$v1 = v$
Concrete guards	$H(c, v1, w)$
$\Rightarrow$	$\Rightarrow$
Mod. conc inv. $J(c, u, v1, w)$	$J(c, E(c, u, v), M(c, u, v),$ $F(c, v1, w))$
Mod. conc. inv. $v1 = v$	$M(c, u, v) = N(c, v1, w)$

Abstract\_Event

**when**

$G(c, u, v)$

**then**

$u := E(c, u, v)$

$v := M(c, u, v)$

**end**

Abstract\_Event

**when**

$H(c, v, w)$

**then**

$v := N(c, v, w)$

$w := F(c, v, w)$

**end**

# Superposition: Introducing a New Rule

- We apply the equality  $v1 = v$
- The previous rule can be decomposed: INV\_REF and **EQL\_REF**
- **Equality of actions on the common variables**

<p>Properties of cons. Abstract invariant Concrete invariant Concrete guards ⇒ Same actions on common variables</p>	<p><math>P(c)</math> <math>I(c, u, v)</math> <math>J(c, u, v, w)</math> <math>H(c, v, w)</math> ⇒ <math>M(c, u, v) = N(c, v, w)</math></p>	<p>EQL_REF</p>
---	--	----------------

- It is trivial since the **actions are the same**

```
(abstract_)ML_out
when
   $c = 0 \wedge a + b < d$ 
then
   $a := a + 1$ 
end
```

```
(concrete_)ML_out
when
   $ml\_tl = \text{green}$ 
then
   $a := a + 1$ 
end
```

- Same thing for refinement of event IL\_out

$$\begin{aligned} & a \in \mathbb{N} \\ & il\_tl = \mathbf{green} \Rightarrow a = 0 \wedge 0 < b \\ & ml\_tl = \mathbf{green} \\ \Rightarrow & \\ & il\_tl = \mathbf{green} \Rightarrow a + 1 = 0 \wedge 0 < b \end{aligned}$$

It cannot be proved when  $il\_tl = \mathbf{green}$  since it reduces to

$$\begin{aligned} & a \in \mathbb{N} \\ & a = 0 \\ \Rightarrow & \\ & a + 1 = 0 \end{aligned}$$

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

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

$$il\_tl = \mathbf{green}$$

$$\Rightarrow$$

$$ml\_tl = \mathbf{green} \Rightarrow c + 1 = 0 \wedge a + b - 1 < d$$

It cannot be proved when  $ml\_tl = \mathbf{green}$  since it reduces to

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

$$c = 0$$

$$\Rightarrow$$

$$c + 1 = 0$$

Proof failures shows that **both lights cannot be green at the same time**

An obvious fact that we had forgotten to state

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

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

This could have been deduced from these requirements

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

FUN-3

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

EQP-3

But the new invariant **inv2\_5** is not preserved by the new events

Unless we correct them as follows:

```
ML_tl_green
  when
    ml_tl = red
    a + b < d
    c = 0
  then
    ml_tl := green
    il_tl := red
  end
```

```
IL_tl_green
  when
    il_tl = red
    0 < b
    a = 0
  then
    il_tl := green
    ml_tl := red
  end
```

$$\begin{array}{l} ml\_tl = \text{green} \Rightarrow c = 0 \wedge a + b < d \\ ml\_tl = \text{green} \\ \Rightarrow \\ ml\_tl = \text{green} \Rightarrow c = 0 \wedge a + 1 + b < d \end{array}$$

It cannot be proved when  $a + 1 + b = d$  unless  $il\_tl$  becomes equal to red

# Solution: Splitting Event ML\_out

---

ML\_out\_1

**when**

$ml\_tl = \text{green}$

$a + b + 1 \neq d$

**then**

$a := a + 1$

**end**

ML\_out\_2

**when**

$ml\_tl = \text{green}$

$a + b + 1 = d$

**then**

$a := a + 1$

$ml\_tl := \text{red}$

**end**

$$\begin{aligned} & il\_tl = \text{green} \Rightarrow a = 0 \wedge 0 < b \\ & il\_tl = \text{green} \\ \Rightarrow & \\ & il\_tl = \text{green} \Rightarrow a = 0 \wedge 0 < b - 1 \end{aligned}$$

It cannot be proved when  $b = 1$  unless  $il\_tl$  becomes equal to red

# Solution: Splitting Event IL\_out

---

```
IL_out_1
  when
    il_tl = green
    b ≠ 1
  then
    b := b - 1
    c := c + 1
  end
```

```
IL_out_2
  when
    il_tl = green
    b = 1
  then
    b := b - 1
    c := c + 1
    il_tl := red
  end
```

# Summary of the Proof Situation

---

- Correct event refinement: **OK**
- Absence of divergence of new events: **FAILURE**
- Absence of deadlock: **?**

ML\_tl\_green

**when**

$ml\_tl = red$

$a + b < d$

$c = 0$

**then**

$ml\_tl := green$

$il\_tl := red$

**end**

IL\_tl\_green

**when**

$il\_tl = red$

$0 < b$

$a = 0$

**then**

$il\_tl := green$

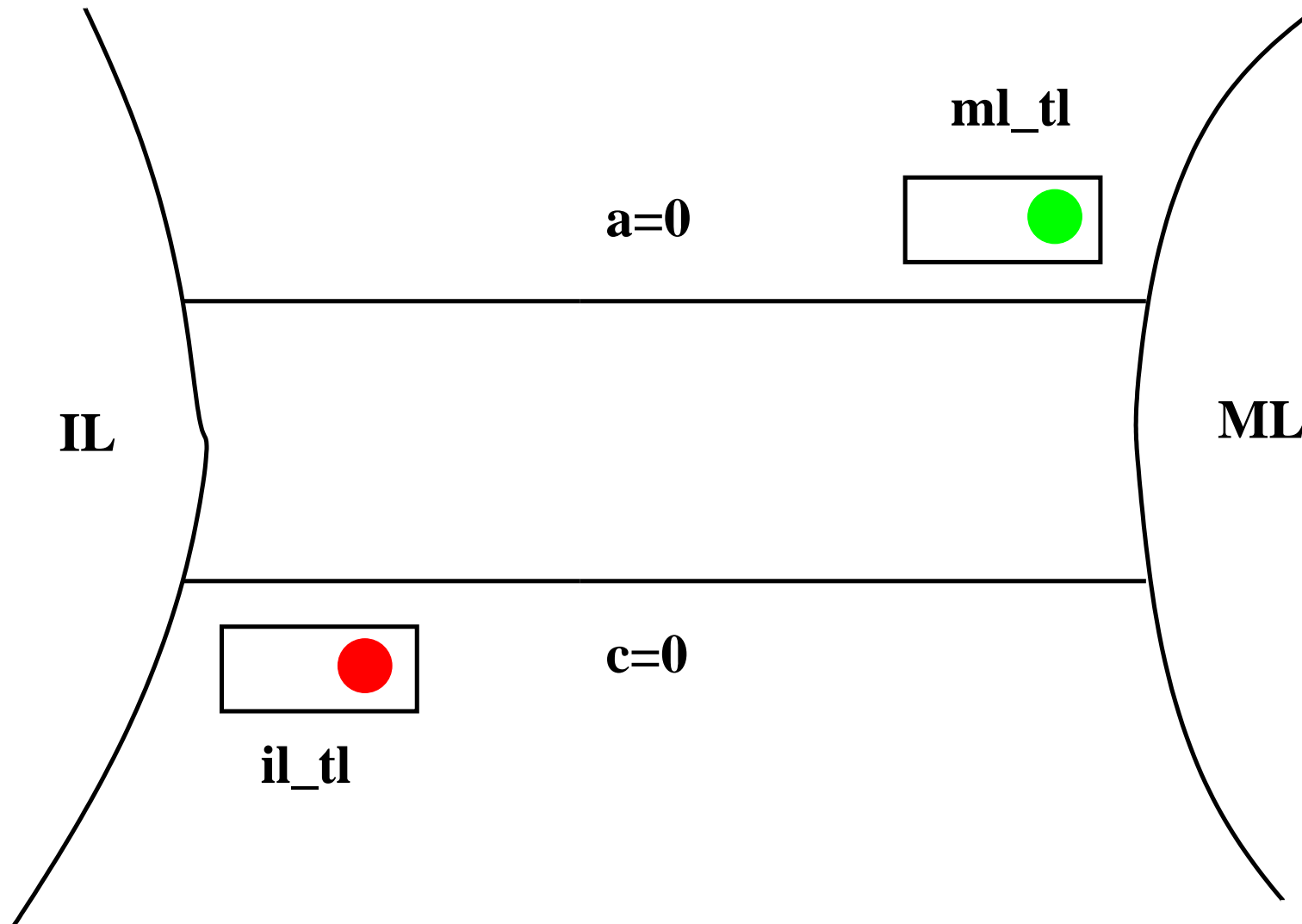
$ml\_tl := red$

**end**

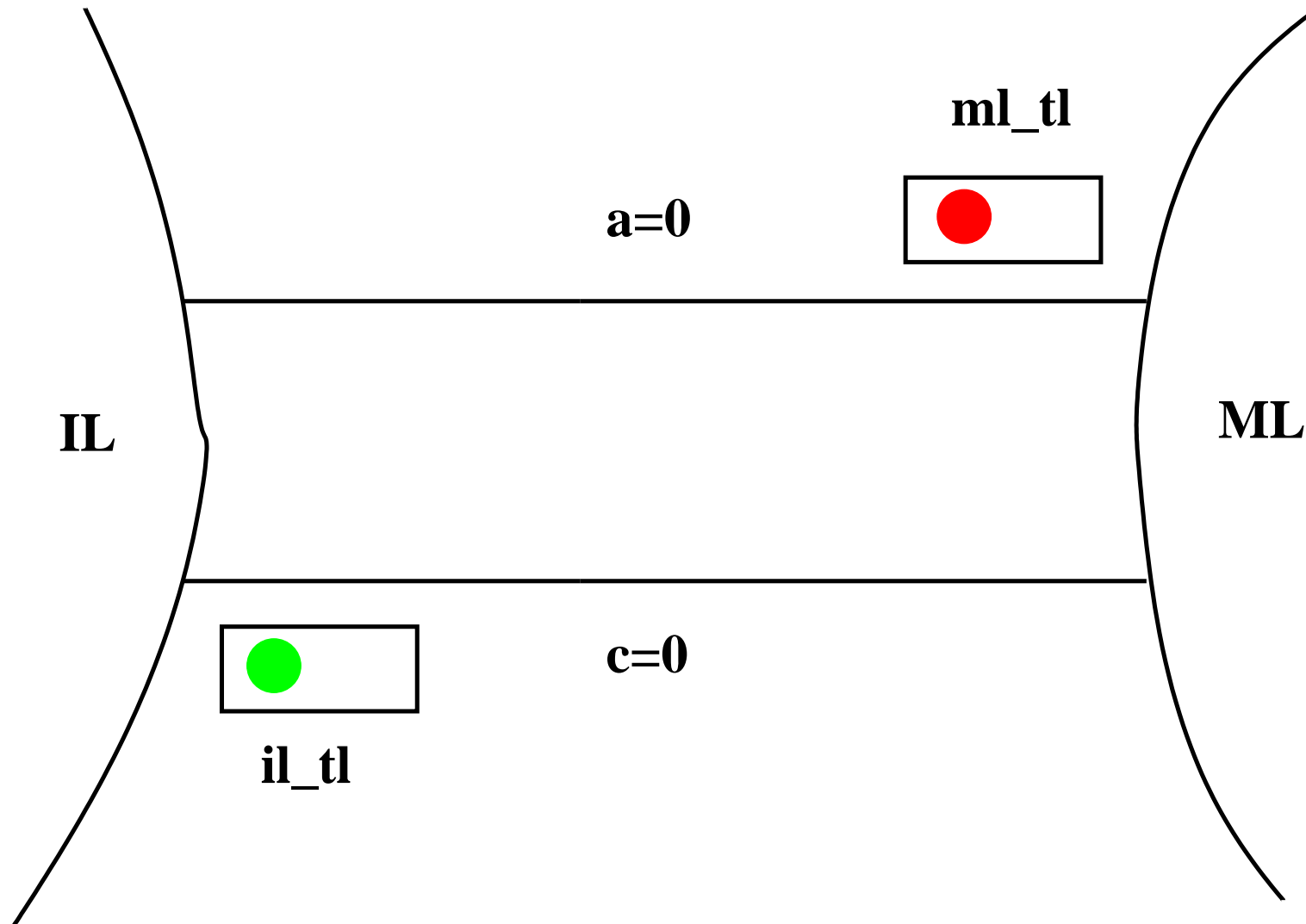
When  $a$  and  $c$  are both equal to 0 and  $b$  is positive, then both events are always alternatively enabled

The lights can change colors very rapidly

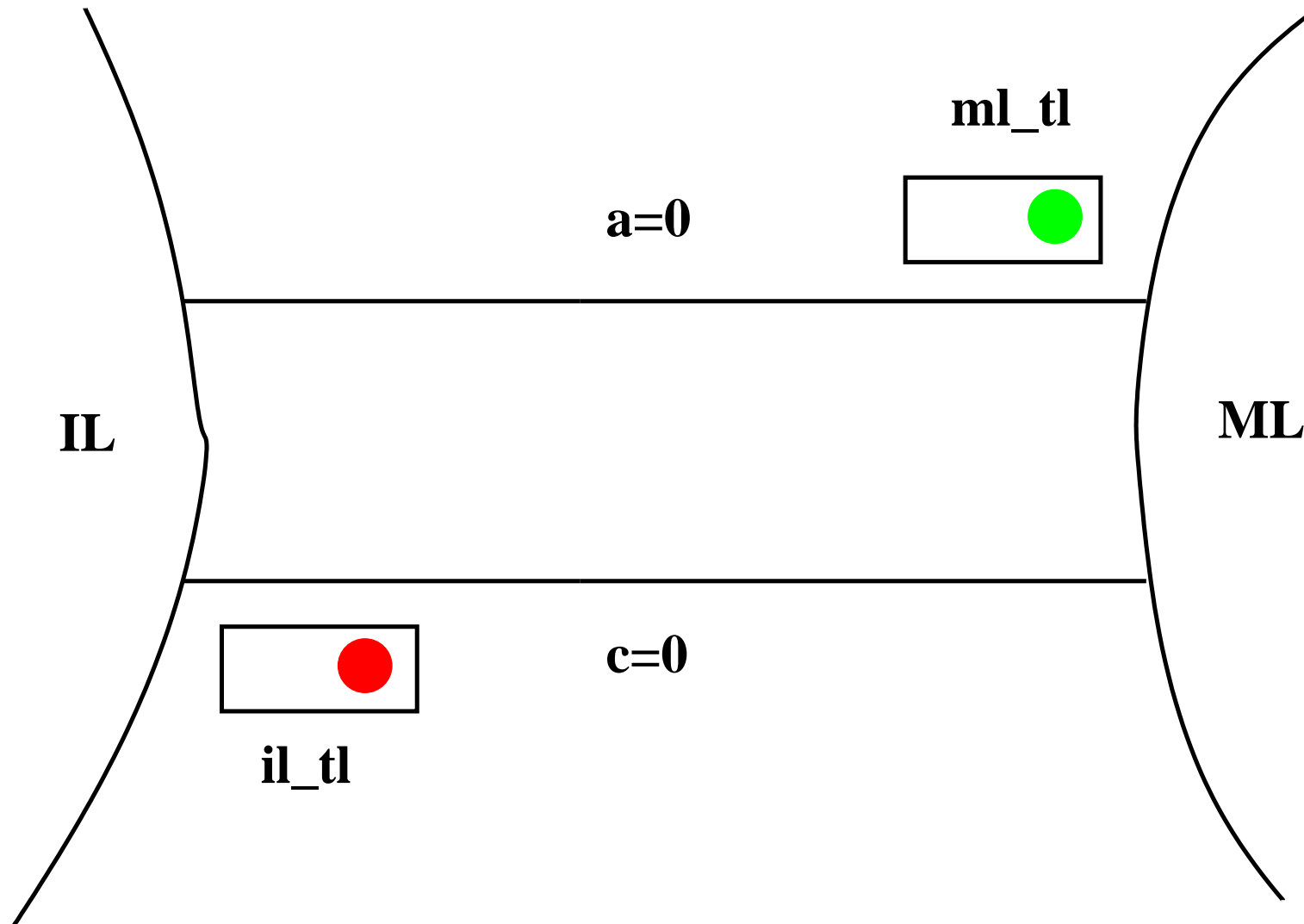
# ML\_tl\_green and IL\_tl\_green can run for ever



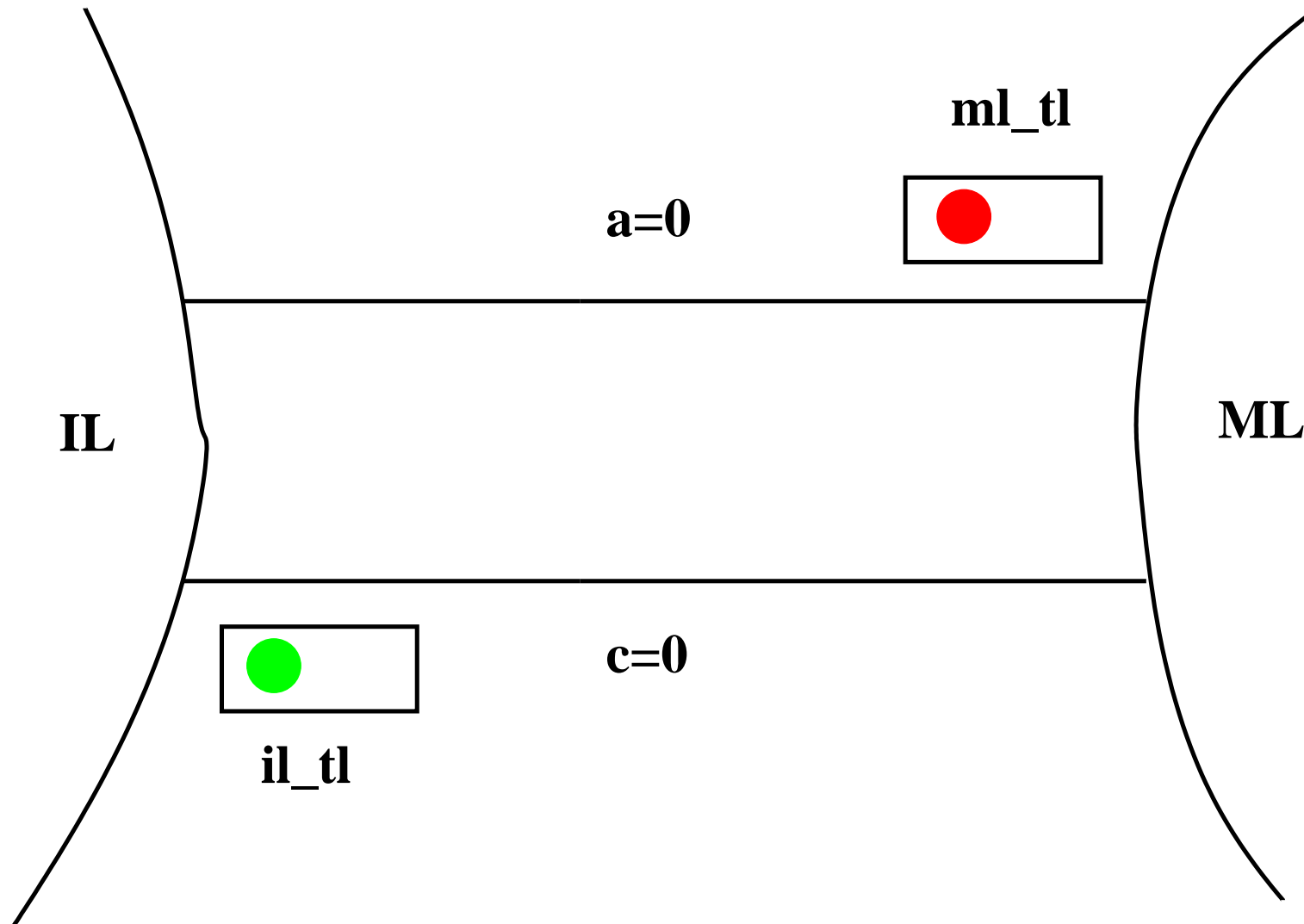
# ML\_tl\_green and IL\_tl\_green can run for ever



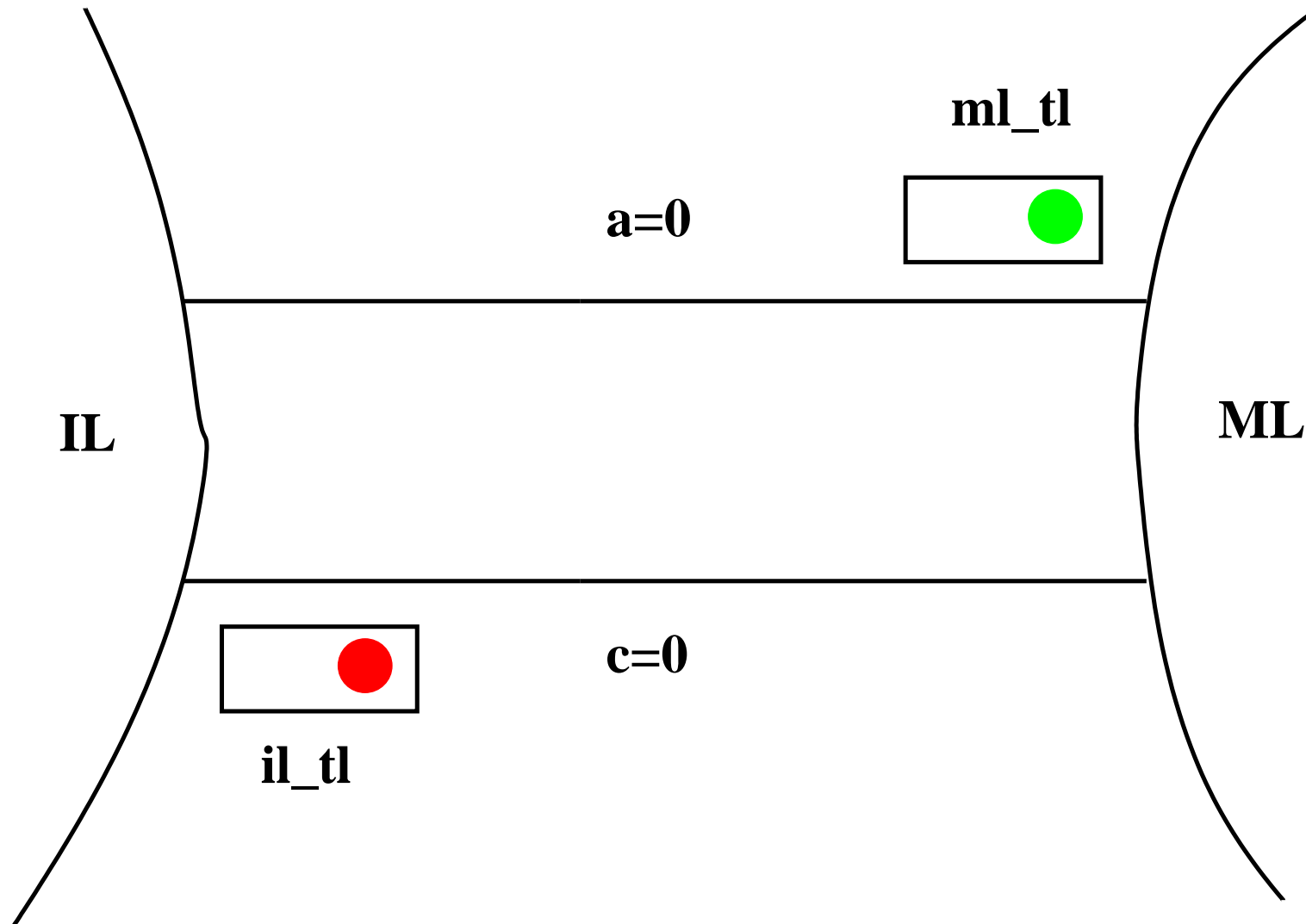
# ML\_tl\_green and IL\_tl\_green can run for ever



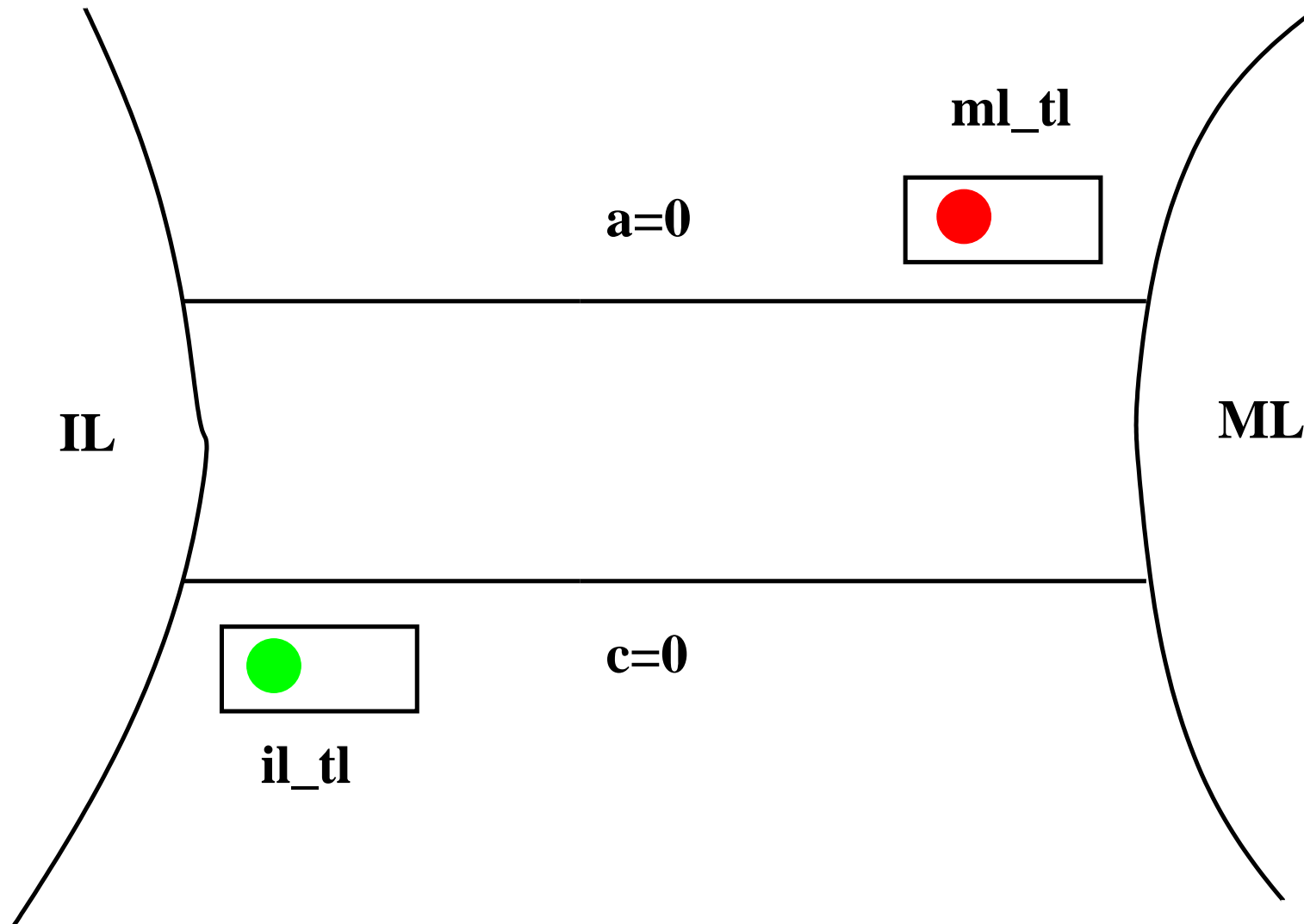
# ML\_tl\_green and IL\_tl\_green can run for ever



# ML\_tl\_green and IL\_tl\_green can run for ever



# ML\_tl\_green and IL\_tl\_green can run for ever



- Allowing each light **to turn green** only when at least one car has **passed in the other direction**
- For this, we introduce **two additional variables**:

**inv2\_6:**  $ml\_pass \in \{0, 1\}$

**inv2\_7:**  $il\_pass \in \{0, 1\}$

# Modifying Events ML\_out\_1 and ML\_out\_2

---

ML\_out\_1

**when**

$ml\_tl = \text{green}$

$a + b + 1 \neq d$

**then**

$a := a + 1$

$ml\_pass := 1$

**end**

ML\_out\_2

**when**

$ml\_tl = \text{green}$

$a + b + 1 = d$

**then**

$a := a + 1$

$ml\_tl := \text{red}$

$ml\_pass := 1$

**end**

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

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

# Modifying Events ML\_tl\_green and IL\_tl\_green

---

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

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

We exhibit the following variant

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

$$ml\_tl = \text{red}$$

$$a + b < d$$

$$c = 0$$

$$il\_pass = 1$$

$\Rightarrow$

$$il\_pass + 0 <$$

$$ml\_pass + il\_pass$$

$$il\_tl = \text{red}$$

$$b > 0$$

$$a = 0$$

$$ml\_pass = 1$$

$\Rightarrow$

$$ml\_pass + 0 <$$

$$ml\_pass + il\_pass$$

This cannot be proved. This suggests the following invariants:

$$\text{inv2\_8: } ml\_tl = \text{red} \Rightarrow ml\_pass = 1$$

$$\text{inv2\_9: } il\_tl = \text{red} \Rightarrow il\_pass = 1$$

$$0 < d$$

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

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

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

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

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

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

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

$$ml\_tl = \text{red} \Rightarrow ml\_pass = 1$$

$$il\_tl = \text{red} \Rightarrow il\_pass = 1$$

$\Rightarrow$

$$(ml\_tl = \text{red} \wedge a + b < d \wedge c = 0 \wedge il\_pass = 1) \vee$$

$$(il\_tl = \text{red} \wedge a = 0 \wedge b > 0 \wedge ml\_pass = 1) \vee$$

$$ml\_tl = \text{green} \vee il\_tl = \text{green} \vee a > 0 \vee c > 0$$

## No Deadlock (2)

The previous statement reduces to the following, which is true

$$0 < d$$

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

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

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

$$\Rightarrow$$

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

$$(a = 0 \wedge b > 0) \vee$$

$$a > 0 \vee$$

$$c > 0$$

$$\rightsquigarrow$$

$$0 < d$$

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

$$\Rightarrow$$

$$b < d \vee b > 0$$

- Thanks to the **proofs**:
  - We discovered 4 errors
  - We introduced several additional invariants
  - We corrected 4 events
  - We introduced 2 more variables

# Conclusion: we Introduced the Superposition Rule

128

<p>Properties of constants Abstract invariants Concrete invariants Concrete guards <math>\Rightarrow</math> Same actions on common variables</p>	<p>EQL_REF</p>
--	----------------

**constants:**  $d$

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

**inv2\_1:**  $ml\_tl \in \{\text{red}, \text{green}\}$

**inv2\_2:**  $il\_tl \in \{\text{red}, \text{green}\}$

**inv2\_3:**  $ml\_tl = 1 \Rightarrow a + b < d \wedge c = 0$

**inv2\_4:**  $il\_tl = 1 \Rightarrow 0 < b \wedge a = 0$

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

**inv2\_6:**  $ml\_pass \in \{0, 1\}$

**inv2\_7:**  $il\_pass \in \{0, 1\}$

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

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

**variant2:**  $ml\_pass + il\_pass$

# Summary of Second Refinement: the Event (1)

---

ML\_out\_1

**when**

$ml\_tl = \text{green}$

$a + b + 1 \neq d$

**then**

$a := a + 1$

$ml\_pass := 1$

**end**

ML\_out\_2

**when**

$ml\_tl = \text{green}$

$a + b + 1 = d$

**then**

$a := a + 1$

$ml\_pass := 1$

$ml\_tl := \text{red}$

**end**

# Summary of Second Refinement: the Event (2)

---

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

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

# Summary of Second Refinement: the Event (3)

---

```
ML_tl_green
  when
     $ml\_tl = \text{red}$ 
     $a + b < d$ 
     $c = 0$ 
     $il\_pass = 1$ 
  then
     $ml\_tl := \text{green}$ 
     $il\_tl := \text{red}$ 
     $ml\_pass := 0$ 
  end
```

```
IL_tl_green
  when
     $il\_tl = \text{red}$ 
     $0 < b$ 
     $a = 0$ 
     $ml\_pass = 1$ 
  then
     $il\_tl := \text{green}$ 
     $ml\_tl := \text{red}$ 
     $il\_pass := 0$ 
  end
```

# Summary of Second Refinement: the Event (4)

---

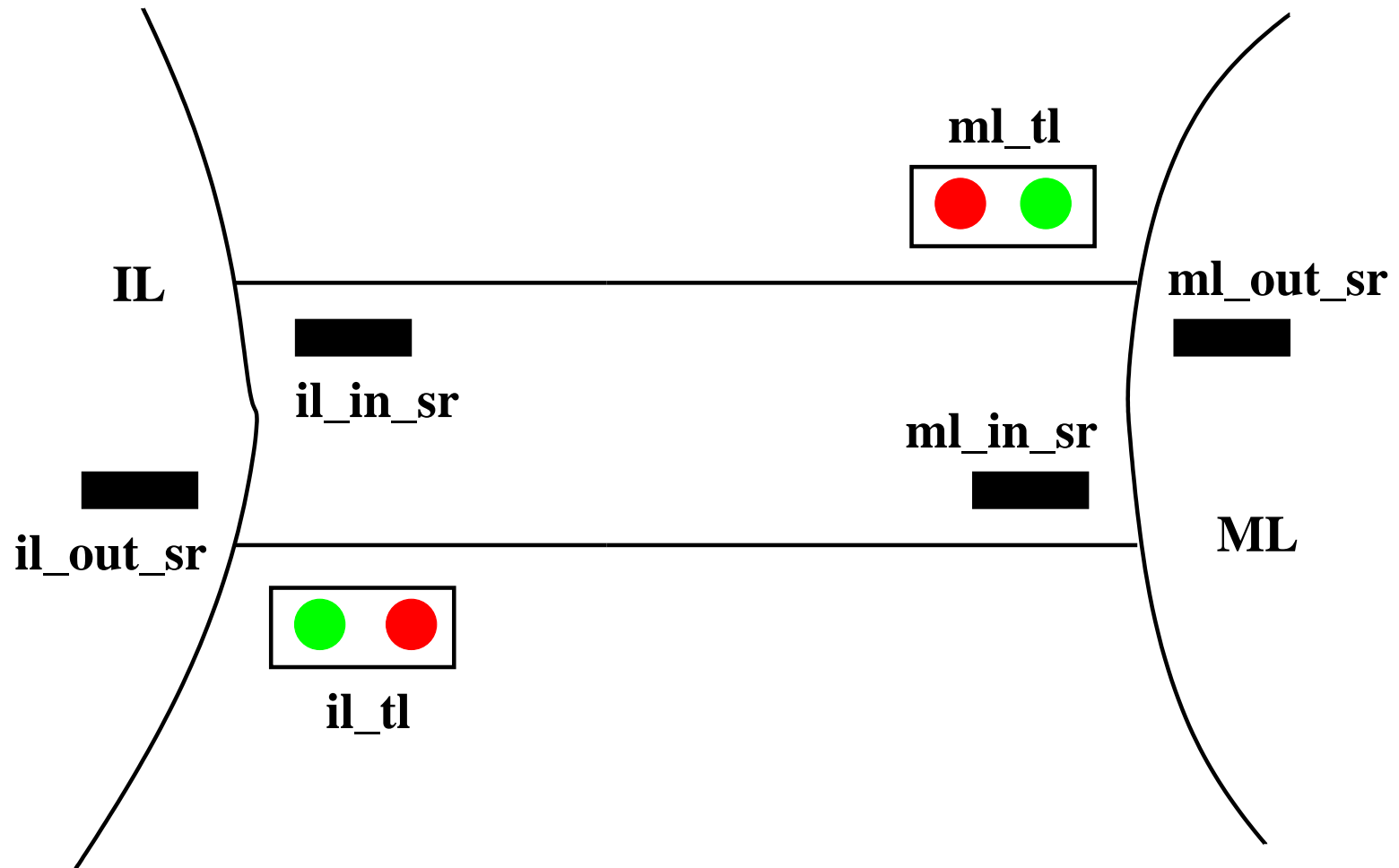
- These events are identical to their abstract versions

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

```
IL_in
  when
     $0 < a$ 
  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)

# Third Refinement: Adding Car Sensors



**constants:**  $d$

**variables:**  $a, b, c,$   
 $ml\_tl,$   
 $il\_tl,$   
 $ml\_pass,$   
 $il\_pass,$   
 $ml\_out\_sr,$   
 $ml\_in\_sr,$   
 $il\_out\_sr,$   
 $il\_in\_sr$

**inv3\_1:**  $ml\_out\_sr \in \{0, 1\}$

**inv3\_2:**  $ml\_in\_sr \in \{0, 1\}$

**inv3\_3:**  $il\_out\_sr \in \{0, 1\}$

**inv3\_4:**  $il\_in\_sr \in \{0, 1\}$

**inv3\_5:**  $il\_in\_sr = 1 \Rightarrow a > 0$

**inv3\_6:**  $il\_out\_sr = 1 \Rightarrow b > 0$

**inv3\_7:**  $ml\_in\_sr = 1 \Rightarrow c > 0$

# Refining Abstract Events (1)

```
ML_out_1
when
   $ml\_tl = \text{green}$ 
   $a + b + 1 \neq d$ 
   $ml\_out\_sr = 1$ 
then
   $a := a + 1$ 
   $ml\_pass := 1$ 
   $ml\_out\_sr := 0$ 
end
```

```
ML_out_2
when
   $ml\_tl = \text{green}$ 
   $a + b + 1 = d$ 
   $ml\_out\_sr = 1$ 
then
   $a := a + 1$ 
   $ml\_tl := \text{red}$ 
   $ml\_pass := 1$ 
   $ml\_out\_sr := 0$ 
end
```

# Refining Abstract Events (2)

```
IL_out_1
  when
    il_tl = green
    b ≠ 1
    il_out_sr = 1
  then
    b := b - 1
    c := c + 1
    il_pass := 1
    il_out_sr := 0
  end
```

```
IL_out_2
  when
    il_tl = green
    b = 1
    il_out_sr = 1
  then
    b := b - 1
    c := c + 1
    il_tl := red
    il_pass := 1
    il_out_sr := 0
  end
```

# Refining Abstract Events (3)

```
ML_in
  when
    ml_in_sr = 1
  then
     $c := c - 1$ 
    ml_in_sr := 0
  end
```

```
IL_in
  when
    il_in_sr = 1
  then
     $a := a - 1$ 
     $b := b + 1$ 
    il_in_sr := 0
  end
```

# Refining Abstract Events (4)

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

```
IL_tl_green
when
  il_tl = red
   $a = 0$ 
  ml_pass = 1
  il_out_sr = 1
then
  il_tl := green
  ml_tl := red
  il_pass := 0
end
```

# Adding New Events (1)

---

```
ML_out_arr
  when
    ml_out_sr = 0
  then
    ml_out_sr := 1
  end
```

```
ML_in_arr
  when
    ml_in_sr = 0
    c > 0
  then
    ml_in_sr := 1
  end
```

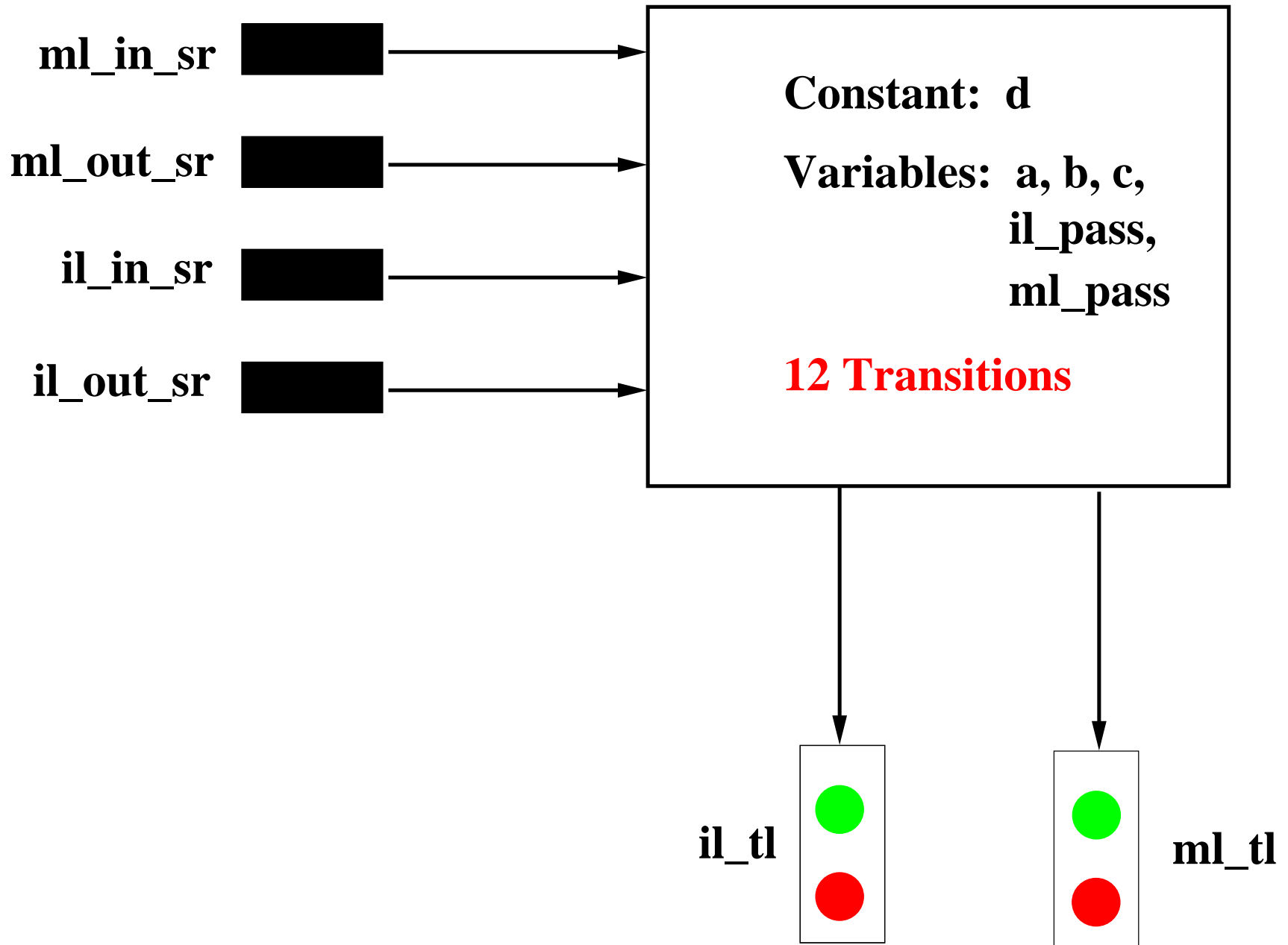
# Adding New Events (1)

---

```
IL_in_arr
  when
    il_in_sr = 0
    a > 0
  then
    il_in_sr := 1
  end
```

```
IL_out_arr
  when
    il_out_sr = 0
    b > 0
  then
    il_out_sr := 1
  end
```

# Final Structure of the Controller



# A Short Demonstration

---

- The constant  $d$  is set to 4
- We have 5 cars named 1, 2, 3, 4, and 5
- The 12 transitions have been translated in an obvious way.
- Transitions  $ML\_out$  and  $IL\_out$  have been randomized
- This simulation worked first time

- **What** is to be **systematically** proved?
  - **Invariant** preservation
  - **Correct refinements** of transitions
  - **No divergence** of new transitions
  - **No deadlock** introduced in refinements
  
- **When** are these proofs done?

- **Who** states what is to be proved?
  - An automatic tool: **the Proof Obligation Generator**
  
- **Who** is going to perform these proofs?
  - An automatic tool: **the Prover**
  - Sometimes helped by the Engineer (**interactive proving**)

- **Three basic tools:**
  - Proof Obligation Generator
  - Prover
  - Model translators into Hardware or Software languages
- These tools are embedded into a **Development Data Base**
- Such tools already exist: **Atelier B** and **Click'n'Prove**

# Summary of Proofs on Example

---

- This development required **125 proofs**
- Only **4 of them were interactive** (easy)
- **97%** proved automatically

# Summary of Mathematical Notations (1)

---

$P \wedge Q$	conjunction
$P \vee Q$	disjunction
$P \Rightarrow Q$	implication
$\neg P$	negation
$x \in S$	set membership operator

# Summary of Mathematical Notations (2)

$\mathbb{N}$	set of Natural Numbers: $\{0, 1, 2, 3, \dots\}$
$\mathbb{Z}$	set of Integers: $\{0, 1, -1, 2, -2, \dots\}$
$\{a, b, \dots\}$	set defined in extension
$a + b$	addition of $a$ and $b$
$a - b$	subtraction of $a$ and $b$

# Summary of Mathematical Notations (3)

---

$a * b$	product of $a$ and $b$
$a = b$	equality relation
$a \leq b$	smaller than or equal relation
$a < b$	smaller than relation

- For the init event in the initial model

Properties of the constants $\Rightarrow$ Modified Invariants	INI_INV
---	---------

- For other events in the initial model

Properties of the constants Invariants Guard of the event $\Rightarrow$ Modified Invariants	INV
---	-----

- This rule is not mandatory

Property of the constant Invariants $\Rightarrow$ Disjunction of the guards	DLF
--	-----

# Refinement Rules (1): Guard Strengthening

---

- For old events only

Properties of the constants Abstract invariants Concrete invariants Concrete guards $\Rightarrow$ Abstract guards	GRD_REF
--	---------

# Refinement Rules (2): Invariant Establishment

---

- For init event only

Properties of the constants $\Rightarrow$ Modified concrete invariants	INI_INV_REF
--	-------------

# Refinement Rules (3): Invariant Preservation

- For all events (except init)
- New events refine an implicit non-guarded event with skip action

Properties of the constants Abstract invariant Concrete invariant Concrete guard $\Rightarrow$ Modified concrete invariant	INV_REF
---	---------

# Refinement Rules (4): Non-divergence of New Events

160

- For new events only

Properties of the constants Abstract invariants Concrete invariants Concrete guard of a new event $\Rightarrow$ Variant $\in \mathbb{N}$	WFD_REF1
---	----------

# Refinement Rules (5): Non-divergence of New Events

161

- For new events only

Properties of the constants Abstract invariants Concrete invariants Disj. of abs. guards $\Rightarrow$ Disj. of conc. guards	WFD_REF2
---	----------

- Global proof rule

Properties of the constants Abstract invariants Concrete invariants Disjunction of abstract guards $\Rightarrow$ Disjunction of concrete guards	DLF_REF
--	---------

- For old events (in case of superposition)

Properties of constants Abstract invariants Concrete invariants Concrete guards $\Rightarrow$ Same actions on common variables	EQL_REF
---	---------