(IV) Event_B: Examples

J.-R. Abrial

April 2005

Version 3

Event_B: Examples

1 Introduction

In this document, we provide some examples to illustrate what has been presented in the three other companion documents. In the first of this examples in section 2 a small reactive system, whose goal is to control cars on a bridge, is presented. The second example in section 4 shows how one can decompose a system. And in the third example in section 5 the usage of generic instantiation is presented.

2 Cars on a Bridge

2.1 Requirements

The system we are going to build is a piece of software connected to some equipment. Its goal is to control cars on a narrow bridge. This bridge is supposed to link the mainland to a small island.

The system is controlling cars on a bridge connecting the mainland to an island	FUN-1

This controller is equipped with two traffic lights.

The system is equipped with two traffic lights with two colors: green and red	EQP-1

One of the traffic lights is situated on the mainland and the other one on the island. Both are close to the bridge.

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

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

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

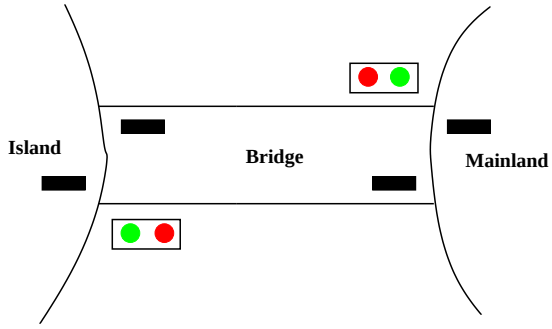
There are also some car sensors situated at both ends of the bridge.

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

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 respectively.

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

The pieces of equipment which have been described are illustrated on the following figure:



This system has two main constraints: the number of cars on the bridge and island is limited,

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

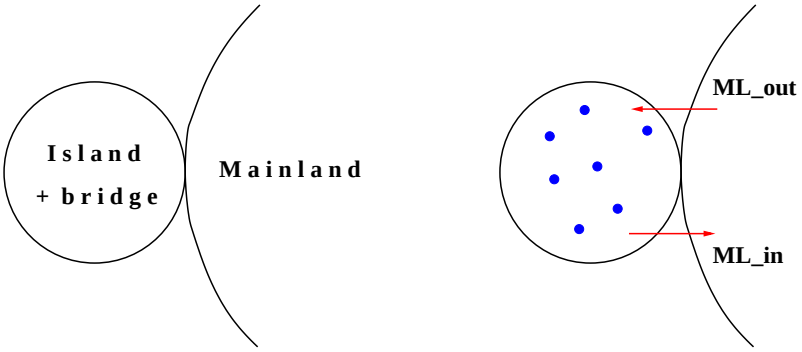
and the bridge is one way.

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

2.2 Initial Model: Limiting the Number of Cars

The first model we are going to construct is very simple. We do not consider at all the various pieces of equipment, namely the traffic lights and sensors. Such equipment will be introduced in subsequent refinements. Likewise, we do not even consider the bridge, only a *compound* made of the bridge and the island together.

As a useful analogy, we suppose to see the situation from very high in the sky. Although we cannot see the bridge, we suppose however that we can "see" the cars in the island-bridge and observe the two transitions, ML_out and ML_in, corresponding to cars entering and leaving the island-bridge compound. All this is illustrated on the following figures:



Formalizing the state. The state is first made of a simple context containing a constant d which is a natural number denoting the maximum number of cars allowed to be in the island-bridge at the same time. This is formalized by the property named prp0_1.

The state is also made of a variable n denoting the actual number of cars in the island-bridge at a given moment. Two invariants named inv0_1 and inv0_2 are used to define the variable n. Invariant inv0_1 says that n is a natural number. The *first basic requirement* of our system, namely FUN_2, is taken into account at this stage by stating in inv0_2 that the number n of cars in the compound is always smaller than or equal to the maximum number d. Here is the formal state:

constants: d **variables:** n	prp0_1 : $d \in \mathbb{N}$	inv0_1 : $n \in \mathbb{N}$ inv0_2 : $n \leq d$

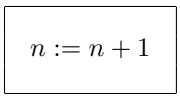
Events. At this stage we can observe two transitions corresponding to cars entering the island-bridge compound or leaving it. Here is an illustration of the situation just before and just after an occurrence of the first event, ML_out. As can be seen, the number of cars in the compound is incremented as a result of this event.

Before **After**

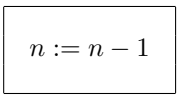
Likewise, here is the situation just before and just after an occurrence of the second event, ML_in. As can be seen, the number of cars in the compound is decremented as a result of this event.

Before **After**

These two events can then be defined in a simple way as follows:



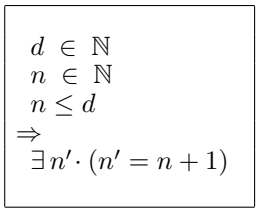
ML_out

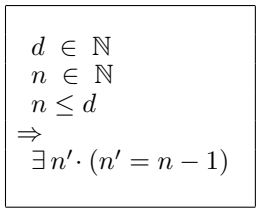


ML_in

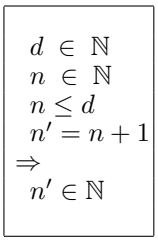
The before-after predicates corresponding to these event actions are straightforward, namely $n' = n + 1$ for ML_out, and $n' = n - 1$ for ML_in.

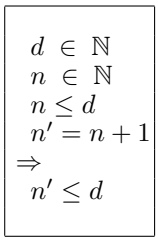
Proving Invariant Preservation. Rule FIS applied to both events leads to the following, which holds trivially:

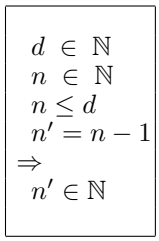


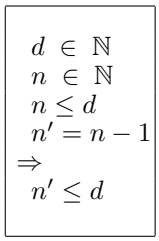


Invariant preservation rule INV applied to both events leads to the following statements:

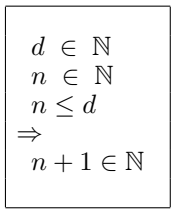


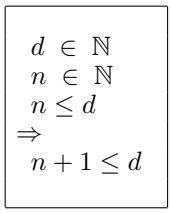


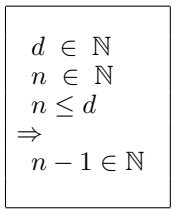


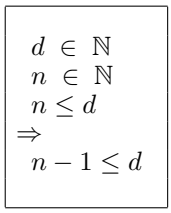


The variable n' can be eliminated by replacing it by its value, yielding:









We notice that in the second case, $n + 1 \leq d$ cannot be proved when n is already equal to d. And in the third case, $n - 1 \in \mathbb{N}$ cannot be proved when n is already equal to 0. This is so because the proposed events ML_out and ML_in are too primitive.

Improving the two Events We have to add *guards* to our events. They denote the necessary conditions for these events to be enabled. For event ML_out to be enabled, it is required that n be strictly smaller than d. And for event ML_in to be enabled, it is required that n be strictly positive. All this is indicated in the following new versions of these events:

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

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

The statements to prove by applying rule INV are modified accordingly and are now easily provable:

$$d \in \mathbb{N} \quad n \in \mathbb{N} \quad n \leq d \quad n < d \Rightarrow n + 1 \in \mathbb{N}$$

$$d \in \mathbb{N} \quad n \in \mathbb{N} \quad n \leq d \quad n < d \Rightarrow n + 1 \leq d$$

$$d \in \mathbb{N} \quad n \in \mathbb{N} \quad n \leq d \quad 0 < n \Rightarrow n - 1 \in \mathbb{N}$$

$$d \in \mathbb{N} \quad n \in \mathbb{N} \quad n \leq d \quad 0 < n \Rightarrow n - 1 \leq d$$

Proving Deadlock Freeness Since our events are now guarded, it means that our system might deadlock when both guard are together false. Clearly, we want to avoid this happening. We have thus to prove rule DLKF stating that one of the two guards is always true. In other words, cars can always either enter the compound or leave it. This is to be proved under the property of the constant and under the invariant:

$$d \in \mathbb{N} \quad n \in \mathbb{N} \quad n \leq d \Rightarrow n < d \ \lor \ 0 < n$$

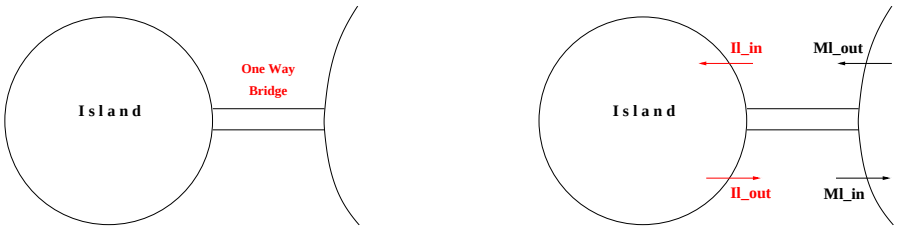
But we now discover that this statement cannot be proved when d is zero, which is quite obvious since then no car can ever enter the compound nor, a fortiori, leave it. We have thus to add the following property, named prp0_2, which was obviously forgotten:

prp0_2 : $0 < d$

Conclusion of the initial model As we have seen, the proofs (or rather the failures of the proofs) allowed us to discover that our events were too primitive and also that one property was missing in the context for the constant d.

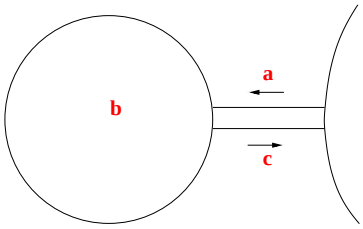
2.3 First Refinement: Introducing the One Way Bridge

In this first refinement, we introduce the bridge. This means that we are able to observe our system more accurately. Together with this more accurate observation, we can also see more events, namely cars entering and leaving the island. These events are called IL_in and IL_out. Note that events ML_out and ML_in which were present in the initial model still exist in this refinement: they now correspond to cars leaving the mainland and entering the bridge or leaving the bridge and entering the mainland. All this is illustrated in the following figures:



Refining the state. The state which was defined by the constant d and variable n in the initial model now becomes more accurate. The constant d remains, but the variable n is now replaced by three new variables. This is because now we can see cars on the bridge and on the island, something which we could not distinguish in the previous abstraction. Moreover, we can see where cars on the bridge are going: either towards the island or towards the mainland.

For these reasons, the state is now represented by means of three variables a, b, and c. Variable a denotes the number of cars on the bridge and going to the island, variable b denotes the number of cars on the island, and variable c denotes the number of cars on the bridge and going to the mainland. This is illustrated on the following figure:



Formally the refined state is represented by a number of new invariants. First, variables a, b, and c are all natural numbers. This is stated in invariant inv1_1, inv1_2, and inv1_3. Then we express in the, so-called, gluing invariant, that the sum of these variables is equal to the previous abstract variable n which now disappears. This is expressed in invariant inv1_4. And finally, we state that the bridge is one way, this is our basic requirement FUN-3, by saying that a or c is 0. Clearly they cannot be both positive since the bridge is one way. Note that they can be both 0 however. This is expressed in invariant inv1_5. Here is the formalization of these invariants:

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

Refining the Abstract Events. The two abstract events ML_out and ML_in have now to be refined as they are not dealing with variable n any more but with variables a, b, and c. Here is the proposed refinement of event ML_out, which is presented together with its abstraction.

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

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

Likewise, here is the proposed refined version of event ML_in, which is also presented together with its abstraction.

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

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

Proving that the Refinement of Abstract Events are Correct. Applying Rule FIS_REF to the refined event holds trivially. Likewise, applying rule GRD_REF to both refined events leads to the following, which holds trivially:

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

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

Applying rule INV_REF to both events leads to the following after some simplifications. Both statements hold trivially.

$d \in \mathbb{N}$
$0 < d$
$n \in \mathbb{N}$
$n \leq d$
$a \in \mathbb{N}$
$b \in \mathbb{N}$
$c \in \mathbb{N}$
$a + b + c = n$
$a = 0 \ \lor \ c = 0$
$a + b < d$
$c = 0$
$\Rightarrow$
$a + 1 \in \mathbb{N}$
$a + 1 + b + c = n + 1$
$a + 1 = 0 \ \lor \ c = 0$

$d \in \mathbb{N}$
$0 < d$
$n \in \mathbb{N}$
$n \leq d$
$a \in \mathbb{N}$
$b \in \mathbb{N}$
$c \in \mathbb{N}$
$a + b + c = n$
$a = 0 \ \lor \ c = 0$
$0 < c$
$\Rightarrow$
$c - 1 \in \mathbb{N}$
$a + b + c - 1 = n - 1$
$a = 0 \ \lor \ c - 1 = 0$

Introducing New Events. We now have to introduce some new events corresponding to cars entering and leaving the island. Next are the proposed new events. A can be seen , such events indeed refine skip as they only modify variables a, b and c in such a way that the abstract variable n remains constant according to invariant inv1_4.

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

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

Proving that the New Events are Correct We leave it as an exercise to the reader to state and prove that these events refine skip. We now have to prove that new events do not diverge. For this, we have to exhibit a variant and prove that is decreased by both new events. The proposed variant is the following:

variant_1 : $2 * a + b$

Applying rule WFD_REF leads to the following obvious statements to prove:

$$
\begin{array}{l}
d \in \mathbb{N} \\
0 < d \\
n \in \mathbb{N} \\
n \leq d \\
a \in \mathbb{N} \\
b \in \mathbb{N} \\
c \in \mathbb{N} \\
a + b + c = n \\
a = 0 \ \lor \ c = 0 \\
0 < a \\
\Rightarrow \\
2 * (a - 1) + b + 1 < 2 * a + b
\end{array}
$$

$$
\begin{array}{l}
d \in \mathbb{N} \\
0 < d \\
n \in \mathbb{N} \\
n \leq d \\
a \in \mathbb{N} \\
b \in \mathbb{N} \\
c \in \mathbb{N} \\
a + b + c = n \\
a = 0 \ \lor \ c = 0 \\
0 < b \\
a = 0 \\
\Rightarrow \\
2 * a + (b - 1) < 2 * a + b
\end{array}
$$

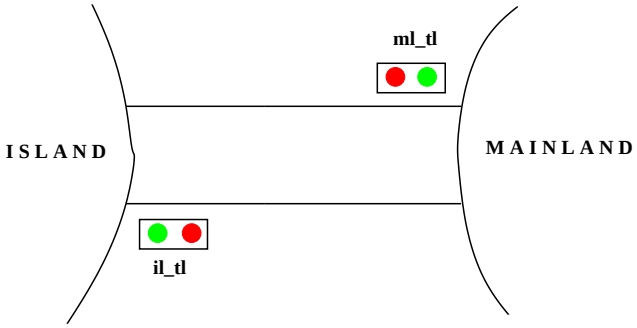
No Deadlock. Finally, we have to prove that the refined events and new events do not together deadlock. This is so because the abstraction did not deadlock. Rule W_DLK leads to the following to prove (it is simplified because we already proved that the abstraction did not deadlock):

$$
\begin{array}{l}
d \in \mathbb{N} \\
0 < d \\
n \in \mathbb{N} \\
n \leq d \\
a \in \mathbb{N} \\
b \in \mathbb{N} \\
c \in \mathbb{N} \\
a + b + c = n \\
a = 0 \ \lor \ c = 0 \\
\Rightarrow \\
(a + b < d \ \land \ c = 0) \ \lor \\
c > 0 \ \lor \\
a > 0 \ \lor \\
(b > 0 \ \land \ a = 0)
\end{array}
$$

2.4 Second Refinement: Introducing the Traffic Lights

At this point, the situation is a bit magic. It seems that car drivers can count cars and thus decide to enter into the bridge from the mainland (event ML_out) or from the island (event IL_out). In reality, as we know, the drivers follows the indication of the traffic lights, they clearly do not the count of the cars, which is impossible.

This refinement then consists in introducing first the two traffic lights, named ml_tl and il_tl, then the corresponding invariants, and finally some new events that are able to change the colors of the traffic lights. The next figure illustrates the new physical situation which can be observed:



Refining the State Two new variables are introduced, ml_tl (for mainland traffic light) and il_tl (for island traffic light). These variables take values 0 (for red) or 1 (for green): this is formalized in invariants named inv2_1 and inv2_2. Since drivers are allowed to pass when traffic lights are green (1), we better ensure by two invariants named inv2_3 and inv2_4 that when ml_tl is green then the guard of events ML_out holds, and that when il_tl is green then the guard of events IL_out holds. Here is the refined state:

constants : d

variables : $a, b, c,$
ml_tl, il_tl

inv2_1 : $ml\_tl \in \{0, 1\}$

inv2_2 : $il\_tl \in \{0, 1\}$

inv2_3 : $ml\_tl = 1 \Rightarrow a + b < d \wedge c = 0$

inv2_4 : $il\_tl = 1 \Rightarrow 0 < b \wedge a = 0$

Refining Abstract Events Events ML_out and IL_out are now refined by changing their guards to the test of the green value of the corresponding traffic lights. This is exactly what car drivers are doing. This is here where we implicitly assume that drivers obey the traffic lights, as indicated by requirements EQP-3. Note that events IL_in and ML_in are not modified in this refinement. Here is the new version of event ML_out presented together with its abstraction:

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

concrete_ML_out
when
$ml\_tl = 1$
then
$a := a + 1$
end

And here is the new version of event IL_out presented together with its abstraction:

abstract_IL_out
when
$a = 0$
$0 < b$
then
$b, c := b - 1, c + 1$
end

concrete_IL_out
when
$il\_tl = 1$
then
$b, c := b - 1, c + 1$
end

Introducing New Events We have to introduce two new events to turn the value of the traffic lights color to green when they are red and when the conditions are appropriate. The appropriate conditions, once again, are exactly the guards of the abstract events ML_out and IL_out. Here are the proposed new events:

ML_tl_green
when
$ml\_tl = 0$
$a + b < d$
$c = 0$
then
$ml\_tl := 1$
end

IL_tl_green
when
$il\_tl = 0$
$0 < b$
$a = 0$
then
$il\_tl := 1$
end

Proving that the Events are Correct. Proving that the concrete events correctly refine their abstraction is easily done and left to the reader. But we have some problems with proving that they maintain the new invariants. For example, here is the statement to be proved (after some simplification) concerning the preservation of invariant inv2_4 by event ML_out:

$$
\begin{array}{l}
a \in \mathbb{N} \\
il\_tl = 1 \;\Rightarrow\; a = 0 \;\wedge\; 0 < b \\
ml\_tl = 1 \\
\Rightarrow \\
il\_tl = 1 \;\Rightarrow\; \underline{a + 1} = 0 \;\wedge\; 0 < b
\end{array}
$$

This statement cannot be proved when il_tl is green (1). Likewise the following statement concerning the preservation of invariant inv2_3 by event IL_out cannot be proved when ml_tl is green (1):

$$
\begin{array}{l}
c \in \mathbb{N} \\
ml\_tl = 1 \;\Rightarrow\; c = 0 \;\wedge\; a + b < d \\
il\_tl = 1 \\
\Rightarrow \\
ml\_tl = 1 \;\Rightarrow\; \underline{c + 1} = 0 \;\wedge\; a + b - 1 < d
\end{array}
$$

What these failures show is that both lights cannot be green at the same time, on obvious fact, which we have forgotten to state. We thus now introduce it as an extra invariant:

inv2_5 : $\quad \text{ml\_tl} = 0 \ \lor \ \text{il\_tl} = 0$

But this new invariant has to be preserved and this is clearly not the case with the proposed new events ML_tl_green and IL_tl_green unless we correct them by turning to red the other traffic light, yielding:

ML_tl_green
when
$ml\_tl = 0$
$a + b < d$
$c = 0$
then
$ml\_tl := 1$
$\underline{il\_tl := 0}$
end

IL_tl_green
when
$il\_tl = 0$
$0 < b$
$a = 0$
then
$il\_tl := 1$
$\underline{ml\_tl := 0}$
end

When trying to prove the preservation of invariant inv2_3 by event ML_out, we are again in trouble. Here is the corresponding (simplified) statement to prove:

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

As can be seen, this statement cannot be proved when $a + 1 + b$ is equal to d unless ml_tl is set to red (0). In fact, when $a + 1 + b$ is equal to d, it means that the entering car is the last one allowed to enter at this stage because more cars would violate requirement FUN_3, which says that there are no more than d cars in the island and bridge. This indicates that event ML_out has to be split into two events (both refining their abstraction however) as follows:

ML_out_1
when
$ml\_tl = 1$
$\underline{a + b + 1 \neq d}$
then
$a := a + 1$
end

ML_out_2
when
$ml\_tl = 1$
$\underline{a + b + 1 = d}$
then
$a := a + 1$
$\underline{ml\_tl := 0}$
end

Likewise, invariant inv2_4 cannot be maintained by event IL_out when b is equal to 1. In this case the last car is leaving the island. As a consequence the island traffic light has to turn red. Here is the simplified statement which has to be proved:

$$\begin{array}{l} il\_tl = 1 \ \Rightarrow \ a = 0 \ \land \ 0 < b \\ il\_tl = 1 \\ \Rightarrow \\ il\_tl = 1 \ \Rightarrow \ a = 0 \ \land \ 0 < \underline{b - 1} \end{array}$$

As for event ML_out, we have to split event IL_out as follows:

IL_out_1
when
$il\_tl = 1$
$b \neq 1$
then
$b, c := b - 1, c + 1$
end

IL_out_2
when
$il\_tl = 1$
$b = 1$
then
$b, c := b - 1, c + 1$
$il\_tl := 0$
end

No divergence of new events We have now to prove that the new events cannot diverge for ever. For this, we must exhibit a certain variant that must be decreased by the new events. In fact, it turns out to be impossible. For instance, when a and c are both 0, meaning that there is no car on the bridge in either direction then the traffic lights could freely change color for ever as one can figure out by looking at the new events ML_tl_green and IL_tl_green:

ML_tl_green
when
$ml\_tl = 0$
$a + b < d$
$c = 0$
then
$ml\_tl := 1$
$il\_tl := 0$
end

IL_tl_green
when
$il\_tl = 0$
$0 < b$
$a = 0$
then
$il\_tl := 1$
$ml\_tl := 0$
end

What could then happen is that the light colors are changing so rapidly that the drivers can never pass. We have to make the color changing in a more disciplined way, that is only when some car has passed in the other direction. For this we introduce two more variables ml_pass and il_pass. Each of them can take two values 0 and 1. When ml_pass is equal to 1 it means that one car at least has passed on the bridge going to the island since mainland traffic light last turned green, and similarly when il_pass is equal to 1. These variables are formalized in the following invariants:

inv2_6 : ml_pass $\in \{0, 1\}$

inv2_7 : il_pass $\in \{0, 1\}$

We must now modify events ML_out_1, ML_out_2, IL_out_1, and IL_out_2 to make ml_pass or il_pass to 1 since a car has passed in the proper direction.

ML_out_1
when
$ml\_tl = 1$
$a + b + 1 \neq d$
then
$a := a + 1$
$ml\_pass := 1$
end

ML_out_2
when
$ml\_tl = 1$
$a + b + 1 = d$
then
$a := a + 1$
$ml\_tl := 0$
$ml\_pass := 1$
end

IL_out_1
when
$il\_tl = 1$
$b \neq 1$
then
$b := b - 1$
$c := c + 1$
$il\_pass := 1$
end

IL_out_2
when
$il\_tl = 1$
$b = 1$
then
$b := b - 1$
$c := c + 1$
$il\_tl := 0$
$il\_pass := 1$
end

But we must also modify event ML_tl_green and IL_tl_green to reset ml_pass and il_pass and also add in their guards the conditions $il\_pass = 1$ and $ml\_pass = 1$ respectively in order to be sure that indeed a car has passed in the other direction. This yields the following:

ML_tl_green
when
$ml\_tl = 0$
$a + b < d$
$c = 0$
$il\_pass = 1$
then
$ml\_tl := 1$
$il\_tl := 0$
$ml\_pass := 0$
end

IL_tl_green
when
$il\_tl = 0$
$0 < b$
$a = 0$
$ml\_pass = 1$
then
$il\_tl := 1$
$ml\_tl := 0$
$il\_pass := 0$
end

Having done all that, we can now state what is to be proved in order to guarantee that there is no divergence of the new events. The variant we can exhibit is the following:

variant_2 : $ml\_pass + il\_pass$

And the statements to be proved are the following:

$ml\_tl = 0$
$a + b < d$
$c = 0$
$il\_pass = 1$
$\Rightarrow$
$il\_pass < ml\_pass + il\_pass$

$il\_tl = 0$
$b > 0$
$a = 0$
$ml\_pass = 1$
$\Rightarrow$
$ml\_pass < ml\_pass + il\_pass$

At this point we figure out that it cannot be proved unless $ml\_pass = 1$ in the first case and $il\_pass = 1$ in the second one. We have two solutions: either to strengthen the guards of events ML_tl_green and IL_tl_green (adding to them the extra guards $ml\_pass = 1$ and $il\_pass = 1$ respectively) or adding some extra invariants. The first solution seems to be the more economical one, yielding:

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

IL_tl_green
when
$il\_tl = 0$
$0 < b$
$a = 0$
$ml\_pass = 1$
$\underline{il\_pass = 1}$
then
$il\_tl := 1$
$ml\_tl := 0$
$il\_pass := 0$
end

No Deadlock It remains now to prove that we have no deadlock. The statement to prove is then the disjunction of the various guards with some simplified assumption (we do not need all invariants):

$$
\begin{array}{l}
d \in \mathbb{N} \\
0 < d \\
ml\_tl \in \{0,1\} \\
il\_tl \in \{0,1\} \\
ml\_pass \in \{0,1\} \\
il\_pass \in \{0,1\} \\
a \in \mathbb{N} \\
b \in \mathbb{N} \\
c \in \mathbb{N} \\
\Rightarrow \\
(ml\_tl = 0 \ \wedge \ a + b < d \ \wedge \ c = 0 \ \wedge \ ml\_pass = 1 \ \wedge \ il\_pass = 1) \ \vee \\
(il\_tl = 0 \ \wedge \ a = 0 \ \wedge \ b > 0 \ \wedge \ ml\_pass = 1 \ \wedge \ il\_pass = 1) \ \vee \\
ml\_tl = 1 \ \vee \\
il\_tl = 1 \ \vee \\
a > 0 \ \vee \\
c > 0
\end{array}
$$

This statement can be simplified to the following:

$$
\begin{array}{l}
d \in \mathbb{N} \\
0 < d \\
b \in \mathbb{N} \\
ml\_tl = 0 \\
il\_tl = 0 \\
\Rightarrow \\
(b < d \ \wedge \ ml\_pass = 1 \ \wedge \ il\_pass = 1) \ \vee \\
(b > 0 \ \wedge \ ml\_pass = 1 \ \wedge \ il\_pass = 1)
\end{array}
$$

Unfortunately, this statement cannot be proved unless we add the following two extra invariants:

inv2_8 : $ml\_tl = 0 \Rightarrow ml\_pass = 1$

inv2_9 : $il\_tl = 0 \Rightarrow il\_pass = 1$

As a consequence, we can now remove the guards $ml\_pass = 1$ and $il\_pass = 1$ in events ML_tl_green and IL_tl_green since they are implied by the guards $ml\_tl = 0$ and $il\_tl = 0$. We figure out that our choice of strengthening the guard of these events was the wrong one. We should have added the two invariants. It remains for us to prove that the two new invariants inv2_8 and inv2_9 are indeed preserved by all events. We leave this as an exercise to the reader.

Conclusion of the Second Refinement. During this refinement, we have seen again how the proofs (or rather the failures of the proofs) have helped us correct our mistake or enlarge our model. If fact, we discovered 4 errors, we introduced several extra invariants, we corrected four events, and we introduced two more variables.

3 Third Refinement: Introducing Car Sensors

Let us consider the events ML_out_1, ML_out_2, IL_out_1, IL_out_2, ML_in, and IL_in:

ML_out_1
when
$ml\_tl = 1$
$a + b + 1 \neq d$
then
$a := a + 1$
$ml\_pass := 1$
end

ML_out_2
when
$ml\_tl = 1$
$a + b + 1 = d$
then
$a := a + 1$
$ml\_tl := 0$
$ml\_pass := 1$
end

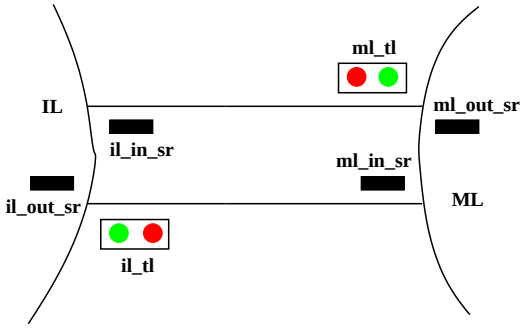
IL_out_1
when
$il\_tl = 1$
$b \neq 1$
then
$b := b - 1$
$c := c + 1$
$il\_pass := 1$
end

IL_out_2
when
$il\_tl = 1$
$b = 1$
then
$b := b - 1$
$c := c + 1$
$il\_tl := 0$
$il\_pass := 1$
end

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

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

We can observe these events happening in the real world, but it is now important to have the controller being aware of them. For this, we introduce in this refinement some sensors able to communicate these transitions to the controllers, namely the passing of cars from the mainland to the bridge and vice-versa and from the bridge to the island and vice-versa. For doing this, we put four such sensors at both ends of the bridge as indicated on the following figure:



A sensor can be in one of two states: either on or off. It is on when a car is “on” it, off otherwise.

As a consequence, we shall enlarge our state with four variables corresponding to each sensor state: ml_out_sr, ml_in_sr, il_out_sr, and il_in_sr. They are defiined in invariants **inv3_1** to **inv3_4**:

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

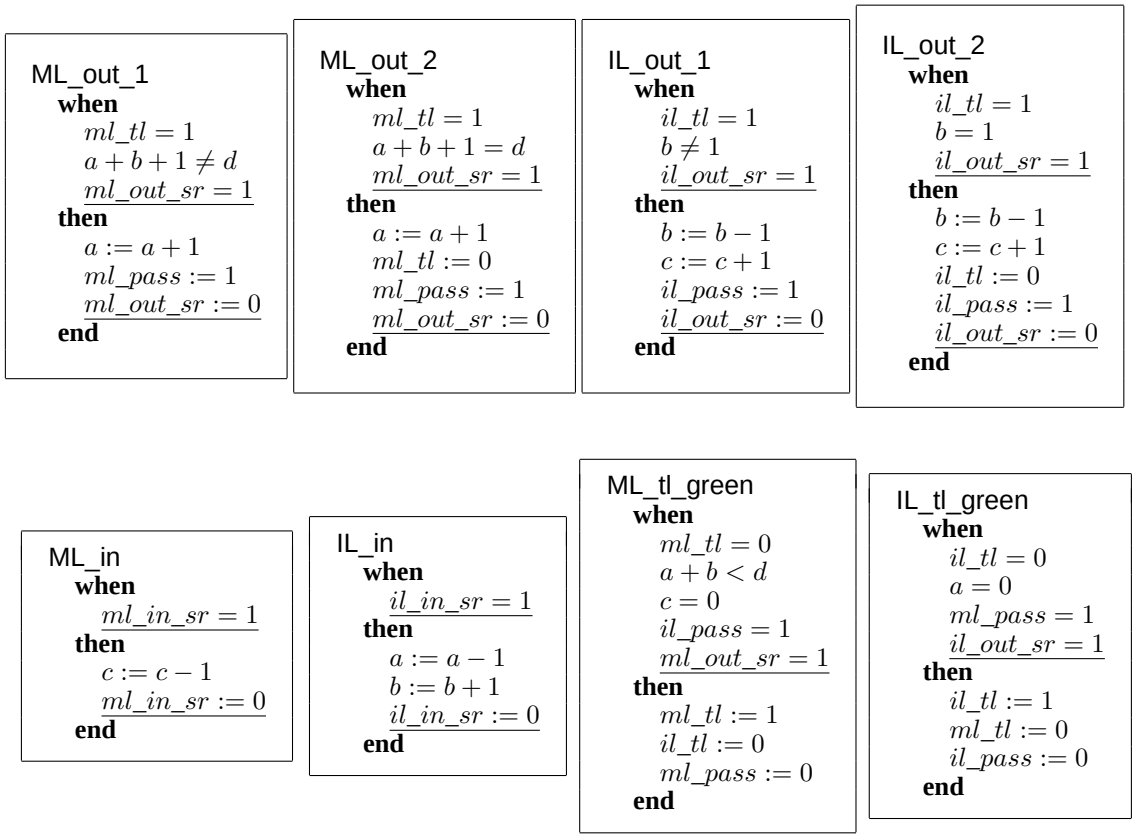
inv3_1 :	$ml\_out\_sr \in \{0, 1\}$
inv3_2 :	$ml\_in\_sr \in \{0, 1\}$
inv3_3 :	$il\_out\_sr \in \{0, 1\}$
inv3_4 :	$il\_in\_sr \in \{0, 1\}$

We also clearly have the new invariants stating that when the state il_in_sr is 1, then a is positive. In other words, there is at least one car on the bridge, namely the one that sits on the sensor il_in_sr. We have similar invariants for il_out_sr and ml_in_sr, yielding:

inv3_5 :	$il\_in\_sr = 1 \Rightarrow a > 0$
inv3_6 :	$il\_out\_sr = 1 \Rightarrow b > 0$
inv3_7 :	$ml\_in\_sr = 1 \Rightarrow c > 0$

3.1 Refining abstract events.

It is now easy to proceed with the refinement of abstract events. This is done in a straightforward fashion as follows:



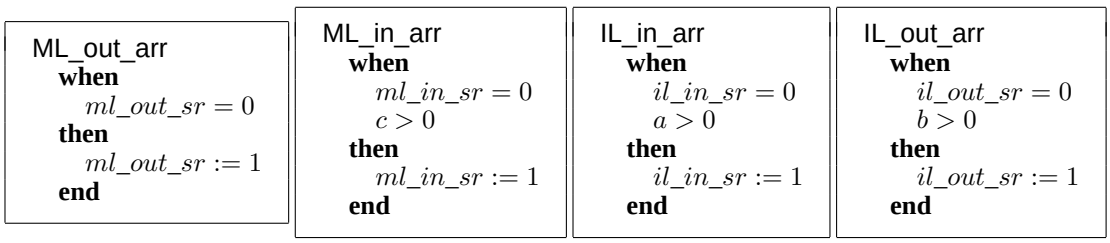
As can be seen, we have added the guards $ml\_out\_sr = 1$ and $il\_out\_sr = 1$ to the events ML_tl_green and IL_tl_green since there is no point in turning a traffic light to green when there is no car willing to pass. Note that this is a *design decision* as opposed to a requirement.

3.2 Correct Refinement

Proving that the previous events refine their abstract counterparts is left as an exercise to the reader.

3.3 Adding New Events

We now add four new events corresponding to cars arriving on the various sensors:



3.4 Refinement of new events.

The new events clearly refine skip as they do not work with old variables. We leave it to the reader to prove that they preserve the new invariants.

3.5 No divergence of new events.

We have to exhibit a variant which is decremented by all new events. Here it is:

variant_3: $4 - (ml\_out\_sr + ml\_in\_sr + il\_out\_sr + il\_in\_sr)$

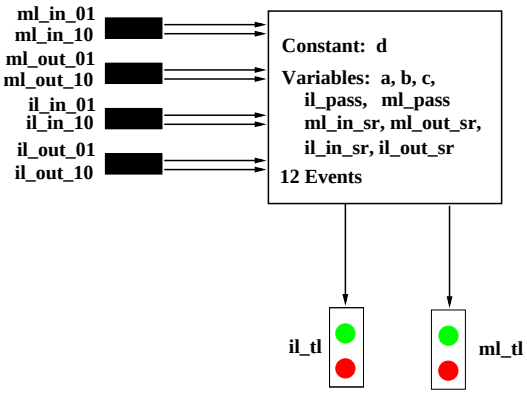
We leave it as an exercise to the reader to prove that this variant is decreased by the new events.

3.6 No Deadlock

Again, we leave it to the reader to prove that this third refinement does not deadlock.

3.7 Conclusion of the Third Refinement

The final structure of the system is shown on the following figure:



As can be seen, we now have one constant, nine variables, eight input wires, two output wires and twelve events.

4 Example with Decomposition: The Two-phase Handshake Protocol

Our next example is a presentation of the very classical two-phase handshake protocol. This protocol is supposed to transfer a file from one agent, the Sender, to another one, the Receiver. These agents are supposed to reside on *different sites*, so that the transfer is not made by a simple copy of the file, it is rather realized gradually by two distinct programs exchanging various kinds of messages on a network. Such programs are working with different contexts and on different machines: the overall protocol is indeed a *distributed program*.